

Microsoft®

РУКОВОДСТВО MICROSOFT® ПО ПРОЕКТИРОВАНИЮ АРХИТЕКТУРЫ ПРИЛОЖЕНИЙ

2^е издание



patterns & practices

Сообщение об авторском праве

Все сведения, представленные в данном документе, включая URL и другие ссылки на Веб-сайты, могут изменяться без уведомления. Если не указано обратное, все компании, организации, продукты, доменные имена, адреса электронной почты, логотипы, люди и события, используемые здесь для примера, являются вымышленными и не имеют никакого отношения к реальным компаниям, организациям, продуктам, доменным именам, адресам электронной почты, логотипам, людям, ситуациям или событиям. Пользователь несет ответственность за соблюдение всех применимых законов об авторском праве. Без ограничения прав, защищенных авторским правом, ни одна часть этого документа не может быть воспроизведена, сохранена или представлена в поисковой системе или передана в любой форме или любыми средствами (электронными, механическими, фотокопированием, звукозаписью или др.) для любых целей без письменного разрешения Microsoft Corporation.

Microsoft может обладать патентами, заявками на патенты, товарными знаками, авторскими или другими правами интеллектуальной собственности, относящимися к предмету этого документа. За исключением случаев, оговоренных лицензионным соглашением Microsoft, предоставление этого документа не дает каких-либо лицензий на эти патенты, товарные знаки, авторские права или другую интеллектуальную собственность.

© 2009 Корпорация Майкрософт. Все права защищены.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, Active Directory, MSDN, Visual Basic, Visual C++, Visual C#, Visual Studio и Win32 являются либо зарегистрированными торговыми марками, либо торговыми марками корпорации Майкрософт в Соединенных Штатах Америки и/или других странах.

Все упоминаемые в данном руководстве имена реально существующих компаний и продуктов могут быть зарегистрированными торговыми марками их владельцев.

Предисловие С. Сомасегара

Данное практическое руководство стало результатом использования собственных технологий для создания продуктов Microsoft и ежедневной работы с заказчиками и партнерами. В нем собраны рекомендации по применению лучших практик для проектирования архитектуры приложения, а также шаблоны и принципы проектирования с использованием наших технологий. Это руководство представляет большую ценность как для разработчиков, так и для архитекторов решений. «*Руководство Microsoft по проектированию архитектуры приложений*» объединило в себе весь опыт, накопленный нами внутри компании, все рекомендации независимых специалистов, пользователей и всего сообщества разработчиков.

Задача этого руководства – помочь архитекторам решений и разработчикам проектировать и создавать более эффективные приложения на платформе Microsoft, обеспечить возможность принятия ключевых решений на ранних этапах создания проекта. В руководстве также рассматриваются конкретные темы, что поможет архитекторам и разработчикам улучшить существующие решения. В создании руководства принимали участие более 25 независимых специалистов и заказчиков.

Продумывая решения в категориях архитектурных шаблонов и принципов построения архитектуры, показателей качества и сквозной функциональности, вы очень быстро придетете к базовой архитектуре приложения и правильному выбору необходимых технологий, шаблонов и ресурсов. После этого руководство поможет определить ключевые области архитектуры приложения, доработка которых обеспечит реализацию конкретного сценария.

Данное руководство включает примеры архитектур для общих типов приложений, таких как Веб-приложения, насыщенные клиентские приложения, RIA, мобильные и сервисные приложения; рекомендации по проектированию показателей качества и сквозной функциональности; подходы к проектированию, которые помогут при создании и доработке архитектуры решений.

Мы уверены, что «*Руководство Microsoft по проектированию архитектуры приложений. 2е издание*» поможет правильно выбрать архитектуру, технологии и шаблоны проектирования, что обеспечит создание более эффективных проектных решений.

С уважением,

С. Сомасегар (S. Somasegar)

Старший вице-президент подразделения разработки

Microsoft

Предисловие Скотта Гатри

Проектирование архитектуры приложения является очень сложной задачей. Доказательство тому – огромное количество книг, статей и документации, посвященных данному вопросу. Понимание архитектуры и лучших практик проектирования для платформы Microsoft по-прежнему вызывает сложности у разработчиков и архитекторов. Руководство «*Application Architecture for .NET: Designing Applications and Services*» (Архитектура приложений для .NET: проектирование приложений и сервисов) сделало огромный вклад в дело раскрытия этой темы, но оно было выпущено в 2002 году.

За это время появилось множество новых технологий. Дж. Д. Мейер (J. D. Meier), Давид Хилл (David Hill) и их команда из группы Microsoft patterns & practices собрали все подробные рекомендации по проектированию приложений и сервисов на платформе Microsoft с использованием передовых практик и технологий. В результате получилось «*Руководство Microsoft по проектированию архитектуры приложений. 2е издание*», руководство, основной целью которого является помочь архитекторам решений и разработчикам создавать эффективные приложения на платформе Microsoft. В руководстве дается обзор .NET Framework, платформы Microsoft, их основных технологий и возможностей, а также приводятся не зависящие от платформы, ориентированные на применение шаблонов и основанные на общих принципах рекомендации, которые обеспечат прочную базу создаваемым приложениям.

В основу данного руководства легли базовые принципы архитектуры и дизайна. Оно включает рекомендации по принятию и проработке ключевых технических решений, дает разъяснения по показателям качества, сквозной функциональности и характеристикам (таким как производительность, безопасность, масштабируемость, удобство и простота обслуживания, развертывание, сетевое взаимодействие и многое другое), которые определяют архитектуру приложения.

Руководство также описывает слои и уровни, которые могут присутствовать в приложении. Каждый уровень/слой рассматривается с точки зрения его основного назначения, функций, возможностей, общих шаблонов проектирования и технологий. Для каждого уровня/слоя предлагаются соответствующие принципы, шаблоны и практики. Наконец, приводятся канонические архетипы приложений для иллюстрации общих типов приложений. Для каждого архетипа приводятся целевые сценарии, технологии, шаблоны и инфраструктура, которую он включает.

Руководство в целом основывается на совокупном опыте и знании специалистов Microsoft, партнеров Microsoft, заказчиков и участников сообщества разработчиков для платформы Microsoft. Оно поможет понять нашу платформу, правильно выбрать архитектуру и технологии и использовать проверенные практики при создании приложений.

С уважением,

Скотт Гатри (Scott Guthrie)

Вице-президент по платформе разработки .NET

Microsoft

Предисловие Девида Хилла

Среди разработчиков популярна старая шутка о том, что чтобы считаться архитектором, надо просто отвечать на все технические вопросы: «ну, зависит от...». – Вопрос: Как лучше всего реализовать аутентификацию и авторизацию в моем решении? – Ответ: Ну, зависит от... – Вопрос: Как реализовать слой доступа к данным? – Ответ: Ну, зависит от... – Вопрос: Какую технологию необходимо применить для UI моего решения? – Ответ: Ну, зависит от... – Вопрос: Как сделать приложение масштабируемым? – Ответ: Ну, зависит от... И так далее, суть ясна.

Но основная идея в том, что ведь на самом деле зависит. В конечном счете, каждое решение уникально, и существует множество факторов, как технических, так и не технических, которые могут существенно влиять на архитектуру и дизайн решения любых масштабов. Задача разработчика и архитектора решения – найти равновесие между (зачастую противоречащими друг другу) требованиями и ограничениями, налагаемыми бизнесом, конечным пользователем, ИТ-средой и инфраструктурой управления организации, экономической ситуацией и, конечно же, технологиями и инструментальными средствами, используемыми для создания решения.

И, чтобы было уж совсем интересно, эти требования и ограничения постоянно меняются по мере появления новых возможностей или выдвижения новых требований к системе. Изменения бизнес-правил или возникновение новых прикладных областей оказывает влияние как на новые, так и на уже существующие приложения. Пользователи хотят получать все более насыщенные, единообразные и высокоинтегрированные пользовательские интерфейсы. Могут появляться новые требования к совместимости, или возникать новые технологии ИТ-инфраструктуры, которые сократят затраты или повысят доступность и масштабируемость. И, конечно же, постоянно выпускаются новые технологии, инфраструктуры и инструменты, которые обещают сократить затраты на разработку или сделать возможными сценарии, которые было трудно реализовать до этого.

Несомненно, разобраться во всем этом и в то же время обеспечить создание эффективного решения, вложившись в предусмотренный бюджет и отведенные сроки, задача не из легких. Она требует от разработчика или архитектора решения учесть все противоречащие и дополняющие друг друга факторы (в том числе и не технические) и найти эффективный баланс между ними. Попытки включения в рассмотрение слишком большого числа факторов может привести к очень сложным решениям. На их создание ушло бы слишком много времени, а результат все равно не обеспечил бы ожидаемой долговечности или гибкости. С другой стороны, если взять слишком мало факторов, получаются слишком ограниченные негибкие наспех скроенные решения, которые не поддаются доработке или плохо масштабируются. Другими словами, разработчикам и архитекторам решений часто приходится балансировать между «идеальным решением» с одной стороны и «решением на данный момент» с другой.

В этом, по моему мнению, и заключается суть проектирования архитектуры приложения – использовать инструменты и технологии сегодняшнего дня для получения максимальной

выгоды, учитывая требования и ограничения, налагаемые бизнесом сегодня, при этом заглядывая в будущее для обеспечения максимальной прибыли через масштабируемость, гибкость и обслуживание. Знание архитектурных принципов и шаблонов позволяет разработчику и архитектору решения понимать и учитывать в процессе проектирования важные аспекты, которые могут иметь огромное влияние на успех решения в целом. Вооруженные этим знанием они могут принимать более осознанные решения, лучше соотносить противоречие и дополняющие друг друга требования и ограничения и создавать решения, которые не просто отвечают всем бизнес-целям, но при этом обеспечивают экономическую эффективность, масштабируемость, гибкость, а также удобство и простоту обслуживания.

Как можно заметить, я обращаюсь и к разработчикам, и к архитекторам решений. Я глубоко уверен, что и тем, и другим чрезвычайно важно иметь твердое понимание шаблонов и принципов, обозначенных в этом руководстве. Мне могут возразить, что детали реализации не так важны, как общий дизайн. Из своего опыта утверждаю, что это далеко не так. Мелкие решения имеют свойство накапливаться со временем. Детали реализации могут иметь очень большое влияние на архитектуру решения в целом, его масштабируемость, удобство обслуживания и гибкость. Поэтому понимание основополагающих принципов и разработчиками, и архитекторами имеет жизненно важное значение. Кроме того, общее знание приводит к тому, что разработчики и архитекторы начинают лучше понимать друг друга, что также имеет огромный положительный эффект.

Цель этого руководства – предоставить обзор архитектуры приложений, принципов и шаблонов проектирования, что поможет делать правильный выбор и создавать более успешные решения. Руководство структурировано таким образом, чтобы его можно использовать и как учебник, и как справочник. Первая часть руководства посвящена общим вопросам построения архитектуры и принципам проектирования и применима к любому типу решения. Во второй части основное внимание направлено на общие типы приложений (такие как Веб-приложения, насыщенные клиентские приложения или мобильные приложения), для каждого из них описывается типовая архитектура и рассматриваются основные вопросы проектирования. Приводимые здесь решения, безусловно, не будут полностью соответствовать реальным задачам, но могут использоваться как базовые архитектуры. Данное руководство содержит советы относительно того, как определять основные элементы архитектуры, чтобы иметь возможность дорабатывать их со временем.

Повсеместно в руководстве особое внимание уделяется разработке решений на платформе Microsoft с применением .NET Framework. В него включены ссылки на статьи и ресурсы, в которых подробно рассматриваются соответствующие технологии и инструменты. Однако, как вы увидите, базовые принципы и шаблоны применимы ко всем платформам. Следует отметить, что данное руководство не является полным и исчерпывающим справочником по каждому аспекту архитектуры и дизайна приложений, оно представляет лишь практический обзор наиболее важных тем и дает ссылки на более подробные руководства или материалы, в которых соответствующие вопросы проработаны более детально.

В области архитектуры и дизайна приложений постоянно происходят изменения, она не стоит на месте. Основные принципы, на которых строились успешные решения в прошлом, будут хорошо служить и в обозримом будущем, но мы также должны быть готовыми к тому, что

темпы инноваций, как в технологиях, так и в новых подходах к проектированию, не будут снижаться. Платформа Microsoft и .NET Framework, а также огромное количество технологий и сценариев, ими поддерживаемые, обширны и глубоки и будут только расширяться и углубляться со временем. С другой стороны, нам не надо ожидать того, что может случиться. Мы можем создавать превосходные решения уже сегодня, и, надеюсь, это руководство поможет в этом.

Дэвид Хилл (David Hill)

patterns and practices

Сентябрь 2009

Введение

Цель данного руководства – помочь разработчикам и архитекторам решений создавать эффективные высококачественные приложения на платформе Microsoft и .NET Framework в более сжатые сроки и с меньшими рисками благодаря использованию проверенных и снискавших доверие архитектурных принципов и шаблонов проектирования.

В руководстве предлагается обзор основных принципов и шаблонов, которые обеспечивают прочную базу для создания хорошей архитектуры и дизайна приложения. В дополнение к этой базе даются общеприменимые рекомендации по разделению функциональности приложения на слои, компоненты и сервисы. Далее приводятся советы по определению и реализации ключевых характеристик дизайна, основных атрибутов качества (таких как производительность, безопасность и масштабируемость) и сквозной функциональности (такой как кэширование и протоколирование). В завершение руководство предлагает рекомендации по архитектуре и дизайну наиболее общих типов приложений, таких как Веб-приложения, насыщенные Интернет-приложения (RIA), насыщенные клиентские приложения, сервисы и мобильные приложения.

Руководство разделено на части соответственно основным аспектам архитектуры и дизайна. Оно скомпоновано таким образом, чтобы могло использоваться и как справочник, и как учебное пособие.

Данное руководство поможет:

- Понять базовые принципы и шаблоны построения архитектуры и дизайна для разработки успешных решений на платформе Microsoft.
- Правильно выбрать стратегии и шаблоны проектирования, которые помогут при проектировании слоев, компонентов и сервисов решения.
- Определить и реализовать ключевые технические решения.
- Определить и реализовать основные показатели качества и сквозные функции для решения.
- Правильно выбрать технологии для реализации решения.
- Создать возможный вариант базовой архитектуры решения.
- Правильно выбрать предлагаемые группой patterns & practices решения и руководства, которые помогут в реализации решения.

Это руководство является развернутым, но его нельзя считать полным и исчерпывающим учебником по архитектуре и дизайну приложений. Оно предназначено для использования в качестве практического и удобного обзора и справочника по общим принципам

проектирования архитектуры и дизайна приложений на платформе Microsoft и .NET Framework.

В частности, данное руководство не пытается предложить определенное или официальное архитектурное решение ни для одного конкретного сценария. Скорее, в нем дается краткий обзор принципов и шаблонов, которые обеспечивают создание хорошей архитектуры и дизайна, и предлагаются рекомендации по некоторым наиболее важным проблемам, которые могут возникнуть.

Основная масса представленного в руководстве материала не ориентирована ни на одну из технологий и может применяться к любой платформе или технологии. Но там, где мы посчитали это необходимым для обеспечения правильного выбора технологий или для использования их с максимальной пользой, мы ввели специальные рекомендации, касающиеся технологий Microsoft и .NET Framework.

Целевая аудитория

Данное руководство ориентировано, главным образом, на разработчиков и архитекторов решений, которые нуждаются в руководстве по разработке архитектуры и проектированию приложений на платформе Microsoft и в .NET Framework.

Однако это руководство будет полезным любому специалисту, который интересуется архитектурой и дизайном приложений, желает разобраться в базовых шаблонах и принципах, стоящих за хорошим дизайном приложений на платформе Microsoft или .NET Framework, а также для новичков, которые только начинают работать с платформой Microsoft или .NET Framework.

Как работать с данным руководством

Данное руководство не является учебником по архитектуре и дизайну приложений, рассматривающим все вопросы шаг за шагом. Скорее, это обзор и справочник. Данное руководство разделено на четыре основных части, каждая из которых включает в себя ряд глав:

- В первом разделе руководства, «[Архитектура и дизайн программного обеспечения](#)», дается обзор базовых принципов и шаблонов, которые являются основой для создания хорошей архитектуры и дизайна приложений, и предлагается подход к созданию дизайна приложения. Те, кто использует это руководство для изучения основ архитектуры приложений, должны начинать с этого раздела и затем переходить к остальным частям, чтобы познакомиться с многослойным дизайном, компонентами, показателями качества, сквозной функциональностью, сетевым взаимодействием, развертыванием и общими типами приложений.
- Во второй части, «[Основы проектирования](#)», предлагается общее практическое руководство по проектированию слоев, компонентов и сервисов решения, а также рекомендации по реализации показателей качества и сквозной функциональности. Также затрагиваются вопросы сетевого взаимодействия и развертывания. Те, кто желает изучить многослойный подход к проектированию архитектуры и дизайна

приложений или проектирование конкретных компонентов и сервисов, должны начинать с этого раздела. Из входящих в него глав можно узнать, как учитывать показатели качества и разрабатывать стратегию физического развертывания.

- В третьем разделе, «[Архетипы приложений](#)», предлагается руководство по проектированию архитектуры и дизайна общих типов приложений, таких как Веб-приложения, RIA, насыщенные клиентские приложений, мобильные и сервисные приложения. Те, кто уже имеет опыт создания архитектуры и дизайна и желает узнать об архитектуре и основных принципах проектирования общих типов приложений, а также получить рекомендации по каждому из них, могут начинать с этого раздела. Во входящих в него главах все эти вопросы рассматриваются более подробно.
- Наконец, в [Приложениях](#) предлагается обзор платформы Microsoft, технологий .NET Framework и их возможностей. В этом разделе также представлены все общие шаблоны проектирования и даются ссылки на дополнительные ресурсы и материалы. Те, кто только начинают работать с .NET Framework или желают узнать, какие технологии предлагает платформа Microsoft, найдут в этом разделе обзор сервисов .NET Framework и платформы, увидят все основные технологии и смогут ознакомиться с описаниями предложений группы patterns & practices, таких как Enterprise Library и библиотека шаблонов patterns & practices.

В зависимости от имеющегося опыта и необходимости можно обращаться непосредственно к определенному(ым) разделу(ам). Если требуется изучить развернутый обзор дизайна и архитектуры на платформе Microsoft и в .NET Framework, можно читать это руководство полностью, от начала до конца, оно поможет понять подходы к созданию архитектуры и дизайна. Это руководство может быть включено в жизненный цикл и процессы разработки приложения в качестве учебного пособия.

Обратная связь и поддержка

Мы постарались обеспечить максимальную точность и безошибочность сведений, приводимых в данном руководстве. Однако будем признательны за любые отзывы по всем затрагиваемым в нем темам, в частности, техническим вопросам, касающимся приведенных рекомендаций, вопросам их применимости и полезности. Для упрощения доступа через Веб список используемых источников также предлагается онлайн по адресу <http://www.microsoft.com/architectureguide>.

Свои комментарии по данному руководству можете оставлять на сайте <http://www.codeplex.com/AppArchGuide>.

Техническая поддержка

Техническую поддержку продуктов и технологий Microsoft, упоминаемых в данном руководстве, обеспечивает Служба технической поддержки Microsoft (Microsoft Product Support Services, PSS). Сведения о продуктах можно получить на Веб-сайте технической поддержки Microsoft по адресу <http://support.microsoft.com>.

Сообщество и группа новостей

Получить поддержку сообщества, обсудить это руководство и оставить свои отзывы можно в группах новостей на сайте Microsoft MSDN® по адресу <http://msdn.microsoft.com/en-us/subscriptions/aa974230.aspx>.

Авторская группа

В создании этого руководства принимали участие такие специалисты по архитектуре и разработке .NET-решений:

- J.D. Meier
- David Hill
- Alex Homer
- Jason Taylor
- Prashant Bansode
- Lonnie Wall
- Rob Boucher Jr.
- Akshay Bogawat

Авторы и рецензенты

Огромная благодарность авторам и рецензентам:

- **Группа тестирования.** Rohit Sharma; Praveen Rangarajan
- **Редактор.** Dennis Rea
- **Независимые авторы и рецензенты.** Adwait Ullal; Andy Eunson; Brian Sletten; Christian Weyer; David Guimbellot; David Ing; David Weller; David Sussman; Derek Greer; Eduardo Jezierski; Evan Hoff; Gajapathi Kannan; Jeremy D. Miller; John Kordyback; Keith Pleas; Kent Corley; Mark Baker; Paul Ballard; Peter Oehlert; Norman Headlam; Ryan Plant; Sam Gentile; Sidney G Pinney; Ted Neward; Udi Dahan; Oren Eini aka Ayende Rahien; Gregory Young
- **Авторы и рецензенты, являющиеся сотрудниками компании Microsoft.** Ade Miller; Amit Chopra; Anna Liu; Anoop Gupta; Bob Brumfield; Brad Abrams; Brian Cawelti; Bhushan Nene; Burley Kawasaki; Carl Perry; Chris Keyser; Chris Tavares; Clint Edmonson; Dan Reagan; David Hill; Denny Dayton; Diego Dagum; Dmitri Martynov; Dmitri Ossipov; Don Smith; Dragos Manolescu; Elisa Flasko; Eric Fleck; Erwin van der Valk; Faisal Mohamood; Francis Cheung; Gary Lewis; Glenn Block; Gregory Leake; Ian Ellison-Taylor; Ilia Fortunov; J.R. Arredondo; John deVadoss; Joseph Hofstader; Kashinath TR; Koby Avital; Loke Uei Tan; Luke Nyswonger; Manish Prabhu; Meghan Perez; Mehran Nikoo; Michael Puleio; Mike Francis; Mike Walker; Mubarak Elamin; Nick Malik; Nobuyuki Akama; Ofer Ashkenazi; Pablo Castro; Pat Helland; Phil Haack; Rabi Satter; Reed Robison; Rob Tiffany; Ryno Rijnsburger; Scott Hanselman; Seema Ramchandani; Serena Yeoh;

Simon Calvert; Srinath Vasireddy; Tom Hollander; Vijaya Janakiraman; Wojtek Kozaczynski

Поделитесь с нами своими успехами

Если это руководство оказалось вам полезным, мы хотели бы знать об этом. Кратко опишите проблемы, с которыми вам приходилось сталкиваться, и то, как это руководство помогло с ними справиться. Присылайте свои отзывы по электронной почте по адресу MyStory@Microsoft.com.

Архитектура и дизайн программного обеспечения

Темы, включенные в данный раздел этого руководства, помогут понять основы архитектуры и дизайна. Обсуждение начинается с определения архитектуры программного обеспечения и объяснения, почему она так важна. Рассматриваются общие вопросы, заслуживающие особого внимания, такие как требования и ограничения, а также точки соприкосновения пользователя, предметной области и системы, в которой будет выполняться приложение. Далее следует описание основных принципов проектирования, архитектурных шаблонов и стилей, обычно применяемых сегодня. Наконец, предлагается подход, которому необходимо следовать при проектировании архитектуры. Раздел содержит следующие главы:

- Глава 1 «[Что такое архитектура программного обеспечения?](#)»
 - Глава 2 «[Основные принципы проектирования архитектуры ПО](#)»
 - Глава 3 «[Архитектурные шаблоны и стили](#)»
 - Глава 4 «[Методика построения архитектуры и дизайна](#)»
-

1

Что такое архитектура программного обеспечения?

Создание архитектуры приложения – это процесс формирования структурированного решения, отвечающего всем техническим и операционным требованиям и обеспечивающего оптимальные общие атрибуты качества, такие как производительность, безопасность и управляемость. Он включает принятие ряда решений на основании широкого диапазона факторов. Каждое из этих решений может иметь существенное влияние на качество, производительность, удобство обслуживания и общий успех приложения.

Филипп Крачтен (Philippe Kruchten), Грейди Буч (Grady Booch), Курт Биттнер (Kurt Bittner) и Рич Рейтман (Rich Reitman) доработали и уточнили определение архитектуры, приведенное в работе Мэри Шоу (Mary Shaw) и Девида Гарлана (David Garlan) (Shaw and Garlan, 1996). Их формулировка звучит так:

«Архитектура программного обеспечения (ПО) заключает в себе ряд важных решений об организации программной системы, среди которых выбор структурных элементов и их интерфейсов, составляющих и объединяющих систему в единое целое; поведение, обеспечиваемое совместной работой этих элементов; организацию этих структурных и поведенческих элементов в более крупные подсистемы, а также архитектурный стиль, которого придерживается данная организация. Выбор архитектуры ПО также касается функциональности, удобства использования, устойчивости, производительности, повторного использования, понятности, экономических и технологических ограничений, эстетического восприятия и поиска компромиссов».

В книге «Архитектура корпоративных программных приложений» Мартин Фаулер (Martin Fowler), рассматривая проектирование архитектуры, выделяет несколько общих элементов в разных определениях этого понятия. Он определяет эти элементы как:

«Разделение системы на составные части в самом первом приближении; принятие решений, которые трудно изменить впоследствии; выработка множества возможных вариантов архитектуры для системы; важность с точки зрения архитектуры различных аспектов может меняться в процессе жизненного цикла системы; и, в конечном счете, под архитектурой можно понимать то, что имеет значение».

[<http://www.pearsonhighered.com/educator/academic/product/0,3110,0321127420,00.html>
]

В книге «Архитектура ПО на практике, 2-е издание» Басс, Клементс и Казман дают следующее определение архитектуре:

«Архитектура программной или вычислительной системы – это структура или структуры системы, включающие программные элементы, видимые извне свойства этих элементов и взаимоотношения между ними. Архитектура касается внешней части интерфейсов; внутренние детали элементов – детали, относящиеся исключительно к внутренней реализации – не являются архитектурными».

[<http://www.aw-bc.com/catalog/academic/product/0,4096,0321154959,00.html>]

Почему архитектура так важна?

Как и любая другая сложная структура, ПО должно строиться на прочном фундаменте. Неправильное определение ключевых сценариев, неправильное проектирование общих вопросов или неспособность выявить долгосрочные последствия основных решений могут поставить под угрозу все приложение. Современные инструменты и платформы упрощают задачу по созданию приложений, но не устраниют необходимости в тщательном их проектировании на основании конкретных сценариев и требований. Неправильно выработанная архитектура обуславливает нестабильность ПО, невозможность поддерживать существующие или будущие бизнес-требования, сложности при развертывании или управлении в среде производственной эксплуатации.

Проектирование систем должно осуществляться с учетом потребностей пользователя, системы (ИТ-инфраструктуры) и бизнес-целей. Для каждой из этих составляющих определяются ключевые сценарии и выделяются важные параметры качества (например, надежность или масштабируемость), а также основные области удовлетворенности и неудовлетворенности. По возможности необходимо выработать и учсть показатели успешности в каждой из этих областей.



Рис. 1

Пользователь, бизнес, система

Необходимо искать компромиссы и находить баланс между конкурирующими требованиями этих трех областей. Например, пользовательский интерфейс решения очень часто является производным от бизнес-целей и ИТ-инфраструктуры, и изменения одного из этих компонентов могут существенно влиять на результирующие механизмы взаимодействия с пользователем. Аналогично, изменения в требованиях по взаимодействию с пользователем могут оказывать сильное воздействие на требования к бизнес-области и ИТ-инфраструктуре. Например,

основной целью пользователя и бизнеса может быть производительность, но системный администратор, вероятно, не сможет инвестировать в оборудование, необходимое для реализации этой цели, 100 % своего рабочего времени. Компромиссным решением может быть выполнение данной цели на 80%.

Основное назначение архитектуры – описание использования или взаимодействия основных элементов и компонентов приложения. Выбор структур данных и алгоритмов их обработки или деталей реализации отдельных компонентов являются вопросами проектирования. Часто вопросы архитектуры и проектирования пересекаются. Вместо того чтобы вводить жесткие правила, разграничающие архитектуру и проектирование, имеет смысл комбинировать эти две области. В некоторых случаях, принимаемые решения, очевидно, являются архитектурными по своей природе, в других – больше касаются проектирования и реализации архитектуры.

Описанные в данном руководстве процессы и предлагаемые в нем сведения помогут создавать архитектуры решений, которые будут реализовывать все требования, могут быть развернуты в выбранной инфраструктуре и обеспечат результаты, отвечающие поставленным целям и задачам.

Рассмотрим основные исходные вопросы при разработке архитектуры ПО:

- Как пользователь будет использовать приложение?
- Как приложение будет развертываться и обслуживаться при эксплуатации?
- Какие требования по атрибутам качества, таким как безопасность, производительность, возможность параллельной обработки, интернационализация и конфигурация, выдвигаются к приложению?
- Как спроектировать приложение, чтобы оно оставалось гибким и удобным в обслуживании в течение долгого времени?
- Основные архитектурные направления, которые могут оказывать влияние на приложение сейчас или после его развертывания?

Цели архитектуры

Архитектура приложения должна объединять бизнес-требования и технические требования через понимание вариантов использования с последующим нахождением путей их реализации в ПО. Цель архитектуры – выявить требования, оказывающие влияние на структуру приложения. Хорошая архитектура снижает бизнес-риски, связанные с созданием технического решения. Хорошая структура обладает значительной гибкостью, чтобы справляться с естественным развитием технологий, как в области оборудования и ПО, так и пользовательских сценариев и требований. Архитектор должен учитывать общий эффект от принимаемых проектных решений, обязательно присутствующие компромиссы между атрибутами качества (такими как производительность и безопасность) и компромиссы, необходимые для выполнения пользовательских, системных и бизнес-требований.

Необходимо помнить, что архитектура должна:

- Раскрывать структуру системы, но скрывать детали реализации.
 - Реализовывать все варианты использования и сценарии.
 - По возможности отвечать всем требованиям различных заинтересованных сторон.
 - Выполнять требования, как по функциональности, так и по качеству.
-

Архитектура сегодня и завтра

Важно понимать основные факторы, которые формируют архитектурные решения сегодня и будут оказывать влияние на то, как изменятся архитектурные решения в будущем. Эти факторы определяются пожеланиями пользователей, а также требованиями бизнеса о получении более быстрых результатов, лучшей поддержки изменяющихся стилей работы и рабочих процессов, а также улучшенной адаптируемости дизайна ПО.

Рассмотрим следующие основные направления:

- **Наделение пользователя полномочиями.** Дизайн, поддерживающий наделение пользователя полномочиями, является гибким, настраиваемым и ориентированным на пользователя. Проектируйте приложения, учитывая соответствующий уровень персонализации и конфигурации для пользователя. Не диктуйте, а позволяйте пользователям определять стиль взаимодействия с приложением, но при этом не перегружайте их ненужными опциями и настройками, что может сбить с толку. Определитесь с ключевыми сценариями и сделайте их предельно простыми; обеспечьте простоту поиска информации и использования приложения.
- **Зрелость рынка.** Используйте преимущества зрелости рынка, применяя существующие платформы и технологии. Создавайте приложения в настолько высокоуровневых средах, насколько это позволяют предъявляемые к приложениям требования, это позволит сконцентрироваться на том, что действительно уникально для вашего приложения, а не на воспроизведении уже существующих функций. Используйте шаблоны, являющиеся богатыми источниками проверенных решений распространенных проблем.
- **Гибкий дизайн.** Все большую популярность приобретают гибкие дизайны, использующие слабое связывание, что делает возможным повторное использование и упрощает поддержку. Архитектура с возможностью подключения модулей позволяет реализовать расширяемость после развертывания. Также для обеспечения взаимодействия с другими системами можно использовать преимущества таких сервис-ориентированных техник, как SOA.
- **Тенденции.** При построении архитектуры необходимо понимать тенденции, которые могут оказывать влияние на дизайн после развертывания. Например, тенденции в насыщенных UI и мультимедиа, составных приложениях, увеличение пропускной способности и доступности сетей, все большее распространение мобильных устройств, продолжающийся рост производительности оборудования, интерес к моделям блогов и сообществ, рост популярности вычислений в облаке и удаленной работы.

Принципы проектирования архитектуры

В настоящее время при продумывании архитектуры мы предполагаем, что наш дизайн будет эволюционировать со временем и что совершенно невозможно наперед знать все то, что необходимо для проектирования системы. Как правило, дизайн изменяется и дорабатывается в ходе реализации приложения по мере выявления новых сведений и в ходе тестирования на соответствие требованиям реального окружения. Создавайте архитектуру, ориентируясь на такие изменения, чтобы иметь возможность адаптировать их к требованиям, которые в начале процесса проектирования известны не в полном объеме.

При проектировании архитектуры необходимо ответить на следующие вопросы:

- Какие части архитектуры являются фундаментальными, изменение которых в случае неверной реализации представляет наибольшие риски?
- Какие части архитектуры вероятнее всего подвергнуться изменениям, а также проектирование каких частей можно отложить?
- Основные допущения, и как они будут проверяться?
- Какие условия могут привести к реструктуризации дизайна?

Не пытайтесь создать слишком сложную архитектуру и не делайте предположений, которые не можете проверить. Лучше оставляйте свои варианты открытыми для изменения в будущем. Некоторые аспекты дизайна должны быть приведены в порядок на ранних стадиях процесса, потому что их возможная переработка может потребовать существенных затрат. Такие области необходимо выявить как можно раньше и уделить им достаточное количество времени.

Основные принципы проектирования архитектуры

При проектировании архитектуры руководствуйтесь следующими основными принципами:

- **Создавайте, чтобы изменять.** Продумайте, как со временем может понадобиться изменить приложение, чтобы оно отвечало вновь возникающим требованиям и задачам, и предусмотрите необходимую гибкость.
- **Создавайте модели для анализа и сокращения рисков.** Используйте средства проектирования, системы моделирования, такие как Унифицированный язык моделирования (Unified Modeling Language, UML), и средства визуализации, когда необходимо выявить требования, принять архитектурные и проектные решения и проанализировать их последствия. Тем не менее, не создавайте слишком формализованную модель, она может ограничить возможности для выполнения итераций и адаптации дизайна.
- **Используйте модели и визуализации как средства общения при совместной работе.** Для построения хорошей архитектуры критически важен эффективный обмен информацией о дизайне, принимаемых решениях и вносимых изменениях. Используйте модели, представления и другие способы визуализации архитектуры для эффективного обмена информацией и связи со всеми заинтересованными

сторонами, а также для обеспечения быстрого оповещение об изменениях в дизайне.

- **Выявляйте ключевые инженерные решения.** Используйте информацию, представленную в данном руководстве, чтобы понять ключевые инженерные решения и области, в которых чаще всего возникают ошибки. В самом начале проекта уделите достаточное количество времени и внимания для принятия правильных решений, это обеспечит создание более гибкого дизайна, внесение изменений в который не потребует полной его переработки.

Рассмотрите возможность использования инкрементного и итеративного подхода при работе над архитектурой. Начинайте с базовой архитектуры, правильно воссоздавая полную картину, и затем прорабатывайте возможные варианты в ходе итеративного тестирования и доработки архитектуры. Не пытайтесь сделать все сразу; проектируйте настолько, насколько это необходимо для начала тестирования вашего дизайна на соответствие требованиям и допущениям. Усложняйте дизайн постепенно, в процессе многократных пересмотров, чтобы убедиться, прежде всего, в правильности принятых крупных решений и лишь затем сосредотачиваться на деталях. Общей ошибкой является быстрый переход к деталям при ошибочном представлении о правильности крупных решений из-за неверных допущений или неспособности эффективно оценить свою архитектуру. При тестировании архитектуры дайте ответы на следующие вопросы:

- Какие допущения были сделаны в этой архитектуре?
- Каким явным или подразумеваемым требованиям отвечает данная архитектура?
- Основные риски при использовании такого архитектурного решения?
- Каковы меры противодействия для снижения основных рисков?
- Является ли данная архитектура улучшением базовой архитектуры или одним из возможных вариантов архитектуры?

Более подробно об основных принципах проектирования архитектуры ПО рассказывается в главе 2, «[Основные принципы проектирования архитектуры ПО](#)».

Инкрементный и итеративный подход к разработке архитектуры, базовая и возможные варианты архитектуры, а также представление и обмен информацией о дизайне рассматриваются в главе 4, «[Методика построения архитектуры и дизайна](#)».

Дополнительные источники

Л.Басс, П.Клементс, Р.Кацман. *Архитектура программного обеспечения на практике, 2-е издание*. Питер, 2006.

Мартин Фаулер. *Архитектура корпоративных программных приложений*. Вильямс, 2007.

2

Основные принципы проектирования архитектуры ПО

Обзор

В данной главе будут рассмотрены основные принципы и рекомендации по проектированию архитектуры ПО. Архитектура ПО часто описывается как организация или структура системы, где *система* представляет набор компонентов, выполняющих определенную функцию или набор функций. Иначе говоря, основное назначение архитектуры – организация компонентов с целью обеспечения определенной функциональности. Такую организацию функциональности часто называют группировкой компонентов по «функциональным областям». На рис. 1 представлена типовая архитектура приложения, компоненты которого сгруппированы по функциональным областям.

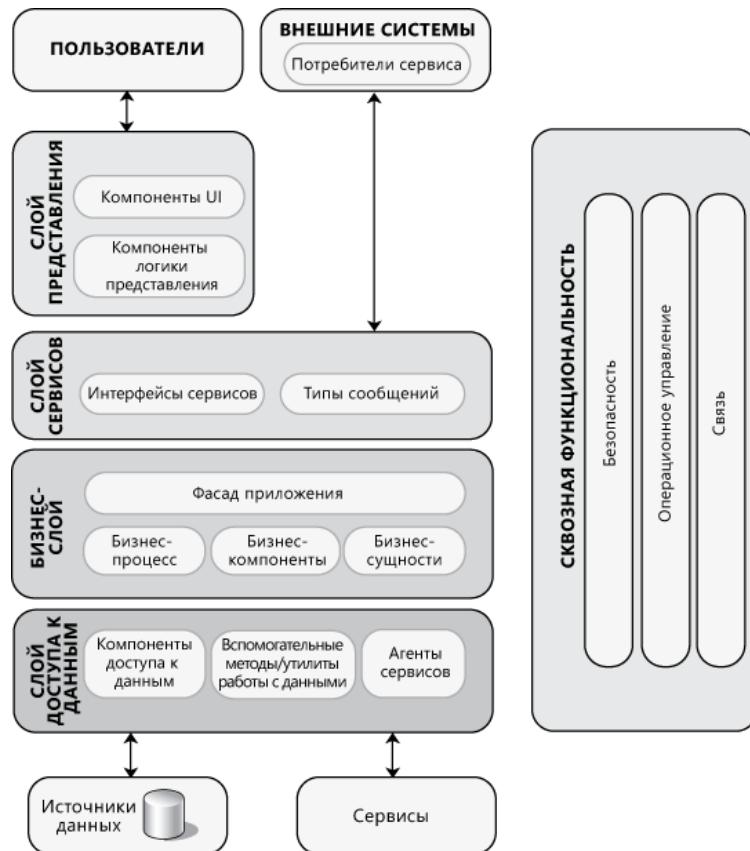


Рис. 2

Типовая архитектура приложения

Но функциональные области используются не только для группировки компонентов, некоторые из них посвящены взаимодействию и организации совместной работы компонентов. В данной главе приводятся рекомендации по различным функциональным областям, которыми необходимо руководствоваться при проектировании архитектуры собственного приложения.

Основные принципы проектирования

Приступая к работе над архитектурой приложения, необходимо помнить об основных принципах проектирования. Это поможет создать архитектуру, которая будет следовать проверенным подходам, обеспечит минимизацию затрат, простоту обслуживания, удобство использования и расширяемость. Рассмотрим основные принципы:

- **Разделение функций.** Разделите приложение на отдельные компоненты с, по возможности, минимальным перекрытием функциональности. Важным фактором является предельное уменьшение количества точек соприкосновения, что обеспечит высокую связность (high cohesion) и слабую связанность (low coupling). Неверное разграничение функциональности может привести к высокой связанности и сложностям взаимодействия, даже несмотря на слабое перекрытие функциональности отдельных компонентов.

- **Принцип единственности ответственности.** Каждый отдельно взятый компонент или модуль должен отвечать только за одно конкретное свойство/функцию или совокупность связанных функций.
- **Принцип минимального знания** (также известный как Закон Деметера (Law of Demeter, LoD)). Компоненту или объекту не должны быть известны внутренние детали других компонентов или объектов.
- **Не повторяйтесь (Don't repeat yourself, DRY).** Намерение должно быть обозначено только один раз. В применении к проектированию приложения это означает, что определенная функциональность должна быть реализована только в одном компоненте и не должна дублироваться ни в одном другом компоненте.
- **Минимизируйте проектирование наперед.** Проектируйте только то, что необходимо. В некоторых случаях, когда стоимость разработки или издержки в случае неудачного дизайна очень высоки, может потребоваться полное предварительное проектирование и тестирование. В других случаях, особенно при гибкой разработке, можно избежать масштабного проектирования наперед (big design upfront, BDUF). Если требования к приложению четко не определены, или существует вероятность изменения дизайна со временем, старайтесь не тратить много сил на проектирование раньше времени. Этот принцип называют YAGNI («You ain't gonna need it»¹).

Цель архитектора ПО при проектировании приложения или системы – максимальное упрощение дизайна через его разбиение на функциональные области. Например, пользовательский интерфейс (user interface, UI), выполнение бизнес-процессов или доступ к данным – все это разные функциональные области. Компоненты в каждой из этих областей должны реализовывать данную конкретную функциональность и не должны смешивать в себе код разных функциональных областей. Так в компонентах UI не должно быть кода прямого доступа к источнику данных; для извлечения данных в них должны использоваться либо бизнес-компоненты, либо компоненты доступа к данным.

Также необходимо проанализировать соотношение затрат/выгод для инвестиций в приложение. В некоторых случаях может быть целесообразным упростить структуру и разрешить, например, связывание элементов UI с результирующими данными. В общем, оценивайте реализацию функциональности также и с коммерческой точки зрения. Далее приводятся обобщенные рекомендации, которые помогут учесть широкий диапазон факторов, влияющих на проектирование, реализацию, развертывание, тестирование и обслуживание приложения.

Практики проектирования

- **Придерживайтесь единства шаблонов проектирования в рамках одного слоя.** По возможности, в рамках одного логического уровня структура компонентов, выполняющих определенную операцию, должна быть единообразной. Например, если для объекта, выступающего в роли шлюза к таблицам или представлениям

¹ Вам это никогда не понадобится (прим. переводчика).

базы данных, решено использовать шаблон Table Data Gateway (Шлюз таблицы данных), не надо включать еще один шаблон, скажем, Repository (Хранилище), использующий другую парадигму доступа к данным и инициализации бизнес-сущностей. Однако для задач с более широким диапазоном требований может потребоваться применить разные шаблоны, например, для приложения, включающего поддержку бизнес-транзакций и составления отчетов.

- **Не дублируйте функциональность в приложении.** Та или иная функциональность должна обеспечиваться только одним компонентом, ее дублирование в любом другом месте недопустимо. Это обеспечивает связность компонентов и упрощает их оптимизацию в случае необходимости изменения отдельной функциональной возможности. Дублирование функциональности в приложении может усложнить реализацию изменений, снизить понятность приложения и создать потенциальную возможность для несогласованностей.
- **Предпочитайте композицию наследованию.** По возможности, для повторного использования функциональности применяйте композицию, а не наследование, потому что наследование увеличивает зависимость между родительским и дочерними классами, ограничивая, таким образом, возможности повторного использования последних. Это также будет способствовать уменьшению глубины иерархии наследования, что упростит работу с ними.
- **Применяйте определенный стиль написания кода и соглашение о присваивании имен для разработки.** Поинтересуйтесь, имеет ли организация сформулированный стиль написания кода и соглашения о присваивании имен. Если нет, необходимо придерживаться общепринятых стандартов. В этом случае вы получите единообразную модель, все участники группы смогут без труда работать с кодом, написанным не ими, т.е. код станет более простым и удобным в обслуживании.
- **Обеспечивайте качество системы во время разработки с помощью методик автоматизированного QA¹.** Используйте в процессе разработки модульное тестирование и другие методики автоматизированного Анализа качества (Quality Analysis), такие как анализ зависимостей и статический анализ кода. Четко определяйте показатели поведения и производительности для компонентов и подсистем и используйте автоматизированные инструменты QA в процессе разработки, чтобы гарантировать отсутствие неблагоприятного воздействия локальных решений по проектированию или реализации на качество всей системы.
- **Учитывайте условия эксплуатации приложения.** Определите необходимые ИТ-инфраструктуре показатели и эксплуатационные данные, чтобы гарантировать эффективное развертывание и работу приложения. Приступайте к проектированию компонентов и подсистем приложений, только имея ясное представление об их индивидуальных эксплуатационных требованиях, что существенно упростит общее развертывание и эксплуатацию. Использование автоматизированных инструментов

¹ Quality Assurance – контроль качества (прим. технического редактора).

QA при разработке гарантированно обеспечит получение необходимых эксплуатационных характеристик компонентов и подсистем приложения.

Слои приложения

- **Разделяйте функциональные области.** Разделите приложение на отдельные функции с, по возможности, минимальным перекрытием функциональности. Основное преимущество такого подхода – независимая оптимизация функциональных возможностей. Кроме того, сбой одной из функций не приведет к сбою остальных, поскольку они могут выполняться независимо друг от друга. Такой подход также упрощает понимание и проектирование приложения и облегчает управление сложными взаимосвязанными системами.
- **Явно определяйте связи между слоями.** Решение, в котором каждый слой приложения может взаимодействовать или имеет зависимости со всеми остальными слоями, является сложным для понимания и управления. Принимайте явные решения о зависимостях между слоями и о потоках данных между ними.
- **Реализуйте слабое связывание слоев с помощью абстракции.** Это можно реализовать, определяя интерфейсные компоненты с хорошо известными входными и выходными характеристиками, такие как фасад¹, которые преобразуют запросы в формат, понятный компонентам слоя. Кроме того, также можно определять общий интерфейс или совместно используемую абстракцию (противоположность зависимости), которые должны быть реализованы компонентами интерфейса, используя интерфейсы или абстрактные базовые классы.
- **Не смешивайте разные типы компонентов на одном логическом уровне.** Начинайте с идентификации функциональных областей и затем группируйте компоненты, ассоциированные с каждой из этих областей в логические уровни. Например, слой UI не должен включать компоненты выполнения бизнес-процессов, в него должны входить только компоненты, используемые для обработки пользовательского ввода и запросов.
- **Придерживайтесь единого формата данных в рамках слоя или компонента.** Смешение форматов данных усложнит реализацию, расширение и обслуживание приложения. Любое преобразование одного формата данных в другой требует реализации кода преобразования и влечет за собой издержки на обработку.

Компоненты, модули и функции

- **Компонент или объект не должен полагаться на внутренние данные других компонентов или объектов.** Каждый метод, вызываемый компонентом или методом другого объекта или компонента, должен располагать достаточными сведениями о том, как обрабатывать поступающие запросы и, в случае необходимости, как перенаправлять их к соответствующим подкомпонентам или

¹ Имеется в виду шаблон проектирования Façade (прим. технического редактора).

другим компонентам. Это способствует созданию более удобных в обслуживании и адаптируемых приложений.

- **Не перегружайте компонент функциональностью.** Например, компонент UI не должен включать код для доступа к данным или обеспечивать дополнительную функциональность. Перегруженные компоненты часто имеют множество функций и свойств, сочетая бизнес-функциональность и сквозную функциональность, такие как протоколирование и обработка исключений. В результате получается очень неустойчивый к ошибкам и сложный в обслуживании дизайн. Применение принципов исключительной ответственности и разделения функциональности поможет избежать этого.
- **Разберитесь с тем, как будет осуществляться связь между компонентами.** Это требует понимания сценариев развертывания, которые должно поддерживать создаваемое приложение. Необходимо определить, будут ли все компоненты выполняться в рамках одного процесса или необходимо обеспечить поддержку связи через физические границы или границы процесса, вероятно, путем реализации интерфейсов взаимодействия на основе сообщений.
- **Максимально изолируйте сквозную функциональность от бизнес-логики приложения.** Сквозная функциональность – это аспекты безопасности, обмена информацией или управляемости, такие как протоколирование и инструментирование. Смешение кода, реализующего эти функции, с бизнес-логикой может привести к созданию дизайна, который будет сложно расширять и обслуживать. Внесение изменений в сквозную функциональность потребует переработки всего кода бизнес-логики. Рассмотрите возможность использования инфраструктур и методик (таких как аспект-ориентированное программирование), которые помогут в реализации такой функциональности.
- **Определяйте четкий контракт для компонентов.** Компоненты, модули и функции должны определять контракт или спецификацию интерфейса, четко оговаривающую их использование и поведение. Контракт должен описывать, как другие компоненты могут выполнять доступ к внутренней функциональности компонента, модуля или функции, и поведение этой функциональности с точки зрения предварительных условий, постусловий, побочных эффектов, исключений, рабочих характеристик и других факторов.

Основные вопросы проектирования

В данном руководстве представлены основные решения, которые должны быть приняты и помогут учесть все важные факторы, когда вы приступаете и затем ведете итеративную разработку проекта архитектуры. Эти основные решения перечислены в списке ниже и кратко рассматриваются в последующих разделах данной главы:

- [Определение типа приложения](#)
- [Выбор стратегии развертывания](#)

- [Выбор соответствующих технологий](#)
- [Выбор показателей качества](#)
- [Решение о путях реализации сквозной функциональности](#)

Более подробное описание процесса проектирования приводится в главе 4, «[Методика построения архитектуры и дизайна](#)».

Определение типа приложения

Выбор соответствующего типа приложения – ключевой момент процесса проектирования приложения. Этот выбор определяется конкретными требованиями и ограничениями среды. От многих приложений требуется поддержка множества типов клиентов и возможность использования более одного базового архетипа. В данном руководстве рассматриваются следующие основные типы приложений:

- Приложения для мобильных устройств.
- Насыщенные клиентские приложения для выполнения преимущественно на клиентских ПК.
- Насыщенные клиентские приложения для развертывания из Интернета с поддержкой насыщенных UI и мультимедийных сценариев.
- Сервисы, разработанные для обеспечения связи между слабо связанными компонентами.
- Веб-приложения для выполнения преимущественно на сервере в сценариях с постоянным подключением.

Кроме того, представлены сведения и рекомендации по некоторым более специальным типам приложений. К ним относятся:

- Приложения и сервисы, размещаемые в центрах обработки данных (ЦОД) и в облаке.
- Офисные бизнес-приложения (Office Business Applications, OBAs), интегрирующие технологии Microsoft Office и Microsoft Server.
- Бизнес-приложения SharePoint (SharePoint Line of Business, LOB), обеспечивающие доступ к бизнес-данным и функциональным возможностям через портал.

Более подробно архетипы приложений рассматриваются в главе 20, «[Выбор типа приложения](#)».

Выбор стратегии развертывания

Приложение может развертываться в разнообразнейших средах, каждая из которых будет иметь собственный набор ограничений, таких как физическое распределение компонентов по серверам, ограничение по используемым сетевым протоколам, настройки межсетевых экранов и маршрутизаторов и многое другое. Существует несколько общих схем развертывания, которые описывают преимущества и мотивы применения ряда

распределенных и нераспределенных сценариев. При выборе стратегии необходимо найти компромисс между требованиями приложения и соответствующими схемами развертывания, поддерживаемым оборудованием, и ограничениями, налагаемыми средой на варианты развертывания. Все эти факторы будут влиять на проектируемую архитектуру.

Более подробно вопросы развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

Выбор соответствующих технологий

Ключевым фактором при выборе технологий для приложения является тип разрабатываемого приложения, а также предпочтительные варианты топологии развертывания приложения и архитектурные стили. Выбор технологий также определяется политиками организации, ограничениями среды, квалификацией ресурсов и т.д. Необходимо сравнить возможности выбираемых технологий с требованиями приложения, принимая во внимание все эти факторы.

Более подробно технологии, предлагаемые платформой Microsoft, рассматриваются в приложении А, «[Платформа приложений Microsoft](#)».

Выбор показателей качества

Показатели качества, такие как безопасность, производительность, удобство и простота использования, помогают сфокусировать внимание на критически важных проблемах, которые должен решать создаваемый дизайн. В зависимости от конкретных требований может понадобиться рассмотреть все упоминаемые в данном руководстве показатели качества или только некоторые из них. Например, вопросы безопасности и производительности необходимо учесть при разработке каждого приложения, тогда как проблемы возможности взаимодействия или масштабируемости стоят далеко не перед всеми проектами. Первым делом, необходимо понять поставленные требования и сценарии развертывания, чтобы знать, какие показатели качества важны для создаваемого приложения. Нельзя также забывать о возможности конфликта между показателями качества. Например, часто требования безопасности идут вразрез с производительностью или удобством использования.

При проектировании с учетом показателей качества следует руководствоваться следующим:

- Показатели качества – это свойства системы, отделенные от ее функциональности.
- С технической точки зрения, реализованные показатели качества отличают хорошую систему от плохой.
- Существует два типа показателей качества: измеряемые во время выполнения и те, оценить которые можно только посредством проверки.
- Необходимо провести анализ и найти оптимальное соотношение между показателями качества.

Вопросы, на которые необходимо ответить при рассмотрении показателей качества:

- Каковы основные показатели качества приложения? Определите их в ходе процесса разработки.

- Каковы основные требования для реализации этих показателей? Поддаются ли они количественному определению?
- Каковы критерии приемки, которые будут свидетельствовать о выполнении требований?

Более подробно показатели качества рассматриваются в главе 16, «[Показатели качества](#)».

Решение о путях реализации сквозной функциональности

Сквозная функциональность представляет ключевую область дизайна, не связанную с конкретным функционалом приложения. Например, необходимо рассмотреть возможности реализации централизованных или общих решений для следующих аспектов:

- Механизм протоколирования, обеспечивающий возможность каждому слою вести журнал в общем хранилище либо в разных хранилищах, но таким образом, чтобы результаты могли быть сопоставлены впоследствии.
- Механизмы аутентификации и авторизации, обеспечивающие передачу удостоверений на разные уровни для предоставления доступа к ресурсам.
- Инфраструктура управления исключениями, которая будет функционировать в каждом слое и между уровнями, если исключения распространяются в рамках системы.
- Подход к реализации связей, используемый для обеспечения обмена информацией между слоями.
- Общая инфраструктура кэширования, позволяющая кэшировать данные в слое представления, бизнес-слое и слое доступа к данным.

В следующем списке приведены основные аспекты сквозной функциональности, которые необходимо рассмотреть при создании архитектуры приложений:

- **Инструментирование и протоколирование.** Обеспечивайте управление и мониторинг всех критически важных для бизнес-логики и системы событий. Протоколируйте достаточное количество сведений для воссоздания событий в системе без включения конфиденциальных данных.
- **Аутентификация.** Определитесь с тем, как будет проходить аутентификация пользователей и передача аутентифицированных удостоверений между слоями.
- **Авторизация.** На каждом уровне и между границами доверия обеспечьте соответствующую авторизацию с необходимой детализацией.
- **Управление исключениями.** Перехватывайте исключения на функциональных, логических и физических границах и избегайте раскрытия конфиденциальных сведений конечным пользователям.
- **Связь.** Выберите соответствующие протоколы, сведите до минимума количество вызовов по сети и защитите передачу конфиденциальных данных по сети.

- **Кэширование.** Определите, что должно кэшироваться и где для улучшения производительности и сокращения времени отклика приложения. При проектировании кэширования не забудьте учесть особенности Веб-фермы и фермы приложений.

Более подробно о сквозной функциональности рассказывает глава 17, «[Сквозная функциональность](#)».

3

Архитектурные шаблоны и стили

Обзор

В данной главе описываются и обсуждаются обобщенные шаблоны и принципы, применяемые при проектировании приложений сегодня. Часто их называют *архитектурными стилями* или *парадигмами*. К ним относят такие шаблоны как клиент/сервер, многоуровневая архитектура, компонентная архитектура, архитектура, основанная на шине сообщений, и сервисно-ориентированная архитектура (service-oriented architecture, SOA). Для каждого стиля предлагается обзор, основные принципы, основные преимущества и сведения, которые помогут выбрать подходящие для приложения архитектурные стили. Важно понимать, что стили описывают разные аспекты приложений. Некоторые архитектурные стили описывают схемы развертывания, некоторые – вопросы структуры и дизайна, другие – аспекты связи. Таким образом, типовое приложение, как правило, использует сочетание нескольких рассматриваемых в данной главе стилей.

Что такое архитектурный стиль?

Архитектурный стиль, иногда называемый архитектурным шаблоном – это набор принципов, высокоуровневая схема, обеспечивающая абстрактную инфраструктуру для семейства систем. Архитектурный стиль улучшает секционирование и способствует повторному использованию дизайна благодаря обеспечению решений часто встречающихся проблем. Архитектурные стили и шаблоны можно рассматривать как набор принципов, формирующих приложение. Гарлан (Garlan) и Шоу (Shaw) определяют архитектурный стиль как:

«... семейство систем с точки зрения схемы организации структуры. Точнее говоря, архитектурный стиль определяет набор компонентов и соединений, которые могут использоваться в экземплярах этого стиля, а также ряд ограничений по их возможным сочетаниям. Сюда могут относиться топологические ограничения на архитектурные решения (например, не использовать циклы). Описание стиля также может включать и другие ограничения, такие как, скажем, необходимость обработки семантики выполнения.»

[Дэвид Гарлан (David Garlan) и Мери Шоу (Mary Shaw), январь 1994, CMU-CS-94-166, «*An Introduction to Software Architecture*» (Введение в архитектуру ПО) по адресу http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf]

Понимание архитектурных стилей обеспечивает несколько преимуществ. Самое главное из них – общий язык. Также они дают возможность вести диалог, не касаясь технологий, т.е. обсуждать схемы и принципы, не вдаваясь в детали. Например, архитектурные стили позволяют сравнивать схему клиент/сервер с *n*-уровневой схемой приложения. Архитектурные стили можно организовать по их фокусу. В следующей таблице перечислены основные фокусные области и соответствующие архитектурные стили.

Категория	Архитектурные стили
Связь	Сервисно-ориентированная архитектура (SOA), шина сообщений
Развертывание	Клиент/сервер, N-уровневая, 3-уровневая
Предметная область	Дизайн на основе предметной области (Domain Driven Design)
Структура	Компонентная, объектно-ориентированная, многоуровневая архитектура

Обзор основных архитектурных стилей

В следующей таблице приводится список типовых архитектурных стилей, рассматриваемых в данной главе, и дается краткое описание каждого из них. В последующих разделах главы все эти стили обсуждаются подробно, а также даются рекомендации по выбору соответствующих стилей для конкретного приложения.

Архитектурный стиль/парадигма	Описание
Клиент/сервер	Система разделяется на два приложения, где клиент выполняет запросы к серверу. Во многих случаях в роли сервера выступает база данных, а логика приложения представлена процедурами хранения.
Компонентная архитектура	Дизайн приложения разлагается на функциональные или логические компоненты с возможностью повторного использования, предоставляющие тщательно проработанные интерфейсы связи.
Дизайн на основе предметной области ⁴	Объектно-ориентированный архитектурный стиль, ориентированный на моделирование сферы деловой активности и определяющий бизнес-объекты на основании сущностей этой сферы.
Многослойная архитектура	Функциональные области приложения разделяются на многослойные группы (уровни).
Шина сообщений	Архитектурный стиль, предписывающий использование

⁴ В некоторых источниках его так же называют проблемно-ориентированным проектированием (прим. научного редактора).

	программной системы, которая может принимать и отправлять сообщения по одному или более каналам связи, так что приложения получают возможность взаимодействовать, не располагая конкретными сведениями друг о друге.
<i>N</i> -уровневая / 3-уровневая	Функциональность выделяется в отдельные сегменты, во многом аналогично многослойному стилю, но в данном случае сегменты физически располагаются на разных компьютерах.
Объектно-ориентированная	Парадигма проектирования, основанная на распределении ответственности приложения или системы между отдельными многократно используемыми и самостоятельными объектами, содержащими данные и поведение.
Сервисно-ориентированная архитектура (SOA)	Описывает приложения, предоставляющие и потребляющие функциональность в виде сервисов с помощью контрактов и сообщений.

Сочетание архитектурных стилей

Архитектура программной системы практически никогда не ограничена лишь одним архитектурным стилем, зачастую она является сочетанием архитектурных стилей, образующих полную систему. Например, может существовать SOA-дизайн, состоящий из сервисов, при разработке которых использовалась многослойная архитектура и объектно-ориентированный архитектурный стиль.

Сочетание архитектурных стилей также полезно при построении Интернет Веб-приложений, где можно достичь эффективного разделения функциональности за счет применения многослойного архитектурного стиля. Таким образом можно отделить логику представления от бизнес-логики и логики доступа к данным. Требования безопасности организации могут обуславливать либо 3-уровневое развертывание приложения, либо развертывание с более чем тремя уровнями. Уровень представления может развертываться в пограничной сети, располагающейся между внутренней сетью организации и внешней сетью. В качестве модели взаимодействия на уровне представления может применяться шаблон представления с отделением (разновидность многослойного стиля), такая как Model-View-Controller (MVC)⁵. Также можно выбрать архитектурный стиль SOA и реализовать связь между Веб-сервером и сервером приложений посредством обмена сообщениями.

Создавая настольное приложение, можно реализовать клиент, который будет отправлять запросы к программе на сервере. В этом случае развертывание клиента и сервера можно выполнить с помощью архитектурного стиля клиент/сервер и использовать компонентную архитектуру для дальнейшего разложения дизайна на независимые компоненты, предоставляющие соответствующие интерфейсы. Применение объектно-ориентированного подхода к этим компонентам повысит возможности повторного использования, тестирования и гибкость.

⁵ Модель-представление-контроллер (прим. переводчика).

На выбор архитектурных стилей оказывает влияние множество факторов. Сюда входят способность организации к проектированию и реализации, возможности и опыт разработчиков, а также ограничения инфраструктуры и организации. Информация, приведенная в следующих разделах, поможет при выборе соответствующих стилей для приложений.

Архитектура клиент/сервер

Клиент/серверная архитектура описывает распределенные системы, состоящие из отдельных клиента и сервера и соединяющей их сети. Простейшая форма системы клиент/сервер, называемая 2-уровневой архитектурой – это серверное приложение, к которому напрямую обращаются множество клиентов.

Исторически архитектура клиент/сервер представляет собой настольное приложение с графическим UI, обменивающееся данными с сервером базы данных, на котором в форме хранимых процедур располагается основная часть бизнес-логики, или с выделенным файловым сервером. Если рассматривать более обобщенно, архитектурный стиль клиент/сервер описывает отношения между клиентом и одним или более серверами, где клиент инициирует один или более запросов (возможно, с использованием графического UI), ожидает ответы и обрабатывает их при получении. Обычно сервер авторизует пользователя и затем проводит обработку, необходимую для получения результата. Для связи с клиентом сервер может использовать широкий диапазон протоколов и форматов данных.

На сегодняшний день примерами архитектурного стиля клиент/сервер могут служить Веб-приложения, выполняющиеся в Интернете или внутренних сетях организаций; настольные приложения для операционной системы Microsoft Windows®, выполняющие доступ к сетевым сервисам данных; приложения, выполняющие доступ к удаленным хранилищам данных (такие как программы чтения электронной почты, FTP-клиенты и средства доступа к базам данных); инструменты и утилиты для работы с удаленными системами (такие как средства управления системой и средства мониторинга сети).

К другим разновидностям стиля клиент/сервер относятся:

- **Системы клиент-очередь-клиент.** Этот подход позволяет клиентам обмениваться данными с другими клиентами через очередь на сервере. Клиенты могут читать данные с и отправлять данные на сервер, который выступает в роли простой очереди для хранения данных. Благодаря этому клиенты могут распределять и синхронизировать файлы и сведения. Иногда такую архитектуру называют *пассивной очередью*.
- **Одноранговые (Peer-to-Peer, P2P) приложения.** Созданный на базе клиент-очередь-клиент, стиль P2P позволяет клиенту и серверу обмениваться ролями с целью распределения и синхронизации файлов и данных между множеством клиентов. Эта схема расширяет стиль клиент/сервер, добавляя множественные ответы на запросы, совместно используемые данные, обнаружение ресурсов и устойчивость при удалении участников сети.
- **Серверы приложений.** Специализированный архитектурный стиль, при котором приложения и сервисы размещаются и выполняются на сервере, и тонкий клиент

выполняет доступ к ним через браузер или специальное установленное на клиенте ПО. Примером является клиент, который работает с приложением, выполняющимся на сервере, через такую среду как Terminal Services (Службы терминалов).

Основные преимущества архитектурного стиля клиент/сервер:

- **Большая безопасность.** Все данные хранятся на сервере, который обычно обеспечивает больший контроль безопасности, чем клиентские компьютеры.
- **Централизованный доступ к данным.** Поскольку данные хранятся только на сервере, администрирование доступа к данным намного проще, чем в любых других архитектурных стилях.
- **Простота обслуживания.** Роли и ответственность вычислительной системы распределены между несколькими серверами, общающимися друг с другом по сети. Благодаря этому клиент гарантированно остается неосведомленным и не подверженным влиянию событий, происходящих с сервером (ремонт, обновление либо перемещение).

Рассмотрите возможность применения архитектуры клиент/сервер, если создаваемое приложение должно размещаться на сервере и не должно поддерживать множество клиентов; если создаются Веб-приложения, предоставляемые через Веб-браузер; если реализуются бизнес-процессы, которые будут использоваться в рамках организации; или если создаются сервисы для использования другими приложениями. Архитектурный стиль клиент/сервер, как и многие стили сетевых приложений, также подходит, если необходимо централизовать хранилище данных, функции резервного копирования и управления, или если разрабатываемое приложение должно поддерживать разные типы клиентов и разные устройства.

Тем не менее, традиционная 2-уровневая архитектура клиент/сервер имеет множество недостатков, включая тенденцию тесного связывания данных и бизнес-логики приложения на сервере, что может иметь негативное влияние на расширяемость и масштабируемость системы, и зависимость от центрального сервера, что негативно сказывается на надежности системы. Для решения этих проблем архитектурный стиль клиент/сервер был развит в более универсальный 3-уровневый (или N-уровневый), описываемый ниже, в котором устранены некоторые недостатки, свойственные 2-уровневой архитектуре клиент/сервер, и обеспечиваются дополнительные преимущества.

Компонентная архитектура

Компонентная архитектура описывает подход к проектированию и разработке систем с использованием методов проектирования программного обеспечения. Основное внимание в этом случае уделяется разложению дизайна на отдельные функциональные или логические компоненты, предоставляющие четко определенные интерфейсы, содержащие методы, события и свойства. В данном случае обеспечивается более высокий уровень абстракции, чем при объектно-ориентированной разработке, и не происходит концентрации внимания на таких вопросах, как протоколы связи или общее состояние.

Основной принцип компонентного стиля – применение компонентов, обладающих следующими качествами:

- **Пригодность для повторного использования.** Как правило, компоненты проектируются с обеспечением возможности их повторного использования в разных сценариях различных приложений. Однако некоторые компоненты создаются специально для конкретной задачи.
- **Замещаемость.** Компоненты могут без труда заменяться другими подобными компонентами.
- **Независимость от контекста.** Компоненты проектируются для работы в разных средах и контекстах. Специальные сведения, такие как данные о состоянии, должны не включаться или извлекаться компонентом, а передаваться в него.
- **Расширяемость.** Компонент может расширять существующие компоненты для обеспечения нового поведения.
- **Инкапсуляция.** Компоненты предоставляют интерфейсы, позволяющие вызывающей стороне использовать их функциональность, не раскрывая при этом детали внутренних процессов либо внутренние переменные или состояние.
- **Независимость.** Компоненты проектируются с минимальными зависимостями от других компонентов. Таким образом, компоненты могут быть развернуты в любой подходящей среде без влияния на другие компоненты или системы.

Обычно в приложениях используются компоненты пользовательского интерфейса (их часто называют *элементами управления*), такие как таблицы и кнопки, а также вспомогательные или служебные компоненты, предоставляющие определенный набор функций, используемых в других компонентах. К другому типу распространенных компонентов относятся ресурсоемкие компоненты, доступ к которым осуществляется нечасто, и активация которых выполняется «точно вовремя» (just-in-time, JIT) (обычно используется в сценариях с удаленными или распределенными компонентами); и компоненты с очередью, вызовы методов которых могут выполняться асинхронно за счет применения очереди сообщений, для хранения и пересылки.

Компоненты зависят от механизмов платформы, обеспечивающей среду выполнения. Часто эти механизмы называют просто *компонентной архитектурой*. Примерами такой архитектуры могут служить Объектная модель программных компонентов (component object model, COM) и Объектная модель распределенных программных компонентов (distributed component object model, DCOM) в Windows, Общая архитектура брокера объектных запросов (Common Object Request Broker Architecture, CORBA) и Серверные компоненты Java (Enterprise JavaBeans, EJB) на других платформах. Используемая компонентная архитектура описывает механизмы размещения компонентов и их интерфейсов, передачи сообщений или команд между компонентами и, в некоторых случаях, сохранения состояния.

Однако термин компонент часто используется в более общем смысле как *составная часть, элемент или составляющая*. Microsoft .NET Framework предоставляет поддержку для построения приложений с использованием такого компонентного подхода. Например, в

данном руководстве обсуждаются бизнес-компоненты и компоненты данных, которые обычно являются классами, скомпилированными в сборки .NET Framework. Они выполняются под управлением среды выполнения .NET Framework. Каждая сборка может включать более одного подобного компонента.

Рассмотрим основные преимущества компонентного архитектурного стиля:

- **Простота развертывания.** Существующие версии компонентов могут заменяться новыми совместимыми версиями, не оказывая влияния на другие компоненты или систему в целом.
- **Меньшая стоимость.** Использование компонентов сторонних производителей позволяет распределять затраты на разработку и обслуживание.
- **Простота разработки.** Для обеспечения заданной функциональности компоненты реализуют широко известные интерфейсы, что позволяет вести разработку без влияния на другие части системы.
- **Возможность повторного использования.** Применение многократно используемых компонентов означает возможность распределения затрат на разработку и обслуживание между несколькими приложениями или системами.
- **Упрощение с технической точки зрения.** Компоненты упрощают систему через использование контейнера компонентов и его сервисов. В качестве примеров сервисов, предоставляемых контейнером, можно привести активацию компонентов, управление жизненным циклом, организацию очереди вызовов методов, обработку событий и транзакции.

Для управления зависимостями между компонентами, поддержки слабого связывания и повторного использования могут использоваться шаблоны проектирования, такие как Dependency Injection (Внедрение зависимостей) или Service Locator (Локатор сервиса). Такие шаблоны часто применяются при создании составных приложений, сочетающих и повторно использующих компоненты во многих приложениях.

Рассмотрите возможность применения компонентной архитектуры, если уже располагаете подходящими компонентами или можете получить их от сторонних производителей; если предполагается, что создаваемое приложение будет преимущественно выполнять процедурные функции, возможно, с небольшим количеством вводимых данных или вообще без таковых; или если хотите иметь возможность сочетать компоненты, написанные на разных языках программирования. Также этот стиль подойдет для создания подключаемой или составной архитектуры, которая позволяет без труда заменять и обновлять отдельные компоненты.

Проектирование на основе предметной области

Проектирование на основе предметной области (Domain Driven Design, DDD) – это объектно-ориентированный подход к проектированию ПО, основанный на предметной области, ее элементах, поведении и отношениях между ними. Целью является создание программных систем, являющихся реализацией лежащей в основе предметной области, путем определения

модели предметной области, выраженной на языке специалистов в этой области. Модель предметной области может рассматриваться как каркас, на основании которого будут реализовываться решения.

Для применения DDD необходимо четко понимать предметную область, которую предполагается моделировать, или иметь способности для овладения такими знаниями. При создании модели предметной области группа разработки нередко работает в сотрудничестве со специалистами в данной области. Архитекторы, разработчики и специалисты в рассматриваемой области обладают разной подготовкой и во многих ситуациях будут использовать разные языки для описания своих целей, желаний и требований. Однако в рамках DDD вся группа договаривается использовать только один язык, ориентированный на предметную область и исключающий все технические жаргонизмы.

В качестве ядра ПО выступает модель предметной области, которая является прямой проекцией этого общего языка; с ее помощью путем анализа языка группа быстро находит пробелы в ПО. Создание общего языка это не просто упражнение по получению сведений от специалистов и их применению. Довольно часто в группах возникают проблемы с обменом информацией не только по причине непонимания языка предметной области, но также и из-за неопределенности языка самого по себе. Процесс DDD имеет целью не только реализацию используемого языка, но также улучшение и уточнение языка предметной области. Это, в свою очередь, положительно отражается на создаваемом ПО, поскольку модель является прямой проекцией языка предметной области.

Чтобы сделать модель строгой и полезной языковой конструкцией, как правило, приходится интенсивно использовать изоляцию и инкапсуляцию в рамках модели предметной области. Это может обусловить относительную дороговизну системы, основанной на DDD. Несмотря на то, что DDD обеспечивает массу преимуществ с технической точки зрения, таких как удобство в обслуживании, эта схема должна применяться лишь для сложных предметных областей, для которых процессы моделирования и лингвистического анализа обеспечивают безусловные преимущества при обмене сложной для понимания информацией и формулировании общего видения предметной области.

Основными преимуществами стиля DDD являются:

- **Обмен информацией.** Все участники группы разработки могут использовать модель предметной области и описываемые ею сущности для передачи сведений и требований предметной области с помощью общего языка предметной области, не прибегая к техническому жаргону.
- **Расширяемость.** Модель предметной области часто является модульной и гибкой, что упрощает обновление и расширение при изменении условий и требований.
- **Удобство тестирования.** Объекты модели предметной области характеризуются слабой связью и высокой связностью, что облегчает их тестирование.

Рассмотрите возможности применения DDD при работе со сложной предметной областью, если хотите улучшить процессы обмена информацией и ее понимания группой разработки, или если необходимо представить проект приложения на общем языке, понятном всем

заинтересованным сторонам. DDD также может быть идеальным подходом для сценариев обработки больших объемов сложных данных предприятия, с которыми сложно справится, используя другие методики.

Обзор методик проектирования на основе предметной области представлен в статье «*Domain Driven Design Quickly*» (Проектирование на основе предметной области, краткий обзор) по адресу <http://www.infoq.com/minibooks/domain-driven-design-quickly>. Также полезными будут книга Эрика Эванса (Eric Evans) «*Domain-Driven Design: Tackling Complexity in the Heart of Software*» (Проектирование на основе предметной области: как справляться со сложностями в сердце ПО) (Addison-Wesley, ISBN: 0-321-12521-5) и книга Джимми Нильссона (Jimmy Nilsson) «*Применение DDD и шаблонов проектирования*» (Вильямс, ISBN 978-5-8459-1296-1, 0-321-26820-2).

Многослойная архитектура⁶

Многоуровневая архитектура обеспечивает группировку связанный функциональности приложения в разных слоях, выстраиваемых вертикально, поверх друг друга. Функциональность каждого слоя объединена общей ролью или ответственностью. Слои слабо связаны, и между ними осуществляется явный обмен данными. Правильное разделение приложения на слои помогает поддерживать строгое разделение функциональности, что в свою очередь, обеспечивает гибкость, а также удобство и простоту обслуживания.

Многослойная архитектура описана как *перевернутая пирамида повторного использования*, в которой каждый слой агрегирует ответственности и абстракции уровня, расположенного непосредственно под ним. При строгом разделении на слои компоненты одного слоя могут взаимодействовать только с компонентами того же слоя или компонентами слоя, расположенного прямо под данным слоем. Более свободное разделение на слои позволяет компонентам слои взаимодействовать с компонентами того же и всех нижележащих слоев.

Слои приложения могут размещаться физически на одном компьютере (на одном уровне) или быть распределены по разным компьютерам (*n*-уровней), и связь между компонентами разных уровней осуществляется через строго определенные интерфейсы. Например, типовое Веб-приложение состоит из слоя представления (функциональность, связанная с UI), бизнес-слоя (обработка бизнес-правил) и слоя данных (функциональность, связанная с доступом к данным, часто практически полностью реализуемая с помощью высокогоуровневых инфраструктур доступа к данным). Подробно *n*-уровневая архитектура приложения рассматривается в данной главе позже в разделе [N-уровневая / 3-уровневая архитектура](#).

Рассмотрим общие принципы проектирования с использованием многослойной архитектуры:

- **Абстракция.** Многослойная архитектура представляет систему как единое целое, обеспечивая при этом достаточно деталей для понимания ролей и ответственостей отдельных слоев и отношений между ними.

⁶ Термины слой (layer) и уровень (tier) часто смешивают. Здесь и далее слой обозначает логическое разделение функциональности, а уровень – физическое разворачивание на разных системах (прим. научного редактора).

- **Инкапсуляция.** Во время проектирования нет необходимости делать какие-либо предположения о типах данных, методах и свойствах или реализации, поскольку все эти детали скрыты в рамках слоя.
- **Четко определенные функциональные слои.** Разделение функциональности между слоями очень четкая. Верхние слои, такие как слой представления, посылают команды нижним слоям, таким как бизнес-слой и слой данных, и могут реагировать на события, возникающие в этих слоях, обеспечивая возможность передачи данных между слоями вверх и вниз.
- **Высокая связность.** Четко определенные границы ответственности для каждого слоя и гарантированное включение в слой только функциональности, напрямую связанной с его задачами, поможет обеспечить максимальную связность в рамках слоя.
- **Возможность повторного использования.** Отсутствие зависимостей между нижними и верхними слоями обеспечивает потенциальную возможность их повторного использования в других сценариях.
- **Слабое связывание.** Для обеспечения слабого связывания между слоями связь между ними основывается на абстракции и событиях.

Примерами многослойных приложений могут служить бизнес-приложения (line-of-business, LOB), такие как системы бухгалтерского учета и управления заказчиками; Веб-приложения и Веб-сайты предприятий; настольные или смарт-клиенты предприятий с централизованными серверами приложений для размещения бизнес-логики.

Многослойная архитектура поддерживается рядом шаблонов проектирования. Например, под названием **Separated Presentation** (Отделение представления) объединяется ряд шаблонов, разделяющих взаимодействие пользователя с UI, представление, бизнес-логику и данные приложения, с которыми работает пользователь. Отделение представления позволяет создавать UI в графических дизайнерах, в то время как разработчики пишут управляющий код. Такое разделение функциональности на роли повышает возможность тестирования поведения отдельных ролей. Рассмотрим основные принципы шаблонов Separated Presentation:

- **Разделение функциональности.** Шаблоны Separated Presentation разделяют функциональность UI на роли; например, в шаблоне MVC имеется три роли: Модель (Model), Представление (View) и Контроллер (Controller). Модель представляет данные (возможно, модель предметной области, которая включает бизнес-правила); Представление отвечает за внешний вид UI; Контроллер обрабатывает запросы, работает с моделью и выполняет другие операции.
- **Уведомление на основе событий.** Шаблон Observer (Наблюдатель) обычно используется для уведомления Представления об изменениях данных, управляемых Моделью.
- **Делегированная обработка событий.** Контроллер обрабатывает события, формируемые элементами управления UI в Представлении.

В качестве примеров шаблонов Separated Presentation рассмотрим шаблон Passive View (Пассивное представление) и шаблон Supervising Presenter (Наблюдающий презентатор), также называемый Supervising Controller (Наблюдающий контроллер).

Основными преимуществами многослойного архитектурного стиля и применения шаблона Separated Presentation являются:

- **Абстракция.** Уровни обеспечивают возможность внесения изменений на абстрактном уровне. Используемый уровень абстракции каждого слоя может быть увеличен или уменьшен.
- **Изоляция.** Обновления технологий могут быть изолированы в отдельных слоях, что поможет сократить риск и минимизировать воздействие на всю систему.
- **Управляемость.** Разделение основных функций помогает идентифицировать зависимости и организовать код в секции, что повышает управляемость.
- **Производительность.** Распределение слоев по нескольким физическим уровням может улучшить масштабируемость, отказоустойчивость и производительность.
- **Возможность повторного использования.** Роли повышают возможность повторного использования. Например, в MVC Контроллер часто может использоваться повторно с другими совместимыми Представлениями для обеспечения характерного для роли или настроенного для пользователя представления одних и тех же данных и функциональности.
- **Тестируемость.** Улучшение тестируемости является результатом наличия строго определенных интерфейсов слоев, а также возможности переключения между разными реализациями интерфейсов слоев. Шаблоны Separated Presentation позволяют создавать во время тестирования фиктивные объекты, имитирующие поведение реальных объектов, таких как Модель, Контроллер или Представление.

Возможность применения многослойной архитектуры необходимо рассмотреть, если в вашем распоряжении имеются уже готовые уровни, подходящие для повторного использования в других приложения; если имеются приложения, предоставляющие подходящие бизнес-процессы через интерфейсы сервисов; или если создается сложное приложение и предварительное проектирование требует разделения, чтобы группы могли сосредоточиться на разных участках функциональности. Многослойная архитектура также будет уместна, если приложение должно поддерживать разные типы клиентов и разные устройства, или если требуется реализовать сложные и/или настраиваемые бизнес-правила и процессы.

Обратите внимание на шаблон Separated Presentation, если хотите получить улучшенную тестируемость и упростить обслуживание функциональности UI или отделить задачу создания дизайна UI от разработки управляющего кода. Эти шаблоны также будут полезны, если представление UI не содержит никакого кода обработки запросов и не реализует никакой бизнес-логики.

Архитектура, основанная на шине сообщений

Основанная на шине сообщений архитектура описывает принцип использования программной системы, которая может принимать и отправлять сообщения по одному или более каналам связи, обеспечивая, таким образом, приложениям возможность взаимодействия без необходимости знания конкретных деталей друг о друге. Это стиль проектирования, в котором взаимодействия между приложениями осуществляются путем передачи (обычно асинхронной) сообщений через общую шину. В типовых реализациях архитектуры, основанной на шине сообщений, используется либо маршрутизатор сообщений, либо шаблон Publish/Subscribe (Публикация/Подписка) и система обмена сообщениями, такая как Message Queuing (Очередь сообщений). Многие реализации состоят из отдельных приложений, обмен данными между которыми осуществляется путем отправки и приема сообщений по общим схемам и инфраструктуре. Шина сообщений обеспечивает возможность обрабатывать:

- **Основанное на сообщениях взаимодействие.** Все взаимодействие между приложениями основывается на сообщениях, использующих известные схемы.
- **Сложную логику обработки.** Сложные операции могут выполняться как часть многошагового процесса путем сочетания ряда меньших операций, каждая из которых поддерживает определенные задачи.
- **Изменения логики обработки.** Взаимодействие с шиной реализуется по общим схемам и с применением обычных команд, что обеспечивает возможность вставки или удаления приложений на шине для изменения используемой для обработки сообщений логики.
- **Интеграцию с разными инфраструктурами.** Использование модели связи посредством сообщений, основанной на общих стандартах, позволяет взаимодействовать с приложениями, разработанными для разных инфраструктур, таких как Microsoft .NET и Java.

Шины сообщений используются для обеспечения сложных правил обработки уже в течение многих лет. Такой дизайн обеспечивает подключаемую архитектуру, которая позволяет вводить приложения в процесс или улучшать масштабируемость, подключая к шине несколько экземпляров одного и того же приложения. К разновидностям шины сообщений относятся:

- **Сервисная шина предприятия (Enterprise Service Bus, ESB).** ESB основывается на шине сообщений и использует сервисы для обмена данными между шиной и компонентами, подключенными к шине. Обычно ESB обеспечивает сервисы для преобразования одного формата в другой, обеспечивая возможность связи между клиентами, использующими несовместимые форматы сообщений.
- **Шина Интернет-сервисов (Internet Service Bus, ISB).** Подобна сервисной шине предприятия, но приложения размещаются не в сети предприятия, а в облаке. Основная идея ISB – использование Унифицированных идентификаторов ресурсов (Uniform Resource Identifiers, URIs) и политик, управляющих логикой маршрутизации через приложения и сервисы в облаке.

К основным преимуществам архитектуры, основанной на шине сообщений, относятся:

- **Расширяемость.** Возможность добавлять или удалять приложения с шины без влияния на существующие приложения.
- **Невысокая сложность.** Приложения упрощаются, потому что каждому из них необходимо знать лишь, как обмениваться данными с шиной.
- **Гибкость.** Приведение набора приложений, составляющих сложный процесс, или схем связи между приложениями в соответствие изменяющимся бизнес-требованиям или требованиям пользователя просто путем внесения изменений в конфигурацию или параметры, управляющие маршрутизацией.
- **Слабое связывание.** Кроме предоставляемого приложением интерфейса для связи с шиной сообщений, нет никаких других зависимостей с самим приложением, что обеспечивает возможность изменения, обновления и замены его другим приложением, предоставляющим такой же интерфейс.
- **Масштабируемость.** Возможность подключения к шине множества экземпляров одного приложения для обеспечения одновременной обработки множества запросов.
- **Простота приложения.** Несмотря на то, что реализация шины сообщений усложняет инфраструктуру, каждому приложению приходится поддерживать лишь одно подключение к шине сообщений, а не множество подключений к другим приложениям.

Рассмотрите возможность применения архитектуры основанной на шине сообщений, если имеете существующие приложения, выполняющие задачи путем взаимодействия друг с другом, либо если хотите объединить множество шагов в одну операцию. Такой архитектурный стиль также подойдет при реализации решений, требующих взаимодействия с внешними приложениями или приложениями, размещенными в разных средах.

N-уровневая / 3-уровневая архитектура

N-уровневая и 3-уровневая архитектура являются стилями развертывания, описывающими разделение функциональности на сегменты, во многом аналогично многослойной архитектуре, но в данном случае эти сегменты могут физически размещаться на разных компьютерах, их называют уровнями. Данные архитектурные стили были созданы на базе компонентно-ориентированного подхода и, как правило, для связи используют методы платформы, а не сообщения.

Характеристиками N-уровневой архитектуры приложения являются функциональная декомпозиция приложения, сервисные компоненты и их распределенное развертывание, что обеспечивает повышенную масштабируемость, доступность, управляемость и эффективность использования ресурсов. Каждый уровень абсолютно независим от всех остальных, кроме тех, с которыми он непосредственно соседствует. N-ному уровню требуется лишь знать, как обрабатывать запрос от n+1 уровня, как передавать этот запрос на n-1 уровень (если таковой имеется), и как обрабатывать результаты запроса. Для обеспечения лучшей масштабируемости связь между уровнями обычно асинхронная.

N-уровневая архитектура обычно имеет не менее трех отдельных логических частей, каждая из которых физически размещается на разных серверах. Каждая часть отвечает за определенную функциональность. При использовании многослойного подхода слой развертывается на уровне, если предоставляемая этим слоем функциональность используется более чем одним сервисом или приложением уровня.

Примером N-уровневого/3-уровневого архитектурного стиля может служить типовое финансовое Веб-приложение с высокими требованиями к безопасности. Бизнес-слой в этом случае должен быть развернут за межсетевым экраном, из-за чего приходится развертывать слой представления на отдельном сервере в пограничной сети. Другой пример – типовой насыщенный клиент, в котором слой представления развернут на клиентских компьютерах, а бизнес-слой и слой доступа к данным развернуты на одном или более серверных уровнях.

Основными преимуществами N-уровневого/3-уровневого архитектурного стиля являются:

- **Удобство поддержки.** Уровни не зависят друг от друга, что позволяет выполнять обновления или изменения, не оказывая влияния на приложение в целом.
- **Масштабируемость.** Уровни организовываются на основании развертывания слоев, поэтому масштабировать приложение довольно просто.
- **Гибкость.** Управление и масштабирование каждого уровня может выполняться независимо, что обеспечивает повышение гибкости.
- **Доступность.** Приложения могут использовать модульную архитектуру, которая позволяет использовать в системе легко масштабируемые компоненты, что повышает доступность.

Рассмотрите возможность применения N-уровневой или 3-уровневой архитектуры, если требования по обработке уровней приложения отличаются настолько сильно, что может возникнуть перекос в распределении ресурсов, или существенно разнятся требования по безопасности уровней. Например, конфиденциальные данные не должны храниться на уровне представления, но могут размещаться на бизнес-уровне или уровне данных. N-уровневая или 3-уровневая архитектура также подойдет в случае, если требуется обеспечить возможность совместного использования бизнес-логики разными приложениями и имеется достаточное количество оборудования для выделения необходимого числа серверов для каждого уровня.

Используйте только три уровня, если создаете приложение для внутренней сети организации, где все серверы будут располагаться в закрытой сети; или Интернет-приложение, требования по безопасности которого не запрещают развертывание бизнес-логики на Веб-сервере или сервере приложений. Рассмотрите возможность применения более трех уровней, если соответственно требованиям по безопасности бизнес-логика не может быть развернута в пограничной сети, или если приложение интенсивно использует ресурсы, и для разгрузки сервера необходимо перенести эту функциональность на другой сервер.

Объектно-ориентированная архитектура

Объектно-ориентированная архитектура – это парадигма проектирования, основанная на разделении ответственостей приложения или системы на самостоятельные пригодные для

повторного использования объекты, каждый из которых содержит данные и поведение, относящиеся к этому объекту. При объектно-ориентированном проектировании система рассматривается не как набор подпрограмм и процедурных команд, а как наборы взаимодействующих объектов. Объекты обособлены, независимы и слабо связаны; обмен данными между ними происходит через интерфейсы путем вызова методов и свойств других объектов и отправки/приема сообщений. Основными принципами объектно-ориентированного архитектурного стиля являются:

- **Абстракция.** Позволяет преобразовать сложную операцию в обобщение, сохраняющее основные характеристики операции. Например, абстрактный интерфейс может быть широко известным описанием, поддерживающим операции доступа к данным через использование простых методов, таких как **Get** (Получить) и **Update** (Обновить). Другая форма абстракции – метаданные, используемые для обеспечения сопоставления двух форматов структурированных данных.
- **Композиция.** Объекты могут быть образованы другими объектами и по желанию могут скрывать эти внутренние объекты от других классов или предоставлять их как простые интерфейсы.
- **Наследование.** Объекты могут наследоваться от других объектов и использовать функциональность базового объекта или переопределять ее для реализации нового поведения. Более того, наследование упрощает обслуживание и обновление, поскольку изменения, вносимые в базовый объект, автоматически распространяются на все наследуемые от него объекты.
- **Инкапсуляция.** Объекты предоставляют функциональность только через методы, свойства и события и скрывают внутренние детали, такие как состояние и переменные, от других объектов. Это упрощает обновление или замену объектов и позволяет выполнять эти операции без влияния на другие объекты и код, требуется лишь обеспечить совместимые интерфейсы.
- **Полиморфизм.** Позволяет переопределять поведение базового типа, поддерживающего операции в приложении, путем реализации новых типов, которые являются взаимозаменяемыми для существующего объекта.
- **Отделение.** Объекты могут быть отделены от потребителя путем определения абстрактного интерфейса, реализуемого объектом и понятного потребителю. Это позволяет обеспечивать альтернативные реализации, не оказывая влияния на потребителей интерфейса.

Обычно объектно-ориентированный стиль используется для описания объектной модели, поддерживающей сложные научные или финансовые операции, либо описания объектов, представляющих реальные артефакты предметной области (такие как покупатель или заказ). Последнее чаще реализуется с применением более специализированного стиля проектирования на основе предметной области, который использует преимущества принципов объектно-ориентированной архитектуры. Более подробно об этом рассказывает раздел «[Проектирование на основе предметной области](#)» ранее в этой главе.

К основным преимуществам объектно-ориентированной архитектуры относятся:

- **Понятность.** Обеспечивается более близкое соответствие приложения реальным объектам, что делает его более понятным.
- **Возможность повторного использования.** Обеспечивается возможность повторного использования через полиморфизм и абстракцию.
- **Тестируемость.** Обеспечивается улучшенная тестируемость через инкапсуляцию.
- **Расширяемость.** Инкапсуляция, полиморфизм и абстракция гарантируют, что изменения в представлении данных не влияют на интерфейсы, предоставляемые объектами, что могло бы ограничить возможности связи и взаимодействия с другими объектами.
- **Высокая связность.** Размещая в объекте только функционально близкие методы и функции и используя для разных наборов функций разные объекты, можно достичь высокого уровня связности.

Рассмотрите возможность применения объектно-ориентированной архитектуры, если хотите моделировать приложение на базе реальных объектов и действий или уже имеете в своем распоряжении подходящие объекты и классы, соответствующие проектным и эксплуатационным требованиям. Объектно-ориентированный стиль также подойдет, если необходимо инкапсулировать логику и данные в компоненты, пригодные для повторного использования, или если имеется сложная бизнес-логика, которая требует абстракции и динамического поведения.

Сервисно-ориентированная архитектура

Сервисно-ориентированная архитектура (Service-oriented architecture, SOA) обеспечивает возможность предоставлять функциональность приложения в виде набора сервисов и создавать приложения, использующие программные сервисы. Сервисы слабо связаны, потому что используют основанные на стандартах интерфейсы, которые могут быть вызваны, опубликованы и обнаружены. Основная задача сервисов в SOA – предоставление схемы и взаимодействия с приложением посредством сообщений через интерфейсы, область действия которых является приложение, а не компонент или объект. Не следует рассматривать SOA-сервис как компонентный поставщик сервисов.

SOA-архитектура может обеспечить упаковку бизнес-процессов в сервисы, поддерживающие возможность взаимодействия и использующие для передачи информации широкий диапазон протоколов и форматов данных. Клиенты и другие сервисы могут выполнять доступ к локальным сервисам, выполняющимся на том же уровне, или к удаленным сервисам по сети.

Основными принципами архитектурного стиля SOA являются:

- **Сервисы автономны.** Обслуживание, разработка, развертывание и контроль версий каждого сервиса происходит независимо от других.
- **Сервисы могут быть распределены.** Сервисы могут размещаться в любом месте сети, локально или удаленно, если сеть поддерживает необходимые протоколы связи.

- **Сервисы слабо связаны.** Каждый сервис совершенно не зависит от остальных и может быть заменен или обновлен без влияния на приложения, его использующие, при условии предоставления совместимого интерфейса.
- **Сервисы совместно используют схему и контракт, но не класс.** При обмене данными сервисы совместно используют контракты и схемы, но не внутренние классы.
- **Совместимость основана на политике.** Политика, в данном случае, означает описание характеристик, таких как транспорт, протокол и безопасность.

Типовые сервисно-ориентированные приложения обеспечивают совместное использование информации, выполнение многоэтапных процессов (системы резервирования и онлайн-магазины), предоставление специальных отраслевых данных или сервисов между организациями и создание составных приложений, которые объединяют данные из многих источников.

Основными преимуществами SOA-архитектуры являются:

- **Согласование предметных областей.** Повторное использование общих сервисов со стандартными интерфейсами расширяет технологические и бизнес-возможности, а также сокращает стоимость.
- **Абстракция.** Сервисы являются автономными, доступ к ним осуществляется по формальному контракту, что обеспечивает слабое связывание и абстракцию.
- **Возможность обнаружения.** Сервисы могут предоставлять описания, что позволяет другим приложениям и сервисам обнаруживать их и автоматически определять интерфейс.
- **Возможность взаимодействия.** Поскольку протоколы и форматы данных основываются на отраслевых стандартах, поставщик и потребитель сервиса могут создаваться и развертываться на разных платформах.
- **Рационализация.** Сервисы обеспечивают определенную функциональность, устранивая необходимость ее дублирования в приложениях.

Рассмотрите возможность применения SOA-подхода, если имеете доступ к сервисам, которые желаете повторно использовать; можете приобрести подходящие сервисы у компании, предоставляющей услуги хостинга; хотите создать приложения, объединяющие различные сервисы в один UI; или создаете приложения в модели ПО + сервисы (Software plus Services, S+S), ПО как сервис (Software as a Service, SaaS) или приложения для размещения в облаке. SOA-стиль подходит, если требуется поддерживать связь посредством обмена сообщениями между сегментами приложения и предоставлять функциональность независимо от платформы, если хотите использовать интегрированные сервисы, такие как аутентификация, или хотите предоставлять сервисы, видимые в каталогах, с возможностью использования клиентами, которые заранее ничего не знают об интерфейсах.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу
<http://www.microsoft.com/architectureguide>.

Evans, Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2004.

Nilsson, Jimmy. Applying Domain-Driven Design and Patterns: With Examples in C# and .NET. Addison-Wesley, 2006.

Больше информации об архитектурных стилях можно найти в следующих источниках:

- «*An Introduction To Domain-Driven Design*» (Введение в проектирование на основе предметной области) по адресу <http://msdn.microsoft.com/en-us/magazine/dd419654.aspx>.
 - «*Domain Driven Design and Development in Practice*» (Проектирование и разработка на основе предметной области на практике) по адресу <http://www.infoq.com/articles/ddd-in-practice>.
 - «*Fear Those Tiers*» (Боязнь ярусов) по адресу <http://msdn.microsoft.com/en-us/library/cc168629.aspx>.
 - «*Layered Versus Client-Server*» (Сравнение многоуровневой архитектуры с архитектурой клиент-сервер) по адресу <http://msdn.microsoft.com/en-us/library/bb421529.aspx>.
 - «*Message Bus*» (Шина сообщений) по адресу <http://msdn.microsoft.com/en-us/library/ms978583.aspx>.
 - «*Microsoft Enterprise Service Bus (ESB) Guidance*» (Руководство по Microsoft Enterprise Service Bus (ESB)) по адресу <http://www.microsoft.com/biztalk/solutions/soa/esb.mspx>.
 - «*Separated Presentation*» (Отделение представления) по адресу <http://martinfowler.com/eaaDev/SeparatedPresentation.html>.
 - «*Services Fabric: Fine Fabrics for New-Era Systems*» (Сервисы: превосходные решения для систем новой эры) по адресу <http://msdn.microsoft.com/en-us/library/cc168621.aspx>.
-

4

Методика построения архитектуры и дизайна

Обзор

В данной главе представлена итеративная техника, которая может использоваться при продумывании и создании прототипа будущей архитектуры. Она поможет свести воедино ключевые решения, обсуждаемые в данном руководстве, включая решения по параметрам качества, архитектурным стилям, типам приложений, технологиям и сценариям развертывания.

Данная методика предполагает создание архитектуры в ходе процесса, состоящего из серий по пять основных шагов каждого. В свою очередь, каждый шаг разбит на отдельные аспекты, рассмотрением которых занимается далее это руководство. Итеративный процесс помогает выработать возможные варианты решений, которые в дальнейшем дорабатываются в ходе итераций и, в конечном счете, обеспечивают создание дизайна архитектуры, наиболее соответствующей разрабатываемому приложению. В конце процесса можно создать обзор архитектуры и представить его всем заинтересованным сторонам.

В зависимости от подхода, используемого вашей организацией для разработки ПО, архитектура может многократно пересматриваться в ходе жизненного цикла проекта. Эта методика подходит для дальнейшей доработки архитектуры, дополнения ее новыми аспектами, выявленными в последующий период сбора сведений, создания прототипов и фактической разработки.

Тем не менее, важно понимать, что это всего лишь один из возможных подходов. Существует множество других более формальных методов определения, анализа и представления архитектуры. Некоторые из них будут рассмотрены кратко в конце этой главы.

Исходные данные, выходные данные и этапы проектирования

Исходные данные проектирования помогают формализовать требования и ограничения, которые должна реализовать создаваемая архитектура. Обычно исходными данными являются варианты использования и сценарии поведения пользователя, функциональные требования, нефункциональные требования (включая параметры качества, такие как производительность, безопасность, надежность и другие), технологические требования, целевая среда развертывания и другие ограничения.

В ходе процесса разработки создается список значимых с точки зрения архитектуры вариантов использования, аспектов архитектуры, требующих специального внимания, и возможных архитектурных решений, которые удовлетворяют требованиям и ограничениям, выявленным в процессе проектирования. Общей техникой постепенной доработки дизайна до тех пор, пока он не будет удовлетворять всем требованиям и ограничениям, является итеративная методика, включающая пять основных этапов, как показано на рис. 1.

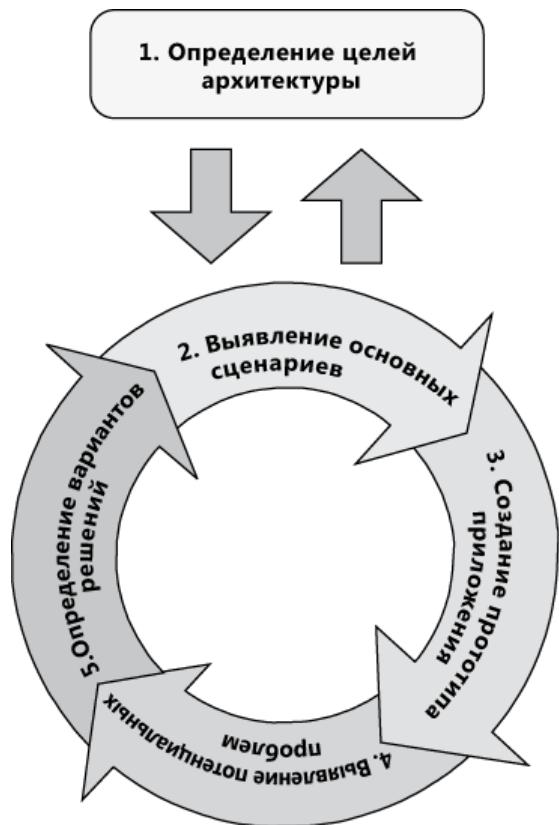


Рис. 3

Основные этапы итеративного процесса проектирования архитектуры

Этими этапами, которые более подробно рассматриваются в следующих разделах, являются:

1. **Определение целей архитектуры.** Наличие четких целей поможет сосредоточиться на архитектуре и правильном выборе проблем для решения. Точно обозначенные цели помогают определить границы каждой фазы: момент, когда завершена текущая фаза и все готово для перехода к следующей.
2. **Основные сценарии.** Используйте основные сценарии, чтобы сосредоточиться на том, что имеет первостепенное значение, и проверяйте возможные варианты архитектур на соответствие этим сценариям.
3. **Общее представление приложения.** Определите тип приложения, архитектуру развертывания, архитектурные стили и технологии, чтобы обеспечить соответствие вашего дизайна реальным условиям, в которых будет функционировать создаваемое приложение.

4. **Потенциальные проблемы.** Выявите основные проблемные области на основании параметров качества и потребности в сквозной функциональности. Это области, в которых чаще всего делаются ошибки при проектировании приложения.
5. **Варианты решений.** В каждой итерации должен быть создан «пилот» или прототип архитектуры, являющийся развитием и доработкой решения. Прежде чем переходить к следующей итерации, необходимо убедиться в соответствии этого прототипа основным сценариям, проблемам и ограничениям развертывания.

Такой процесс создания архитектуры предполагает итеративный и инкрементный подход. Сначала создается возможный вариант архитектуры – обобщенный дизайн, который может тестироваться по основным сценариям, требованиям, известным ограничениям, параметрам качества и Архитектурной Базе¹. В ходе доработки варианта архитектуры, выявляются дополнительные детали и сведения о дизайне, результатом чего становится расширение основных сценариев, корректировка общего представления приложения и подхода к решению проблем.

При итеративном походе к архитектуре часто есть соблазн выполнять итерации в горизонтальном направлении, в рамках отдельных слоев приложения, а не в вертикальном направлении, что заставляет думать о функциональности, выходящей за рамки слоев и составляющей отдельную возможность (вариант использования) значимую для пользователей. При выполнении итераций в горизонтальной плоскости есть угроза реализации большого числа функций до того, как пользователи смогут их проверить.

Не стремитесь создать архитектуру за одну итерацию. Каждая итерация должна раскрывать дополнительные детали. Но не увязайтесь в деталях, сосредоточьтесь на основных этапах и создавайте инфраструктуру, на которой может быть основана ваша архитектура и дизайн. В следующих разделах предлагаются рекомендации и сведения по каждому из этапов.

Определение целей архитектуры

Цели архитектуры – это задачи и ограничения, очерчивающие архитектуру и процесс проектирования, определяющие объем работ и помогающие понять, когда пора остановиться. Рассмотрим ключевые моменты в определении целей архитектуры:

- **Начальное определение задач архитектуры.** От этих задач будет зависеть время, затрачиваемое на каждую фазу проектирования архитектуры. Необходимо решить, что вы делаете: создаете прототип, проводите тестирование возможных вариантов

¹ Architecture Frame – термин предложенный авторами руководства, означает коллекцию архитектурных аспектов на которые нужно акцентировать внимание, а также шаблонов и инженерных подходов. Подробнее по адресу

<http://blogs.msdn.com/jmeier/archive/2008/09/22/architecture-frame.aspx> (прим. технического редактора).

реализации или выполняете длительный процесс разработки архитектуры для нового приложения.

- **Определение потребителей архитектуры.** Определите, будет ли разрабатываемая конструкция использоваться другими архитекторами, либо она предназначается для разработчиков и тестировщиков, ИТ-специалистов и руководителей. Учтите нужды и подготовленность целевой аудитории, чтобы сделать разрабатываемую конструкцию максимально удобной для них.
- **Определение ограничений.** Изучите все опции и ограничения применяемой технологии, ограничения использования и развертывания. Полностью разберитесь со всеми ограничениями в начале работы, чтобы не тратить время или не сталкиваться с сюрпризами в процессе разработки приложения.

Время и объем работ

На основании общих целей архитектуры можно планировать время, затрачиваемое на каждую из работ по проектированию. Например, на разработку прототипа может уйти лишь несколько дней, тогда как для создания полной и детально проработанной архитектуры сложного приложения потребуются месяцы и множество итераций. Исходя из своего понимания целей, оценивайте необходимое количество времени и сил для каждого этапа, это поможет прийти к видению результата и четкому определению целей и приоритетов архитектуры. Перечислим возможные цели:

- Создание полного дизайна приложения.
- Создание прототипа.
- Определение основных технических рисков.
- Тестирование возможных вариантов реализации.
- Создание общих моделей для облегчения понимания системы.

Каждая из этих целей обуславливает акцентирование внимания на разных аспектах при проектировании и требует разных временных затрат. Например, если требуется определить основные риски архитектуры системы аутентификации, на установление сценариев аутентификации, ограничений, налагаемых на архитектуру аутентификации, и возможных вариантов используемых технологий уйдет максимальное количество времени и энергии. Тогда как на ранних этапах разработки общей архитектуры приложения аутентификация будет лишь одним из многих аспектов, требующих выработки решений и их документирования.

Примерами работ по созданию архитектуры могут быть разработка прототипа для получения отзывов по пользовательскому интерфейсу Веб-приложения для обработки заказов, тестирование различных способов сопоставления данных о местоположении с результатами поиска, создание приложения для отслеживания заказов клиентов и проектирование архитектуры аутентификации и авторизации для приложения с целью обеспечения безопасности.

Ключевые сценарии

В контексте архитектуры и дизайна *вариант использования* (*use case*) – это описание ряда взаимодействий между системой и одним или более действующими лицами (либо пользователем, либо другой системой). *Сценарий* – это более широкое и всеобъемлющее описание взаимодействия пользователя с системой, чем ветвь варианта использования. Основной целью при продумывании архитектуры системы должно быть выявление нескольких ключевых сценариев, что поможет при принятии решения об архитектуре. Задача – найти баланс между целями пользователя, бизнеса и системы (как показано на рис. 1 главы 1, «[Что такое архитектура программного обеспечения?](#)»).

Ключевые сценарии – это наиболее важные сценарии для успеха создаваемого приложения. Ключевой сценарий можно определить как любой сценарий, отвечающий одному или более из следующих критериев:

- Он представляет проблемную область – значительную неизвестную область или область значительного риска.
- Он ссылается на существенный для архитектуры вариант использования (описывается в следующем разделе).
- Он представляет взаимодействие параметров качества с функциональностью.
- Он представляет компромисс между параметрами качества.

Например, сценарии аутентификации пользователей могут быть ключевыми сценариями, потому что являются пересечением параметра качества (безопасность) с важной функциональностью (регистрация пользователя в системе). В качестве другого примера можно привести сценарий, основанный на незнакомой или новой технологии.

Важные с точки зрения архитектуры варианты использования

Важные с точки зрения архитектуры варианты использования оказывают влияние на многие аспекты дизайна. Они играют особо важную роль в обеспечении будущего успеха создаваемого приложения. Эти варианты использования важны для приемки развернутого приложения и должны охватывать достаточно большую часть дизайна, чтобы быть полезными при оценке архитектуры. К важным с точки зрения архитектуры вариантам использования относятся:

- **Бизнес-критический (Business Critical).** Вариант использования, имеющий высокий уровень использования либо особую важность для пользователей или других заинтересованных сторон, по сравнению с другими функциями, или предполагающий высокий риск.
- **Имеющий большое влияние (High Impact).** Вариант использования охватывает и функциональность, и параметры качества, либо представляет сквозную функцию, имеющую глобальное влияние на слои и уровни приложения. Примерами могут служить особо уязвимые с точки зрения безопасности операции Create, Read, Update, Delete (CRUD).

После выявления важных с точки зрения архитектуры вариантов использования они могут применяться как средство оценки применимости или неприменимости возможных вариантов архитектуры приложения. Если вариант архитектуры охватывает больше вариантов использования или описывает существующие варианты использования более эффективно, обычно это свидетельствует о том, что данный вариант архитектуры является улучшением базовой архитектуры. Определение термина *вариант использования* дается в статье «*What is a Use Case?*» (Что такое вариант использования?) по адресу http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci334062,00.html.

Хороший вариант использования будет совпадать с пользовательским представлением, системным представлением и бизнес-представлением архитектуры. Используйте эти сценарии и варианты использования для тестирования своего дизайна и выявления возможных проблем. При продумывании вариантов использования и сценариев обратите внимание на следующее:

- На ранних этапах разработки дизайна сократите риск путем создания варианта архитектуры, поддерживающего важные с точки зрения архитектуры сквозные сценарии, затрагивающие все слои архитектуры.
- Используя модель архитектуры как руководство, вносите изменения в архитектуру, дизайн и код для реализации сценариев, функциональных требований, технологических требований, параметров качества и ограничений.
- Создайте модель архитектуры на основании известных на данный момент сведений и составьте список вопросов, ответы на которые должны быть даны в последующих сценариях и итерациях.
- Внеся существенные изменения в архитектуру и дизайн, создайте вариант использования, который будет отражать и применять эти изменения.

Общее представление приложения

Создайте общее представление того, как будет выглядеть готовое приложение. Это общее представление позволит сделать архитектуру более осозаемой, свяжет ее с реальными ограничениями и решениями. Создание общего представления приложения включает следующие действия:

1. **Определение типа приложения.** Прежде всего, определите, приложение какого типа создается. Будет ли это мобильное приложение, насыщенный клиент, насыщенное Интернет-приложение, сервис, Веб-приложение или некоторое сочетание этих типов? Более подробно о типовых архетипах приложений рассказывается в главе 20, «[Выбор типа приложения](#)».
2. **Определение ограничений развертывания.** При проектировании архитектуры приложения необходимо учесть корпоративные политики и процедуры, а также среду, в которой планируется развертывание приложения. Если целевая среда фиксированная или негибкая, конструкция приложения должна отражать существующие в этой среде ограничения. Также в конструкции приложения должны

быть учтены нефункциональные требования (Quality-of-Service, QoS), такие как безопасность и надежность. Иногда необходимо поступиться чем-либо в дизайне из-за ограничений в поддерживаемых протоколах или топологии сети. Выявление требований и ограничений, присутствующих между архитектурой приложения и архитектурой среды на ранних этапах проектирования позволяет выбрать соответствующую топологию развертывания и разрешить конфликты между приложением и целевой средой. Более подробно сценарии развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

3. **Определение значащих архитектурных стилей проектирования.** Определите, какие архитектурные стили будут использоваться при проектировании. Архитектурный стиль – это набор принципов. Он может рассматриваться как обобщенный шаблон, обеспечивающий абстрактную базу для семейства систем. Каждый стиль определяет набор правил, задающих типы компонентов, которые могут использоваться для компоновки системы, типы отношений, применяемых в компоновке, ограничения по способам компоновки и допущения о семантике компоновки. Архитектурный стиль улучшает секционирование и способствует возможности повторного использования дизайна благодаря предоставлению решений часто встречающихся проблем. Типовыми архитектурными стилями являются сервисно-ориентированная архитектура (Service Oriented Architecture, SOA), клиент/сервер, многослойная, шина сообщений и проектирование на основе предметной области. Приложения часто используют сочетание стилей. Более подробно наиболее распространенные в настоящее время архитектурные стили рассматриваются в главе 3, «[Архитектурные шаблоны и стили](#)».
4. **Выбор подходящих технологий.** Наконец, на основании типа приложения и других ограничений выбираем подходящие технологии и определяем, какие технологии будут использоваться в будущей системе. Основными факторами являются тип разрабатываемого приложения, предполагаемая топология развертывания приложения и предпочтительные архитектурные стили. Выбор технологий также зависит от политик организации, ограничений среды, квалификации штата и т.д. В следующем разделе описываются некоторые основные технологии Microsoft для каждого типа приложения.

Подходящие технологии

При выборе технологий для использования при проектировании обращайте внимание на то, что обеспечит выбранный архитектурный стиль, тип и основные параметры качества для приложения. Рассмотрим рекомендации, которые помогут выбрать технологии представления, реализации и связи, наиболее подходящие для каждого типа приложений на платформе Microsoft:

- **Мобильные приложения.** Для разработки приложения для мобильных устройств могут использоваться технологии слоя представления, такие как .NET Compact Framework, ASP.NET для мобильных устройств и Silverlight для мобильных устройств.
- **Насыщенные клиентские приложения.** Для разработки приложений с насыщенными UI, развертываемыми и выполняемыми на клиенте, могут

использоваться сочетания технологий слоя представления Windows Presentation Foundation (WPF), Windows Forms и XAML Browser Application (XBAP)¹.

- **Насыщенные клиентские Интернет-приложения (RIA).** Для развертывания насыщенных UI в рамках Веб-браузера могут использоваться подключаемый модуль Silverlight™ или Silverlight в сочетании с AJAX.
- **Веб-приложения.** Для создания Веб-приложений могут применяться ASP.NET WebForms², AJAX, Silverlight, ASP.NET MVC и ASP.NET Dynamic Data³.
- **Сервисные приложения.** Для создания сервисов, предоставляющих функциональность внешним потребителям систем и сервисов, могут использоваться Windows Communication Foundation (WCF) и ASP.NET Web services (ASMX)⁴.

Более подробно доступные для разных типов приложений технологии рассматриваются в следующих темах приложений в конце данного руководства:

- [Платформа приложений Microsoft](#)
- [Матрица технологий слоя представления](#)
- [Матрица технологий слоя доступа к данным](#)
- [Матрица интеграционных технологий](#)
- [Матрица технологий рабочего процесса](#)

Графическое представление архитектуры

Важно, чтобы вы могли графически представить разрабатываемую архитектуру. Независимо от того, делается ли это на бумаге, в виде слайдов или в другом формате, главное – показать основные ограничения и принятые решения для того, чтобы обозначить границы и начать обсуждение. На самом деле, это имеет двойную ценность. Если вы не можете наглядно представить архитектуру, значит, вы не полностью понимаете ее. Если вы смогли изобразить четкую и краткую диаграмму, она будет понятной остальным, и будет намного проще объяснить детали.

¹ Приложения браузера XAML (прим. переводчика).

² Веб-формы ASP.NET (прим. переводчика).

³ Динамические данные ASP.NET (прим. переводчика).

⁴ Веб-сервисы ASP.NET (прим. переводчика).

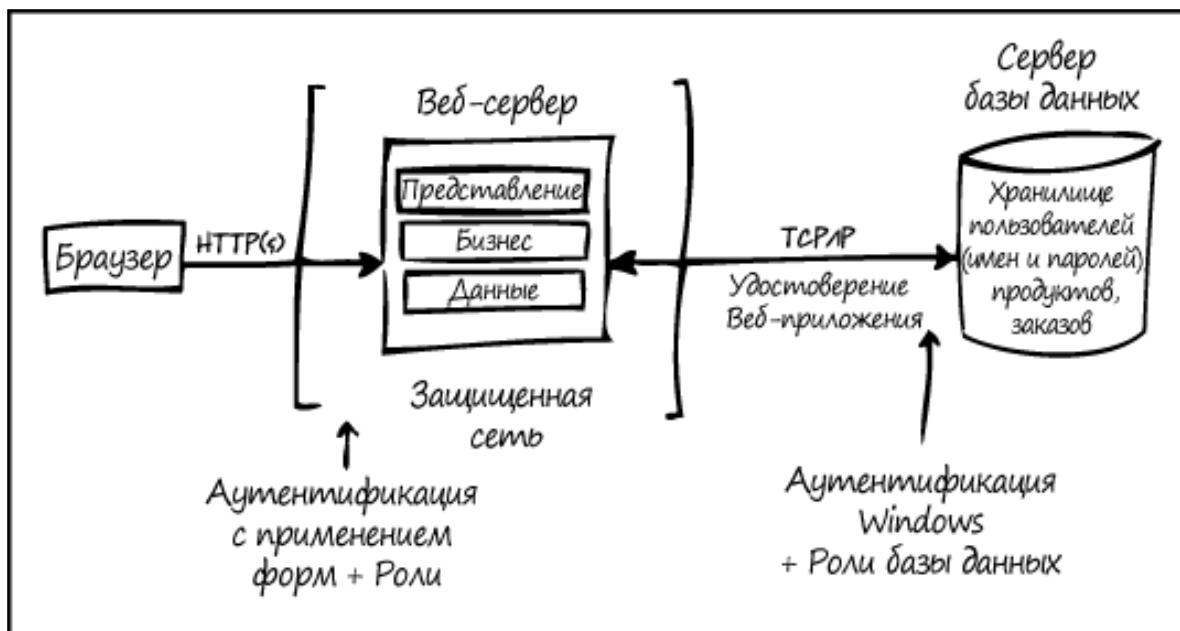


Рис. 4

Пример графического представления дизайна Веб-приложения в первом приближении с указанием протоколов и методов аутентификации, которые предполагается использовать.

Основные проблемы

Определите основные потенциальные проблемы архитектуры своего приложения, чтобы понять области, в которых наиболее вероятно возникновение ошибок. К потенциальным проблемам относятся появление новых технологий и критически важные бизнес-требования. Например, «Могу ли я переходить с одного сервиса стороннего производителя к другому?», «Могу ли я добавить поддержку нового типа клиента?», «Могу ли я быстро менять бизнес-правила оплаты услуг?» и «Могу ли я перейти к новой технологии для компонента X?». Несмотря на то, что это крайне обобщенные аспекты, как правило, при реализации они (и другие зоны риска) проецируются в *параметры качества и сквозную функциональность*.

Параметры качества

Параметры качества – это общие свойства архитектуры, которые оказывают влияние на поведение во время выполнения, дизайн системы и взаимодействие с пользователем. Та степень, с которой приложение обеспечивает требуемое сочетание параметров качества, таких как удобство и простота использования, производительность, надежность и безопасность, определяет успешность дизайна и общее качество программного продукта. При проектировании приложения, отвечающего любому из этих параметров, необходимо учесть влияние и других требований, должны быть проанализированы плюсы и минусы по отношению к другим параметрам качества. Важность или приоритетность каждого из параметров качества для разных систем разная. Например, для бизнес-приложения (line-of-business, LOB) производительность, масштабируемость, безопасность и удобство использования будут более важны, чем возможность взаимодействия с другими системами. А

вот для коробочного приложения такая возможность будет иметь большее значение, чем для LOB-приложения.

Параметры качества представляют функциональные области, которые потенциально могут оказывать влияние на все приложение, на все его слои и уровни. Некоторые параметры относятся ко всему дизайну системы, тогда как другие касаются только времени выполнения, времени проектирования или взаимодействия с пользователем. Следующий список систематизирует сведения о параметрах качества и помогает понять, на какие сценарии их влияние наиболее вероятно:

- **Общесистемные качества.** Общие качества системы в целом, такие как возможность технической поддержки и тестируемость.
 - **Качества времени выполнения.** Качества системы, проявляемые непосредственно во время выполнения, такие как доступность, возможность взаимодействия с другими системами, управляемость, производительность, надежность, масштабируемость и безопасность.
 - **Конструктивные качества.** Качества, отражающие дизайн системы, такие как концептуальная целостность, гибкость, удобство и простота обслуживания и возможность повторного использования.
 - **Пользовательские качества.** Удобство и простота использования системы.
-

Более подробно о том, как обеспечить реализацию соответствующих параметров качества конструкцией, рассказывается в главе 16, «[Параметры качества](#)».

Сквозная функциональность

Сквозная функциональность – это аспекты дизайна, которые могут применяться ко всем слоям, компонентам и уровням. Также это те области, в которых чаще всего делаются ошибки, имеющие большое влияние на дизайн. Приведем примеры сквозной функциональности:

- **Аутентификация и авторизация.** Как правильно выбрать стратегию аутентификации и авторизации, передачи идентификационных данных между слоями и уровнями и хранения удостоверений пользователей.
- **Кэширование.** Как правильно выбрать технику кэширования, определить данные, подлежащие кэшированию, где кэшировать данные и как выбрать подходящую политику истечения срока действия.
- **Связь.** Как правильно выбрать протоколы для связи между слоями и уровнями, обеспечения слабого связывания между слоями, осуществления асинхронного обмена данными и передачи конфиденциальных данных.
- **Управление конфигурацией.** Как выявить данные, которые должны быть настраиваемыми, где и как хранить данные конфигурации, как защищать конфиденциальные данные конфигурации и как обрабатывать их в серверной ферме или кластере.

- **Управление исключениями.** Как обрабатывать и протоколировать исключения и обеспечивать уведомления в случае необходимости.
- **Протоколирование и инструментирование.** Как выбрать данные, подлежащие протоколированию, как сделать протоколирование настраиваемым, и как определить необходимый уровень инструментирования.
- **Валидация.** Как определить, где и как проводить валидацию; как выбрать методики для проверки длины, диапазона, формата и типа; как предотвратить и отклонить ввод недопустимых значений; как очистить потенциально злонамеренный и опасный ввод; как определить и повторно использовать логику валидации на разных слоях и уровнях приложения.

Подробнее о том, как обеспечить правильную обработку сквозной функциональности, рассказывается в главе 17, «[Сквозная функциональность](#)».

Вопросы, требующие особого внимания при проектировании

Анализ параметров качества и сквозной функциональности в связи с имеющимися требованиями позволяет сосредоточиться на конкретных функциональных областях. Например, безопасность, несомненно, является жизненно важным фактором при проектировании и присутствует во многих слоях и во многих аспектах архитектуры. Сквозная функциональность, относящаяся к безопасности, является ориентиром, указывающим на области, на которых следует заострить внимание. Категории сквозной функциональности могут использоваться для разделения архитектуры приложения для дальнейшего анализа и выявления уязвимых мест приложения. Такой подход обеспечивает создание дизайна с оптимальным уровнем безопасности. При обсуждении сквозной функциональности относящейся к безопасности рекомендуется обратить внимание на следующие вопросы:

- **Аудит и протоколирование.** Кто, что сделал и когда? Приложение функционирует в нормальном режиме? Аудит занимается вопросами регистрации событий, связанных с безопасностью. Протоколирование касается того, как приложение публикует данные о своей работе.
- **Аутентификация.** Кто вы? Аутентификация – это процесс, при котором одна сущность четко и однозначно идентифицирует другую сущность, обычно это делается с помощью таких учетных данных, как имя пользователя и пароль.
- **Авторизация.** Что вы можете делать? Авторизация определяет, как приложение управляет доступом к ресурсам и операциям.
- **Управление конфигурацией.** В каком контексте выполняется приложение? К каким базам данных подключается? Как выполняется администрирование приложения? Как защищены эти настройки? Управление конфигурацией определяет, как приложение реализует эти операции и задачи.
- **Шифрование.** Как реализована защита секретов (конфиденциальных данных)? Как осуществляется защита от несанкционированного доступа данных и библиотек (целостности)? Как передаются случайные значения, которые должны быть

криптографически стойкими? Шифрование и криптография занимается вопросами реализации конфиденциальности и целостности.

- **Обработка исключений.** Что делает приложение при сбое вызова его метода? Насколько полные данные об ошибке оно предоставляет? Обеспечивает ли оно понятные для конечных пользователей сообщения об ошибках? Возвращает ли оно ценные сведения об исключении вызывающей стороне? Выполняется ли корректная обработка произошедшего сбоя? Предоставляет ли приложение администраторам необходимую информацию для проведения анализа основных причин сбоя? Обработка исключений касается того, как исключения обрабатываются в приложении.
- **Валидация входных данных.** Как определить, что поступаемые в приложение данные действительные и безопасные? Выполняется ли ограничение ввода через точки входа и кодировка вывода через точки выхода? Можно ли доверять таким источникам данных, как базы данных и общие файлы? Проверка ввода касается вопросов фильтрации, очистки или отклонения вводимых в приложение данных перед их дополнительной обработкой.
- **Конфиденциальные данные.** Как приложение работает с конфиденциальными данными? Обеспечивает ли оно защиту конфиденциальных данных пользователей и приложения? Здесь решаются вопросы обработки приложением любых данных, которые должны быть защищены либо при хранении в памяти, либо при передаче по сети, либо при хранении в постоянных хранилищах.
- **Управление сессиями.** Как приложение обрабатывает и защищает сеансы пользователей? Сеанс – это ряд взаимосвязанных взаимодействий пользователя и приложения.

Эти вопросы помогут принять основные проектные решения по безопасности приложения и задокументировать их как часть архитектуры. Например, на рис. 3 показано, как аспекты безопасности обозначены в архитектуре типового Веб-приложения.

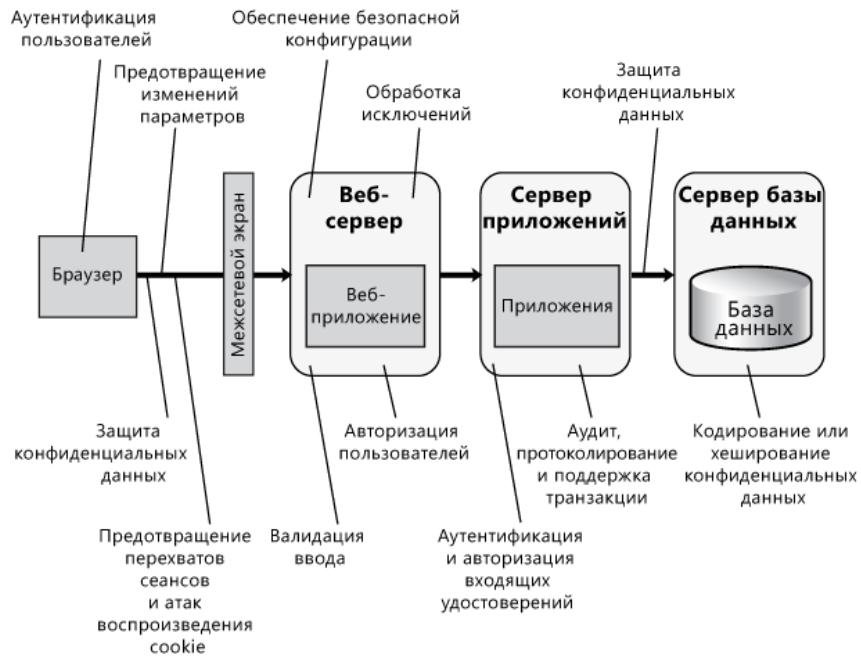


Рис. 5

Аспекты безопасности в архитектуре типового Веб-приложения.

Варианты решений

Определив основные проблемы, можно приступить к созданию исходной базовой архитектуры и ее детализации для получения возможного варианта архитектуры. В ходе этого процесса можно использовать пилотные архитектуры для более подробного рассмотрения определенных областей дизайна или для проверки новых идей. Затем выполняется проверка нового варианта архитектуры на соответствие основным сценариям и заданным требованиям и вновь повторяется итеративный цикл по доработке дизайна.

Важно, особенно если проектирование и разработка ведутся по гибкому процессу, чтобы итерация включала как проектирование архитектуры, так и разработку реализации. Это поможет избежать масштабного проектирования наперед.

Базовая архитектура и возможные варианты архитектуры

Базовая архитектура описывает существующую систему, то как она выглядит сегодня. Для нового проекта исходная базовая архитектура – это первое высокоуровневое представление архитектуры, на основании которого будут создаваться возможные варианты архитектуры. Возможный вариант архитектуры включает тип приложения, архитектуру развертывания, архитектурный стиль, выбранные технологии, параметры качества и сквозную функциональность.

На каждом этапе разработки дизайна будьте уверены, что понимаете основные риски и предпринимаете меры по их сокращению, проводите оптимизацию для эффективной и рациональной передачи проектных сведений и создаете архитектуру, обеспечивая гибкость и возможность реструктуризации. Возможно, архитектуру придется изменять несколько раз, использовать несколько итераций, возможных вариантов и множество пилотных архитектур.

Если возможный вариант архитектуры является улучшением, он может стать базой для создания и тестирования новых возможных вариантов.

Итеративный и инкрементный подход позволяет избавиться от крупных рисков сначала, итеративно формировать архитектуру и через тестирование подтверждать, что каждая новая базовая архитектура является улучшением предыдущей. Следующие вопросы помогут протестировать новый вариант архитектуры, полученный на основании «пилота» архитектуры:

- Данная архитектура обеспечивает решение без добавления новых рисков?
- Данная архитектура устраниет больше известных рисков, чем предыдущая итерация?
- Данная архитектура реализует дополнительные требования?
- Данная архитектура реализует важные с точки зрения архитектуры варианты использования?
- Данная архитектура реализует аспекты, связанные с параметрами качества?
- Данная архитектура реализует дополнительные аспекты сквозной функциональности?

Пилотные архитектуры

Пилотная архитектура (architectural spike) – это тестовая реализация небольшой части общего дизайна или архитектуры приложения. Ее назначение – анализ технических аспектов конкретной части решения для проверки технических допущений, выбора дизайна из ряда возможных вариантов и стратегий реализации или иногда оценка сроков реализации.

Пилотные архитектуры часто применяются в процессах гибкого или экстремального проектирования, но могут быть очень эффективным способом улучшения и доработки дизайна решения независимо от подхода к разработке. Благодаря их сфокусированности на основных частях общего проекта решения, пилотные архитектуры могут использоваться для решения важных технических проблем и для сокращения общих рисков и неопределенностей в дизайне.

Что дальше?

После завершения моделирования архитектуры можно приступать к доработке дизайна, планированию тестов и представлению решений остальным участникам процесса.

Руководствуйтесь следующими рекомендациями:

- При документировании возможных вариантов архитектуры и вариантов ее тестирования старайтесь не загромождать этот документ, что обеспечит простоту его обновления. Такой документ может включать сведения о целях, типе приложения, топологии развертывания, основных сценариях и требованиях, технологиях, параметрах качества и тестах.
- Используйте параметры качества для определения очертаний дизайна и реализации. Например, разработчики должны знать антишаблоны для выявленных

архитектурных рисков и использовать соответствующие проверенные схемы для решения данных проблем.

- Делитесь получаемыми сведениями с участниками группы и другими заинтересованными сторонами. К ним могут относиться группа разработки приложения, группа тестирования и администраторы сети или системные администраторы.

Анализ архитектуры

Анализ архитектуры приложения – критически важная задача, поскольку позволяет сократить затраты на исправление ошибок, как можно раньше выявить и исправить возможные проблемы. Анализ архитектуры следует выполнять часто: по завершении основных этапов проекта и в ответ на существенные изменения в архитектуре. Создавайте архитектуру, помня об основных вопросах задаваемых при таком анализе, это позволит как улучшить архитектуру, так и сократить время, затрачиваемое на каждый анализ.

Основная цель анализа архитектуры – подтверждение применимости базовой архитектуры и ее возможных вариантов, и также проверка соответствия предлагаемых технических решений функциональным требованиям и параметрам качества. Кроме того, анализ помогает обнаружить проблемы и выявить области, требующие доработки.

Оценки на основании сценариев

Оценки на основании сценариев – это мощный метод анализа дизайна архитектуры. При такой оценке основное внимание направлено на наиболее важные с точки зрения бизнеса и имеющие наибольшее влияние на архитектуру сценарии. Рассмотрите возможность применения одной из следующих типовых методик:

- **Метод анализа архитектуры ПО (Software Architecture Analysis Method, SAAM).** Изначально SAAM создавался для оценки модифицируемости, но позже был расширен для анализа архитектуры относительно показателей качества, таких как модифицируемость, портируемость, расширяемость, интегрируемость и функциональный охват.
- **Метод анализа архитектурных компромиссов (Architecture Tradeoff Analysis Method, ATAM).** ATAM – это доработанная и улучшенная версия SAAM, которая позволяет пересматривать архитектурные решения относительно требований параметров качества и того, насколько хорошо эти решения отвечают конкретным целевым показателям качества.
- **Активный анализ конструкции (Active Design Review, ADR).** ADR больше всего подходит для незавершенных архитектур или архитектур, находящихся в процессе разработки. Основное отличие этого метода в том, что анализ более сфокусирован на наборе проблем или отдельных разделах, а не на проведении общего анализа.
- **Активный анализ промежуточных конструкций (Active Reviews of Intermediate Designs, ARID).** ARID сочетает в себе подход ADR анализа архитектуры, находящейся в процессе разработки, с фокусом на наборе проблем и подход методов ATAM и

SAAM анализа на основании сценария с основным вниманием на параметрах качества.

- **Метод анализа рентабельности (Cost Benefit Analysis Method, СВАМ).** Метод СВАМ основное внимание уделяет анализу затрат, выгод и планированию последствий архитектурных решений.
- **Анализ модифицируемости на уровне архитектуры (Architecture Level Modifiability Analysis, ALMA).** ALMA оценивает модифицируемость архитектуры для систем бизнес-аналитики (business information systems, BIS).
- **Метод оценки семейства архитектур (Family Architecture Assessment Method, FAAM).** FAAM оценивает архитектуры семейства информационных систем с точки зрения возможности взаимодействия и расширяемости.

Методики анализа и оценки дизайнов архитектур рассматриваются в книге Пола Клементса (Paul Clements), Рика Казмана (Rick Kazman) и Марка Клейна (Mark Klein) «*Evaluating Software Architectures: Methods and Case Studies (SEI Series in Software Engineering)*» (Оценка программных архитектур: методы и практические примеры (Серия SEI по разработке ПО)) (Addison-Wesley Professional , ISBN-10: 020170482X, ISBN-13: 978-0201704822).

Представление дизайна архитектуры

Представление дизайна является очень важным для проведения анализа архитектуры, также это гарантирует, что все реализовано правильно. Дизайн архитектуры должен быть представлен всем заинтересованным сторонам, включая группу разработки, системных администраторов и операторов, владельцев бизнеса и др.

Один из способов представления архитектуры – карта важных решений. Карта это не территория, а абстракция, которая помогает раскрыть и представить архитектуру. Существует несколько широко известных методов описания архитектуры для ее представления:

- **4+1.** В данном подходе используется пять представлений готовой архитектуры. Четыре представления описывают архитектуру с разных точек зрения: логическое представление (например, объектная модель), представление процессов (например, аспекты параллелизма и синхронизации), физическое представление (схема программных уровней и функций в распределенной аппаратной среде) и представление для разработчиков. Пятое представление показывает сценарии и варианты использования ПО. Более подробно данный подход рассматривается в статье «*Architectural Blueprints—The “4+1” View Model of Software Architecture*» (Архитектурные проекты – модель программной архитектуры '4+1') по адресу <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>.
- **Гибкое моделирование.** Данный подход следует идеи того, что содержимое важнее представления. Это обеспечивает простоту, понятность, достаточную точность и единообразие создаваемых моделей. Простота документа гарантирует активное участие заинтересованных сторон в моделировании артефактов. Более подробно этот подход рассматривается в книге Скотта Амблера «*Гибкие*

технологии: экстремальное программирование и унифицированный процесс разработки (J. Wiley, ISBN-10: 0471202827, ISBN-13: 978-0471202820; Питер, ISBN 5-94723-545-5, 0-471-20282-7).

- **IEEE 1471.** IEEE 1471 – сокращенное название стандарта, формально известного как ANSI/IEEE 1471-2000, который обогащает описание архитектуры, в частности, придавая конкретное значение контексту, представлениям и срезам. Больше информации об этом можно найти в статье «*Recommended Practice for Architecture Description of Software-Intensive Systems*» (Рекомендованная практика описания архитектуры преимущественно программных систем) по адресу http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html.
- **Унифицированный язык моделирования (Unified Modeling Language, UML).** Этот подход обеспечивает три представления модели системы. Представление функциональных требований (функциональные требования системы с точки зрения пользователя, включая варианты использования); статическое структурное представление (объекты, атрибуты, отношения и операции, включая диаграммы классов); и представление динамического поведения (взаимодействие объектов и изменения внутреннего состояния объектов, включая диаграммы последовательностей, деятельности и состояний). Подробно об этом рассказывается в книге Мартина Фаулера «*UML, основы: краткое руководство по стандартному языку объектного моделирования*» (Addison-Wesley Professional, ISBN-10: 0321193687, ISBN-13: 978-0321193681; Символ-Плюс, ISBN 5-93286-060-X).

Дополнительные источники

Скотт Амблер. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. J. Wiley, 2002.

Paul Clements, Rick Kazman, and Mark Klein. Evaluating Software Architectures: Methods and Case Studies (SEI Series in Software Engineering). Addison-Wesley Professional, 2001.

Мартин Фаулер. UML, основы: краткое руководство по стандартному языку объектного моделирования, 2003.

Основы проектирования

В данный раздел руководства включены ряд тем, которые помогут понять основы многослойной архитектуры и обеспечат практическое руководство для создания некоторых типовых уровней¹², используемых большинством приложений, таких как уровень представления, бизнес-уровень, уровень данных и уровень сервисов. Раздел включает следующие главы:

- Глава 5 «[Рекомендации по проектированию многослойных приложений](#)»
- Глава 6 «[Рекомендации по проектированию слоя представления](#)»
- Глава 7 «[Рекомендации по проектированию бизнес-слоя](#)»
- Глава 8 «[Рекомендации по проектированию слоя доступа к данным](#)»
- Глава 9 «[Рекомендации по проектированию слоя сервисов](#)»

Как правило, каждый слой содержит ряд компонентов. При проектировании компонентов каждого слоя необходимо учесть широкий диапазон факторов, которые будут влиять на общий успех дизайна. В данный раздел руководства включены рекомендации по проектированию компонентов, которые помогут избежать типовых проблем и реализовать лучшие практики. Подробнее смотрите в следующих главах:

- Глава 10 «[Рекомендации по проектированию компонентов](#)»
- Глава 11 «[Проектирование компонентов представления](#)»
- Глава 12 «[Проектирование компонентов бизнес-слоя](#)»
- Глава 13 «[Проектирование бизнес-сущностей](#)»
- Глава 14 «[Проектирование компонентов рабочего процесса](#)»
- Глава 15 «[Проектирование компонентов слоя доступа к данным](#)»

Общее качество и последующий успех дизайна приложения зависит от того, насколько хорошо оно отвечает атрибутам качества, таким как безопасность, возможность повторного использования, производительность и удобство обслуживания. Кроме того, приложение, скорее всего, будет включать реализации общих аспектов, таких как обработка исключений, кэширование и протоколирование. В данном разделе представлены рекомендации по тому,

¹² Понятия слой (layer) и уровень (tier) очень близки, слой используется для логического разделения, а уровень для физического разворачивания. Но когда в контексте не важно или абсолютно понятно, какое именно разделение имеется в виду, используется слово уровень, хотя в английском языке наоборот (*прим. технического редактора*).

как необходимо реализовывать показатели качества и сквозную функциональность в своих приложениях. Этим вопросам посвящены следующие главы:

- Глава 16 «[Показатели качества](#)»
 - Глава 17 «[Сквозная функциональность](#)»
-

При проектировании приложения, особенно распределенного приложения, правильное проектирование инфраструктуры связи является ключом к успеху дизайна. Данный раздел руководства также поможет разобраться с требованиями к связи и реализовать дизайны, обеспечивающие соответствующие уровни отделения, безопасности и производительности. Подробно эти вопросы рассматриваются в Главе 18, «[Взаимодействие и обмен сообщениями](#)».

Наконец, необходимо подумать о развертывании приложения и учесть все ограничения, налагаемые физической средой, условиями работы с сетью и оборудованием, которое будет поддерживать приложение во время работы. Последняя глава данного раздела обсуждает сценарии физического развертывания и описывает некоторые проблемы, с которыми вам придется столкнуться при использовании многоуровневой модели развертывания, такие как безопасность. Подробнее смотрите в Главе 19, «[Физические уровни и развертывание](#)».

5

Рекомендации по проектированию многослойных приложений

Обзор

В данной главе обсуждается общая структура приложений с точки зрения логической группировки компонентов в отдельные слои, взаимодействующие друг с другом и с другими клиентами и приложениями. Разбиение на слои выполняется соответственно логическому делению компонентов и функциональности и не учитывает физического размещения компонентов. Слои могут размещаться как на разных уровнях, так и на одном. В данной главе будет рассмотрено, как разделять приложения на логические части, как выбирать соответствующую функциональную компоновку приложения и как обеспечить поддержку приложением множества типов клиентов. Также мы расскажем о сервисах, которые могут использоваться для предоставления логики в слоях приложений.

Важно понимать разницу между слоями и уровнями. *Слои (Layers)* описывают логическую группировку функций и компонентов в приложении, тогда как *уровни (tiers)* описывают физическое распределение функций и компонентов по серверам, компьютерам, сетям или удаленным местоположениям. Несмотря на то, что и для слоев, и для уровней применяется одна и так же терминология (представление, бизнес, сервисы и данные), следует помнить, что только уровни подразумевают физическое разделение. Размещение нескольких слоев на одном компьютере (одном уровне) – довольно обычное явление. Термин уровень используется в применении к схемам физического распределения, например, двухуровневое, трехуровневое, *n*-уровневое. Более подробно о физических уровнях и развертывании рассказывает глава 19, «[Физические уровни и развертывание](#)».

Логическое разделение на слои

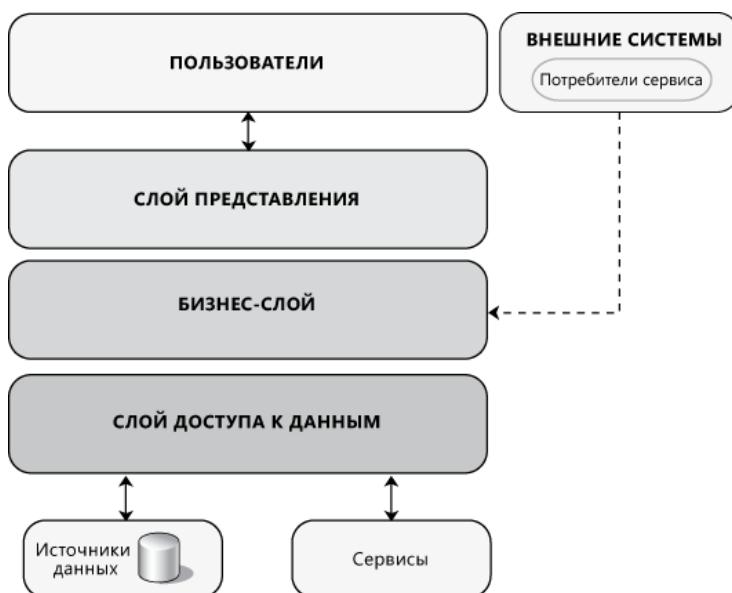
Независимо от типа проектируемого приложения и того, имеется ли у него пользовательский интерфейс или оно является сервисным приложением, которое просто предоставляет сервисы (не путайте со слоем сервисов приложения), его структуру можно разложить на логические группы программных компонентов. Эти логические группы называются слоями. Слои помогают разделить разные типы задач, осуществляемые этими компонентами, что упрощает создание

дизайна, поддерживающего возможность повторного использования компонентов. Каждый логический слой включает ряд отдельных типов компонентов, сгруппированных в подслои, каждый из подслоев выполняет определенный тип задач.

Определяя универсальные типы компонентов, которые присутствуют в большинстве решений, можно создать схему приложения или сервиса и затем использовать эту схему как эскиз создаваемого дизайна. Разделение приложения на слои, выполняющие разные роли и функции, помогает максимально повысить удобство и простоту обслуживания кода, оптимизировать работу приложения при различных схемах развертывания и обеспечивает четкое разграничение областей применения определенной технологии или принятия определенных проектных решений.

Слой представления, бизнес-слой и слой данных

На самом высоком и наиболее абстрактном уровне логическое представление архитектуры системы может рассматриваться как набор взаимодействующих компонентов, сгруппированных в слои. На рис. 1 показано упрощенное высокоуровневое представление этих слоев и их взаимоотношений с пользователями, другими приложениями, вызывающими сервисы, реализованные в бизнес-слое приложения, источниками данных, такими как реляционные базы данных или Веб-сервисы, обеспечивающие доступ к данным, и внешними или удаленными сервисами, используемыми приложением.



Логическое представление архитектуры многослойной системы

Эти слои физически могут располагаться на одном или разных уровнях. Если они размещаются на разных уровнях или разделены физическими границами, дизайн должен обеспечивать это. Более подробно данные вопросы рассматриваются в разделе [Этапы проектирования многослойной структуры](#) далее в данной главе.

Как показано на рис. 1, приложение может состоять из ряда базовых слоев. Типовой трехслойный дизайн, представленный на рис. 1, включает следующие слои:

- **Слой представления.** Данный слой содержит ориентированную на пользователя функциональность, которая отвечает за реализацию взаимодействием пользователя с системой, и, как правило, включает компоненты, обеспечивающие общую связь с основной бизнес-логикой, инкапсулированной в бизнес-слое. Более подробно о проектировании слоя представления рассказывает глава 6, «[Рекомендации по проектированию слоя представления](#)».
- **Бизнес-слой¹.** Этот слой реализует основную функциональность системы и инкапсулирует связанную с ней бизнес-логику. Обычно он состоит из компонентов, некоторые из которых предоставляют интерфейсы сервисов, доступные для использования другими участниками взаимодействия. Проектированию бизнес-слоя посвящена глава 7, «[Рекомендации по проектированию бизнес-слоя](#)». Более подробно проектирование компонентов бизнес-слоя рассматривается в главе 12, «[Проектирование компонентов бизнес-слоя](#)».
- **Слой доступа к данным.** Этот слой обеспечивает доступ к данным, хранящимся в рамках системы, и данным, предоставляемым другими сетевыми системами. Доступ может осуществляться через сервисы. Слой данных предоставляет универсальные интерфейсы, которые могут использоваться компонентами бизнес-слоя. Проектированию слоя данных посвящена глава 8, «[Рекомендации по проектированию слоя доступа к данным](#)». Больше информации по проектированию компонентов данных можно найти в главе 15, «[Проектирование компонентов слоя доступа к данным](#)».

Сервисы и слои

В первом приближении решение, основанное на сервисах, можно рассматривать как набор сервисов, взаимодействующих друг с другом путем передачи сообщений. Концептуально эти сервисы можно считать компонентами решения в целом. Однако каждый сервис образован программными компонентами, как любое другое приложение, и эти компоненты могут быть логически сгруппированы в слой представления, бизнес-слой и слой данных. Другие приложения могут использовать сервисы, не задумываясь о способе их реализации. Принципы многослойного дизайна, обсуждаемые в предыдущем разделе, в равной степени применяются и к основанным на сервисах решениям.

Слой сервисов

Обычным подходом при создании приложения, которое должно обеспечивать сервисы для других приложений, а также реализовывать непосредственную поддержку клиентов, является использование слоя сервисов, который предоставляет доступ к бизнес-функциональности приложения (рис. 2). Слой сервисов обеспечивает альтернативное представление, позволяющее клиентам использовать другой механизм для доступа к приложению.

¹ Его еще называют слоем бизнес-логики (прим. научного редактора).

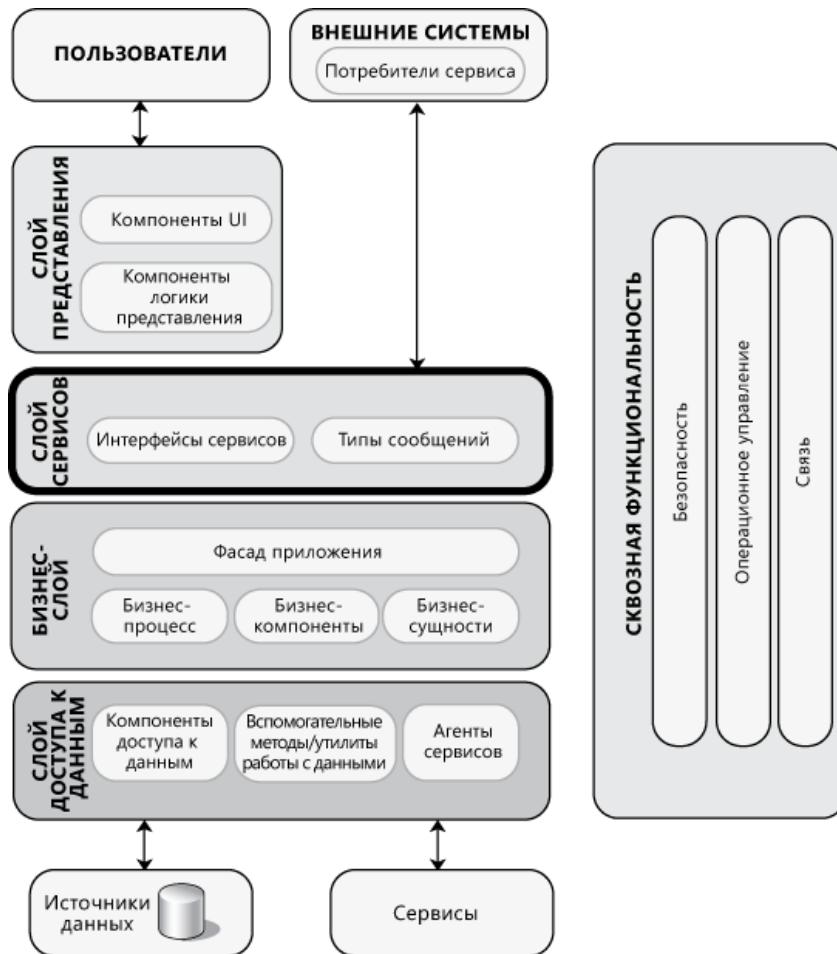


Рис. 2

Включение слоя сервисов в приложение

В данном сценарии пользователи могут выполнять доступ к приложению через слой представления, который обменивается данными с компонентами бизнес-слоя либо напрямую, либо через фасад приложения в бизнес-слое, если методы связи требуют композиции функциональности. Между тем, внешние клиенты и другие системы могут выполнять доступ к приложению и использовать его функциональность путем взаимодействия с бизнес-слоем через интерфейсы сервисов. Это улучшает возможности приложения для поддержки множества типов клиентов, способствует повторному использованию и более высокому уровню композиции функциональности в приложениях.

В некоторых случаях слой представления может взаимодействовать с бизнес-слоем через слой сервисов. Но это не является обязательным условием. Если физически слой представления и бизнес-слой располагаются на одном уровне, они могут взаимодействовать напрямую. Проектированию слоя сервисов посвящена глава 9, «[Рекомендации по проектированию слоя сервисов](#)». Более подробно взаимодействие слоев рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Этапы проектирования многослойной структуры

Приступая к проектированию приложения, прежде всего, сосредоточьтесь на самом высоком уровне абстракции и начинайте с группировки функциональности в слои. Далее следует

определить открытый интерфейс для каждого слоя, который зависит от типа создаваемого приложения. Определив слои и интерфейсы, необходимо принять решение о том, как будет развертываться приложение. Наконец, выбираются протоколы связи для обеспечения взаимодействия между слоями и уровнями приложения. Несмотря на то, что разрабатываемая структура и интерфейсы могут изменяться со временем, особенно в случае применения гибкой разработки, следование этим этапам гарантированно обеспечит рассмотрение всех важных аспектов в начале процесса. Обычно при проектировании используется следующая последовательность шагов:

- [Шаг 1 – Выбор стратегии разделения на слои](#)
 - [Шаг 2 – Выбор необходимых слоев](#)
 - [Шаг 3 – Принятие решения о распределении слоев и компонентов](#)
 - [Шаг 4 – Выяснение возможности сворачивания слоев](#)
 - [Шаг 5 – Определение правил взаимодействия между слоями](#)
 - [Шаг 6 – Определение сквозной функциональности](#)
 - [Шаг 7 – Определение интерфейсов между слоями](#)
 - [Шаг 8 – Выбор стратегии развертывания](#)
 - [Шаг 9 – Выбор протоколов связи](#)
-

[Шаг 1 – Выбор стратегии разделения на слои](#)

Разделение на слои представляет логическое распределение компонентов приложения по группам, выполняющим определенные роли и функции. Использование многослойного подхода может повысить удобство обслуживания приложения и упростить его масштабирование, если необходимо повысить производительность. Существует множество разных способов группировки взаимосвязанных функций в слои. Однако неправильное разделение на слои (слишком мало или слишком много) может лишь усложнить приложение, приводя к снижению общей производительности, удобства обслуживания и гибкости. Определение соответствующего уровня детализации при разделении приложения на слои – критически важный первый шаг в определении стратегии разделения на слои.

Также следует учесть, применяются ли слои исключительно для логического разделения либо для обеспечения физического разделения в случае необходимости. Пересечение границ слоев, особенно границ между физически удаленными компонентами, обуславливает возникновение издержек и снижение производительности. Однако общее повышение масштабируемости и гибкости приложения может быть намного более сильным аргументом по сравнению с падением производительности. Кроме того, разделение на слои может упростить оптимизацию производительности отдельных слоев без влияния на смежные слои.

В случае логического разделения взаимодействующие слои приложения будут развертываться на одном уровне и выполняться в одном процессе, что позволит применять более производительные механизмы связи, такие как прямые вызовы через интерфейсы компонентов. Однако чтобы воспользоваться преимуществами логического разделения на

слои и гарантировать гибкость в будущем, следует серьезно и тщательно подойти к вопросам обеспечения инкапсуляции и слабого связывания между слоями.

Для слоев, развертываемых на разных уровнях (разные физические компьютеры), взаимодействие со смежными слоями будет происходить по сети, и необходимо обеспечить, чтобы выбранный дизайн поддерживал подходящий механизм связи, который будет учитывать задержку связи и обеспечивать слабое связывание между слоями.

Определение того, какие слои приложения вероятнее всего будут развертываться на разных уровнях, а какие на одном, также является важной частью стратегии разделения на слои. Для обеспечения гибкости необходимо гарантированно обеспечить слабую связанность слоев. Это позволяет использовать преимущества большей производительности при размещении слоев на одном уровне и в случае необходимости развертывать их на множестве уровней.

Применение многослойного подхода может несколько усложнить дизайн и увеличить продолжительность подготовительного этапа разработки, но в случае правильной реализации существенно улучшит обслуживаемость, расширяемость и гибкость приложения. Сопоставьте преимущества, обеспечиваемые возможностью повторного использования и слабым связыванием при разделении на слои, с негативными последствиями их применения, такими как снижение производительности и повышение сложности. Тщательно продумайте, как разделить приложение на слои, и как слои будут взаимодействовать друг с другом; тем самым вы обеспечите хороший баланс производительности и гибкости. Как правило, выигрыш в гибкости и удобстве обслуживания, обеспечиваемый многослойной схемой, намного превышает сомнительное повышение производительности, которого можно достичь в тесно связанном дизайне, не использующем слои.

Описание общепринятых типов слоев и руководство по выбору необходимых слоев можно найти в разделе «[Логическое разделение на слои](#)» ранее в данной главе.

Шаг 2 – Выбор необходимых слоев

Существует множество разных способов группировки взаимосвязанных функций в слои. Самый распространенный в бизнес-приложениях подход – распределение функциональности представления, сервисов, доступа к данным и бизнес-функциональности по разным слоям. В некоторых приложениях также используются слои составления отчетов, управления и инфраструктуры.

Внимательно подходите к вопросу введения дополнительных слоев. Слой должен обеспечивать логическую группировку взаимосвязанных компонентов, которая заметно увеличивает удобство обслуживания, масштабируемость и гибкость приложения. Например, если приложение не предоставляет сервисы, возможно, отдельный слой сервисов не понадобится, тогда приложение будет включать только слой представления, бизнес-слой и слой доступа к данным.

Шаг 3 – Принятие решения о распределении слоев и компонентов

Слои и компоненты должны распределяться по разным физическим уровням, только если в этом есть необходимость. К типовым причинам реализации распределенного развертывания

относятся политики безопасности, физические ограничения, совместно используемая бизнес-логика и масштабируемость.

- Если компоненты представления Веб-приложения осуществляют синхронный доступ к компонентам бизнес-слоя и ограничения безопасности не требуют наличия границы доверия между слоями, рассмотрите возможность развертывания компонентов бизнес-слоя и слоя представления на одном уровне, это обеспечит максимальную производительность и управляемость.
- В насыщенных клиентских приложениях, в которых обработка UI выполняется на клиентском компьютере, вариант развертывания компонентов бизнес-слоя на отдельном бизнес-уровне может быть выбран по соображениями безопасности и для повышения управляемости.
- Развертывайте бизнес-сущности на одном уровне с кодом, их использующим. Это может означать их развертывание в нескольких местах, например, размещение копий на отдельном уровне представления или данных, логика которого использует или ссылается на эти бизнес-сущности. Развертывайте компоненты агентов сервиса на том же уровне, что и код, вызывающий эти компоненты, если ограничения безопасности не требуют наличия границы доверия между ними.
- Рассмотрите возможность развертывания асинхронных компонентов бизнес-слоя, компонентов рабочего процесса и сервисов с одинаковыми характеристиками загрузки и ввода/вывода на отдельном уровне. Это позволит настраивать инфраструктуру для обеспечения максимальной производительности и масштабируемости.

Шаг 4 – Выяснение возможности сворачивания слоев

В некоторых случаях имеет смысл свернуть слои. Например, в приложении, имеющем очень ограниченный набор бизнес-правил или использующем правила преимущественно для валидации, бизнес-логика и логика представления могут быть реализованы в одном слое. В приложении, которое просто извлекает данные с Веб-сервиса и отображает их, может иметь смысл просто добавить ссылки на Веб-сервис непосредственно в слой представления и использовать данные Веб-сервиса напрямую. В этом случае логически объединяются слои доступа к данным и представления.

Это лишь некоторые примеры того, когда имеет смысл сворачивание слоев. Тем не менее, группировка функциональности в слои является общим правилом. В некоторых случаях слой может выступать в роли прокси- или транзитного уровня, который обеспечивает инкапсуляцию или слабое связывание практически без предоставления функциональности. Но отделение этой функциональности позволит расширять ее в будущем без оказания влияния или с небольшим влиянием на другие слои.

Шаг 5 – Определение правил взаимодействия между слоями

Когда дело доходит до стратегии разделения на слои, необходимо определить правила взаимодействия слоев друг с другом. Основная цель задания правил взаимодействия – минимизация зависимостей и исключение циклических ссылок. Например, если два слоя

имеют зависимости от компонентов третьего слоя, появляется циклическая зависимость. Общим правилом, которого следует придерживаться в данном случае, является разрешение только одностороннего взаимодействия между слоями через применение одного из следующих подходов:

- **Взаимодействие сверху вниз.** Слои могут взаимодействовать со слоями, расположенными ниже, но нижние слои никогда не могут взаимодействовать с расположенными выше слоями. Это правило поможет избежать циклических зависимостей между слоями. Использование событий позволит оповещать компоненты расположенных выше слоев об изменениях в нижних слоях без введения зависимостей.
- **Строгое взаимодействие.** Каждый слой должен взаимодействовать только со слоем, расположенным непосредственно под ним. Это правило обеспечит строгое разделение, при котором каждый слой знает только о слое сразу под ним. Положительный эффект от этого правила в том, что изменения в интерфейсе слоя будут оказывать влияние только на слой, расположенный непосредственно над ним. Применяйте данный подход при проектировании приложения, которое предполагается расширять новой функциональностью в будущем, если хотите максимально сократить воздействие этих изменений; или при проектировании приложения, для которого необходимо обеспечить возможность распределения на разные уровни.
- **Свободное взаимодействие.** Более высокие слои могут взаимодействовать с расположенными ниже слоями напрямую, в обход других слоев. Это может повысить производительность, но также увеличит зависимости. Иначе говоря, изменения в нижнем слое может оказывать влияние на несколько расположенных выше слоев. Этот подход рекомендуется применять при проектировании приложения, которое гарантированно будет размещаться на одном уровне (например, самодостаточное насыщенное клиентское приложение), или при проектировании небольшого приложения, для которого внесение изменений, затрагивающих множество слоев, не потребует больших усилий.

Шаг 6 – Определение сквозной функциональности

Определившись со слоями, необходимо обратить внимание на функциональность, охватывающую все слои. Такую функциональность часто называют *сквозной функциональностью*. К ней относится протоколирование, валидация, аутентификация и управление исключениями. Важно выявить все сквозные функции приложения и по возможности спроектировать для каждой из них отдельные компоненты. Такой подход поможет обеспечить лучшую возможность повторного использования и обслуживания.

Избегайте смешения такого общего кода с кодом компонентов слоев, чтобы слои и их компоненты вызывали компоненты сквозной функциональности только для выполнения таких действий, как протоколирование, кэширование или аутентификация. Поскольку эта функциональность должна быть доступна для всех слоев, способ развертывания компонентов

сквозной функциональности должен обеспечивать это, даже если слои физически размещаются на разных уровнях.

Существуют разные подходы к реализации сквозной функциональности, от общих библиотек, таких как Enterprise Library группы patterns & practices, до методов Aspect Oriented Programming (AOP)¹, в которых код сквозной функциональности вставляется прямо в откомпилированный файл с помощью метаданных. Более подробно сквозная функциональность рассматривается в главе 17, «[Сквозная функциональность](#)».

Шаг 7 – Определение интерфейсов между слоями

Основная цель при определении интерфейса слоя – обеспечить слабое связывание между слоями. Это означает, что слой не должен раскрывать внутренние детали, от которых может зависеть другой слой. Вместо этого интерфейс слоя должен быть спроектирован так, чтобы свести до минимума зависимости путем предоставления открытого интерфейса, скрывающего детали компонентов слоя. Такое скрытие называется *абстракцией*. Существует множество способов реализовать ее. Предлагаем подходы, которые могут использоваться для определения интерфейса слоя:

- **Абстрактный интерфейс.** Может быть определен с помощью абстрактного базового класса или интерфейса, который выступает в роли описания типа для конкретных классов. Этот тип определяет общий интерфейс, используемый для взаимодействия с этим слоем. Такой подход также улучшает тестируемость, потому что позволяет использовать тестовые объекты (иногда называемые mock-объектами или фиктивными объектами), реализующие абстрактный интерфейс.
- **Общий тип проектирования.** Многие шаблоны проектирования определяют конкретные типы объектов, которые представляют интерфейс в разных слоях. Эти типы объектов обеспечивают абстракцию, которая скрывает детали, касающиеся слоя. Например, шаблон Table Data Gateway определяет типы объектов, которые представляют таблицы в базе данных и отвечают за реализацию SQL-запросов, необходимых для доступа к данным. Сущности, работающие с объектом, ничего не знают о SQL-запросах или деталях того, как объект подключается к базе данных и выполняет команды. Многие шаблоны проектирования базируются на абстрактных интерфейсах, но в основе некоторых из них лежат конкретные классы. Большинство шаблонов, такие как Table Data Gateway, хорошо задокументированы в этом отношении. Общие типы проектирования следует применять, если необходим способ быстро и просто реализовать интерфейс слоя или при реализации шаблона проектирования для интерфейса слоя.
- **Инверсия зависимостей**². Это такой стиль программирования, при котором абстрактные интерфейсы определяются вне или независимо от слоев. Тогда слои зависят не друг от друга, а от общих интерфейсов. Шаблон Dependency Injection

¹ Аспектно-ориентированное программирование (прим. переводчика).

² Подразумевается Dependency Inversion Principle (DIP, Принцип инверсии зависимости) (прим. научного редактора).

является типовой реализацией инверсии зависимостей. При использовании Dependency Injection контейнер описывает сопоставления, определяющие как находить компоненты, от которых могут зависеть другие компоненты, и контейнер может создавать и вводить эти зависимые компоненты автоматически. Подход с инверсией зависимостей обеспечивает гибкость и может помочь в реализации модульного дизайна, поскольку зависимости определяются конфигурацией, а не кодом. Также такой подход максимально упрощает тестирование, потому что позволяет вводить тестовые классы на разные уровни дизайна.

- **Основанный на обмене сообщениями.** Вместо взаимодействия с компонентами других слоев напрямую через вызов их методов или доступ к свойствам можно использовать связь посредством обмена сообщениями для реализации интерфейсов и обеспечения взаимодействия между слоями. Существует несколько решений для обмена сообщениями, такие как Windows Communication Foundation, Веб-сервисы и Microsoft Message Queuing, которые поддерживают взаимодействие через физические границы и границы процессов. Можно также комбинировать абстрактные интерфейсы с общим типом сообщений, используемым для определения структур данных для взаимодействия. Основное отличие подхода на основе сообщений в том, что для взаимодействия между слоями используется общий интерфейс, инкапсулирующий все детали взаимодействия. Этот интерфейс может определять операции, схемы данных, контракты уведомления о сбоях, политики системы безопасности и многие другие аспекты, относящиеся к обмену данными между слоями. Основанный на обмене сообщениями подход рекомендуется применять при реализации Веб-приложения и описании интерфейса между слоем представления и бизнес-слоем, который должен поддерживать множество типов клиентов, или если требуется поддерживать взаимодействие через физические границы и границы процессов. Также рассмотрите возможность применения такого подхода, если хотите формализовать взаимодействие или взаимодействовать с интерфейсом, не сохраняющим состояние, когда данные о состоянии передаются с сообщением.

Для реализации взаимодействия между слоем представления Веб-приложения и слоем бизнес-логики рекомендуется использовать подход на основе сообщений. Если бизнес-слой не сохраняет состояния между вызовами (другими словами, каждый вызов между слоем представления и бизнес-слоем представляет новый контекст), можно передавать данные контекста вместе с запросом и обеспечить общую модель обработки исключений и ошибок в слое представления.

Шаг 8 – Выбор стратегии развертывания

Существует несколько общих шаблонов, которые представляют структуры развертывания приложений, применяемые во многих решениях. Когда требуется выбрать наиболее подходящее решение развертывания для приложения, полезно сначала рассмотреть общие шаблоны. Только полностью разобравшись с разными схемами развертывания, можно переходить к конкретным сценариям, требованиям и ограничениям безопасности, чтобы определиться с наиболее подходящим шаблоном или шаблонами. Более подробно шаблоны развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

Шаг 9 – Выбор протоколов связи

Физические протоколы, используемые для связи между слоями или уровнями, играют основную роль в обеспечении производительности, безопасности и надежности приложения. Выбор протокола связи имеет еще большее значение для распределенного развертывания. Если компоненты размещаются физически на одном уровне, часто можно положиться на прямое взаимодействие этих компонентов. Но если компоненты и слои развернуты физически на разных серверах и клиентских компьютерах, как это происходит в большинстве сценариев, необходимо продумать, как обеспечить эффективную и надежную связь между компонентами этих слоев. Более подробно протоколы и технологии связи рассматриваются в главе 18, «[Взаимодействие и обмен сообщениями](#)».

6

Рекомендации по проектированию слоя представления

Обзор

В данной главе приводятся основные рекомендации по проектированию слоя представления. Из нее вы узнаете, какое место слой представления занимает в типовой архитектуре многослойного приложения, какие компоненты обычно включает и с какими основными проблемами придется столкнуться при проектировании слоя представления. Здесь вы найдете советы по проектированию, рекомендуемые этапы проектирования, соответствующие шаблоны проектирования и технологии.

Слой представления содержит компоненты, реализующие и отображающие пользовательский интерфейс, а также управляющие взаимодействием с пользователем. Этот слой, кроме компонентов, организовывающих взаимодействие с пользователем, включает элементы управления для ввода данных пользователем и их отображения. На рис. 1 показано место слоя представления в общей архитектуре приложения.

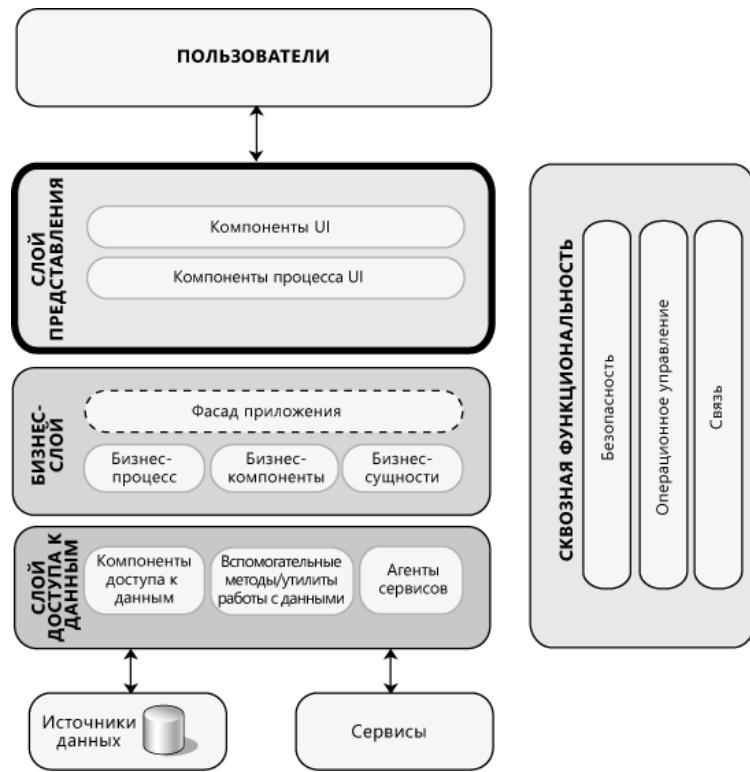


Рис. 7

Слой представления в типовом приложении и компоненты, которые он может включать

Слой представления обычно включает следующие компоненты:

- **Компоненты пользовательского интерфейса.** Это визуальные элементы приложения, используемые для отображения данных пользователю и приема пользовательского ввода.
- **Компоненты логики представления.** Логика представления – это код приложения, определяющий поведение и структуру приложения и не зависящий от конкретной реализации пользовательского интерфейса. При реализации шаблона Separated Presentation могут использоваться следующие компоненты логики представления: Презентатор (Presenter), Модель презентации (Presentation Model) и Модель Представления (View Model). Слой представления также может включать компоненты Модели слоя представления (Presentation Layer Model), которые инкапсулируют данные бизнес-слоя, или компоненты Сущности представления (Presentation Entity), которые инкапсулируют бизнес-логику и данные в форме, удобной для использования слоем представления.

Подробно компоненты, обычно применяемые в слое представления, рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)». Проектированию компонентов слоя представления посвящена глава 11, «[Проектирование компонентов представления](#)».

Общие принципы проектирования

При проектировании слоя представления необходимо учесть несколько основных факторов. Чтобы создаваемая конструкция гарантированно отвечала требованиям приложения, следуйте лучшим практикам и руководствуйтесь такими принципами:

- **Выбирайте соответствующий тип приложения.** От выбранного типа приложения будут существенно зависеть доступные варианты реализации слоя представления. Определитесь, будете ли вы реализовывать насыщенный (смарт) клиент, Веб-клиент или насыщенное Интернет-приложение (rich Internet application, RIA). Решение должно приниматься на основании требований, предъявляемых к приложению, и ограничений, накладываемых организацией или средой. Более подробно основные архетипы приложения, их преимущества и недостатки рассматриваются в главе 20, «[Выбор типа приложения](#)».
- **Выбирайте соответствующую технологию UI.** Разные типы приложений обеспечивают разные наборы технологий для разработки слоя представления. Каждый тип технологии обладает индивидуальными преимуществами, которые определяют возможность создания соответствующего слоя представления. Технологии, доступные для каждого типа приложений, рассматриваются в приложении А, «[Матрица технологий слоя представления](#)».
- **Используйте соответствующие шаблоны.** Шаблоны слоя представления (их список приводится в конце данной главы) предлагают проверенные решения обычных проблем, возникающих при проектировании слоя представления. Помните, что не все шаблоны применимы в равной степени ко всем архетипам, но общий шаблон Separated Presentation, в котором аспекты, касающиеся представления, отделены от базовой логики приложения, подходит для всех типов приложений. Специальные шаблоны, такие как MVC, MVP и Supervising Presenter, обычно используются в слое представления насыщенных клиентских приложений и RIA. Разновидности шаблонов Model-View-Controller (MVC) и Model-View-Presenter (MVP) могут применяться в Веб-приложениях.
- **Разделяйте функциональные области.** Используйте специальные компоненты UI для формирования визуального представления, отображения и взаимодействия с пользователем. В сложных случаях, или если хотите обеспечить возможность модульного тестирования, обратите внимание на специальные компоненты логики представления для управления обработкой взаимодействия с пользователем. Также применение специальных сущностей представления позволит представлять бизнес-логику и данные в форме, удобной для использования компонентами UI и логики представления. Сущности представления инкапсулируют бизнес-логику и данные бизнес-слоя в рамках слоя представления и используют их во многом так же, как используются бизнес-сущности в бизнес-слое. Разные типы компонентов слоя представления более подробно рассматриваются в главе 11, «[Проектирование компонентов представления](#)».
- **Учитывайте рекомендации по проектированию пользовательского интерфейса.** При проектировании слоя представления придерживайтесь рекомендаций своей

организации по UI, включая такие аспекты, как удобство и простота доступа, локализация и удобство использования. Для выбранных типа клиента и технологий ознакомьтесь с установленными рекомендациями по интерактивности UI, удобству использования, совместимости с системой, соответствуя предъявляемым требованиям, а также с шаблонами проектирования UI, и примените те из них, которые соответствуют дизайну и требованиям создаваемого приложения.

- **Придерживайтесь принципов ориентированного на пользователя проектирования.** До того, как приступать к проектированию слоя представления, поймите своего потребителя. Используйте опросы, исследования удобства использования, и интервью для выбора варианта пользовательского интерфейса, наиболее соответствующего требованиям заказчика.
-

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Кэширование](#)
 - [Сетевое взаимодействие](#)
 - [Композиция](#)
 - [Управление исключениями](#)
 - [Навигация](#)
 - [Взаимодействие с пользователем](#)
 - [Пользовательский интерфейс](#)
 - [Валидация](#)
-

Кэширование

Кэширование – один из лучших механизмов повышения производительности приложений и сокращения времени отклика UI. Кэширование данных в слое представления позволит оптимизировать поиск данных и избежать повторных сетевых вызовов, а также обеспечит возможность сохранять результаты ресурсоемких или часто повторяющихся процессов во избежание ненужной повторной обработки. При разработке стратегии кэширования руководствуйтесь следующими рекомендациями:

- Выберите подходящее размещение для кэша, например, в памяти или на диске. Если приложение развертывается на Веб-ферме, избегайте применения локальных кэшей, для которых необходима синхронизация. Для развертываний на Веб-фермах и фермах серверов приложений рекомендуется использовать систему управления транзакционными ресурсами, такую как Microsoft SQL Server®, или продукт,

поддерживающий распределенное кэширование, такой как технология Memcached производства компании Danga Interactive или механизм кэширования Velocity¹ от компании Microsoft. Тем не менее, если отличия между серверами не существенные или данные изменяются очень медленно, подойдет кэширование в памяти.

- При работе с кэшем в памяти применяйте кэширование данных в готовом к использованию виде. Например, кэшируйте не просто необработанные данные базы данных, а используйте необходимые приложению объекты. Однако избегайте кэширования часто меняющихся данных, поскольку затраты на поддержание актуальности кэша в этом случае могут превышать затраты на воссоздание или повторный запрос таких данных.
- Не кэшируйте незашифрованные конфиденциальные данные.
- Не полагайтесь на кэшированные данные, они могут быть удалены. Также не забывайте, что кэшированные данные могут устаревать. Например, использование кэшированных данных недопустимо при выполнении коммерческих операций, для которых требуется извлекать самые свежие данные.
- Позаботьтесь о правах доступа к кэшированным данным. В случае возможности доступа к данным пользователей, выполняющих разные роли, кэшируйте только те данные, для которых можете применить соответствующую авторизацию.
- При использовании множества потоков убедитесь, что любой доступ к кэшу является потокобезопасным.

Более подробно методики кэширования рассматриваются в главе 17, «[Сквозная функциональность](#)».

Сетевое взаимодействие

Обработка длительных запросов должна реализовываться с учетом времени отклика пользователя, а также удобства обслуживания и тестируемости кода. При проектировании обработки запросов руководствуйтесь следующими рекомендациями:

- Используйте асинхронные операции или фоновые потоки, чтобы избежать блокировки UI при выполнении длительных действий в приложениях Windows Forms и WPF. Применяйте AJAX для осуществления асинхронных запросов в ASP.NET. Обеспечивайте пользователю обратную связь о ходе выполнения процесса для длительных операций. Предоставьте пользователю возможность отменить длительную операцию.
- Избегайте смешения логики обработки UI и формирования визуального представления.

¹ В ноябре 2009 года данный продукт вошел в состав Windows Server AppFabric (прим. научного редактора).

- При выполнении ресурсоемких вызовов к удаленным ресурсам или слоям, таких как вызов Веб-сервисов или запрос к базе данных, проведите анализ, возможно, будет целесообразным делать эти запросы к сервису с детализированным интерфейсом (много мелких запросов) либо с обобщенным интерфейсом (один большой запрос). Если пользователь запрашивает большие объемы данных для выполнения операции, первым делом извлекайте лишь то, что необходимо для отображения и начала работы, а затем постепенно догружайте данные в фоновом потоке или по мере необходимости (примерами такого подхода является разделение данных на страницы и виртуализация UI). Используйте более масштабные вызовы к обобщенному сервису, когда пользователю нет необходимости ожидать завершения вызова.

Композиция

Подумайте, не упростится ли разработка и обслуживание приложения, если слой представления будет использовать независимые модули и блоки представления, компонуемые во время выполнения. Шаблоны композиции UI поддерживают создание блоков представления и компоновку пользовательского интерфейса во время выполнения. Эти шаблоны также помогают максимально сократить зависимости кода и библиотек, в противном случае, каждое изменение зависимостей требовало бы проведения повторной компиляции и развертывания модуля. Шаблоны композиции помогают реализовать совместное использование, повторное использование и замещение логики представления и блоков представления. При разработке стратегии композиции UI руководствуйтесь следующими рекомендациями:

- Избегайте зависимостей между компонентами. Например, по возможности широко используйте шаблоны абстракции во избежание проблем с обслуживанием. Применяйте шаблоны, поддерживающие внедрение зависимостей во время выполнения.
- Создавайте шаблоны с заполнителями. Например, используйте шаблон Template View (Представление по шаблону) для компоновки динамических Веб-страниц, чтобы обеспечить единообразие и возможность повторного использования.
- Используйте для создания представления модульные части, пригодные для повторного использования, например, шаблон Composite View (Составное представление) позволяет компоновать представление из модульных неделимых составляющих. Реализуйте в своем приложении разделение, применяя самостоятельные модули, добавление которых не составляет труда.
- Будьте внимательны при использовании компоновок, формируемых динамически во время выполнения, что может вызывать сложности с загрузкой и обслуживанием. Рассматривайте шаблоны и библиотеки сторонних производителей, которые поддерживают динамическую компоновку и внедрение визуальных блоков и представления во время выполнения.

- Для обмена данными между компонентами представления используйте шаблоны связи со слабой связанностью, такие как Publish/Subscribe. Это обеспечит снижение связанности компонентов и повышение тестируемости и гибкости.
-

Управление исключениями

Проектируйте для приложения централизованный механизм управления исключениями, который обеспечит единый подход к перехвату и обработке непредвиденных исключений (исключений, которые невозможно обработать локально). Особое внимание обратите на исключения, распространяющиеся через границы слоев и уровней, а также исключения, пересекающие границы доверия. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Обеспечьте понятные пользователю сообщения об ошибках для уведомления об ошибках приложения, но убедитесь, что не включаете конфиденциальные данные в страницы ошибок, в сообщения об ошибках, файлы журналов и аудита.
Постарайтесь по возможности привести приложение в согласованное состояние или, если это невозможно, обеспечьте завершение выполнения.
- Убедитесь, что перехватываете исключения, которые не будут перехвачены где-либо в другом месте (например, глобальном обработчике ошибок), и очищайте ресурсы и состояние после возникновения исключения. Глобальный обработчик исключений, отображающий глобальную страницу ошибок или сообщение об ошибке, пригодится для всех необрабатываемых исключений. Как правило, необрабатываемые исключения указывают на то, что система находится в несогласованном состоянии и, возможно, требует корректного завершения.
- Различайте системные исключения и ошибки бизнес-логики. Для ошибок бизнес-логики выводите на экран понятное пользователю сообщение об ошибке и предоставьте пользователю возможность повторить операцию. Для системных ошибок проведите проверку причины возникновения исключения (например, сбой сервиса или базы данных), выведите на экран понятное пользователю сообщение об ошибке и запротоколируйте сообщение об ошибке, что будет полезным при диагностике и устраниении неисправностей.
- Перехватывайте только те исключения, которые можете обработать, и избегайте использования собственных исключений без крайней надобности. Не применяйте исключения для управления потоком логики приложения.

Более подробно методики управления исключениями рассматриваются в главе 17, «[Сквозная функциональность](#)».

Навигация

Разрабатывайте стратегию навигации так, чтобы пользователи могли без труда перемещаться по экранам или страницам приложения, и чтобы можно было отделить функциональность навигации от формирования и обработки UI. Обеспечьте единообразное представление навигационных ссылок и элементов управления во всем приложении, чтобы не запутывать

пользователя и скрыть сложность приложения. При проектировании стратегии навигации руководствуйтесь следующими рекомендациями:

- Спроектируйте панели инструментов и меню, чтобы пользователи могли находить функции, предоставляемые UI.
- Обеспечьте предсказуемость реализации навигации между формами с помощью мастеров и определитесь с тем, как будете сохранять состояние навигации между сессиями в случае необходимости.
- Избегайте дублирования логики для обработчиков событий навигации и по возможности не указывайте в коде пути переходов. Для обработки обычных действий из множества источников используйте шаблон Command (Команда).

Взаимодействие с пользователем

Хороший стиль взаимодействия с пользователем и пользовательский интерфейс – вот что отличает хорошее приложение от плохого. Видимая производительность имеет намного большее значение, чем фактическая, поэтому решающее значение имеют управление ожиданиями и знание типовых схем взаимодействия с пользователем. Например, возможно, пользователи не будут против немногого дольше подождать загрузки страницы, если получают обратную связь со сведениями о предположительных сроках загрузки, и это ожидание не мешает продолжению их работы. В других ситуациях очень короткая задержка, для некоторых действий UI даже доли секунды, может создать ощущение, что приложение не отвечает. Изучите исследования удобства использования, опросы и интервью, чтобы понять, чего пользователи хотят и ожидают от приложения, и на основании этих сведений спроектируйте эффективный UI. При проектировании взаимодействия с пользователем руководствуйтесь следующими рекомендациями:

- Не создавайте перегруженные или слишком сложные интерфейсы. Обеспечивайте четкую схему работы с приложением для всех основных сценариев пользователя, применяйте цвета и ненавязчивую анимацию для привлечения внимания пользователя к важным изменениям в UI, таким как изменение состояния.
- Обеспечьте полезные и информативные сообщения об ошибках без предоставления в них конфиденциальных данных.
- Избегайте блокирования пользовательского интерфейса при выполнении действий, для завершения которых может потребоваться довольно длительное время. Как минимум, обеспечьте обратную связь о ходе выполнения действия и продумайте возможность отмены операции.
- Рассмотрите возможность предоставления пользователю дополнительных возможностей по конфигурированию и, где это возможно, персонализации UI, обеспечивая тем самым гибкий и настраиваемый UI.
- Продумайте, как будет обеспечиваться поддержка локализации и глобализации, даже если это не является основным требованием на начальных этапах проектирования. Добавление поддержки локализации и глобализации после

завершения проектирования может потребовать выполнения большого объема переработки и реструктуризации.

Пользовательский интерфейс

Пользовательский интерфейс должен реализовывать требования, касающиеся ввода данных и их валидации. Для обеспечения максимального удобства использования руководствуйтесь установленными в организации рекомендациями и отраслевыми рекомендациями, которые были выработаны в течение многих лет исследований проектирования и механизмов ввода. При выборе стратегии компоновки пользовательского интерфейса учтите, кто будет заниматься созданием UI: отдельная группа дизайнеров или группа разработки. Если UI будет создаваться дизайнерами, выбирайте подход к компоновке, не требующий написания кода или использования инструментов разработки. При проектировании пользовательского интерфейса руководствуйтесь следующими рекомендациями:

- Используйте шаблон Separated Presentation, такой как MVP, для отделения компоновки пользовательского интерфейса от его обработки. С помощью шаблонов обеспечьте единообразие внешнего вида и взаимодействия со всеми окнами UI и единый внешний вид и стиль взаимодействия для всех элементов UI, чтобы обеспечить максимальное удобство доступа и простоту использования. Избегайте слишком сложных компоновок.
- Используйте основанные на применении форм элементы управления для задач сбора данных; механизм ввода на базе документов для ввода данных в более свободной форме, таких как текстовые или графические документы; или подход с применением мастеров для более упорядоченных или управляемых рабочим процессом задач сбора данных.
- Избегайте применения жестко кодированных строк и внешних ресурсов для текстовых данных или данных компоновки (например, для поддержки языков с написанием справа налево), особенно если приложение будет подлежать локализации.
- Учтите удобство и простоту доступа. При продумывании стратегии ввода следует подумать о пользователях с ограниченными возможностями; например, реализовать ПО для преобразования текста в речь для слепых пользователей или увеличить текст и изображение для пользователей с проблемами зрения. По возможности поддерживайте сценарии работы только через клавиатуру для пользователей, которые не могут работать с координатно-указательными устройствами.
- Принимайте во внимание наличие экранов разных размеров и с разными разрешениями, существование разных типов устройств и разных типов ввода, таких как мобильные устройства, сенсорные экраны и устройства с возможностью рукописного ввода. Например, для сенсорных экранов обычно используются большие кнопки с большими расстояниями между ними, чем для обычных UI, предназначенных для ввода только с помощью мыши или клавиатуры. Для создания компоновки Веб-приложения пользуйтесь Каскадными таблицами стилей

(Cascading Style Sheets, CSS). Это обеспечит более высокую производительность формирования визуального представления и удобство обслуживания.

Валидация

Эффективная стратегия проверки пользовательского ввода и данных имеет критически важное значение для безопасности и корректной работы приложения. Определите правила валидации пользовательского ввода и бизнес-правила, существующие в слое представления. При проектировании стратегии проверки пользовательского ввода и данных руководствуйтесь следующими рекомендациями:

- Проверка пользовательского ввода должна проводиться в слое представления, тогда как проверка на соответствие бизнес-правилам – в бизнес-слое. Однако если бизнес-слой и слой представления разнесены физически, логика проверки на соответствие бизнес-правилам должна дублироваться в слое представления для улучшения удобства использования и уменьшения времени отклика. Этого можно достичь с помощью метаданных или путем применения одинаковых компонентов правил проверки в обоих слоях.
- Проектируйте стратегию проверки, руководствуясь целью ограничить, предотвратить и очистить злонамеренный ввод. Рассматривайте шаблоны и библиотеки сторонних производителей, которые могут помочь в реализации проверки. Определяйте бизнес-правила, обеспечивающие проверку, такие как границы транзакции, и реализуйте достаточно глубокую проверку, чтобы гарантировать выполнение этих правил.
- Убедитесь, что правильно обрабатываете ошибки валидации и избегайте предоставления конфиденциальных данных в сообщениях об ошибках. Кроме того, обеспечьте протоколирование сбоев при проверке, что поможет при выявлении злонамеренных действий.

Более подробно методики проверки рассматриваются в главе 17, «[Сквозная функциональность](#)».

Выбор технологии

Следующие рекомендации помогут выбрать соответствующую технологию реализации слоя представления для платформы Microsoft. Также они предлагают общие шаблоны, которые будут полезны для определенных типов приложений и технологий.

Мобильные приложения

Рассмотрим рекомендации по проектированию мобильного приложения:

- Используйте Microsoft Windows Compact Framework для создания полнофункциональных подключенных исполняемых приложений, приложений без постоянного подключения или приложений без подключения с возможностью

выполнения на широком диапазоне устройств, работающих под управлением Microsoft Windows Mobile.

- Для создания подключенных приложений, поддерживающих широкий диапазон мобильных устройств или требующих Wireless Application Protocol (WAP)¹, compact HTML (cHTML) либо подобные форматы формирования визуального представления, используйте ASP.NET для мобильных устройств.

Насыщенные клиентские приложения

Руководствуйтесь следующими рекомендациями при проектировании насыщенных клиентских приложений:

- Для создания приложений с поддержкой насыщенных мультимедийных и графических возможностей используйте Windows Presentation Foundation (WPF).
- Для создания приложений, загружаемых с Веб-сервера и выполняемых на клиенте Windows, используйте XAML Browser Applications (XBAP)².
- Для создания приложений, предназначенных преимущественно для работы с документами или для формирования отчетов, применяйте технологию Microsoft Office Business Application (OBA)³.
- При желании воспользоваться преимуществами широкого набора элементов управления сторонних производителей и инструментов быстрой разработки приложений примените Windows Forms. При проектировании составного приложения с использованием Windows Forms обратите внимание на предложение группы patterns & practices Smart Client Software Factory.
- При построении приложения с использованием WPF рассмотрите следующие возможности:
 - Для составных приложений воспользуйтесь предлагаемым группой patterns & practices руководством Composite Client Application Guidance (Руководство по проектирования составных клиентских приложений).
 - В случаях, когда созданием представления занимаются дизайнеры, а не обычные разработчики, используйте шаблон Presentation Model (Model-View-ViewModel), который является разновидностью шаблона Model-View-Controller (MVC), специально приспособленной для платформ разработки UI. Чтобы предоставить дизайнерам больше возможностей, реализуйте шаблоны данных (DataTemplate) через пользовательские элементы управления. Также для организации связи между представлением (View) и

¹ Протокол беспроводного доступа (прим. переводчика).

² Приложения браузера XAML (прим. переводчика).

³ Бизнес-приложения для Microsoft Office (прим. переводчика).

презентатором (Presenter) или моделью представления (View Model) применяйте WPF Commands.

Насыщенные Интернет-приложения

При проектировании насыщенного Интернет-приложения (Rich Internet Application, RIA) руководствуйтесь следующими рекомендациями:

- Для создания подключенных приложений, выполняющихся в браузере и поддерживающих широкий диапазон платформ, насыщенных графикой и поддерживающих богатые возможности работы с мультимедиа и представления, используйте Silverlight.
- Если принято решение создавать приложение с использованием Silverlight, рассмотрите следующие аспекты:
 - Рассмотрите возможность применения шаблона Presentation Model (Model-View-ViewModel), как описывалось ранее в данной главе.
 - При проектировании приложения, для которого планируется длительный срок эксплуатации и изменения в его ходе, воспользуйтесь предлагаемым группой patterns & practices руководством Composite Client Application Guidance.

Веб-приложения

При проектировании Веб-приложения руководствуйтесь следующими рекомендациями:

- Для создания приложений, доступ к которым будет осуществляться через Веб-браузер или специальный агент пользователя, используйте ASP.NET.
- Если принято решения создавать приложение с использованием ASP.NET, обратите внимание на следующие аспекты:
 - Используйте шаблоны страниц, чтобы упростить разработку и реализовать единообразный UI для всех страниц приложения.
 - Применение AJAX с ASP.NET Web Forms позволит повысить интерактивность и перенести больший объем обработки в фоновый режим с меньшим числом обновлений страниц.
 - Если требуется включить области насыщенного медиа содержимого и интерактивности, используйте элементы управления Silverlight с ASP.NET.
 - Повысить тестируемость приложения или реализовать более четкое разделение между пользовательским интерфейсом приложения и бизнес-логикой позволит использование ASP.NET MVC Framework. Эта инфраструктура поддерживает подход на базе model-view-controller к разработке Веб-приложений.

Предложения группы patterns & practices, Smart Client Software Factory и Composite Client Application Guidance, более подробно рассматриваются в разделе «[Предложения patterns & practices](#)» далее в этой главе.

Аспекты производительности

Для обеспечения максимальной производительности слоя представления руководствуйтесь следующими рекомендациями:

- Очень внимательно подходите к вопросу проектирования слоя представления, так чтобы он включал лишь функциональность, необходимую для обеспечения насыщенного взаимодействия с пользователем с быстрым откликом. Например, обеспечьте, чтобы слой представления мог проверять пользовательский ввод без необходимости обмена данными между уровнями. Это может потребовать присутствия правил проверки данных бизнес-слоя в слое представления, возможно, через применение метаданных или совместно используемых компонентов.
- Взаимодействие между слоем представления и бизнес-слоем или слоем сервисов приложения должно быть асинхронным. Это устранит неблагоприятное влияние на удобство использования и время отклика приложения при возникновении большой задержки или обрывов связи.
- Реализуйте кэширование данных, которые будут выводиться на экран пользователя, в слое представления. Например, можно кэшировать статистические данные, отображаемые в биржевых сводках.
- Как правило, избегайте сохранения данных сеанса или кэширования данных отдельных пользователя. Это можно делать, только если число пользователей ограничено или объем данных относительно невелик. Однако для коротких сеансов, возможно, удобным будет подход с кэшированием данных каждого пользователя на короткий промежуток времени. Не забывайте о привязанности данных в Веб-фермах или фермах серверов приложений при хранении или кэшировании данных сеансов или данных отдельных пользователей.
- При отображении данных всегда применяйте разделение данных на страницы. Не используйте запросы, которые могут возвращать неограниченное количество данных, и задавайте размер страницы, соответствующий отображаемому объему данных. Разбиение на страницы на стороне клиента должно применяться только в случае крайней необходимости.
- Пользуйтесь ViewState в ASP.NET аккуратно, поскольку это приводит к увеличению объема данных, участвующего в каждом обращении к серверу, и может негативно сказаться на производительности приложения. Где это необходимо, настройте страницы на использование сеансов только для чтения или выполнение вообще без сохранения сеансов.

Этапы проектирования слоя представления

Далее приводится рекомендуемый процесс проектирования слоя представления приложения. Данный подход гарантирует, что при проектировании архитектуры будут учтены все необходимые аспекты.

1. **Выберите тип клиента.** Выберите тип клиента, удовлетворяющий требованиям и соответствующий ограничениям инфраструктуры и развертывания организации. Например, если пользователи имеют мобильные устройства и будут подключаться к сети периодически, вероятно, мобильный клиент будет оптимальным выбором. Сведения, которые будут полезны при выборе соответствующего типа клиента, можно найти в главе 20, «[Выбор типа приложения](#)».
2. **Выберите технологию для реализации слоя представления.** Определитесь в общих чертах с функциональностью UI и слоя представления и выберите технологию UI, отвечающую этим требованиям и доступную для выбранного типа клиента. На данном этапе, если доступные технологии не подходят, возможно, необходимо пересмотреть выбранный тип клиента. Технологии, предлагаемые для каждого типа приложения, приведены в приложении В, «[Матрица технологий слоя представления](#)».
3. **Спроектируйте пользовательский интерфейс.** Определитесь с тем, должен ли UI быть модульным, и примите решение о том, как будут реализовано разделение функциональности в слое представления. Рассмотрите возможность использования шаблонов раздельного представления, таких как Presentation Model, MVC и MVP. Следуйте рекомендациям, приведенным в разделах Композиция, Навигация, Взаимодействие с пользователем и Пользовательский интерфейс данной главы ранее, это гарантирует создание UI, отвечающего всем поставленным требованиям. Доступные типы компонентов рассматриваются в главе 11, «[Проектирование компонентов представления](#)».
4. **Выберите стратегию валидации данных.** Используйте методики проверки данных для защиты своей системы от не пользующегося доверием ввода. Также определите подходящую стратегию для обработки и протоколирования исключений. Реализация соответствующих стратегий проверки, обработки исключений и протоколирования рассматривается в главе 17, «[Сквозная функциональность](#)».
5. **Выберите стратегию реализации бизнес-логики.** Вынесите бизнес-логику, чтобы отделить ее от кода слоя представления. Это сделает приложение более удобным в обслуживании, поскольку позволит изменять бизнес-логику без влияния на слой представления. Выбор методики зависит от сложности приложения; приводим общие принципы выбора:
 - **Валидация в UI.** В простых приложениях, где бизнес-логика используется только для проверки пользовательского ввода, ее можно разместить в компонентах UI. При этом бизнес-логику, не касающуюся валидации, включать в компоненты UI нельзя.

- **Компоненты обработки бизнес-задач.** Для более сложных приложений, поддерживающих транзакции или включающих базовую бизнес-логику, которая выходит за рамки проверки UI, продумайте размещение бизнес-логики в отдельных компонентах, которые будут использоваться компонентами UI.
 - **Модель предметной области.** В сложных корпоративных приложениях, в которых бизнес-логика используется совместно многими приложениями, вынесите компоненты бизнес-слоя в отдельный логический уровень. Это позволит развертывать бизнес-слой физически на отдельном уровне, что обеспечит масштабируемость и повторное использование другими приложениями.
 - **Управление бизнес-правилами.** В приложениях, которые должны поддерживать сложную проверку, координацию процессов и логику предметной области, размещайте бизнес-логику в подсистеме управления бизнес-правилами, такой как Microsoft BizTalk® Server.
6. **Определите стратегию связи с другими слоями.** Если приложение включает несколько слоев, таких как слой доступа к данным и бизнес-слой, определите стратегию связи между слоем представления и остальными слоями. При наличии выделенного бизнес-слоя слой представления будет обмениваться данными с ним. Если бизнес-слой нет, слой представления будет взаимодействовать непосредственно со слоем доступа к данным. Для доступа к другим слоям используйте следующие техники:
- **Прямые вызовы методов.** Если слой, с которым происходит взаимодействие, физически располагается на одном уровне со слоем представления, можно выполнять прямые вызовы методов.
 - **Веб-сервисы.** Используйте интерфейс Веб-сервиса, если хотите разделить доступ к данным или бизнес-логику с другими приложениями, если бизнес-слой или слой доступа к данным развернут на другом уровне, или если важно реализовать разделение функциональности. Если бизнес-логика или логика доступа к данным будет использоваться слоем представления в рамках внутренней сети, используйте WCF-службу,ирующую по протоколу TCP. Если бизнес-логика или логика доступа к данным будет использоваться слоем представления через Интернет, используйте WCF-службу,ирующую по протоколу HTTP. Если в бизнес-логике или логике доступа к данным предполагаются длительные вызовы, реализуйте асинхронную связь с помощью WCF и очереди сообщений.

Более подробно реализация соответствующих стратегий связи рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Шаблоны проектирования

Основные шаблоны проектирования для слоя представления организованы по категориям и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Кэширование	Cache Dependency (Кэш с зависимостью). Использует внешние данные для определения состояния данных, хранящихся в кэше. Page Cache (Кэш страниц). Улучшает время ответа динамических Веб-страниц, доступ к которым осуществляется довольно часто, но сами они меняются реже и потребляют большое количество ресурсов системы для воссоздания.
Композиция и компоновка	Composite View (Составное представление). Сочетает отдельные представления в композитное представление. Шаблон Presentation Model (Model-View-ViewModel) . Разновидность шаблона Model-View-Controller (MVC), приспособленная для современных платформ разработки UI, на которых созданием представления (View) занимаются, главным образом, дизайнеры, а не обычные разработчики. Template View (Представление по шаблону). Реализует представление общего шаблона и создает представления на базе этого шаблонного представления. Transform View (Представление с преобразованием). Преобразует данные, переданные на уровень представления, в HTML для отображения в UI. Two-Step View (Двухэтапное представление). Преобразует модель данных в логическое представление без какого-либо специального форматирования и затем преобразует это логическое представление, добавляя необходимое форматирование.
Управление исключениями	Exception Shielding (Экранирование исключений). При возникновении исключения предотвращает предоставление сервисом данных о его внутренней реализации.
Навигация	Application Controller (Контроллер приложений). Единое место обработки навигации между окнами. Front Controller (Контроллер запросов). Шаблон только для Веб, консолидирующий обработку запросов путем направления всех запросов через один объект-обработчик, который можно изменять во время выполнения с помощью декораторов. Page Controller (Контроллер страниц). Принимает ввод из запроса и обрабатывает его для конкретной страницы или действия Веб-сайта. Command (Команда). Инкапсулирует обработку запроса в отдельный командный объект с обычным интерфейсом выполнения.
Взаимодействие с пользователем	Asynchronous Callback (Асинхронный обратный вызов). Выполняет длительные задачи в отдельном потоке, выполняющемся в фоновом режиме, и обеспечивает потоку функцию для обратного вызова по завершении выполнения задачи. Chain of Responsibility (Цепочка обязанностей). Представляя

	возможность обработать запрос нескольким объектам, устраниет возможность связывания отправителя запроса с его получателем.
--	--

Более подробно шаблон Page Cache рассматривается в статье «*Enterprise Solution Patterns Using Microsoft .NET*» (Шаблоны корпоративных решений для Microsoft .NET) по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.

Более подробно шаблоны Application Controller, Front Controller, Page Controller, Template View, Transform View и Two-Step View рассматриваются в книге Мартина Фаулера «Архитектура корпоративных приложений». Addison-Wesley, 2002. Или по адресу <http://martinfowler.com/eaaCatalog>.

Более подробно шаблоны Composite View и Presentation Model рассматриваются в статье «*Patterns in the Composite Application Library*» (Шаблоны в библиотеке составного приложения) по адресу <http://msdn.microsoft.com/en-us/library/dd458924.aspx>.

Более подробно шаблон Chain of Responsibility рассматривается в статье «*Patterns in Practice*» (Шаблоны на практике) по адресу <http://msdn.microsoft.com/en-us/magazine/cc546578.aspx>.

Шаблону Command посвящена глава 5, «Поведенческие шаблоны», в книге Эрика Гамма, Ричарда Хелма, Ральфа Джонсона и Джона Влиссидеса «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2007.

Более подробно шаблон Asynchronous Callback рассматривается в статье «*Creating a Simplified Asynchronous Call Pattern for Windows Forms Applications*» (Создание упрощенного шаблона асинхронных вызовов для приложений Windows Forms) по адресу <http://msdn.microsoft.com/en-us/library/ms996483.aspx>.

Более подробно шаблоны Exception Shielding и Entity Translator рассматриваются в статье «*Useful Patterns for Services*» (Полезные шаблоны для сервисов) по адресу <http://msdn.microsoft.com/en-us/library/cc304800.aspx>.

Предложения patterns & practices

Узнать о дополнительных предложениях группы Microsoft patterns & practices можно из следующих источников:

- «*Composite Client Application Guidance*» по адресу <http://msdn.microsoft.com/en-us/library/cc707819.aspx>.
- «*Smart Client Software Factory*» по адресу <http://msdn.microsoft.com/en-us/library/aa480482.aspx>.
- «*Web Client Software Factory*» по адресу <http://msdn.microsoft.com/en-us/library/bb264518.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- «*Choosing the Right Presentation Layer Architecture*» (Как правильно выбрать архитектуру слоя представления) по адресу <http://msdn.microsoft.com/en-us/library/aa480039.aspx>.
 - Распределенная система кэширования объектов в памяти «*memcached*» по адресу <http://www.danga.com/memcached/>.
 - «*Microsoft Inductive User Interface Guidelines*» (Руководство по *Microsoft Inductive User Interface*) по адресу <http://msdn.microsoft.com/en-us/library/ms997506.aspx>.
 - «*Microsoft Project Code Named ‘Velocity’*» (Проект Microsoft под кодовым названием ‘Velocity’) по адресу <http://msdn.microsoft.com/en-us/data/cc655792.aspx>.
 - «*User Interface Text Guidelines*» (Руководство по проектированию текстовой части пользовательского интерфейса) по адресу <http://msdn.microsoft.com/en-us/library/bb158574.aspx>.
 - «*Design and Implementation Guidelines for Web Clients*» (Руководство по проектированию и реализации Веб-клиентов) по адресу <http://msdn.microsoft.com/en-us/library/ms978631.aspx>.
 - «*Web Presentation Patterns*» (Шаблоны представления в Веб) по адресу <http://msdn.microsoft.com/en-us/library/ms998516.aspx>.
-

7

Рекомендации по проектированию бизнес-слоя

Обзор

В этой главе приводятся основные рекомендации по проектированию бизнес-слоя приложения. Данная информация поможет разобраться, какое место бизнес-слой занимает в типовой архитектуре многослойного приложения, какие компоненты обычно включает и с какими основными проблемами придется столкнуться при проектировании бизнес-слоя. Здесь вы найдете советы по проектированию, рекомендуемые этапы проектирования, соответствующие шаблоны проектирования и технологии. На рис. 1 показано место бизнес-слоя в общей архитектуре приложения.

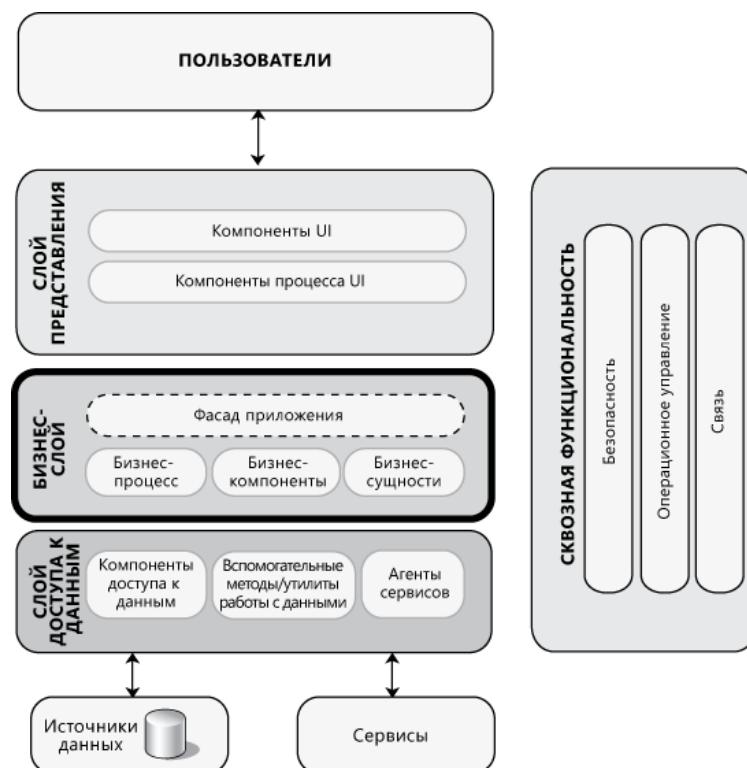


Рис. 8

Бизнес-слой в типовом приложении и компоненты, которые он может включать

Бизнес-слой обычно включает следующие компоненты:

- **Фасад приложения.** Этот необязательный компонент обычно обеспечивает упрощенный интерфейс для компонентов бизнес-логики, часто сочетая множество бизнес-операций в одну, что упрощает использование бизнес-логики. Это сокращает количество зависимостей, потому что вызывающим сторонам извне нет необходимости знать детали компонентов бизнес-слоя и отношения между ними.
- **Компоненты бизнес-логики.** Бизнес-логика, как и любая логика приложения, занимается вопросами извлечения, обработки, преобразования и управления данными приложения; применением бизнес-правил и политик и обеспечением непротиворечивости и действительности данных. Чтобы создать наилучшие условия для повторного использования, компоненты бизнес-логики не должны содержать поведения или логики приложения конкретного варианта использования или пользовательской истории. Компоненты бизнес-логики можно подразделить на две категории:
 - **Компоненты бизнес-процесса.** После того, как компоненты UI получили необходимые данные от пользователя и передали их в бизнес-слой, приложение может использовать эти данные для осуществления бизнес-процесса. Большинство бизнес-процессов включают множество этапов, которые должны выполняться в установленном порядке и могут взаимодействовать друг с другом через различные механизмы координирования. Компоненты бизнес-процесса определяют и координируют долгосрочные многоэтапные бизнес-процессы и могут быть реализованы с помощью инструментов управления бизнес-процессами. Они работают с компонентами бизнес-процесса, которые создают экземпляры и осуществляют операции с компонентами рабочего процесса. Более подробно компоненты бизнес-процесса рассматриваются в главе 14, «[Проектирование компонентов рабочего процесса](#)».
 - **Компоненты бизнес-сущностей.** Бизнес-сущности, или более общее название – бизнес-объекты, инкапсулируют бизнес-логику и данные, необходимые для представления в приложении объектов реального мира, таких как заказчики (Customers) или заказы (Orders). Они хранят значения данных и предоставляют их через свойства; содержат бизнес-данные приложения и управляют ими; и предоставляют программный доступ с сохранением состояния к бизнес-данным и связанной функциональности. Бизнес-сущности также проверяют данные, содержащиеся в сущности; они инкапсулируют бизнес-логику для обеспечения непротиворечивости данных, а также реализации бизнес-правил и поведения. Более подробно компоненты бизнес-сущностей рассматриваются в главе 13, «[Проектирование бизнес-сущностей](#)».

Подробно компоненты, обычно используемые в бизнес-слое, рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)».

Проектированию компонентов бизнес-слоя посвящена глава 12, «[Проектирование компонентов бизнес-слоя](#)».

Общие принципы проектирования

При проектировании бизнес-слоя перед архитектором ПО стоит задача максимально упростить приложение путем разделения задач на разные функциональные области. Например, логика обработки бизнес-правил, бизнес-процессов и бизнес-сущностей – все они представляют разные функциональные области. В рамках каждой области проектируемые компоненты должны быть сфокусированы на решении конкретной задачи и не должны включать код, связанный с другими функциональными областями. При проектировании бизнес-слоя придерживайтесь следующих рекомендаций:

- **Убедитесь в необходимости отдельного бизнес-слоя.** По возможности всегда создавайте отдельный бизнес-слой, это способствует повышению удобства обслуживания приложения. Исключением могут быть лишь приложения с небольшим числом или вообще без бизнес-правил (кроме валидации данных).
- **Определитесь с ответственностями и потребителями бизнес-слоя.** Это поможет принять решение о том, какие задачи должен выполнять бизнес-слой, и каким образом будет предоставляться доступ к нему. Используйте бизнес-слой для обработки сложных бизнес-правил, преобразования данных, применения политик и валидации. Если бизнес-слой будет использоваться и слоем представления, и внешними приложениями, можно предоставить бизнес-слой в виде сервиса.
- **Не смешивайте в бизнес-слое компоненты разных типов.** Используйте бизнес-слой как средство избежать смешения кода представления и доступа к данным с бизнес-логикой, чтобы отделить бизнес-логику от логики представления и доступа к данным и упростить тестирование бизнес-функциональности. Также используйте бизнес-слой, чтобы централизовать функции бизнес-логики и обеспечить возможность повторного использования.
- **Сократите количество сетевых вызовов при доступе к удаленному бизнес-слою.** Если бизнес-слой размещается на другом уровне, физически отдельно от других слоев и клиентов, с которыми должен взаимодействовать, рассмотрите возможность реализации удаленного управляемого сообщениями фасада приложения или слоя сервисов, который объединит мелкие операции в более крупные. Используйте большие пакеты для передачи данных по сети, такие как Объекты переноса данных (Data Transfer Objects, DTO).
- **Избегайте тесного связывания между слоями.** При создании интерфейса бизнес-слоя применяйте принципы абстракции для максимального ослабления связывания. К техникам абстракции относятся использование открытых объектных интерфейсов, общих описаний интерфейсов, абстрактных базовых классов или связи через обмен сообщениями. Для Веб-приложений создайте управляемый сообщениями интерфейс между слоем представления и бизнес-слоем. Более подробно эти вопросы рассматриваются в главе 5, «[Рекомендации по проектированию многослойных приложений](#)».

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Аутентификация](#)
 - [Авторизация](#)
 - [Кэширование](#)
 - [Связывание и связность](#)
 - [Управление исключениями](#)
 - [Протоколирование, аудит и инstrumentирование](#)
 - [Валидация](#)
-

Аутентификация

Проектирование эффективной стратегии аутентификации для бизнес-слоя имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак с подделкой пакетов, атак перебором по словарю, перехватом сеансов и других типов атак. При проектировании стратегии аутентификации руководствуйтесь следующими рекомендациями:

- Избегайте аутентификации в бизнес-слое, если она будет использоваться только слоем представления или слоем сервисов, располагающимися на том же уровне в пределах границы доверия. Передавайте удостоверениезывающего в бизнес-слой, только если требуется проводить аутентификацию или авторизацию на основании ID исходнойзывающей стороны.
- Если предполагается разделять бизнес-слой между многими приложениями, использующими разные хранилища пользователей, рассмотрите возможность применения механизма единой регистрации. Избегайте создания собственных механизмов аутентификации, по возможности пользуйтесь встроенными механизмами платформы.
- Если слой представления и бизнес-слой развернуты на одном компьютере и требуется выполнять доступ к ресурсам на основании разрешений Списка управления доступом (access control list, ACL) исходногозывающего, рассмотрите возможность использования олицетворения. Если слой представления и бизнес-слой развернуты на разных компьютерах и требуется выполнять доступ к ресурсам на основании разрешений ACL исходногозывающего, рассмотрите возможность использования делегирования. Однако делегирование обуславливает более интенсивное использование ресурсов и, кроме того, не поддерживается многими средствами, поэтому должно применяться только в случае крайней необходимости. Если требования безопасности позволяют, выполняйте аутентификацию

пользователя на границе и реализуйте обращения к нижним слоям с помощью доверенной подсистемы. В качестве альтернативы предлагается подход обеспечения безопасности на основании утверждений (особенно для приложений на базе сервисов), который использует преимущества механизмов интегрированной идентификации и обеспечивает возможность целевой системе проводить аутентификацию утверждений пользователя.

Авторизация

Проектирование эффективной стратегии авторизации для бизнес-слоя имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для разглашения данных, повреждения или подделки данных и несанкционированного получения прав доступа. При проектировании стратегии авторизации руководствуйтесь следующими рекомендациями:

- Защитите ресурсы, применяя авторизацию на основании удостоверений, учетных групп, ролей или других контекстных данных вызывающей стороны. Максимально сократите дробление на роли, чтобы уменьшить число необходимых сочетаний разрешений.
 - Применяйте авторизацию на основании ролей для бизнес-решений, авторизацию на базе ресурсов для аудита системы и авторизацию на основании утверждений, если требуется поддерживать интегрированную авторизацию по совокупности данных, таких как удостоверение, роль, разрешения, права и другие факторы.
 - По возможности избегайте применения олицетворения и делегирования, потому что это может существенно повлиять на производительность и возможности масштабирования. Как правило, олицетворение клиента при вызове требует больших ресурсов, чем вызов напрямую.
 - Не смешивайте код авторизации и код обработки бизнес-задач в одном компоненте.
 - Поскольку, как правило, авторизация охватывает все приложение, обеспечьте, чтобы инфраструктура авторизации не создавала существенные издержки производительности.
-

Кэширование

Проектирование соответствующей стратегии кэширования для бизнес-слоя важно с точки зрения производительности и скорости ответа приложения. Кэширование позволит оптимизировать поиск данных, сократить количество повторных обращений к сети и избежать ненужной повторной обработки. Как часть стратегии кэширования должно быть принято решение о том, когда и как будут загружаться кэшированные данные. Чтобы обеспечить отсутствие задержек на клиенте, кэш должен загружаться асинхронно или с помощью процесса пакетной передачи. При проектировании стратегии кэширования руководствуйтесь следующими рекомендациями:

- Кэшируйте статические данные, которые будут регулярно использоваться бизнес-слоем, но избегайте кэширования часто меняющихся данных. Кэшируйте данные, которые нельзя быстро и эффективно извлечь из базы данных, но избегайте кэширования очень больших объемов данных, что может привести к снижению производительности обработки. Кэшируйте только минимально необходимое количество данных.
- Рассмотрите возможность кэширования данных в бизнес-слой в готовом к использованию виде.
- По возможности избегайте кэширования конфиденциальных данных или обеспечьте механизм защиты таких данных в кэше.
- Рассмотрите, как развертывание Веб-фермы повлияет на дизайн механизмов кэширования для бизнес-слоя. Если запросы от одного клиента могут обрабатываться несколькими серверами фермы, решение кэширования должно поддерживать синхронизацию кэшированных данных.

Более подробно методики кэширования рассматриваются в главе 17, «[Сквозная функциональность](#)».

Связывание и связность

При проектировании компонентов бизнес-слоя, убедитесь в их высокой связности и реализуйте слабое связывание между слоями. Это поможет улучшить масштабируемость приложения. При проектировании связывания и связности руководствуйтесь следующими рекомендациями:

- Избегайте циклических зависимостей. Бизнес-слой должен знать только о слое, расположенному под ним (слое доступа к данным), и ничего не знать о слое над ним (слое представления или внешних приложениях, которые обращаются к бизнес-слою напрямую).
- Реализуйте слабо связанный интерфейс через абстракцию. Этого можно достичь за счет применения интерфейсных компонентов, общих описаний интерфейса или совместно используемой абстракции, когда конкретные компоненты зависят от абстракций, а не от других компонентов (принцип инверсии зависимостей). Этапы проектирования многослойной структуры подробно рассматриваются в главе 5, «[Рекомендации по проектированию многослойных приложений](#)».
- Если динамическое поведение не требует слабого связывания, обеспечивайте тесное связывание в рамках бизнес-слоя при проектировании.
- Обеспечивайте высокую связность при проектировании. Компонент должен включать только ту функциональность, которая касается именно данного компонента. Всегда избегайте смешения логики доступа к данным с бизнес-логикой в компонентах бизнес-слоя.

- Использование основанных на обмене сообщениями механизмов доступа к компонентам бизнес-слоя обеспечивает снижение связанности и возможность размещения их на других уровнях в случае необходимости.
-

Управление исключениями

Проектирование эффективного решения управления исключениями для бизнес-слоя имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак Отказа в обслуживании (Denial of Service, DoS) и может допустить разглашение конфиденциальных или критически важных данных о приложении. Формирование и обработка исключений является ресурсоемкой операцией, поэтому важно, чтобы стратегия управления исключениями разрабатывалась с учетом влияния на производительность. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Перехватывайте только те внутренние исключения, которые можете обработать, или если требуется добавить данные. Например, перехватывайте исключения, возникающие при попытках преобразования значений null. Не используйте исключения как средство управления бизнес-логикой или потоком выполнения приложения.
 - Правильно спроектируйте стратегию распространения исключений. Например, разрешите исключениям распространяться вверх к граничным слоям, где они могут быть запротоколированы и преобразованы соответствующим образом для передачи на следующий слой. Применяйте контекстные идентификаторы, это позволит находить взаимосвязанные исключения на разных слоях при проведении анализа основных причин ошибок и сбоев.
 - Убедитесь, что перехватываете исключения, которые не будут перехвачены где-либо в другом месте (например, глобальном обработчике ошибок), и очищайте ресурсы и состояние после возникновения исключения.
 - Правильно выберите стратегию протоколирования и уведомления о критических ошибках и исключениях, чтобы регистрировать достаточный объем сведений об исключениях и при этом не разглашать конфиденциальных данных.
-

Более подробно методики управления исключениями рассматриваются в главе 17, «[Сквозная функциональность](#)».

Протоколирование, аудит и инструментирование

Проектирование эффективных механизмов протоколирования, аудита и инструментирования для бизнес-слоя имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, пользователи смогут безнаказанно отказываться от своих действий. Также файлы журналов могут пригодиться для доказательства правонарушений в случае судебного разбирательства. Аудит считается наиболее достоверным, если данные журнала формируются непосредственно в момент доступа к ресурсу той же процедурой, которая выполняет этот доступ. Инструментирование можно

реализовать, используя счетчики производительности и события. Инструменты мониторинга системы используют этот инструментарий или другие точки доступа для обеспечения администраторов сведениями о состоянии, производительности и работоспособности приложения. При проектировании стратегии протоколирования и инstrumentирования руководствуйтесь следующими рекомендациями:

- Обеспечьте централизацию протоколирования, аудита и инструментирования для бизнес-слоя. Для реализации функциональности обработки исключений и протоколирования используйте библиотеку Enterprise Library от patterns & practices или решения сторонних производителей, такие как Сервисы протоколирования Apache (Apache Logging Services) «log4Net» или сервис «NLog» Ярослава Ковальского (Jarosław Kowalski).
- Включите в компоненты бизнес-слоя инструментарий для критических событий системы и бизнес-процесса.
- Не храните конфиденциальные бизнес-данные в файлах журнала.
- Обеспечьте, чтобы сбой протоколирования не оказывал влияния на нормальную функциональность бизнес-слоя.
- Рассмотрите возможность аудита и протоколирования всех эпизодов доступа к функциям бизнес-слоя.

Валидация

Проектирование эффективного механизма валидации для бизнес-слоя имеет большое значение с точки зрения обеспечения удобства и простоты использования и надежности приложения, в противном случае, оно может остаться открытым для несогласованности данных и нарушений бизнес-правил, а также обеспечивать неудовлетворительный уровень взаимодействия с пользователем. Кроме того, приложение может оказаться уязвимым к таким угрозам безопасности, как межсайтовые атаки внедрением сценариев (cross-site scripting, XSS), атаки типа внедрение SQL-кода, переполнение буфера и другие типы атак посредством входных данных. Четкого и исчерпывающего определения действительного или злонамеренного ввода нет. Возможные риски, обуславливаемые злонамеренным вводом, зависят также от того, как приложение использует ввод. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- Проверяйте все вводимые данные и параметры методов в бизнес-слое, даже если проверка ввода выполняется в слое представления.
- Обеспечьте централизованный подход к валидации, чтобы обеспечить наилучшие условия для тестирования и повторного использования.
- Ограничивайте, отклоняйте и очищайте пользовательский ввод. Иначе говоря, предполагайте, что весь пользовательский ввод является злонамеренным. Проводите проверку длины, диапазона, формата и типа вводимых данных.

Вопросы развертывания

При развертывании бизнес-слоя необходимо обратить внимание на вопросы производительности и безопасности в среде производственной эксплуатации. При развертывании бизнес-слоя руководствуйтесь следующими рекомендациями:

- Если не предъявляются особые требования по размещению бизнес-слоя на отдельном уровне в связи с вопросами масштабируемости и безопасности, развертывайте его на одном уровне со слоем представления. Это обеспечит максимальную производительность приложения.
- Если требуется поддерживать удаленный бизнес-слой, для улучшения производительности приложения используйте протокол TCP.
- Используйте набор протоколов Internet Protocol Security (IPSec)¹ для защиты данных, передаваемых между физическими уровнями.
- Используйте шифрование по протоколу Secure Sockets Layer (SSL)² для защиты вызовов удаленных Веб-сервисов компонентами бизнес-слоя.

Этапы проектирования бизнес-слоя

При проектировании бизнес-слоя также необходимо учитывать технические требования для основных составляющих этого слоя: компонентов бизнес-слоя, бизнес-сущностей и компонентов бизнес-процесса. В данном разделе кратко описываются основные действия по проектированию самого бизнес-слоя. Приводим рекомендуемый процесс проектирования бизнес-слоя:

1. **Создайте дизайн бизнес-слоя в первом приближении.** Определите, кто будет использовать бизнес-слой: слой представления, слой сервисов или другие приложения. Это определит тактику методов доступа к бизнес-слою. Далее определите требования безопасности для бизнес-слоя, требования и стратегию валидации. Руководствуйтесь рекомендациями раздела «[Специальные вопросы проектирования](#)» данной главы, это поможет учесть все факторы при создании дизайна в первом приближении.
2. **Спроектируйте компоненты бизнес-слоя.** При проектировании и реализации приложения могут использоваться несколько типов компонентов бизнес-слоя. Примерами таких компонентов являются компоненты бизнес-процесса, служебные компоненты и вспомогательные компоненты. К факторам, влияющим на выбор структуры компонентов бизнес-слоя, относятся разные аспекты дизайна приложения, требования к транзакциям и правила обработки. Более подробно эти вопросы рассматриваются в главе 12, «[Проектирование компонентов бизнес-слоя](#)».
3. **Спроектируйте компоненты бизнес-сущностей.** Бизнес-сущности используются для размещения и управления бизнес-данными, используемыми приложением. Бизнес-

¹ Безопасные Интернет-протоколы (прим. переводчика).

² Протокол безопасных соединений (прим. переводчика).

сущности должны обеспечивать проверку данных, размещаемых в сущности. Кроме того, бизнес-сущности обеспечивают свойства и операции для доступа и инициализации данных сущности. Более подробно эти вопросы рассматриваются в главе 13, «[Проектирование бизнес-сущностей](#)».

4. **Спроектируйте компоненты рабочего процесса.** Существует масса сценариев, в которых задачи должны выполняться в определенном порядке в зависимости от завершения определенных этапов или координируются человеком. Эти требования можно отобразить в основных сценариях рабочего процесса. Необходимо понимать, как требования и правила влияют на выбор при реализации компонентов рабочего процесса. Более подробно эти вопросы рассматриваются в главе 14, «[Проектирование компонентов рабочего процесса](#)».

Проектированию и использованию компонентов в приложениях посвящена глава 10, «[Рекомендации по проектированию компонентов](#)».

Шаблоны проектирования

Основные шаблоны проектирования для бизнес-слоя организованы по категориям и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны проектирования
Компоненты бизнес-слоя	Application Façade (Фасад приложения). Централизует и агрегирует поведение для обеспечения унифицированного слоя сервисов. Chain of Responsibility (Цепочка обязанностей). Предоставляя возможность обработать запрос нескольким объектам, устраняет возможность связывания отправителя запроса с его получателем. Command (Команда). Инкапсулирует обработку запроса в отдельный командный объект с общим интерфейсом выполнения.
Бизнес-сущности	Domain Model (Модель предметной области). Набор бизнес-объектов, представляющих сущности предметной области и отношения между ними. Entity Translator (Транслятор сущностей). Объект, преобразующий типы данных сообщения в бизнес-типы для запросов и выполняющий обратные преобразования для ответов. Table Module (Модуль таблицы). Единый компонент, реализующий бизнес-логику для всех строк таблицы или представления базы данных.
Рабочие процессы	Data-Driven Workflow (Управляемый данными рабочий процесс). Рабочий процесс, включающий задачи, последовательность выполнения которых определяется значениями данных в рабочем процессе или системе. Human Workflow (Рабочий процесс, управляемый оператором). Рабочий процесс, включающий задачи, выполняемые вручную. Sequential Workflow (Последовательный рабочий процесс). Рабочий процесс, включающий задачи, выполняющиеся в

	<p>определенной последовательности, когда выполнение одной задачи запускается только после завершения предыдущей.</p> <p>State-Driven Workflow (Управляемый состоянием рабочий процесс). Рабочий процесс, включающий задачи, последовательность выполнения которых определяется состоянием системы.</p>
--	--

Более подробно шаблон Façade рассматривается в главе 4, «Структурные шаблоны», книги Эрика Гамма, Ричарда Хелма, Ральфа Джонсона и Джона Влиссидеса «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2007.

Более подробно шаблон Chain of Responsibility рассматривается в статье «*Patterns in Practice*» по адресу <http://msdn.microsoft.com/en-us/magazine/cc546578.aspx>.

Шаблону Command посвящена глава 5, «Поведенческие шаблоны», книги Эрика Гамма, Ричарда Хелма, Ральфа Джонсона и Джона Влиссидеса «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2007.

Более подробно шаблон Entity Translator рассматривается в статье «*Useful Patterns for Services*» (Полезные шаблоны для сервисов) по адресу <http://msdn.microsoft.com/en-us/library/cc304800.aspx>.

Более подробно шаблоны Data-Driven Workflow, Human Workflow, Sequential Workflow и State-Driven Workflow рассматриваются в статье «*Windows Workflow Foundation Overview*» (Обзор Windows Workflow Foundation) по адресу <http://msdn.microsoft.com/en-us/library/ms734631.aspx> и «*Workflow Patterns*» (Шаблоны рабочего процесса) по адресу <http://www.workflowpatterns.com/>.

Предложения группы patterns & practices

Узнать о дополнительных предложениях группы Microsoft patterns & practices можно из следующих источников:

- «Enterprise Library» по адресу <http://msdn.microsoft.com/en-us/library/cc467894.aspx>.
 - «Unity» (механизм внедрения зависимостей) по адресу <http://msdn.microsoft.com/en-us/library/dd203101.aspx>.
-

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

Более подробно интеграция бизнес-слоев рассматривается в статье «*Integration Patterns*» (Шаблоны интеграции) по адресу <http://msdn.microsoft.com/en-us/library/ms978729.aspx>.

Сведения о сервисах протоколирования Apache (Apache Logging Services) «log4Net» можно найти по адресу <http://logging.apache.org/log4net/index.html>.

Сведения о сервисе «NLog» Ярослава Ковальского можно найти по адресу <http://www.nlog-project.org/introduction.html>.

8

Рекомендации по проектированию слоя доступа к данным

Обзор

В данной главе приводятся основные рекомендации по проектированию слоя доступа к данным приложения. Из нее вы узнаете, какое место слой доступа к данным занимает в типовой архитектуре многослойного приложения, какие компоненты обычно включает и с какими основными проблемами придется столкнуться при проектировании слоя доступа к данным. Здесь вы найдете советы по проектированию, рекомендуемые этапы проектирования, соответствующие шаблоны проектирования и технологии. На рис. 1 показано место слоя доступа к данным в общей архитектуре приложения.

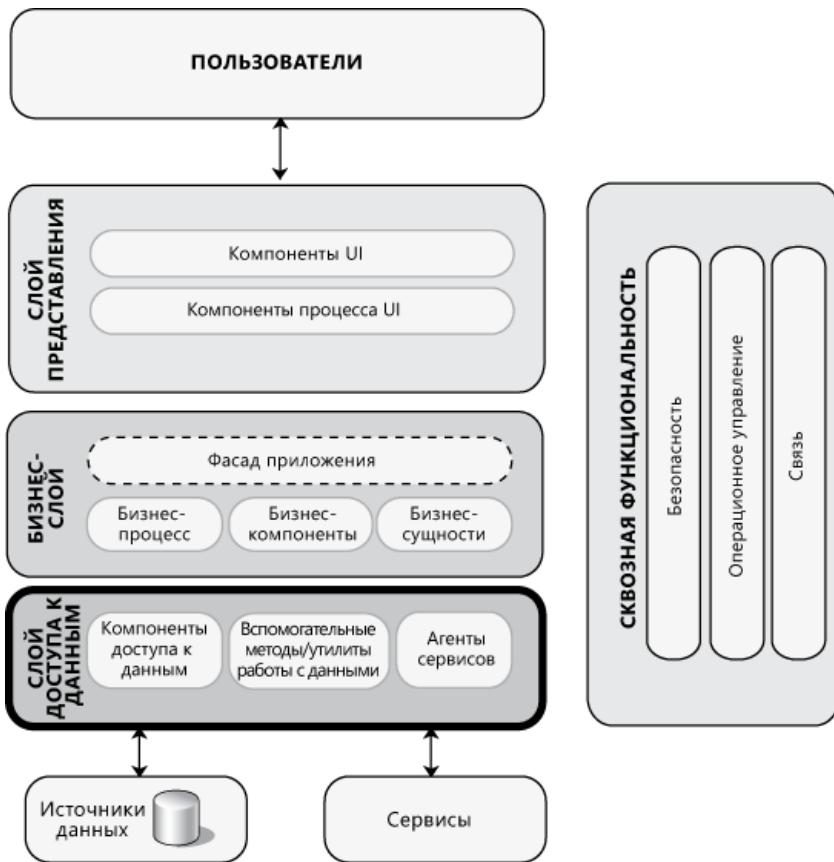


Рис. 9

Слой данных в типовом приложении и компоненты, которые он может включать

Слой доступа к данным может включать следующие компоненты:

- **Компоненты доступа к данным.** Эти компоненты абстрагируют логику, необходимую для доступа к базовым хранилищам данных. Они обеспечивают централизацию общей функциональности доступа к данным, что способствует упрощению настройки и обслуживания приложения. Некоторые инфраструктуры доступа к данным могут требовать, чтобы общая логика доступа к данным была определена и реализована в отдельных вспомогательных или служебных компонентах доступа к данным, пригодных для повторного использования. Другие инфраструктуры доступа к данным, включая многие инфраструктуры объектно-реляционного сопоставления (Object/Relational Mapping, O/RM), реализуют такие компоненты автоматически, сокращая объем кода доступа к данным, который должен написать разработчик.
- **Агенты сервисов.** Если компонент бизнес-слоя должен выполнять доступ к данным, предоставляемым внешним сервисом, может понадобиться реализовать код управления семантикой взаимодействия с этим конкретным сервисом. Агенты сервисов реализуют компоненты доступа к данным, которые изолируют меняющиеся требования вызова сервисов от приложения и могут обеспечивать дополнительные сервисы, такие как кэширование, поддержку работы в автономном режиме и базовое преобразование между форматом данных, предоставляемых сервисом, и форматом, поддерживаемым вашим приложением.

Более подробно компоненты, обычно используемые в слое доступа к данным, рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)». Вопросам проектирования компонентов доступа к данным посвящена глава 15, «[Проектирование компонентов слоя доступа к данным](#)».

Общие принципы проектирования

Слой доступа к данным должен отвечать требованиям приложения, работать эффективно и безопасно и обеспечивать простоту обслуживания и расширения в случае изменения бизнес-требований. При проектировании слоя доступа к данным руководствуйтесь следующими общими рекомендациями:

- **Правильно выберите технологию доступа к данным.** Выбор технологии доступа к данным зависит от типа данных, с которыми придется работать, и того, как предполагается обрабатывать данные в приложении. Для каждого сценария существуют наиболее подходящие технологии. Все технологии доступа к данным с перечислением преимуществ и соображений относительно применения в том или ином сценарии обсуждаются в конце данного руководства в приложении С, «[Матрица технологий доступа к данным](#)».
- **Используйте абстракцию для реализации слабо связанного интерфейса слоя доступа к данным.** Этот подход можно реализовать путем определения интерфейсных компонентов, таких как шлюз с общеизвестными входными и выходными параметрами, который преобразует запросы в формат, понятный компонентам слоя. Кроме того, с помощью интерфейсных типов или абстрактных базовых классов можно определить совместно используемую абстракцию, которая должна быть реализована интерфейсными компонентами. Абстракция слоя более подробно рассматривается в главе 5, «[Рекомендации по проектированию многослойных приложений](#)».
- **Инкапсулируйте функциональность доступа к хранилищу данных в слое доступа к данным.** Слой доступа к данным должен скрывать детали доступа к источнику данных. Он должен обеспечивать управление подключениями, формирование запросов и сопоставление сущностей приложения со структурами источника данных. Потребители слоя доступа к данным взаимодействуют с ним через абстрактные интерфейсы с использованием сущностей приложения, таких как объекты предметной области, **типовизированные наборы данных (Typed DataSet)** и XML, и не должны знать внутренних деталей слоя доступа к данным. Такое разделение функциональных областей упрощает разработку и обслуживание приложения.
- **Примите решение о том, как будет выполняться сопоставление сущностей приложения со структурами источника данных.** Тип сущности, используемой в приложении, является основным фактором при принятии решения о методе сопоставления этих сущностей со структурами источника данных. Обычно для этого используются шаблоны Domain Model или Table Module либо механизмы Объектно-реляционного сопоставления (Object/Relational Mapping, O/RM), однако, бизнес-

сущности могут быть реализованы с использованием разнообразных подходов²². Необходимо определить стратегию заполнения бизнес-сущностей или структур данных из источника данных и их предоставления для доступа с бизнес-слоя или слоя представления приложения. Шаблоны Domain Model и Table Module более подробно рассматриваются в разделе «[Шаблоны проектирования](#)» ближе к окончанию данной главы. Больше сведений о бизнес-сущностях и форматах данных можно найти в главе 13, «[Проектирование бизнес-сущностей](#)».

- **Рассмотрите возможность объединения структур данных.** При предоставлении данных через сервисы воспользуйтесь Объектами передачи данных (Data Transfer Objects, DTO), они помогут организовать данные в унифицированные структуры. Кроме того, объекты DTO позволяют реализовать слабо детализированные операции, обеспечивая при этом структуру для перемещения данных через границы. Объекты DTO также могут объединять бизнес-сущности при выполнении агрегированных операций. При использовании шаблона Table Data Gateway или Active Record данные могут быть представлены с помощью класса **DataTable** (Таблица данных).
- **Примите решение о том, как будет реализовано управление подключениями.** Обычно слой доступа к данным должен создавать и управлять подключениями ко всем источникам данных, используемым приложением. Необходимо выбрать метод хранения и защиты данных подключения, соответствующий требованиям безопасности предприятия. Возможными вариантами могут быть шифрование разделов конфигурационного файла или сохранение конфигурационных данных исключительно на сервере. Подробнее эти вопросы рассматриваются в главе 15, «[Проектирование компонентов слоя доступа к данным](#)».
- **Определите, как будут обрабатываться исключения, возникающие при обработке данных.** Слой доступа к данным должен перехватывать и обрабатывать (по крайней мере, обеспечивать начальный этап) все исключения, связанные с источниками данных и операциями CRUD (Create, Read, Update и Delete). Исключения, касающиеся самих данных и доступа к источникам данных, и ошибки истечения времени ожидания должны обрабатываться в этом слое и передаваться в другие слои, только если сбои оказывают влияние на скорость ответа и функциональность приложения.
- **Учтите риски безопасности.** Слой доступа к данным должен обеспечивать защиту от попыток похищения или повреждения данных и защищать механизмы, используемые для получения доступа к источнику данных. Например, очищать детализированную информацию об ошибках или исключениях, чтобы не было возможности разглашения данных из источника данных, и использовать менее привилегированные учетные записи, обладающими только необходимыми для осуществления операций приложения правами. Даже если сам источник данных обладает возможностью ограничивать привилегии, защита должна быть

²² В том числе и несовместимых с описанными выше шаблонами сопоставления (прим. научного редактора).

реализована и в слое доступа к данным, и в источнике данных. Доступ к базе данных должен осуществляться через параметризованные запросы, чтобы предотвратить возможность успеха атак с внедрением SQL-кода. Никогда не применяйте конкатенацию строк для построения динамических запросов на основе вводимых пользователем данных.

- **Сократите количество сетевых вызовов и обращений к базе данных.** Рассмотрите возможность группировки команд в одну операцию базы данных.
 - **Учтите требования производительности и масштабируемости.** Для слоя доступа к данным требования масштабируемости и производительности должны учитываться во время проектирования. Например, для приложения электронной коммерции именно производительность слоя доступа к данным, скорее всего, будет «узким местом» приложения. Если производительность слоя доступа к данным критична, используйте инструменты профилирования, чтобы понять и затем сократить количество или разбить ресурсоемкие операции с данными.
-

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Пакетная обработка](#)
 - [Большие бинарные объекты](#)
 - [Подключения](#)
 - [Формат данных](#)
 - [Управление исключениями](#)
 - [Объектно-реляционное сопоставление](#)
 - [Запросы](#)
 - [Хранимые процедуры](#)
 - [Сравнение хранимых процедур и динамического SQL](#)
 - [Транзакции](#)
 - [Валидация](#)
 - [XML](#)
-

Пакетная обработка

Пакетная обработка команд базы данных может обеспечить повышение производительности слоя данных. Каждый запрос к среде выполнения базы данных порождает издержки.

Пакетирование может сократить общие издержки за счет увеличения пропускной способности и уменьшения задержки. Пакетирование схожих запросов может повысить производительность, поскольку это позволяет базе данных выполнять кэширование и повторно использовать план выполнения для аналогичного запроса. При проектировании пакетной обработки руководствуйтесь следующими рекомендациями:

- Рассмотрите возможность пакетирования команд для сокращения количества обращений к базе данных и минимизации трафика. Однако наибольший положительный эффект будет иметь пакетирование схожих запросов. Объединение разных или случайных запросов не обеспечивает значительного уменьшения издержек.
- Используйте пакетные команды и **DataReader** (Модуль чтения данных) для загрузки и копирования множества наборов данных. Но при загрузке больших объемов файлов данных в базу данных рекомендуется применять утилиты базы данных для массового копирования.
- Не используйте транзакции для длительных пакетных команд, это может заблокировать ресурсы базы данных.

Большие бинарные объекты

Если данные хранятся и извлекаются как единый поток, их можно рассматривать как большой бинарный объект или BLOB (Binary Large Object). BLOB может иметь внутреннюю структуру, но эта структура скрыта от базы данных, в которой он хранится, или слоя доступа к данным, который выполняет чтение и запись этого объекта. BLOB-данные, как правило, хранятся непосредственно в базе данных или в файловой системе. BLOB-объекты обычно используются для хранения изображений, но также могут использоваться для хранения бинарного представления объектов. При проектировании BLOB-объектов руководствуйтесь следующими рекомендациями:

- Определитесь в необходимости хранения BLOB-данных в базе данных. Современные базы данных обеспечивают намного лучшую обработку BLOB-данных, предоставляя возможность выбора соответствующего типа данных столбца, а также удобство обслуживания и работы, контроль версий и хранение соответствующих метаданных. Тем не менее, подумайте, возможно, практичнее будет хранить BLOB на диске, а в базе данных размещать только ссылку на эти данные.
- Рассмотрите возможность применения BLOB-объектов для упрощения синхронизации больших бинарных объектов между серверами.
- Если необходимо обеспечить возможность поиска в BLOB-данных, предусмотрите в базе данных дополнительные поля для поиска, чтобы не приходилось проводить синтаксический анализ данных в BLOB-полях.
- При извлечении приводите BLOB к соответствующему типу для работы в бизнес-слое или слое представления.

Подключения

Подключения к источникам данных являются фундаментальной частью слоя доступа к данным. Все подключения к источнику данных должны управляться слоем доступа к данным. Создание и управление подключениями занимает ценные ресурсы, как слоя доступа к данным, так и источника данных. Чтобы обеспечить максимальную производительность и безопасность, при проектировании подключений слоя доступа к данным руководствуйтесь следующими рекомендациями:

- Открывайте подключения как можно позже и закрывайте их как можно раньше. Никогда не оставляйте подключения открытыми без надобности.
- По возможности осуществляйте транзакции через одно подключение.
- Применение модели доверенной подсистемы безопасности позволит воспользоваться преимуществами пула подключений; по возможности избегайте олицетворения или использования личных удостоверений.
- В целях безопасности не храните данные подключения в системных или пользовательских Data Source Name (DSN)²³.
- В случае необходимости, предусмотрите логику повторного подключения для обработки ситуаций, когда подключение к источнику данных потеряно либо закрыто по истечении времени ожидания. Однако в случае возникновения некоторых проблем, таких как конкуренция за ресурсы, повторные попытки выполнения операции могут усугублять их, негативно сказываясь на масштабировании. Более подробно эти вопросы рассматриваются в главе 15, «[Проектирование компонентов слоя доступа к данным](#)».

Формат данных

Правильный выбор формата данных обеспечивает возможность взаимодействия с другими приложениями и способствует обмену сериализованными данными между разными процессами или компьютерами. Формат данных и сериализация также важны для хранения и извлечения состояния приложения бизнес-слоем. При выборе формата данных руководствуйтесь следующими рекомендациями:

- Используйте XML для обеспечения возможности взаимодействия с другими системами и платформами или при работе со структурами данных, которые могут меняться со временем.
- Используйте объекты **DataSet** для сценариев без постоянного подключения в простых приложениях, выполняющих операции CRUD.
- Если требуется передавать данные через физические границы, обратите внимание на требования сериализации и возможности взаимодействия. Например, продумайте, как будет выполняться сериализация бизнес-объектов, как они будут

²³ Имя источника данных (прим. переводчика).

транслироваться в объекты передачи данных (Data Transfer Objects, DTOs), если есть такая необходимость, и какие форматы может принимать целевой слой.

Более подробно форматы данных рассматриваются в главе 15, «[Проектирование компонентов слоя доступа к данным](#)». Проектированию и использованию компонентов в приложении посвящена глава 10, «[Рекомендации по проектированию компонентов](#)».

Управление исключениями

Проектируйте стратегию централизованного управления исключениями, чтобы обеспечить единообразие при перехвате и формировании исключений в слое доступа к данным. По возможности концентрируйте логику обработки исключений в компонентах приложения, реализующих сквозную функциональность. Особое внимание уделяйте исключениям, распространяющимся через границы доверия и на другие слои и уровни. Учитывайте при проектировании и необрабатываемые исключения, чтобы они не приводили к снижению надежности приложения или разглашению конфиденциальных данных приложения. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Выявите исключения, которые должны перехватываться и обрабатываться слоем доступа к данным. Например, обычно проблемы взаимоблокировки и подключений могут быть решены в слое доступа к данным. Тем не менее, некоторые исключения, такие как нарушения параллелизма, должны быть представлены пользователю для разрешения возникшей ситуации.
- Выберите соответствующую стратегию распространения исключений. Например, разрешите исключениям распространяться к граничным слоям, где они могут быть зарегистрированы и в случае необходимости преобразованы перед передачей в следующий слой. Применяйте контекстные идентификаторы, это позволит находить взаимосвязанные исключения в разных слоях при проведении анализа основных причин ошибок и сбоев.
- Где это возможно, реализуйте процесс повторного подключения для операций, в которых могут возникать ошибки источника данных или ошибки в результате истечения времени ожидания.
- Убедитесь, что перехватываете исключения, которые не будут перехвачены где-либо в другом месте (например, глобальном обработчике ошибок), и очищайте ресурсы и состояние после возникновения исключения.
- Правильно выберите стратегию протоколирования и уведомления для критических ошибок и исключений, чтобы регистрировать достаточный объем сведений об исключениях и не разглашать конфиденциальных данных.

Объектно-реляционное сопоставление

При проектировании объектно-ориентированного (ОО) приложения учтите несоответствия модели ОО и реляционной модели и факторы, которые могут усложнить задачу по передаче данных между ними. Например, инкапсуляция в ОО-дизайнах, где все поля скрыты, может

противоречить открытой природе свойств в базе данных. К другим примерам несоответствия относятся отличия в применяемых типах данных, структуре, транзакциях и механизмах обработки данных. Два основных средства решения этой проблемы несоответствия – шаблоны проектирования для доступа к данным, такие как Repository, и инструменты Объектно-реляционного сопоставления (Object/Relational Mapping, O/RM). Часто наилучшим подходом является Проектирование на основе предметной области (Domain Driven Design), которое основывается на моделировании сущностей, соответствующих объектам предметной области. О Проектировании на основе предметной области рассказывается в главе 3, «[Архитектурные схемы и стили](#)», и главе 13, «[Проектирование бизнес-сущностей](#)».

При проектировании объектно-реляционного сопоставления руководствуйтесь следующими рекомендациями:

- Используйте инфраструктуру, обеспечивающую механизмы Объектно-реляционного сопоставления (O/RM) между сущностями предметной области и базой данных. В настоящее время предлагаются O/RM решения, которые значительно упрощают задачу разработчика, избавляя от необходимости реализации большого объема собственного кода.
- При работе над абсолютно новым приложением либо в ситуациях, когда обеспечивается полный контроль над схемой базы данных²⁴, инструмент O/RM может применяться для формирования схемы, поддерживающей заданную объектную модель, и для обеспечения сопоставления сущностей базы данных и предметной области.
- В ситуациях, когда приходится работать с существующей схемой базы данных²⁵, инструмент O/RM может применяться для сопоставления модели предметной области и существующей реляционной модели.
- Если вы работаете с небольшим приложением или не имеете доступа к инструментам O/RM, реализуйте шаблон доступа к данным, такой как Repository. В случае применения шаблона Repository объекты хранилища позволяют работать с сущностями предметной области так, как будто они располагаются в памяти.
- При работе с Веб-приложениями или сервисами группируйте сущности и поддерживайте механизмы, которые обеспечат загрузку сущностей предметной области с включением в них только необходимых данных. Такой процесс обычно называют отложенной загрузкой. Это позволяет приложениям обеспечивать более высокую пропускную способность, необходимую для поддержки операций без сохранения состояния, и ограничить использование ресурсов благодаря отсутствию необходимости удерживать в памяти инициализированные модели предметной области для каждого пользователя.

²⁴ Так называемая greenfield среда (прим. научного редактора).

²⁵ Так называемая brownfield среда (прим. научного редактора).

Запросы

Запросы – это основные операции манипуляции данными в слое доступа к данным. Они являются механизмом преобразования запросов приложения в CRUD-действия базы данных. Поскольку запросы имеют такое большое значение, они должны быть оптимизированы для обеспечения максимальной производительности и пропускной способности базы данных. При использовании запросов в слое доступа к данным руководствуйтесь следующими рекомендациями:

- Используйте параметризованные SQL-запросы и типизированные параметры, чтобы повысить безопасность и снизить шансы успеха атак с внедрением SQL-кода. Не применяйте конкатенацию строк для построения динамических запросов на базе вводимых пользователем данных
- Рассмотрите возможность использования объектов для построения запросов. Например, реализуйте шаблон Query Object (Объект запроса) или используйте поддержку параметризованных запросов, предоставляемую ADO.NET. Также оптимизируйте схему базы данных для выполнения запросов.
- При построении динамических SQL-запросов избегайте смешения бизнес-логики с логикой, используемой для формирования SQL-выражений, поскольку это может сильно усложнить обслуживание и отладку кода.

Хранимые процедуры

В прошлом хранимые процедуры обеспечивали лучшую производительность по сравнению с динамическими SQL-выражениями²⁶. Однако сегодня современные ядра СУБД практически уравняли производительность обработки хранимых процедур и динамических SQL-выражений (использующих параметризованные запросы). Основными факторами при принятии решения об использовании хранимых процедур являются абстракция, удобство обслуживания и среда выполнения. В данном разделе приводятся рекомендации по проектированию приложения с использованием хранимых процедур. В следующем разделе даются рекомендации по выбору хранимых процедур или динамических SQL-выражений.

С точки зрения безопасности и производительности основными рекомендациями для хранимых процедур является использовать типизированные параметры и избегать динамического SQL в хранимых процедурах. Параметры – один из факторов, определяющих использование кэшированных планов запросов, а не их повторное создание. При изменении типов параметров и большого числа параметров формируются новые планы выполнения запросов, что может снизить производительность. При проектировании хранимых процедур руководствуйтесь следующими рекомендациями:

- Используйте типизированные параметры как входные значения процедуры и выходные параметры для возвращения одиночных значений. Рассмотрите возможность применения XML-параметров или табличных параметров для

²⁶ Здесь и далее под динамическим SQL понимается, произвольный SQL запрос, приходящий из приложения (прим. научного редактора).

передачи списков данных или табличных данных. В хранимых процедурах не форматируйте данные для отображения; обеспечьте возвращение соответствующих типов и выполняйте форматирование в слое представления.

- Применяйте параметры или переменные базы данных, если требуется формировать динамический SQL в хранимой процедуре. Однако помните, что использование динамического SQL в хранимых процедурах может негативно сказываться на производительности, безопасности и удобстве обслуживания.
- Избегайте создания временных таблиц при обработке данных. Тем не менее, если временные таблицы должны использоваться, создавайте их в памяти, а не на жестком диске.
- Реализуйте соответствующие стратегии обработки ошибок и возвращайте ошибки, которые код приложения может обработать.

Сравнение хранимых процедур и динамического SQL

Выбор между хранимыми процедурами и динамическим SQL заключается, главным образом, в принятии решения об использовании SQL-выражений, динамически генерируемых в коде, либо SQL, реализованного в хранимой в базе данных процедуре. При выборе между хранимыми процедурами и динамическим SQL необходимо учесть требования абстракции, удобства обслуживания и ограничения среды. Кроме того, во многих случаях немаловажную роль при выборе играют предпочтения или квалификация разработчика.

Основное преимущество хранимых процедур в том, что они обеспечивают уровень абстракции для базы данных, а это минимизирует зависимость кода приложения от изменений схемы базы данных. Также упрощается реализация и управление безопасностью, поскольку можно ограничить доступ ко всему, кроме хранимой процедуры, и использовать механизмы безопасности, обеспечивающие детализированную защиту и поддерживаемые большинством баз данных (хотя не забывайте, что это может помешать использовать преимущества пула подключений).

Основное преимущество динамических SQL-выражений в том, что зачастую они считаются более гибкими, чем хранимые процедуры, и могут обеспечить более быструю обработку. Многие инфраструктуры O/RM самостоятельно генерируют динамические запросы, существенно сокращая объем кода, который должен быть написан разработчиками.

Выбирая между хранимыми процедурами и динамическим SQL, руководствуйтесь следующими рекомендациями:

- Для небольшого приложения с единственным клиентом и несколькими бизнес-правилами динамический SQL часто является лучшим выбором.
- Для большого приложения с множеством клиентов продумайте, как обеспечить необходимую абстракцию. Примите решение, где эта абстракция должна находиться: в базе данных в форме хранимых процедур или в слое доступа к данным приложения в форме шаблонов доступа к данным или продуктов O/RM.

- Хранимые процедуры позволяют осуществлять операции с использованием большого количества данных ближе к данным, что может улучшить производительность.
- Использование хранимых процедур для доступа к базе данных позволит максимально сократить зависимость кода приложения от изменений схемы базы данных. Это поможет изолировать и максимально сократить количество изменений в коде приложения при нормализации или оптимизации схемы. Изменения во входных и выходных параметрах хранимой процедуры могут влиять на код приложения, но часто эти изменения можно изолировать в отдельных компонентах, выполняющих доступ к хранимой процедуре. Изолировать и максимально сократить изменения в коде приложения при обновлении схемы помогут и инфраструктуры О/RM.
- Принимая решение об использовании динамических SQL-запросов необходимо понимать, какое влияние будут оказывать изменения в схемах базы данных на приложение. Поэтому следует реализовывать слой доступа к данным таким образом, чтобы он обеспечивал отделение бизнес-компонентов от выполнения запросов базы данных. Такое отделение помогут реализовать такие шаблоны как Query Object и Repository. О/RM-инструменты позволяют полностью отделить бизнес-компоненты от выполнения запросов к базе данных.
- Учтите квалификацию группы, которая будет заниматься разработкой приложения. Если ваши разработчики не знакомы с программированием баз данных, выбирайте инструменты и шаблоны, более привычные вашим разработчиком.
- Подумайте о поддержке отладки. Разработчикам приложения будет проще проводить отладку динамического SQL.

Транзакции

Транзакция – это обмен последовательными данными и связанными с ними действиями, которые рассматриваются как единое целое, с целью выполнить запрос и гарантировать целостность базы данных. Транзакция считается завершенной, только если обработка всех данных и все действия завершены, тогда изменения соответствующей базы данных становятся постоянными. Транзакции поддерживают отмену (откат) действий в случае возникновения ошибки, что помогает сохранить целостность данных базы данных.

Важно правильно выбрать модель параллельной обработки запросов и определить, как будет осуществляться управление транзакциями. Существуют модели параллельной обработки запросов с пессимистической и оптимистической блокировкой. При *оптимистической блокировке* данные не блокируются, и обновления требуют кода для проверки. Обычно проверяется, не изменились ли данные с момента последнего извлечения, для этого используются временные метки. При *пессимистической блокировке* данные блокируются и не могут обновляться другими операциями до снятия блокировки.

При проектировании транзакций руководствуйтесь следующими рекомендациями:

- Используйте транзакции только в случае необходимости. Тщательно продумайте границы транзакции, чтобы обеспечить возможность повторных попыток и композиции. Возможно, для простых запросов явная транзакция не нужна, но вы должны убедиться, что не нарушаете стандартного поведения завершения и изоляции транзакции для базы данных. По умолчанию база данных Microsoft SQL Server® выполняет каждое SQL-выражение как отдельную транзакцию (режим автоматического завершения транзакции).
- Транзакции должны быть предельно короткими, чтобы максимально сократить время блокировки. Страйтесь избегать использования блокировки для продолжительных транзакций или при доступе к совместно используемым данным, что может блокировать доступ к этим данным другого кода. Не используйте блокировки взаимоисключающего доступа, что может приводить к конфликтам и взаимным блокировкам.
- Используйте соответствующий уровень изоляции, который определяет, как и когда изменения становятся доступными другим операциям. Необходимо найти компромисс между обеспечением непротиворечивости данных и частотой конфликтов. Высокий уровень изоляции обеспечит большую непротиворечивость данных, но негативно скажется на параллельной обработке в целом. Более низкий уровень изоляции обеспечит лучшую производительность, снизив количество конфликтов, но и более низкую согласованность данных.
- При использовании классов пространства имен **System.Transactions** рекомендуется применять модель неявных транзакций, обеспечиваемую объектом **TransactionScope** (Область действия транзакции) пространства имен **System.Transactions**. Неявные транзакции выполняются не так быстро, как транзакции, созданные вручную, или явные, но их проще реализовывать, и они обеспечивают гибкие и простые в обслуживании решения промежуточного слоя. Создаваемые вручную транзакции лучше реализовывать в хранимой процедуре.
- Если нет возможности завершить или откатить транзакцию или при использовании длительных транзакций, реализуйте компенсационные методы для возвращения хранилища данных в предыдущее состояние в случае сбоя операции в рамках транзакции.
- Если требуется выполнять множество запросов к базе данных, рассмотрите возможность применения множества активных результирующих наборов (multiple active result sets, MARS), что обеспечит поддержку множества результирующих наборов только для пересылки и только для чтения и позволит выполнять множество запросов с использованием одного подключения. MARS могут быть полезны в приложениях с параллельной обработкой множества транзакций.

Валидация

Создание эффективного решения валидации пользовательского ввода и данных имеет решающее значение для безопасности приложения. Определите правила валидации данных, поступающих с других слоев и от компонентов сторонних производителей, а также из базы

данных или хранилища данных. Поймите свои границы доверия и проверяйте все данные, пересекающие их. При проектировании стратегии проверки руководствуйтесь следующими рекомендациями:

- Проверяйте все данные, получаемые слоем доступа к данным от всех вызывающих сторон. Гарантируйте правильную обработку значений NULL и фильтрацию недействительных символов.
- При проектировании механизмов валидации учитывайте назначение данных. Например, пользовательский ввод, используемый для создания динамического SQL, должен проверяться на наличие символов или шаблонов, встречающихся в атаках внедрением SQL-кода.
- Возвращайте информативные сообщения об ошибках в случае сбоя валидации.

Более подробно методики проверки рассматриваются в главе 17, «[Сквозная функциональность](#)».

XML

Расширяемый язык разметки (Extensible Markup Language, XML) обеспечивает возможность взаимодействия и обслуживания структур данных вне базы данных. По соображениям производительности будьте осторожны с применением XML для очень больших объемов данных. Если требуется обрабатывать большие объемы данных в виде XML, применяйте не схемы на базе элементов, в которых значения данных хранятся как значения элементов, а схемы на базе атрибутов: в них значения данных хранятся в виде атрибутов, следовательно, такие схемы меньшего размера. При проектировании использования XML руководствуйтесь следующими рекомендациями:

- Для доступа к форматированным XML-данным используйте легковесные методы чтения и записи XML²⁷, особенно для очень больших наборов XML-данных. Если требуется взаимодействовать с реляционной базой данных, используйте объекты, которые поддерживают эту функциональность, такие как **DataSet** ADO.NET. При чтении и записи XML используйте общие настройки для обработки пробелов и комментариев.
- Рассмотрите возможность применения схемы XML для описания форматов и обеспечения проверки данных, хранящихся и передаваемых как XML. Используйте расширенные средства проверки для комплексных параметров данных схемы XML. Но не забывайте, что проверка приведет к снижению производительности.
- Храните XML в типизированных столбцах базы данных, если таковые имеются. Это обеспечит максимальную производительность. Если предполагается, что XML-данные будут регулярно запрашиваться, задайте индексы (если используемая база данных их поддерживает).

²⁷ То есть, не используйте XML DOM без крайней необходимости (прим. научного редактора).

Вопросы выбора технологий

Следующие рекомендации помогут правильно выбрать технологию и методики для реализации проектируемого приложения на основании его типа и предъявляемых к нему требований:

- Если требуется обеспечить поддержку запросов и параметров, используйте объекты ADO.NET напрямую.
- Если требуется поддерживать более сложные сценарии доступа к данным или упростить код доступа к данным, используйте Data Access Application Block (Блок доступа к данным) библиотеки Enterprise Library. Более подробно Enterprise Library рассматривается в приложении F, «[Enterprise Library от patterns & practices](#)».
- При создании управляемого данными Веб-приложения, страницы которого используют модель данных базовой базы данных, используйте технологию ASP.NET Dynamic Data (Динамические данные ASP.NET).
- Если требуется работать с форматированными XML-данными, используйте классы пространства имен **System.Xml** и его дочерних пространств имен или Linq to XML (X.Linq).
- При создании пользовательского интерфейса с помощью ASP.NET рассмотрите возможность применения **DataReader** для доступа к данным, чтобы максимально увеличить производительность формирования визуального представления. Объекты **DataReader** идеально подходят для ситуаций, где необходимо только чтения и только движение вперед, в которых каждая строка обрабатывается быстро.
- Для реализации доступа к SQL Server используйте классы пространства имен **SqlClient** ADO.NET, они обеспечивают максимальную производительность.
- Для реализации доступа к SQL Server 2008 используйте FILESTREAM, что обеспечит большую гибкость для хранения и доступа к BLOB-данным.
- При проектировании объектно-ориентированного бизнес-слоя на основании шаблона Domain Model используйте O/RM-средства, такие как ADO.NET Entity Framework или инструментарий с открытым исходным кодом NHibernate (для получения более развернутой информации обратитесь в раздел [Дополнительные источники](#) в конце данной главы).

Руководство по выбору технологии доступа к данным приведено в главе 15, «[Проектирование компонентов слоя доступа к данным](#)». Технологии, доступные на платформе Microsoft, представлены в приложении С, «[Матрица технологий доступа к данным](#)».

Вопросы производительности

Производительность – это функция как дизайна слоя доступа к данным, так и дизайна базы данных. Для обеспечения максимальной пропускной способности при настройке системы

рассматривайте их вместе. При проектировании производительности руководствуйтесь следующими рекомендациями:

- Используйте пул подключений и оптимизируйте производительность, исходя из результатов, полученных при моделировании нагрузочных сценариев.
- Рассмотрите возможность настройки уровней изоляции для запросов к данным. Для приложения с высокими требованиями по пропускной способности специальные операции с данными могут выполняться на более низких уровнях изоляции, чем вся остальная транзакция. Комбинирование различных уровней изоляции может иметь негативное влияние на непротиворечивость данных, поэтому необходимо тщательно анализировать этот аспект для каждого отдельного случая.
- Выполняйте пакетирование команд, чтобы сократить количество обращений к серверу базы данных.
- Для нечасто меняющихся данных используйте оптимистическую блокировку. Это позволит сократить издержки на блокировку строк базы данных, в том числе и на подключение, которое должно оставаться открытым в течение всей блокировки.
- В случае применения **DataReader** используйте последовательный поиск, это обеспечит большую производительность.

Вопросы безопасности

Слой доступа к данным должен обеспечивать защиту базы данных от попыток похищения или повреждения данных. Для различных частей источника данных должен предоставляться только необходимый уровень доступа. Также слой доступа к данным должен защищать механизмы, используемые для получения доступа к источнику данных. При проектировании системы безопасности руководствуйтесь следующими рекомендациями:

- При работе с SQL Server используйте аутентификацию Windows с реализацией модели доверенной подсистемы безопасности. Подробно модель доверенной подсистемы безопасности рассматривается в главе 19, «[Физические уровни и развертывание](#)».
- Шифруйте строки подключения в конфигурационных файлах, вместо того чтобы использовать системные или пользовательские DSN.
- Для хранения паролей применяйте нешифрованную версию пароля, а хеш с шумом.
- Требуйте, чтобы вызывающая сторона передавала данные удостоверения в слой доступа к данным для аудита.
- Используйте параметризованные SQL-запросы и типизированные параметры, чтобы снизить риски безопасности и сократить шансы на успех атак с внедрением SQL-кода. Не применяйте конкатенацию строк для построения динамических запросов на базе вводимых пользователем данных

Вопросы развертывания

При проектировании развертывания слоя доступа к данным архитектор ПО должен учесть производительность и вопросы безопасности среды производственной эксплуатации. При развертывании слоя доступа к данным руководствуйтесь следующими рекомендациями:

- Если это не противоречит требованиям масштабирования и безопасности, размещайте слой доступа к данным на одном уровне с бизнес-слоем, это обеспечит лучшую производительность приложения.
- Если требуется поддерживать удаленный уровень доступа к данным, улучшить производительность позволит применение протокола TCP.
- Рассмотрите возможность размещения слоя доступа к данным на другом сервере, отличном от сервера базы данных. Физические характеристики сервера базы данных обычно оптимизированы для этой роли и редко соответствуют параметрам, являющимся оптимальными для слоя доступа к данным, поэтому размещение этих двух слоев физически на одном уровне, практически, безусловно приведет к снижению производительности приложения.

Этапы проектирования слоя доступа к данным

Правильный подход к проектированию слоя доступа к данным обеспечит сокращение времени на разработку и большее удобство обслуживания слоя доступа к данным после развертывания приложения. В этом разделе кратко представлен эффективный подход к проектированию слоя доступа к данным. Рассмотрим основные этапы проектирования слоя доступа к данным:

1. **Создайте общий дизайн слоя доступа к данным.** Выберите, с какой средой будете работать, greenfield или brownfield, и, исходя из этого, определите ограничения источника данных и соответствующие ограничения слоя данных. Кроме того, в случае необходимости доработки, проанализируйте, как новый код будет сосуществовать с источником данных в его текущем состоянии.
 - В сценарии с использованием среды **greenfield** вы имеете полный контроль над схемой, используемой источником данных. Ограничения обуславливаются самим источником данных.
 - В сценарии с использованием среды **brownfield** вы не обладаете контролем над схемами источника данных, и в его качестве может выступать все, что угодно, начиная от базы данных, до компонентов шлюза, используемых для взаимодействия с существующими компонентами. Необходимо понимать структуру и ограничения существующего бизнеса. Например, выясните, имеется ли заданное хранилище операционных данных или другие ограничения, которые помешают изменить существующую схему. Но, как правило, есть возможность добавлять новые таблицы или представления в существующую схему. Также определитесь, как будет осуществляться

взаимодействие со слоем данных: через Веб-сервисы или с помощью устаревшего приложения, использующего компоненты шлюза. В любом случае взаимодействие будет ограничено операциями, определенными в контракте Веб-сервиса или в интерфейсе, предоставляемом компонентами шлюза.

2. **Выберите необходимые типы сущностей.** Компоненты доступа к данным работают с сущностями. Сущности используются для размещения и управления данными, используемыми приложением, и в них желательно включить весь необходимый код валидации. Также важно правильно выбрать тип и формат данных бизнес-сущностей, поскольку это определит выполнение требований по обеспечению возможности взаимодействия и сериализации. Руководство по выбору типов сущностей и типы, обычно используемые в бизнес-компонентах и компонентах данных, можно найти в главе 13, «[Проектирование бизнес-сущностей](#)». При выборе формата данных обратите внимание на следующие аспекты:

- Если требуется поддерживать сценарии без постоянного подключения в простых приложениях, выполняющих CRUD-операции, используйте объекты **DataSet** или отдельные **DataTable**. Самым распространенным подходом является применение ADO.NET. Это идеальный вариант при работе с существующим приложением, в котором уже используются ADO.NET. При проектировании нового приложения можно применять LINQ to Datasets для заполнения **DataSet** с помощью запросов LINQ.
- Если к слою доступа к данным будут обращаться другие приложения или требуется обеспечить возможность взаимодействия с другими системами или платформами, используйте формат XML.
- Если большое значение имеет удобство и простота обслуживания приложения, применяйте специальные бизнес-сущности. Для этого придется писать дополнительный код для сопоставления сущностей с операциями базы данных, но решения O/RM могут облегчить задачу разработчика. Для обеспечения большей гибкости используйте ADO.NET Entity Framework или другие O/RM-средства, такие как инструментарий с открытым исходным кодом NHibernate.
- Реализуйте сущности, наследуя их от базового класса, который обеспечивает основную функциональность и инкапсулирует выполнение общих задач. Однако не перегружайте базовый класс ненужными операциями, что может понизить связность сущностей, унаследованных от этого базового класса, и негативно сказаться на удобстве обслуживания и производительности.
- В проектируемых сущностях реализуйте взаимодействие с базой данных, полагаясь на компоненты логики доступа к данным. Обеспечьте централизованную реализацию всех политик доступа к данным и соответствующей бизнес-логики. Например, если бизнес-сущности выполняют прямой доступ к базам данных SQL Server, всем приложениям,

развернутым на клиентах, использующих бизнес-сущности, потребуется разрешение на подключение к SQL и вход.

3. **Выберите технологию доступа к данным.** Определитесь с тем, какая функциональность необходима для реализации логики доступа к данным, и выберите технологию, отвечающую этим требованиям. Доступные на платформе Microsoft технологии доступа к данным представлены в Приложении С, «[Матрица технологий доступа к данным](#)».
4. **Спроектируйте компоненты доступа к данным.** Перечислите все источники данных, которые хотите использовать, и примите решение о методе доступа для каждого из них. Примите решение о том, потребуются ли вспомогательные компоненты или желательно упростить разработку и обслуживание компонентов доступа к данным. Наконец, выберите соответствующие шаблоны проектирования. Например, рассмотрите такие шаблоны, как Table Data Gateway, Query Object, Repository и другие. Подробнее эти вопросы рассматриваются в главе 15, «[Проектирование компонентов слоя доступа к данным](#)».
5. **Спроектируйте агенты сервисов.** Используйте соответствующий инструмент для добавления ссылки на сервис. Это обеспечит создание прокси-классов и классов данных, которые представляют контракт данных сервиса. Затем определите, как сервис будет использоваться в приложении. В большинстве приложений доступ к функциональности и данным, предоставляемым агентами сервисов, реализуется посредством компонентов доступа к данным, что обеспечивает единый интерфейс независимо от источника данных. Для небольших приложений бизнес-слой, или даже слой представления, может работать с агентом сервиса напрямую.

Шаблоны проектирования

Основные шаблоны организованы по категориям и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Общие	<p>Active Record (Активная запись). Включает объект доступа к данным в сущность предметной области.</p> <p>Data Mapper (Преобразователь данных). Реализует слой преобразования между объектами и структурой базы данных, используемый для перемещения данных из одной структуры в другую, обеспечивая при этом их независимость.</p> <p>Data Transfer Object (Объект передачи данных). Объект, в котором сохраняются данные, передаваемые между процессами, что обеспечивает сокращение необходимого числа вызовов методов.</p> <p>Domain Model (Модель предметной области). Набор бизнес-объектов, представляющих сущности предметной области и отношения между ними.</p> <p>Query Object (Объект запроса). Объект, представляющий запрос к базе данных.</p> <p>Repository (Хранилище). Представление источника данных в памяти, работающее с сущностями предметной области.</p>

	<p>Row Data Gateway (Шлюз записи данных). Объект, выступающий в роли шлюза к отдельной записи источника данных.</p> <p>Table Data Gateway (Шлюз таблицы данных). Объект, выступающий в роли шлюза к таблице или представлению источника данных и выполняющий сериализацию всех запросов на выборку, вставку, обновление и удаление.</p> <p>Table Module (Модуль таблицы). Единый компонент, реализующий бизнес-логику для всех строк таблицы или представления базы данных.</p>
Пакетная обработка	<p>Parallel Processing (Параллельная обработка). Позволяет обрабатывать множество пакетных операций одновременно, чтобы сократить время обработки.</p> <p>Partitioning (Секционирование). Разбивает большие пакеты, чтобы обрабатывать их параллельно.</p>
Транзакции	<p>Capture Transaction Details (Перехват данных транзакции). Создает объекты базы данных, такие как триггеры и теневые таблицы, для записи всех изменений, вносимых в ходе транзакции.</p> <p>Coarse Grained Lock (Блокировка крупных фрагментов данных). Одной блокировкой блокирует набор взаимосвязанных объектов.</p> <p>Implicit Lock (Неявная блокировка). Использует код инфраструктуры для запроса блокировки от имени кода, выполняющего доступ к совместно используемым ресурсам.</p> <p>Optimistic Offline Lock (Оптимистическая блокировка в автономном режиме). Обеспечивает, чтобы изменения, вносимые в одном сеансе, не конфликтовали с изменениями другого сеанса.</p> <p>Pessimistic Offline Lock (Пессимистическая блокировка в автономном режиме). Предотвращает конфликты, вынуждая транзакцию блокировать данные перед их использованием.</p> <p>Transaction Script (Сценарий транзакции). Организует бизнес-логику каждой транзакции в одну процедуру, обращаясь к базе данных напрямую либо через тонкую оболочку над базой данных.</p>

Более подробно шаблоны Domain Model, Table Module, Coarse-Grained Lock, Implicit Lock, Transaction Script, Active Record, Data Mapper, Data Transfer Object, Optimistic Offline Locking, Pessimistic Offline Locking, Query Object, Repository, Row Data Gateway и Table Data Gateway рассматриваются в книге Мартина Фаулера «Архитектура корпоративных программных приложений». Вильямс, 2007. Или по адресу <http://martinfowler.com/eaaCatalog/>.

Более подробно шаблон Capture Transaction Details рассматривается в статье «*Data Patterns*» (Шаблоны данных) по адресу <http://msdn.microsoft.com/en-us/library/ms998446.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*.NET Data Access Architecture Guide*» (Руководство по архитектуре доступа к данным в .NET) по адресу <http://msdn.microsoft.com/en-us/library/ms978510.aspx>.
- «*Concurrency Control*» (Управление параллельной обработкой) по адресу <http://msdn.microsoft.com/en-us/library/ms978457.aspx>.
- «*Data Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/ms998446.aspx>.

- «*Designing Data Tier Components and Passing Data Through Tiers*» (Проектирование компонентов уровня доступа к данных и передача данных по уровням) по адресу <http://msdn.microsoft.com/en-us/library/ms978496.aspx>.
 - «*Typing, storage, reading, and writing BLOBs*» (Ввод, хранение, чтение и запись BLOB-объектов) по адресу http://msdn.microsoft.com/en-us/library/ms978510.aspx#daag_handlingblobs.
 - «*Using stored procedures instead of SQL statements*» (Использование хранимых процедур вместо SQL-выражений) по адресу <http://msdn.microsoft.com/en-us/library/ms978510.aspx>.
 - Сайт сообщества «*NHibernate Forge*» по адресу <http://nhforge.org/Default.aspx>.
-

9

Рекомендации по проектированию слоя сервисов

Обзор

При предоставлении доступа к функциональности приложения через сервисы функции сервиса должны быть выделены в отдельный слой сервисов. Из данной главы вы поймете, какое место слой сервисов занимает в архитектуре приложения, узнаете, из каких этапов складывается проектирование слоя сервисов, какие шаблоны и технологии используются. В главу также включено руководство по решению типовых проблем, обычно возникающих при проектировании слоя сервисов.

В слое сервисов определяется и реализуется интерфейс сервиса и контракты данных (или типы сообщений). Одним из самых важных моментов, о котором нельзя забывать, является то, что сервис никогда не должен раскрывать детали внутренних процессов или бизнес-сущностей, используемых в приложении. В частности, необходимо гарантировать, что сущности бизнес-слоя не будут оказывать большого влияния на контракты данных. Слой сервисов должен предоставлять компоненты-трансляторы, которые будут обеспечивать преобразование форматов данных между сущностями бизнес-слоя и контрактами данных.

На рис. 1 показано место слоя сервисов в общей архитектуре приложения.

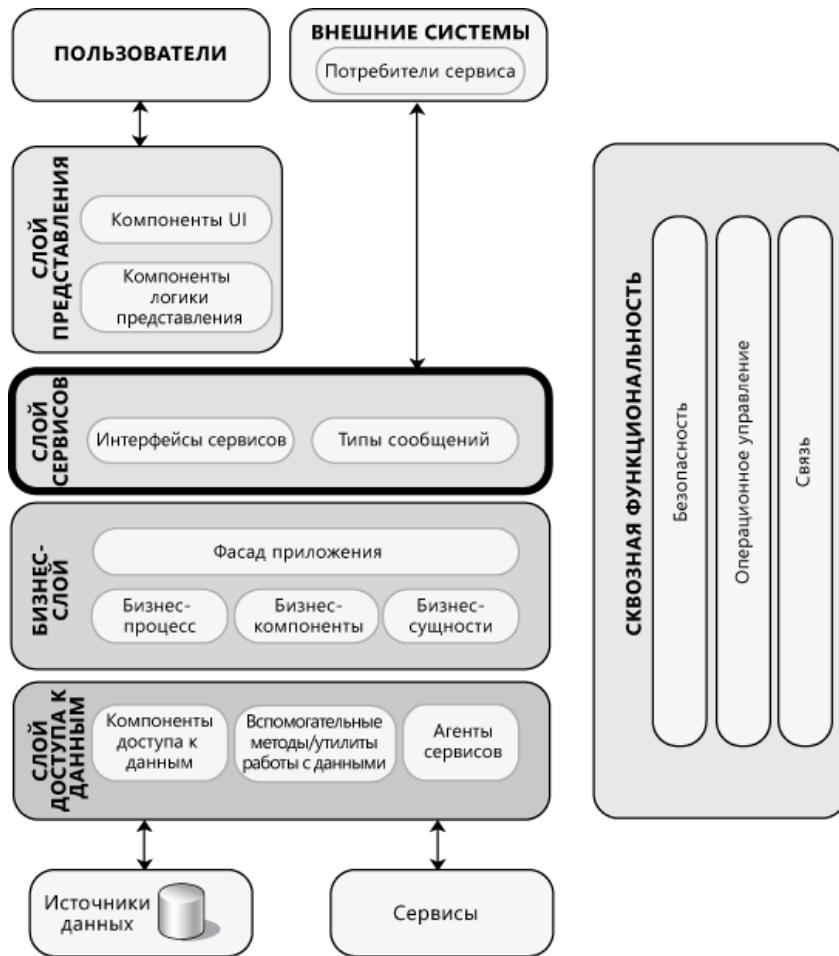


Рис. 10

Типовое приложение со слоем сервисов

Слой сервисов обычно включает следующие компоненты:

- **Интерфейсы сервисов.** Сервисы предоставляют интерфейсы, в которые передаются все входящие сообщения. Интерфейс сервиса можно рассматривать как фасад, предоставляющий потенциальным потребителям доступ к бизнес-логике, реализованной в приложении (как правило, логику бизнес-слоя).
- **Типы сообщений.** При обмене данными через слой сервисов структуры данных заключаются в структуры сообщений, поддерживающие разные типы операций. Слой сервисов также обычно включает типы и контракты данных, которые определяют используемые в сообщениях типы данных.

Более подробно компоненты, обычно используемые в слое сервисов, рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)». О проектировании интерфейсов сервисов рассказывает глава 18, «[Взаимодействие и обмен сообщениями](#)».

Общие принципы проектирования

При проектировании слоя сервисов необходимо учесть множество факторов. Многие из этих аспектов проектирования касаются проверенных практик, имеющих отношение к созданию многослойных архитектур. Однако для сервисов необходимо рассматривать также и факторы,

связанные с обменом сообщениями. Основное, на что требуется обратить внимание: сервисы взаимодействуют посредством обмена сообщениями, как правило, по сети, что по сути своей медленнее, чем прямое взаимодействие внутри процесса, и обычно сервисы взаимодействуют с потребителями асинхронно. Кроме того, сообщения, передаваемые между сервисом и потребителем, могут быть маршрутизированы, изменены, доставлены в порядке, отличном от порядка, в котором они отправлялись, или даже утрачены, если не реализован механизм гарантированной доставки. Все эти аспекты требуют создания дизайна, в котором будет учтено такое недетерминированное поведение процесса обмена сообщениями. При проектировании слоя сервисов руководствуйтесь следующими рекомендациями:

- **Проектируйте сервисы, областью действия которых является приложение, а не компонент.** Операции сервисов должны охватывать большие фрагменты функциональности и сосредотачиваться на операциях приложения.
Проектирование слишком узконаправленных операций может иметь негативное влияние на производительность и масштабируемость. Тем не менее, необходимо обеспечить, чтобы сервис не возвращал неограниченно большие объемы данных. Например, для сервиса, который может возвращать большие объемы демографических данных, необходимо создать операцию, которая будет возвращать не все данные за один вызов, а подмножества данных определенного размера. Размер подмножества данных должен соответствовать возможностям сервиса и требованиям его потребителей.
- **Проектируйте расширяемые сервисы и контракты данных, не делая допущений о предполагаемом клиенте.** Иначе говоря, контракты данных должны быть спроектированы так, чтобы их можно было расширять, не оказывая влияния на потребителей сервиса. Однако чтобы избежать излишней сложности или для поддержки изменений, не обладающих обратной совместимостью, возможно, придется создавать новые версии интерфейса сервиса, которые будут функционировать параллельно с существующими версиями. Нельзя делать предположения о клиентах или о том, как они планируют использовать предоставляемый сервис.
- **Проектируйте только соответственно контракту сервиса.** Слой сервисов должен реализовывать и обеспечивать только функциональность, оговоренную контрактом сервиса. Внутренняя реализация и детали сервиса никогда не должны раскрываться внешним потребителям. Также если возникает необходимость изменить контракт сервиса для включения новой функциональности, реализованной сервисом, и новые операции и типы не являются обратно совместимыми с существующими контрактами, рассмотрите возможность создания версий контрактов. Определите новые операции, предоставляемые сервисом в новой версии контракта сервиса, и новые типы передаваемых сообщений в новой версии контракта данных.
Проектирование контрактов сообщений рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».
- **Отделяйте функциональность слоя сервиса от функциональности инфраструктуры.** В слое сервисов реализация сквозной функциональность не должен сочетаться с кодом логики сервиса, поскольку это может усложнить

расширение и обслуживание реализаций. Обычно сквозная функциональности реализуется в отдельных компонентах, доступных для компонентов бизнес-слоя.

- **Создавайте сущности из стандартных элементов.** По возможности компонуйте сложные типы и объекты передачи данных, используемые сервисом, из стандартных элементов.
- **При проектировании учитывайте возможность поступления некорректных запросов.** Никогда нельзя делать предположение о том, что все поступающие в сервис сообщения будут корректными. Реализуйте логику валидации всех входных данных на основании значений, диапазонов и типов данных, и отклоняйте недопустимые данные. Более подробно валидация рассматривается в главе 17, «[Сквозная функциональность](#)».
- **Обеспечьте в сервисе функциональность для выявления и обработки повторяющихся сообщений (идемпотентность).** Реализация широко известных шаблонов, таких как Receiver и Replay Protection, при проектировании сервиса обеспечит, что дублирующие сообщения не будут обрабатываться, или что повторная обработка не будет влиять на результат.
- **Проектируйте сервис, чтобы он мог обрабатывать сообщения, поступающие в произвольном порядке (коммутативность).** Если существует вероятность поступления сообщений в порядке, отличном от того, в котором они отправлялись, реализуйте возможность сохранения сообщений с последующей обработкой в правильном порядке.

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании слоя сервисов. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по проектированию аспектов, в которых чаще всего возникают ошибки:

- [Аутентификация](#)
- [Авторизация](#)
- [Сетевое взаимодействие](#)
- [Управление исключениями](#)
- [Каналы обмена сообщениями](#)
- [Структура сообщения](#)
- [Конечная точка сообщения](#)
- [Безопасность сообщений](#)
- [Маршрутизация сообщений](#)
- [Преобразование сообщений](#)

- [Интерфейс сервиса](#)
 - [Валидация](#)
-

Более подробно протоколы сообщений, асинхронная связь, возможность взаимодействия, производительность и доступные технологии рассматриваются в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Аутентификация

Аутентификация используется для установления подлинности потребителя сервиса. Проектирование эффективной стратегии аутентификации для слоя сервисов имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак с подделкой пакетов, атак перебором по словарю, перехватом сеансов и других типов атак. При проектировании стратегии аутентификации руководствуйтесь следующими рекомендациями

- Найдите походящий механизм безопасно идентифицировать пользователей, по возможности используя функции базовой платформы, и определите границы доверия, при пересечении которых должна применяться аутентификация.
 - Учтите последствия применения разных настроек уровней доверия для выполнения кода сервиса.
 - Для обычной аутентификации или при передаче учетных данных в открытом виде используйте такие безопасные протоколы, как Secure Sockets Layer (SSL). Для SOAP-сообщений применяйте механизмы безопасности уровня сообщения, поддерживаемые стандартами WS* (Web Services Security, Web Services Trust и Web Services Secure Conversation).
-

Авторизация

Авторизация позволяет определить, какие ресурсы или действия доступны тому или иному аутентифицированному потребителю сервиса. Проектирование эффективной стратегии авторизации для слоя сервисов имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для разглашения сведений, повреждения или подделки данных и несанкционированного получения прав. Как правило, стратегия авторизации работает с обобщенными действиями или операциями, а не ресурсами, которые их выполняют. При проектировании стратегии авторизации руководствуйтесь следующими рекомендациями:

- Задайте соответствующие права доступа к ресурсам для пользователей, групп и ролей. Выполняйте сервисы под учетной записью, обладающей лишь необходимым набором разрешений.
- По возможности избегайте слишком подробного деления прав при авторизации, чтобы обеспечить эффективность и удобство обслуживания применяемой стратегии авторизации.

- Для аутентификации Windows используйте авторизацию URL и/или авторизацию доступа к файлам.
- Где это возможно, ограничьте доступ к открытым Веб-методам путем использования декларативных методов проверки прав.

Сетевое взаимодействие

При проектировании стратегии сетевого взаимодействия для сервиса выбор протокола должен основываться на сценарии развертывания, который должен поддерживать ваш сервис. При проектировании стратегии связи руководствуйтесь следующими рекомендациями

- На основании требований, предъявляемых к механизмам сетевого взаимодействия, определите используемый механизм: запрос-ответ или двусторонняя связь, и то, должен ли обмен сообщениями быть односторонним или двунаправленным. Также выясните необходимость в асинхронных вызовах.
- Примите решение о том, как будет обрабатываться ненадежная или неустойчивая связь, возможно, путем реализации агента сервиса или использования надежной системы очереди сообщений, такой как Message Queuing.
- Если сервис предполагается развертывать в закрытой сети, используйте протокол Transmission Control Protocol (TCP)¹, что обеспечит максимальную эффективность сетевого взаимодействия. Если сервис будет развертываться в Интернете, используйте протокол Hypertext Transfer Protocol (HTTP)².
- Для обеспечения максимальной гибкости используйте динамические URL для конфигурации конечных точек. Например, вместо указания URL конечных точек в коде, по возможности используйте конфигурацию или сервис каталогов, такой как Universal Discovery Description and Integration (UDDI)³.
- Проверяйте адреса конечных точек и обеспечьте защиту конфиденциальных данных в сообщениях.

Управление исключениями

Проектирование эффективной стратегии управления исключениями для слоя сервисов имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак Отказа в обслуживании (Denial of Service, DoS) и может допустить разглашение конфиденциальных или наиболее важных данных о приложении. Формирование и обработка исключений является ресурсоемкой операцией, поэтому важно, чтобы проектирование стратегии управления исключениями велось с учетом влияния на производительность. Хорошей практикой является проектирование

¹ Протокол управления передачей (прим. переводчика).

² Протокол передачи гипертекста (прим. переводчика).

³ Универсальный поиск, описание и взаимодействие (прим. переводчика).

централизованного механизма управления исключениями и протоколирования. Также предусмотрите механизмы управления, поддерживающие инструментирование и централизованный мониторинг, чтобы облегчить работу системным администраторам. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Перехватывайте только те исключения, которые можете обработать, и продумайте, как обеспечить целостность сообщения при возникновении исключения. Обеспечьте корректную обработку необрабатываемых исключений и не используйте исключения для управления бизнес-логикой.
- При работе с SOAP используйте элементы Fault (Сбой) или специальные расширения для доставки данных исключения вызывающей стороне.
- Обеспечьте протоколирование исключений, обеспечивающее неразглашение конфиденциальных данных в сообщениях об ошибках или файлах журнала.

Более подробно методики управления исключениями рассматриваются в главе 17, «[Сквозная функциональность](#)».

Каналы обмена сообщениями

Связь между сервисом и его потребителями состоит в передаче данных по каналу. В большинстве случаев используются каналы, предоставляемые выбранной инфраструктурой сервиса, такой как Windows Communication Foundation (WCF). Необходимо понимать, какие шаблоны поддерживает выбранная инфраструктура, и определить подходящий канал для взаимодействия с потребителями сервиса. При проектировании каналов обмена сообщениями руководствуйтесь следующими рекомендациями:

- Из имеющихся шаблонов каналов обмена сообщениями, таких как Channel Adapter, Messaging Bus и Messaging Bridge, выберите наиболее подходящие для вашего сценария. Убедитесь также в правильности выбора инфраструктуры сервисов и ее соответствии всем требованиям.
- Продумайте, как будете перехватывать и проверять данные между конечными точками, если в этом появится необходимость.
- Опишите условия возникновения исключений в канале и способы их обработки.
- Продумайте, как будете обеспечивать доступ к клиентам, не поддерживающим обмен сообщениями.

Структура сообщения

Данные, которыми обмениваются сервис и его потребитель, должны быть заключены в сообщение. Формат этого сообщения определяется типами поддерживаемых операций. Например, может происходить обмен документами, выполнение команд или формирование событий. При проектировании структуры сообщения руководствуйтесь следующими рекомендациями:

- Из имеющихся шаблонов структуры сообщения, таких как Command, Document, Event и Request-Reply, выберите наиболее подходящие для вашего сценария.
 - Разделяйте очень большие объемы данных на меньшие блоки и отправляйте их последовательно.
 - При использовании медленных каналов доставки сообщений включайте в сообщения данные о сроке действия. Сервис должен игнорировать сообщения, срок действия которых истек.
-

Конечная точка для передачи сообщения

Конечная точка для передачи сообщения представляет подключение, которое приложения используют для взаимодействия с сервисом. Реализация интерфейса сервиса представляет конечную точку для передачи сообщений. При проектировании реализации сервиса необходимо учесть возможность поступления дублирующихся или недействительных сообщений. При проектировании конечных точек для передачи сообщений руководствуйтесь следующими рекомендациями:

- Из доступных шаблонов конечных точек для передачи сообщений, таких как Gateway, Mapper, Competing Consumers и Message Dispatcher, выберите наиболее подходящие вашему сценарию.
 - Определитесь с тем, должны ли приниматься все сообщения или необходимо реализовать фильтр для обработки лишь определенных сообщений.
 - Обеспечьте идемпотентность интерфейса для передачи сообщений.
Идемпотентность – это ситуация, когда при поступлении от одного потребителя дублирующихся сообщений обрабатывается только одно из них. Иначе говоря, идемпотентная конечная точка гарантирует, что будет обработано только одно сообщение и все дублирующиеся сообщения будут проигнорированы.
 - Обеспечьте коммутативность интерфейса для передачи сообщений.
Коммутативность касается порядка приема сообщений. В некоторых случаях может понадобиться сохранять входящие сообщения, чтобы обеспечить их обработку в правильном порядке.
 - Проектируйте с учетом сценариев без постоянного подключения. Например, реализуйте поддержку гарантированной доставки или сохранения сообщений для отложенной доставки. Обеспечьте, чтобы при отсутствии подключения не выполнялись попытки соединения с конечными точками.
-

Безопасность сообщений

При передаче конфиденциальных данных между сервисом и его потребителем необходимо предусмотреть защиту сообщений. Можно использовать защиту на транспортном уровне (IPSec или SSL) или защиту на уровне сообщения (шифрование и цифровые подписи). При проектировании безопасности сообщений руководствуйтесь следующими рекомендациями:

- В большинстве случаев необходимо предусмотреть возможности защиты содержимого сообщений методами обеспечения безопасности на уровне сообщения. Безопасность на уровне сообщения помогает защищать конфиденциальные данные, передаваемые в сообщениях, путем их шифрования и предоставления цифровой подписи, которая защитит от повреждений или подделки сообщений, а также позволит выявить отправителя сообщения. Однако нельзя забывать, что каждый уровень защиты негативно сказывается на производительности.
- Если взаимодействие между сервисом и потребителем осуществляется без посредников, таких как другие серверы и маршрутизаторы, можно использовать защиту на транспортном уровне, например, с помощью протоколов IPSec или SSL. Но если сообщение передается через одного или более посредников, должна использоваться только безопасность на уровне сообщения, поскольку при первом варианте защиты сообщение дешифруется и затем повторно шифруется на каждом посреднике, что представляет угрозу безопасности.
- Применение обоих типов защиты, и на транспортном уровне, и на уровне сообщений, обеспечит максимальную безопасность. Безопасность на транспортном уровне поможет защитить данные заголовков, которые не могут быть зашифрованы при использовании безопасности на уровне сообщений.

Маршрутизация сообщений

Маршрутизатор сообщений обеспечивает отделение потребителя сервиса от реализации сервиса. Существует три основных типа маршрутизаторов: простые, составные и шаблонные. В простых маршрутизаторах конечная точка назначения сообщения определяется с помощью единственного маршрутизатора. Составные маршрутизаторы объединяют в себе множество простых маршрутизаторов для обработки более сложных потоков сообщений. Архитектурные шаблоны используются для описания разных стилей маршрутизации на основе простых маршрутизаторов сообщений. При проектировании маршрутизации сообщений руководствуйтесь следующими рекомендациями:

- Из доступных шаблонов маршрутизации сообщений, таких как Aggregator, Content-Based Router, Dynamic Router и Message Filter, выберите наиболее подходящие для вашего сценария.
- При поступлении от потребителя последовательных сообщений маршрутизатор должен гарантировать их доставку в одну конечную точку в заданном порядке (коммутативность).
- Маршрутизатор сообщений может просматривать данные сообщения, чтобы понять, как выполнять его маршрутизацию. Таким образом, необходимо гарантировать возможность доступа маршрутизатора к этим данным. Возможно, сведения маршрутизации придется вынести в заголовок. При шифровании сообщения необходимо гарантированно обеспечить, что в нешифрованном заголовке имеются данные, необходимые для маршрутизации сообщения.

Преобразование сообщений

Часто при передаче сообщений между сервисом и потребителем приходится преобразовывать их в формат, понятный потребителю. Обеспечить доступ к каналу обмена сообщениями для клиентов, не поддерживающих обмен сообщениями, можно с помощью адаптеров; а для преобразования данных сообщений в формат, понятный всем потребителям, используются трансляторы. При проектировании преобразования сообщений руководствуйтесь следующими рекомендациями:

- Определите требования и место выполнения преобразования. Учтите издержки производительности при преобразовании и постарайтесь максимально сократить количество выполняемых преобразований.
- Выберите соответствующие шаблоны для преобразования сообщений, такие как Canonical Data Mapper, Envelope Wrapper и Normalizer. Однако применяйте модель Canonical Data Mapper только по необходимости.
- Определяйте формат сообщения с помощью метаданных.
- Рассмотрите возможность использования внешнего хранилища для хранения метаданных.

Интерфейс сервиса

Интерфейс сервиса представляет контракт, предоставляемый сервисом. Контракт определяет операции, поддерживаемые сервисом, и связанные с ними параметры и объекты передачи данных. При проектировании интерфейса сервиса необходимо учесть пересекаемые границы и тип потребителей, которые будут осуществлять доступ к сервису. Например, операции сервиса должны быть операциями уровня приложения и слабо детализированными. Одна из самых больших ошибок при проектировании интерфейса сервиса – интерпретация сервиса как компонента с детализированными операциями. Это приводит к созданию дизайна, требующего большого количества вызовов через физические границы или границы процесса, что может снизить производительность и увеличить задержки при выполнении операций. При проектировании интерфейса сервиса руководствуйтесь следующими рекомендациями:

- Используйте слабо детализированный интерфейс для обеспечения пакетирования запросов и максимального сокращения количества вызовов по сети.
- Проектируйте интерфейсы сервисов таким образом, чтобы изменения бизнес-логики не оказывали на них влияния. Однако при изменении бизнес-требований, вероятно, это будет просто неизбежно.
- Не реализуйте бизнес-правила в интерфейсе сервиса.
- Для обеспечения максимальной совместимости с разными типами клиентов используйте параметры стандартных форматов. При проектировании интерфейса не делайте допущений о возможных вариантах использования сервиса клиентом.
- Не используйте наследование объектов для реализации контроля версий интерфейса сервиса.

- Включайте трассировку и компиляцию в режиме отладки для всех сервисов только на время разработки и тестирования.
-

Валидация

Для обеспечения защиты слоя сервисов необходимо проверять все получаемые им запросы, в противном случае приложение будет уязвимым и к злонамеренным атакам, и к ошибкам, обусловленным некорректными входными данными. Точного и исчерпывающего определения действительного или злонамеренного ввода не существует, кроме того, риск эксплуатации уязвимостей зависит от того, как приложение использует поступающие данные. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- Используйте централизованный подход к проверке, чтобы максимально повысить тестируемость и пригодность для повторного использования.
 - Ограничивайте, отклоняйте и очищайте все содержимое сообщений, включая параметры. Проводите проверку длины, диапазона, формата и типа.
 - Используйте схемы проверки сообщений. Валидация с применением схем рассматривается в статье «Message Validation» (Валидация сообщений) по адресу <http://msdn.microsoft.com/en-us/library/cc949065.aspx> и в статье «Input/Data Validation» (Валидация ввода/данных) по адресу <http://msdn.microsoft.com/en-us/library/cc949061.aspx>.
-

REST и SOAP

Representational State Transfer (REST)¹ и SOAP представляют два разных стиля реализации сервисов. Фактически, REST – это архитектурный шаблон, создаваемый с использованием простых команд и хорошо ложащийся на протокол HTTP. Хотя архитектурные принципы REST могли бы применяться и с другими протоколами, на практике реализации REST используются в сочетании с HTTP. SOAP – это основанный на XML протокол обмена сообщениями, который может использоваться с любым протоколом связи, в том числе и с HTTP.

Основное различие между этими двумя подходами в способе сохранения состояния сервиса. Состояние сервиса не должно рассматриваться как состояние приложения или сеанса; это разные состояния, которые приложение проходит в течение его жизненного цикла. При использовании SOAP переход из состояния в состояние может быть реализован через взаимодействие с единственной конечной точкой сервиса, которая может инкапсулировать и предоставлять доступ ко многим операциям и типам сообщений.

При использовании REST набор операций ограничен, и эти операции применяются к ресурсам, которые предоставляются и доступны через URI (HTTP-адреса). В сообщениях фиксируется текущее или необходимое состояние ресурса. REST хорошо работает с Веб-приложениями, что позволяет обеспечивать преимущество поддержки HTTP не-XML MIME-типам или потоковую передачу содержимого. Потребители сервиса при перемещении по REST-ресурсам

¹ Передача репрезентативного состояния (прим. переводчика).

взаимодействуют с URI так же, как обычный пользователь перемещается и взаимодействует с Веб-страницами.

С большинством реализаций сервисов могут использоваться и REST, и SOAP, часто подход с применением REST лучше подходит для публичных сервисов или в случаях, когда доступ к сервису может осуществляться неизвестными потребителями. SOAP намного лучше подходит для реализации широкого диапазона процедурных взаимодействий, таких как интерфейс между слоями приложения. С SOAP вы не ограничены одним только протоколом HTTP. Стандарты WS-*, которые могут использоваться в SOAP, предоставляют стандартный и, следовательно, обеспечивающий возможность взаимодействия метод решения общих проблем обмена сообщениями, таких как безопасность, транзакции, адресация и надежность. REST тоже может обеспечивать такую же функциональность, но в настоящее время эта область практически не стандартизована, поэтому часто приходится создавать собственные механизмы.

В общем, при проектировании взаимодействий на базе SOAP можно использовать те же принципы, чтобы и для REST-взаимодействий без сохранения состояния. В обоих подходах для описания обмена данными (полезной нагрузкой) используются обычные глаголы. В случае с SOAP набор глаголов не является окончательным и определяется конечной точкой сервиса. Для REST набор глаголов ограничен предопределенными словами, отражающими протокол HTTP. При выборе между REST и SOAP руководствуйтесь следующими рекомендациями:

- SOAP – протокол, обеспечивающий базовую инфраструктуру обмена сообщениями, на которой могут строиться абстрактные слои. Он обычно применяется как RPC-инфраструктура¹, передающая запросы и ответы по сети с помощью XML-форматированных сообщений.
- SOAP обеспечивает учет таких аспектов, как безопасность и адресация, через внутреннюю реализацию протокола, но требует, чтобы был доступен стек SOAP.
- REST – методика, которая может использовать другие протоколы, такие как JavaScript Object Notation (JSON)², протокол публикации Atom и собственные форматы Plain Old XML (POX)³.
- REST предоставляет приложение и данные как конечный автомат, а не просто конечную точку сервиса. Он позволяет использовать для выполнения запросов и изменения состояния системы стандартные HTTP-методы вызова, такие как GET и PUT. REST по своей природе не сохраняет состояния, поэтому каждый отдельный запрос, присыпаемый от клиента серверу, должен содержать все необходимые данные для его интерпретации, поскольку сервер не сохраняет данных о состоянии сеанса.

¹ Механизм удаленного вызова процедур (*прим. научного редактора*).

² Объектная нотация JavaScript (*прим. переводчика*).

³ Обычный старый XML (*прим. переводчика*).

Аспекты проектирования при использовании REST

REST представляет архитектурный стиль для распределенных систем и создан с целью упростить систему путем ее деления на ресурсы. Ресурсы и операции, поддерживаемые ими, представляются и предоставляются как набор URI, передаваемых по протоколу HTTP. При проектировании REST-ресурсов руководствуйтесь следующими рекомендациями

- Обозначьте и распределите по категориям ресурсы, которые будут предоставляться клиентам.
- Выберите подход для представления ресурсов. Хорошей практикой является использование значащих имен для исходных точек REST и уникальных идентификаторов для конкретных экземпляров ресурсов. Например, <http://www.contoso.com/employee/> представляет исходную точку служащий. <http://www.contoso.com/employee/smithah01> использует ID служащего для обозначения конкретного служащего.
- Примите решение о необходимости поддержки множества реализаций для разных ресурсов. Например, о том, должен ли ресурс поддерживать форматы XML, Atom или JSON, и сделайте это частью запроса к ресурсу, тогда ресурс будет предоставляться в разных форматах (например, <http://www.contoso.com/example.atom> и <http://www.contoso.com/example.json>).
- Примите решение о необходимости поддерживать множество представлений для разных ресурсов. Например, должен ли ресурс поддерживать операции GET и POST или только GET. По возможности не увлекайтесь применением операций POST и избегайте размещения действий в URI.
- Не реализуйте сохранения состояния сеанса пользователя в сервисе и не пытайтесь использовать гипертекст (такой как скрытые элементы управления в Веб-страницах) для управления состоянием. Например, когда пользователь передает запросы, например, для добавления элемента в корзину, сохраняйте данные в постоянном хранилище, таком как база данных.

Аспекты проектирования при использовании SOAP

SOAP – основанный на сообщениях протокол, используемый для реализации слоя обмена сообщениями сервиса. Сообщение представляет собой конверт, в котором содержатся заголовок и тело. Заголовок может использоваться для предоставления сведений, внешних по отношению к операции, осуществляющей сервисом. Например, в заголовок могут быть включены данные безопасности, транзакции или маршрутизации. Тело включает контракты в форме XML-схем, используемые для реализации сервиса. При проектировании SOAP-сообщений руководствуйтесь следующими рекомендациями:

- Примите решение о том, как будут обрабатываться сбои и ошибки, и как будут возвращаться клиентам соответствующие сведения об ошибке. Более подробно эти вопросы рассматриваются в статье «*Exception Handling in Service Oriented Applications*» (Обработка исключений в сервисно-ориентированных приложениях) по адресу <http://msdn.microsoft.com/en-us/library/cc304819.aspx>.

- Определите набор операций, осуществляемых сервисом; структуры данных, передаваемые с запросом к сервису; и ошибки или сбои, возвращаемые запросом к сервису.
- Выберите соответствующую модель безопасности для сервисов. Более подробно эти вопросы рассматриваются в статье «*Improving Web Services Security: Scenarios and Implementation Guidance for WCF*» (Повышение безопасности Веб-сервисов: сценарии и руководство по реализации для WCF) по адресу <http://msdn.microsoft.com/en-us/library/cc949034.aspx>.
- Избегайте использования составных типов в сообщениях. Применение только простых типов обеспечит максимальную возможность взаимодействия.

Более подробно REST и SOAP рассматриваются в главе 25, «[Проектирование приложений сервисов](#)».

Вопросы выбора технологий

Следующие рекомендации помогут правильно выбрать технологию для реализации слоя сервисов:

- Для простоты используйте ASP.NET Web services (ASMX), но только если доступен Веб-сервер, работающий под управлением Microsoft Internet Information Services (IIS).
- Используйте сервисы WCF, если требуется реализовать расширенные возможности, такие как надежные сеансы с гарантированной доставкой сообщений и транзакции, трассировка действий, протоколирование сообщений, счетчики производительности и поддержка нескольких транспортных протоколов.
- Если для сервиса решено использовать ASMX и требуется обеспечить безопасность на уровне сообщения и передачу двоичных данных, можно применить Web Service Extensions (WSE)¹. Однако, как правило, в случае необходимости реализации функциональность WSE следует переходить к WCF.
- Если решено использовать WCF для сервиса:
 - Используйте HTTP-транспорт на базе спецификаций SOAP для обеспечения возможности взаимодействия с не-WCF и не-Windows клиентами.
 - Используйте протокол TCP и двоичное кодирование сообщений с безопасностью на транспортном уровне и аутентификацией Windows, если требуется поддерживать клиентов во внутренней сети.
 - Используйте протокол именованных каналов и бинарное кодирование сообщений, если требуется поддерживать WCF-клиентов, размещенных на том же компьютере.

¹ Расширения Веб-сервисов (прим. переводчика).

- Определяйте контракты сервисов, которые используют явную, а не неявную, оболочку сообщений. Это позволит определять контракты сообщений как вход и выход для операций, таким образом, вы сможете расширять контракты данных, включенные в контракты сообщения, не оказывая влияния на контракт сервиса.

Доступные технологии обмена сообщениями представлены в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Вопросы развертывания

Слой сервисов может развертываться на одном уровне с другими слоями приложения или на разных уровнях, если это обусловлено требованиями производительности и изоляции. Тем не менее, в большинстве случаев слой сервисов располагается физически на одном уровне с бизнес-слоем, это позволяет обеспечить наилучшую производительность при предоставлении бизнес-функциональности. При развертывании слоя сервисов руководствуйтесь следующими рекомендациями:

- Если это не идет в разрез с требованиями производительности и безопасности среды производственной эксплуатации, развертывайте слой сервисов на одном уровне с бизнес-слоем, это обеспечит максимальную производительность приложения.
- Если сервис размещается на одном уровне с потребителем сервиса, используйте именованные каналы или протоколы общей памяти.
- Если сервис используется только приложениями локальной сети, используйте для взаимодействия протокол TCP.
- Если сервис публично доступен через Интернет, используйте HTTP в качестве транспортного протокола

Более подробно схемы развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

Этапы проектирования слоя сервисов

Проектирование слоя сервисов начинается с определения интерфейса сервиса, состоящего из контрактов, которые планируется предоставлять. Обычно такой подход называют Contract First Design (Контрактно-ориентированное проектирование). Следующий шаг после определения интерфейса сервиса – проектирование реализации сервиса, который используется для преобразования контрактов данных в бизнес-сущности и для взаимодействия с бизнес-слоем. Проектирование слоя сервисов может включать следующие основные этапы:

1. Определение контрактов данных и сообщений, которые представляют используемую для сообщений схему.
2. Определение контрактов сервиса, которые представляют операции, поддерживаемые сервисом.

3. Определение контрактов сбоев, которые возвращают данные об ошибках потребителям сервиса.
4. Проектирование объектов преобразований, которые обеспечивают преобразования между бизнес-сущностями и контрактами данных.
5. Проектирования абстракции, используемой для взаимодействия с бизнес-слоем.

Для разработки Веб-сервисов можно использовать инструменты проектирования, такие как Web Service Software Factory: Modeling Edition (также известный как Service Factory) от группы patterns & practices. Этот интегрированный набор ресурсов создан с целью обеспечить возможность быстро и единообразно создавать Веб-сервисы с использованием широко известных архитектурных схем и шаблонов проектирования. Более подробно данные вопросы рассматриваются в статье «*Web Service Software Factory: Modeling Edition*» (Фабрика ПО для разработки Веб-сервисов: издание для построения моделей) по адресу <http://msdn.microsoft.com/en-us/library/cc487895.aspx>.

Проектирование контрактов сообщений и подход Contract-First рассматриваются в главе 18, «[Взаимодействие и обмен сообщениями](#)». Абстракции в многослойных архитектурах посвящена глава 5, «[Рекомендации по проектированию многослойных приложений](#)».

Шаблоны проектирования

Основные шаблоны организованы по категориям и представлены в следующей таблице.

Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны проектирования
Сетевое взаимодействие	Duplex (Двусторонний обмен сообщениями) . Двунаправленный обмен сообщениями, при котором и сервис, и клиент отправляют сообщения друг другу независимо и без учета того, какой шаблон используется, One-Way (Однонаправленный) или Request-Reply (Запрос-ответ). Fire and Forget (Отправил и забыл) . Однонаправленный обмен сообщениями, используемый, когда ожидать ответа нет необходимости. Reliable Sessions (Надежные сеансы) . Надежная передача сообщений из конца в конец между источником и точкой назначения, не зависящая от количества или типа посредников между конечными точками. Request Response (Запрос-ответ) . Механизм двунаправленного обмена сообщениями, при котором клиент ожидает ответа на каждое отправленное сообщение.
Каналы обмена сообщениями	Channel Adapter (Адаптер канала) . Компонент, который может выполнять доступ к API или данным приложения и на основании этих данных публиковать сообщения в канале, а также может принимать сообщения и вызывать функции приложения. Message Bus (Шина сообщений) . Структурирует связующее промежуточное ПО между приложениями как шину связи, что позволяет приложениям взаимодействовать, используя обмен сообщениями. Messaging Bridge (Мост обмена сообщениями) . Компонент, соединяющий обменивающиеся сообщениями системы и тиражирующий

	<p>сообщения между этими системами.</p> <p>Point-to-Point Channel (Канал «точка-точка»). Передача сообщения по каналу «точка-точка» гарантирует, что получит это конкретное сообщение только один получатель.</p> <p>Publish-Subscribe Channel (Канал публикации-подписки). Создает механизм отправки сообщений только приложениям, заинтересованным в получении этих сообщений, без идентификации получателей.</p>
<i>Структура сообщения</i>	<p>Command Message (Сообщение с командой). Структура сообщения, используемая для поддержки команд.</p> <p>Document Message (Сообщение с данными документа). Структура, используемая для передачи документов или структуры данных между приложениями.</p> <p>Event Message (Сообщение о событии). Структура, обеспечивающая надежное асинхронное уведомление о событиях между приложениями.</p> <p>Request-Reply (Запрос-отклик). Запрос и отклик передаются по разным каналам.</p>
<i>Конечная точка сообщения</i>	<p>Competing Consumer (Конкурирующий потребитель). Задает несколько потребителей для одной очереди сообщений и заставляет их конкурировать за право обрабатывать сообщения, что позволяет обменивающемуся сообщениями клиенту обрабатывать множество сообщений одновременно.</p> <p>Durable Subscriber (Постоянный подписчик). Чтобы обеспечить гарантированную доставку в сценарии без подключения, сообщения сохраняются и затем предоставляются для доступа клиенту при подключении к каналу сообщений.</p> <p>Idempotent Receiver (Идемпотентный получатель). Гарантирует, что сервис обрабатывает сообщение только один раз.</p> <p>Message Dispatcher (Диспетчер сообщений). Компонент, рассылающий сообщения множеству потребителей.</p> <p>Messaging Gateway (Шлюз обмена сообщениями). Инкапсулирует вызовы, осуществляемые посредством обмена сообщениями, в один интерфейс, чтобы отделить их от остального кода приложения.</p> <p>Messaging Mapper (Преобразователь обмена сообщениями). Преобразует запросы в бизнес-объекты для входящих сообщений и выполняет обратный процесс для преобразования бизнес-объектов в ответные сообщения.</p> <p>Polling Consumer (Опрашивающий потребитель). Потребитель сервиса, который проверяет канал на наличие сообщений через равные промежутки времени.</p> <p>Selective Consumer (Избирательный потребитель). Потребитель сервиса, использующий фильтры для получения только сообщений, соответствующих определенному критерию.</p> <p>Service Activator (Активатор сервиса). Сервис, принимающий асинхронные запросы для вызова операций в компонентах бизнес-слоя.</p> <p>Transactional Client (Транзакционный клиент). Клиент, который может реализовать транзакции при взаимодействии с сервисом.</p>
<i>Безопасность сообщений</i>	Data Confidentiality (Конфиденциальность данных). Использует шифрование на уровне сообщений для защиты конфиденциальных

	<p>данных в сообщении.</p> <p>Data Integrity (Целостность данных). Гарантирует защиту сообщений от повреждения или подделки при передаче.</p> <p>Data Origin Authentication (Аутентификация происхождения данных). Проводит проверку источника сообщения как расширенный метод обеспечения целостности данных.</p> <p>Exception Shielding (Экранирование исключений). При возникновении исключения предотвращает разглашение сервисом данных о его внутренней реализации.</p> <p>Federation (Объединение). Интегрированное представление данных, распределенных по многим сервисам и потребителям.</p> <p>Replay Protection (Защита от атак повторов). Обеспечивает идемпотентность сообщения, предотвращая возможность его перехвата и многократного выполнения злоумышленниками.</p> <p>Validation (Валидация). Проверяет содержимое и значения сообщений для обеспечения защиты сервиса от неправильно сформированного или злонамеренного содержимого.</p>
Маршрутизация сообщения	<p>Aggregator (Агрегатор). Фильтр, обеспечивающий сбор и сохранение взаимосвязанных сообщений, объединение этих сообщений и публикацию в выходном канале одного агрегатного сообщения для дальнейшей обработки.</p> <p>Content-Based Router (Маршрутизатор на основе содержимого). Выполняет маршрутизацию каждого сообщения к соответствующему потребителю, исходя из содержимого сообщения, например, наличия определенных полей, заданных значений полей и т.д.</p> <p>Dynamic Router (Динамический маршрутизатор). Компонент, выполняющий динамическую маршрутизацию сообщения к потребителю на основании оценки условий/правил, заданных потребителем.</p> <p>Message Broker (Hub and Spoke) (Брокер сообщений (веерная структура)). Центральный компонент, взаимодействующий с множеством приложений для получения сообщений из многих источников; определяет место назначения сообщения и направляет его в соответствующий канал.</p> <p>Message Filter (Фильтр сообщений). На основании заданных критериев предотвращает передачу по каналу потребителю нежелательных сообщений.</p> <p>Process Manager (Диспетчер процесса). Компонент, обеспечивающий маршрутизацию сообщений через множество этапов рабочего процесса.</p>
Преобразование сообщения	<p>Canonical Data Mapper (Канонический преобразователь данных). Использует общий формат данных для осуществления преобразований между двумя несопоставимыми форматами данных.</p> <p>Claim Check (Проверка утверждений). Извлекает данные из постоянного хранилища по необходимости.</p> <p>Content Enricher (Расширитель содержимого). Дополняет сообщения недостающими данными, полученными из внешнего источника данных.</p> <p>Content Filter (Фильтр содержимого). Удаляет конфиденциальные данные из сообщения и максимально сокращает объем сетевого трафика, удаляя ненужные данные из сообщения.</p> <p>Envelope Wrapper (Оболочка конверта). Оболочка сообщений, включающая данные заголовка, используемые, например, для защиты,</p>

	маршрутизации или аутентификации сообщения. Normalizer (Нормализатор). Преобразует или трансформирует данные в общий формат обмена, если организации используют разные форматы.
REST	Behavior (Поведение). Применяется к ресурсам, выполняющим операции. Обычно такие ресурсы не содержат состояния и поддерживают только операцию POST. Container (Контейнер). Создает шаблон сущности, обеспечивая средства для динамического добавления и/или обновления вложенных ресурсов. Entity (Сущность). Ресурсы, чтение которых может быть осуществлено операцией GET, но изменение возможно только с помощью операций PUT и DELETE. Store (Хранилище). Обеспечивает возможность создания и обновления сущностей с помощью операции PUT. Transaction (Транзакция). Ресурсы, поддерживающие транзакционные операции.
Интерфейс сервиса	Façade (Фасад). Реализует унифицированный интерфейс для набора операций, чтобы обеспечить упрощенный интерфейс и уменьшить связанность систем. Remote Façade (Удаленный фасад). Создает обобщенный унифицированный интерфейс для набора операций или процессов в удаленной подсистеме, обеспечивая обобщенный интерфейс для детализированных операций, чтобы упростить использование этой подсистемы и свести до минимума вызовы по сети. Service Interface (Интерфейс сервиса). Программный интерфейс, который может использоваться другими системами для взаимодействия с сервисом.
SOAP	Data Contract (Контракт данных). Схема, определяющая структуры данных, передаваемые с запросом к сервису. Fault Contracts (Контракты сбоев). Схема, определяющая ошибки или сбои, которые могут быть возвращены из запроса к сервису. Service Contract (Контракт сервиса). Схема, определяющая операции, которые может осуществлять сервис.

Более подробно шаблоны Duplex и Request Response рассматриваются в статье «*Designing Service Contracts*» (Проектирование контрактов сервисов) по адресу <http://msdn.microsoft.com/en-us/library/ms733070.aspx>.

Более подробно шаблон Request-Reply рассматривается в статье «*Request-Reply*» (Запрос-ответ) по адресу <http://www.eaipatterns.com/RequestReply.html>.

Более подробно шаблоны Command, Document Message, Event Message, Durable Subscriber, Idempotent Receiver, Polling Consumer и Transactional Client рассматриваются в статье «*Messaging Patterns in Service-Oriented Architecture, Part I*» (Шаблоны обмена сообщениями в сервисно-ориентированной архитектуре. Часть I) по адресу <http://msdn.microsoft.com/en-us/library/aa480027.aspx>.

Более подробно шаблоны Data Confidentiality и Data Origin Authentication рассматриваются в материале «*Chapter 2: Message Protection Patterns*» (Глава 2: Механизмы защиты сообщений) по адресу <http://msdn.microsoft.com/en-us/library/aa480573.aspx>.

Более подробно шаблоны Replay Detection, Exception Shielding и Validation рассматриваются в материале «*Chapter 5: Service Boundary Protection Patterns*» (Глава 5: Механизмы защиты границ сервиса) по адресу <http://msdn.microsoft.com/en-us/library/aa480597.aspx>.

Более подробно шаблоны Claim Check, Content Enricher, Content Filter и Envelope Wrapper рассматриваются в материале «*Messaging Patterns in Service Oriented Architecture, Part 2*» по адресу <http://msdn.microsoft.com/en-us/library/aa480061.aspx>.

Более подробно шаблон Remote Façade рассматриваются в статье «*Patterns of Enterprise Application Architecture: Remote Façade*» (Архитектура корпоративных программных приложений) по адресу <http://martinfowler.com/eaaCatalog/remoteFacade.html>.

Более подробно шаблоны REST, такие как Behavior, Container и Entity, рассматриваются в статье «*REST Patterns*» (Шаблоны REST) по адресу http://wiki.developer.mindtouch.com/REST/REST_Patterns.

Более подробно шаблоны Aggregator, Content-Based Router, Publish-Subscribe, Message Bus и Point-to-Point рассматриваются в материале «*Messaging patterns in Service-Oriented Architecture, Part 1*» по адресу <http://msdn.microsoft.com/en-us/library/aa480027.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Enterprise Solution Patterns Using Microsoft .NET*» по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.
- «*Web Service Security Guidance*» (Руководство по безопасности Веб-сервисов) по адресу <http://msdn.microsoft.com/en-us/library/aa480545.aspx>.
- «*Improving Web Services Security: Scenarios and Implementation Guidance for WCF*» по адресу <http://www.codeplex.com/WCFSecurityGuide>.
- «*WS-* Specifications*» (Спецификации WS-*) по адресу <http://www.ws-standards.com/ws-atomictransaction.asp>.

10

Рекомендации по проектированию компонентов

Обзор

Компоненты являются средством изоляции определенных наборов функций в элементах, которые могут распространяться и устанавливаться отдельно от другой функциональности. Данная глава содержит общие рекомендации по созданию компонентов и описывает типы компонентов, обычно применяемые в слоях приложений, проектируемых с использованием многослойного подхода, обсуждаемого в этом руководстве. Хотя, методики построения компонентов обычно не зависят от структуры приложения.

Общие рекомендации по проектированию компонентов

Рассмотрим общие рекомендации проектирования компонентов приложений:

- **Применяйте принципы SOLID при проектировании классов, входящих в компонент.** Принципы SOLID – это:
 - **Принцип единственности ответственности (Single responsibility).** Класс должен отвечать только за один аспект.
 - **Принцип открытости/закрытости (Open/closed principle).** Классы должны быть расширяемыми без необходимости доработки.
 - **Принцип замещения Лискова (Liskov substitution principle).** Подтипы и базовые типы должны быть взаимозаменяемы.
 - **Принцип отделения интерфейса (Interface segregation principle).** Интерфейсы классов должны быть клиент-специфическими и узконаправленными. Классы должны предоставлять разные интерфейсы для клиентов, имеющих разные требования к интерфейсам.
 - **Принцип инверсии зависимостей (Dependency inversion principle).** Зависимости между классами должны заменяться абстракциями, что обеспечит возможность проектирования сверху вниз без необходимости проектирования сначала модулей нижнего уровня. Абстракции не должны зависеть от деталей – детали должны зависеть от абстракций.

- **Проектируйте сильно связные компоненты.** Не перегружайте компоненты введением в них невзаимосвязанной или смешанной функциональности. Например, всегда избегайте смешения в компонентах бизнес-слоя логики доступа к данным и бизнес-логики. Обеспечив связность функциональности, можно создавать сборки, включающие более одного компонента, и устанавливать компоненты в соответствующих слоях приложения, даже если эти слои разделены физически.
- **Компонент не должен зависеть от внутренних деталей других компонентов.** Каждый компонент или объект должен вызывать метод другого объекта или компонента, и этот метод должен знать, как обрабатывать запрос и, если необходимо, как направить его к соответствующим подкомпонентам или другим компонентам. Такой подход позволяет создавать более адаптируемые и удобные в обслуживании приложения.
- **Продумайте, как компоненты будут взаимодействовать друг с другом.** Для этого требуется понимать, какие сценарии развертывания должно поддерживать создаваемое приложение, должно ли оно поддерживать взаимодействие через физические границы или границы процесса, либо все компоненты будут выполняться в одном процессе.
- **Не смешивайте код сквозной функциональности и прикладную логику приложения.** Код, реализующий сквозную функциональность – это код, связанный с безопасностью, связью или управлением, таким как протоколированием и инструментированием. Смешение кода, реализующего эти функции, с логикой компонентов может привести к созданию плохо расширяемого и сложного в обслуживании дизайна.
- **Применяйте основные принципы компонентного архитектурного стиля.** Эти принципы состоят в том, что компоненты должны быть пригодными для повторного использования, заменяемыми, расширяемыми, инкапсулированными, независимыми и не зависеть от контекста. Компонентный архитектурный стиль рассматривается в главе 3, «[Архитектурные шаблоны и стили](#)».

Распределение компонентов по слоям

Каждый слой приложения содержит наборы компонентов, реализующих функциональность данного слоя. Эти компоненты должны быть связными и слабо связанными, чтобы обеспечить возможность повторного использования и упростить обслуживание. На рис. 1 показано, какие типы компонентов обычно используются в каждом из слоев.

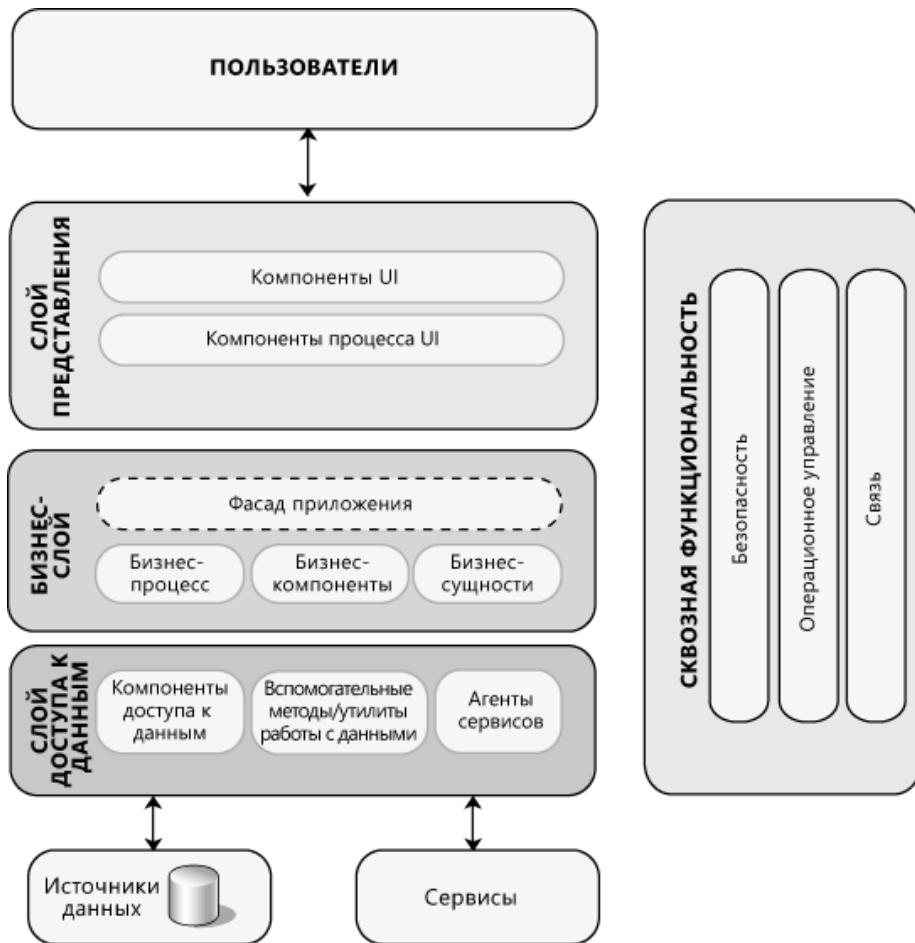


Рис. 1

Типы компонентов, обычно используемые в каждом из слоев

Следующие разделы посвящены рассмотрению компонентов, представленных на рис. 1.

Компоненты слоя представления

Компоненты слоя представления реализуют функциональность, необходимую для обеспечения взаимодействия пользователей с приложением. Обычно в слое представления располагаются следующие типы компонентов:

- **Компоненты пользовательского интерфейса.** Конкретная реализация пользовательского интерфейса приложения инкапсулирована в компоненты пользовательского интерфейса (UI). Это визуальные элементы приложения, используемые для отображения данных пользователю и приема пользовательского ввода. Компоненты UI, спроектированные для реализации шаблона Separated Presentation, иногда называют Представлениями (Views). В большинстве случаев их роль заключается в предоставлении пользователю интерфейса, который обеспечивает наиболее соответствующее представление данных и логики приложения, а также в интерпретации пользовательского ввода и передаче его в компоненты логики представления, которые определяют влияние ввода на данные и состояние приложения. В некоторых случаях в компонентах пользовательского интерфейса может содержаться специальная логика реализации пользовательского

интерфейса, однако, как правило, они включают минимальный объем логики приложения, поскольку это может негативно сказаться на удобстве обслуживания и возможности повторного использования, а также усложнить модульное тестирование.

- **Компоненты логики представления.** Логика представления – это код приложения, определяющий поведение и структуру приложения таким образом, что они не зависят от какой-либо конкретной реализации пользовательского интерфейса. Компоненты логики представления, главным образом, обеспечивают реализацию вариантов использования приложения (или пользовательских историй) и координируют взаимодействия пользователя с базовой логикой и состоянием приложения независимо от UI. Также они отвечают за организацию поступающих с бизнес-слоя данных в формат, пригодный для потребления компонентами UI. Например, они могут агрегировать данные из многих источников и преобразовывать их для большего удобства отображения. Компоненты логики представления можно подразделить на две категории:
 - **Компоненты Presenter, Controller, Presentation Model и ViewModel.** Данные типы компонентов используются при реализации шаблона Separated Presentation и часто инкапсулируют логику представления слоя представления. Чтобы обеспечить максимальные возможности повторного использования и удобство тестирования, эти компоненты не привязаны ни к одному конкретному классу, элементу или элементу управления UI.
 - **Компоненты сущностей представления.** Эти компоненты инкапсулируют бизнес-логику и данные и упрощают их потребление пользовательским интерфейсом и компонентами логики представления, например, путем преобразования типов данных или агрегации данных из нескольких источников. В некоторых случаях, это бизнес-сущности бизнес-слоя, используемые напрямую слоем представления. В других случаях, они могут представлять подмножество компонентов бизнес-сущностей и создаваться специально для поддержки слоя представления приложения. Сущности представления помогают обеспечить непротиворечивость и действительность данных в слое представления. В некоторых шаблонах раздельного представления эти компоненты называют моделями.

Более подробно о проектировании слоя представления рассказывает глава 6, «[Рекомендации по проектированию слоя представления](#)». Проектированию компонентов представления посвящена глава 11, «[Проектирование компонентов представления](#)».

Компоненты слоя сервисов

Приложение может предоставлять слой сервисов для взаимодействия с клиентами или использования другими системами. Компоненты слоя сервисов обеспечивают другим клиентам и приложениям способ доступа к бизнес-логике приложения и используют функциональность приложения путем обмена сообщениями по каналу связи. Обычно в слое сервисов располагаются следующие типы компонентов:

- **Интерфейсы сервисов.** Сервисы предоставляют интерфейс сервисов, в который передаются все входящие сообщения. Описание набора сообщений, которыми необходимо обмениваться с сервисом для осуществления им определенной бизнес-задачи, называется контрактом. Интерфейс сервиса можно рассматривать как фасад, предоставляющий потенциальным потребителям бизнес-логику, реализованную в приложении (как правило, это логика бизнес-слоя).
- **Типы сообщений.** При обмене данными в слое сервисов структуры данных заключены в структуры сообщений, поддерживающие разные типы операций. Например, существуют такие типы сообщений, как Command (Команда), Document (Документ) и другие. Типы сообщений – это контракты сообщений, используемых для взаимодействия потребителей и провайдеров сервиса. Также слой сервисов обычно предоставляет типы данных и контракты, которые определяют типы данных, используемые в сообщениях, и изолируют внутренние типы данных от данных, содержащихся в типе сообщения. Это предотвращает раскрытие внутренних типов данных внешним потребителям, что могло бы привести к сложностям с контролем версий интерфейса.

Более подробно взаимодействие и обмен сообщениями рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Компоненты бизнес-слоя

Компоненты бизнес-слоя реализуют основную функциональность системы и инкапсулируют соответствующую бизнес-логику. Бизнес-слой обычно включает следующие типы компонентов:

- **Фасад приложения.** Этот необязательный компонент обычно обеспечивает упрощенный интерфейс для компонентов бизнес-логики зачастую путем объединения множества бизнес-операций в одну, что упрощает использование бизнес-логики и сокращает количество зависимостей, поскольку внешним вызывающим сторонам нет необходимости знать детали бизнес-компонентов и отношения между ними.
- **Компоненты бизнес-логики.** Бизнес-логика – это логика приложения, связанная с извлечением, обработкой, преобразованием и управлением данными приложения; применением бизнес-правил и политик и обеспечением непротиворечивости и действительности данных. Чтобы обеспечить наилучшие условия для повторного использования, компоненты бизнес-логики не должны включать поведение или логику приложения, относящиеся к конкретному варианту использования или пользовательской истории. Компоненты бизнес-логики можно разделить на следующие две категории:
 - **Компоненты рабочего процесса.** После того как данные введены в компоненты UI и переданы в бизнес-слой, приложение может использовать их для выполнения бизнес-процесса. Многие бизнес-процессы состоят из множества этапов, которые должны осуществляться в соответствующем порядке и могут

взаимодействовать друг с другом посредством механизмов координаирования. Компоненты рабочего-процесса определяют и управляют длительными многоэтапными бизнес-процессами и могут быть реализованы с использованием инструментов управления бизнес-процессами. Компоненты рабочего процесса работают с компонентами бизнес-процесса, которые создают экземпляры компонентов рабочего процесса и осуществляют операции с ними. Более подробно компоненты рабочего процесса рассматриваются в главе 14, «[Проектирование компонентов рабочего процесса](#)».

- **Компоненты бизнес-сущностей.** Бизнес-сущности, или, более обобщенно, бизнес-объекты, инкапсулируют бизнес-логику и данные, необходимые для представления в приложении элементов реального мира, таких как заказчики (Customers) или заказы (Orders). Они сохраняют значения данных и предоставляют их через свойства; содержат и управляют бизнес-данными, которые используются приложением; и обеспечивают программный доступ с сохранением состояния к бизнес-данным и соответствующей функциональности. Также бизнес-сущности проводят проверку содержащихся в них данных и инкапсулируют бизнес-логику для обеспечения непротиворечивости данных и реализации бизнес-правил и поведения. Более подробно компоненты бизнес-сущностей рассматриваются в главе 13, «[Проектирование бизнес-сущностей](#)».

Очень часто бизнес-сущности должны быть доступными компонентам и сервисам как бизнес-слоя, так и слоя данных. Например, бизнес-сущности могут сопоставляться с источником данных, и к ним могут выполнять доступ бизнес-компоненты. Если слои располагаются на одном уровне, бизнес-сущности могут использоваться совместно непосредственно через указатели. Однако при этом все равно должно быть обеспечено разделение бизнес-логики и логики доступа к данным. Этого можно достичь путем перемещения бизнес-сущностей в отдельную сборку, доступную для использования сборками и бизнес-сервисов, и сервисов данных. Этот подход аналогичен использованию шаблона инверсии зависимостей, когда бизнес-сущности отделяются от бизнес-слоя и слоя данных, и их зависимость от бизнес-сущностей реализуется как совместно используемый контракт.

Более подробно проектирование бизнес-слоя рассматривается в главе 7, «[Рекомендации по проектированию бизнес-слоя](#)». О проектировании компонентов бизнес-слоя рассказывает глава 12, «[Проектирование компонентов бизнес-слоя](#)». Проектированию компонентов бизнес-сущностей посвящена глава 13, «[Проектирование бизнес-сущностей](#)». Проектирование компонентов рабочего процесса более подробно обсуждается в главе 14, «[Проектирование компонентов рабочего процесса](#)».

Компоненты слоя доступа к данным

Компоненты слоя доступа к данным обеспечивают доступ к данным, размещенным в рамках системы, и к данным, предоставляемым другими сетевыми системами. Обычно слой доступа к данным включает следующие типы компонентов:

- **Компоненты доступа к данным.** Эти компоненты абстрагируют логику, необходимую для доступа к базовым хранилищам данных. Для большинства задач доступа к данным необходима общая логика, которая может быть выделена и реализована в отдельных вспомогательных компонентах, доступных для повторного использования, или подходящей вспомогательной инфраструктуре. Это может упростить компоненты доступа к данным и централизовать логику, что облегчает обслуживание. Остальные задачи, общие для компонентов слоя данных и не относящиеся ни к одному набору компонентов, могут быть реализованы как отдельные служебные компоненты. Вспомогательные и служебные компоненты часто объединяются в библиотеку или инфраструктуру, что облегчает их повторное использование в других приложениях.
- **Агенты сервисов.** Если бизнес-компонент должен использовать функциональность, предоставляемую внешним сервисом, вероятно, потребуется реализовать код для управления семантикой взаимодействия с конкретным сервисом. Агенты сервисов изолируют специальные аспекты вызова разных сервисов в приложении и могут обеспечивать дополнительные сервисы, такие как кэширование, поддержка работы в автономном режиме и базовое сопоставление форматов данных, предоставляемых сервисом, и форматов, требуемых приложением.

Более подробно проектирование слоя доступа к данным рассматривается в главе 8, «[Рекомендации по проектированию слоя доступа к данным](#)». Проектированию компонентов слоя доступа к данным посвящена глава 15, «[Проектирование компонентов слоя доступа к данным](#)».

Компоненты сквозной функциональности

Некоторые задачи необходимо выполнять во многих слоях. Компоненты сквозной функциональности реализуют специальные типы функциональности, доступ к которым могут осуществлять компоненты любого слоя. Рассмотрим основные типы компонентов сквозной функциональности:

- **Компоненты для реализации безопасности.** Сюда относятся компоненты, осуществляющие аутентификацию, авторизацию и валидацию.
- **Компоненты для реализации задач операционного управления.** Сюда относятся компоненты, реализующие политики обработки исключений, протоколирование, счетчики производительности, конфигурацию и трассировку.
- **Компоненты для реализации взаимодействия.** Сюда относятся компоненты, взаимодействующие с другими сервисами и приложениями.

Реализации сквозной функциональности посвящена глава 17, «[Сквозная функциональность](#)».

Шаблоны проектирования

Основные шаблоны для компонентов организованы по категориям и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны проектирования
Бизнес-компоненты	Application Façade (Фасад приложения). Централизует и агрегирует поведение для обеспечения унифицированного слоя сервисов. Chain of Responsibility (Цепочка обязанностей). Предоставляя возможность обработать запрос нескольким объектам, устраниет возможность связывания отправителя запроса с его получателем. Command (Команда). Инкапсулирует обработку запроса в отдельный командный объект с общим интерфейсом выполнения.
Бизнес-сущности	Domain Model (Модель предметной области). Набор бизнес-объектов, представляющих сущности предметной области и отношения между ними. Entity Translator (Транслятор сущностей). Объект, преобразующий типы данных сообщения в бизнес-типы для запросов и выполняющий обратные преобразования для ответов. Table Module (Модуль таблицы). Единый компонент, реализующий бизнес-логику для всех строк таблицы или представления базы данных.
Сущности представления	Entity Translator (Транслятор сущностей). Объект, преобразующий типы данных сообщения в бизнес-типы для запросов и выполняющий обратные преобразования для ответов.
Логика представления	Application Controller (Контроллер приложений). Реализует централизованную точку обработки навигации по экрану и потока приложения. Model-View-Controller (Модель-Представление-Контроллер). Разделяет код UI на три модуля: Модель (данные), Представление (интерфейс) и Контроллер (логика обработки), уделяя основное внимание Представлению. Существует две разновидности этого шаблона: Passive View (Пассивное представление) и Supervising Presenter (Наблюдающий презентатор), которые определяют взаимодействие Представления с Моделью. Model-View-ViewModel (Модель-Представление-Модель представления). Разновидность шаблона MVC, в которой взаимодействие Представления и Модели представления реализуются с помощью шаблона Command. Model-View-Presenter (Модель-Представление-Презентатор). Разделяет обработку запроса на три отдельные роли, при этом Представление отвечает за обработку пользовательского ввода и передачу управления объекту Презентатора. Passive View (Пассивное представление). Сокращает представление до необходимого минимума, перенося в контроллер функциональность обработки пользовательского ввода и реализацию обновления представления. Presentation Model (Модель презентации). Выносит всю логику представления и состояние из представления и реализует

	<p>формирование визуального представления через связывание данных и шаблоны.</p> <p>Supervising Presenter (или Supervising Controller) (Наблюдающий презентатор или Наблюдающий контроллер). Разновидность шаблона MVC, где контроллер отвечает за обработку сложной логики, в частности, согласование представлений, а представление отвечает за простую логику, касающуюся представления.</p>
<i>Интерфейс сервиса</i>	<p>Façade (Фасад). Реализует унифицированный интерфейс для набора операций, чтобы обеспечить упрощенный интерфейс и уменьшить связанность систем.</p> <p>Remote Façade (Удаленный фасад). Создает обобщенный унифицированный интерфейс для набора операций или процессов в удаленной подсистеме, обеспечивая слабо детализированный интерфейс для детализированных операций, чтобы упростить использование этой подсистемы и свести до минимума вызовы по сети.</p> <p>Service Interface (Интерфейс сервиса). Программный интерфейс, который может использоваться другими системами для взаимодействия с сервисом.</p>
<i>Рабочие процессы</i>	<p>Data-Driven Workflow (Управляемый данными рабочий процесс). Рабочий процесс, включающий задачи, последовательность выполнения которых определяется значениями данных в рабочем процессе или системе.</p> <p>Human Workflow (Рабочий процесс оператора). Рабочий процесс, включающий задачи, выполняемые вручную.</p> <p>Sequential Workflow (Последовательный рабочий процесс). Рабочий процесс, включающий задачи, выполняющиеся в определенной последовательности, когда выполнение одной задачи запускается только после завершения предыдущей.</p> <p>State-Driven Workflow (Управляемый состоянием рабочий процесс). Рабочий процесс, включающий задачи, последовательность выполнения которых определяется состоянием системы.</p>

Более подробно шаблон Façade рассматривается в главе 4, «Структурные шаблоны», книги Эрика Гамма, Ричарда Хельма, Ральфа Джонсона и Джона Влиссисда «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2006.

Более подробно шаблон Chain of Responsibility рассматривается в статье «*Patterns in Practice*» по адресу <http://msdn.microsoft.com/en-us/magazine/cc546578.aspx>.

Более подробно шаблон Command рассматривается в главе 5, «Поведенческие шаблоны», книги Эрика Гамма, Ричарда Хельма, Ральфа Джонсона и Джона Влиссисда «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2006.

Более подробно шаблон Entity Translator рассматривается в статье «*Useful Patterns for Services*» по адресу <http://msdn.microsoft.com/en-us/library/cc304800.aspx>.

Более подробно шаблоны Data-Driven Workflow, Human Workflow, Sequential Workflow и State-Driven Workflow рассматриваются в статье «*Windows Workflow Foundation Overview*» (Обзор

Windows Workflow Foundation) по адресу <http://msdn.microsoft.com/en-us/library/ms734631.aspx> и "Workflow Patterns" по адресу <http://www.workflowpatterns.com/>.

Более подробно шаблоны Application Controller и Model-View-Controller (MVC) рассматриваются в книге Мартина Фаулера «Архитектура корпоративных программных приложений». Вильямс, 2007. Или по адресу <http://martinfowler.com/eaaCatalog>.

Более подробно шаблоны Supervising Presenter и Presentation Model рассматриваются в статье «*Patterns in the Composite Application Library*» по адресу <http://msdn.microsoft.com/en-us/library/dd458924.aspx>.

Более подробно шаблон Remote Façade рассматривается в статье «*P of EAA: Remote Façade*» по адресу <http://martinfowler.com/eaaCatalog/remoteFacade.html>.

Предложения группы patterns & practices

Узнать о дополнительных предложениях группы Microsoft patterns & practices можно из следующих источников:

- «*Composite Client Application Guidance*» по адресу <http://msdn.microsoft.com/en-us/library/cc707819.aspx>.
- «*Enterprise Library*» по адресу <http://msdn.microsoft.com/en-us/library/cc467894.aspx>.
- «*Smart Client Software Factory*» по адресу <http://msdn.microsoft.com/en-us/library/aa480482.aspx>.
- «*Unity*» (механизм внесения зависимостей) по адресу <http://msdn.microsoft.com/en-us/library/dd203101.aspx>.
- «*Web Client Software Factory*» по адресу <http://msdn.microsoft.com/en-us/library/bb264518.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Integration Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/ms978729.aspx>.
- Martin, Robert C. and Micah Martin. *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, 2006.
- «*User Interface Control Guidelines*» по адресу <http://msdn.microsoft.com/en-us/library/bb158625.aspx>.

11

Проектирование компонентов представления

Обзор

Данная глава описывает этапы проектирования компонентов пользовательского интерфейса и компонентов логики представления, которые являются частью слоя представления. Некоторые из шагов касаются проектирования самого слоя представления, тогда как другие больше посвящены отдельным типам компонентов, которые могут использоваться.

Прежде всего, требуется понять, какие требования предъявляются к UI, и суметь выбрать соответствующую технологию. После этого уже можно принимать решения о связывании логики представления и данных с элементами управления UI. Также необходимо иметь четкое представление о требованиях к обработке ошибок и проверке в UI. В следующих разделах этой главы более подробно рассматриваются шаги по проектированию компонентов представления.

Шаг 1 – Понимание предъявляемых к UI требований

Понимание предъявляемых к UI требований – ключ к принятию решений по типу UI, технологии и типу элементов управления, используемым для его реализации. Требования к UI определяются функциональностью, которую должно поддерживать приложение, и ожиданиями пользователей.

Начните с выяснения, кто будет пользователями приложения, и понимания целей и задачей, которые эти пользователи желают реализовывать при использовании приложения. Особое внимание уделите вопросу последовательности задач или операций; выясните, ожидают ли пользователи структурированного последовательного взаимодействия или неструктурированного взаимодействия с произвольным порядком операций, когда существует возможность выполнения множества задач одновременно. Как часть этого процесса, также выясните, какие данные понадобятся пользователям, и формат, в котором они ожидают их увидеть. Возможно, придется провести исследования, чтобы лучше понять среду, в которой пользователь будет взаимодействовать с приложением. Кроме того, рассмотрите текущие уровни взаимодействия с пользователем и сравните их с требованиями к взаимодействию с пользователем, предъявляемыми для разрабатываемого UI, чтобы убедиться в их логичности и понятности. Все эти факторы помогут создать ориентированный на пользователя дизайн.

Один из факторов, имеющих большое влияние на выбор технологии – требуемая функциональность UI. Выясните, должен ли UI предоставлять насыщенную функциональность или взаимодействие с пользователем, должен ли он обеспечивать минимальное время отклика или требует графической или анимационной поддержки. Также рассмотрите требования с точки зрения локализации к типам данных, форматам и форматам представления для таких данных, как даты, время и валюты. Кроме того, определите требования по персонализации приложения, такие как предоставление пользователю возможности менять компоновку и стили во время выполнения.

Чтобы сделать UI интуитивно понятным и простым в использовании, продумайте компоновку или композицию интерфейса, а также перемещение пользователя по UI приложения. Это поможет выбрать соответствующие элементы управления и технологии для UI. Разберитесь с тем, какие требования физического устройства отображения (такие как размер и разрешение экрана) и специальные возможности (такие как крупный текст или кнопки, рукописный ввод и т.д.) необходимо поддерживать. Примите решение о том, как будете выполнять группировку взаимосвязанных данных в разделах UI, избегать конфликтов или неоднозначностей интерфейса и выделять важные элементы. Обеспечьте пользователям возможность быстро и легко находить сведения в приложении посредством навигационных элементов управления, функций поиска, четко именованных разделов, карт сайта и других соответствующих возможностей.

Шаг 2 – Выбор необходимого типа UI

На основании предъявляемых к UI требований можно принять решение о типе UI для приложения. Существует ряд разных типов UI, каждый из которых обладает определенными преимуществами и недостатками. Часто обнаруживается, что предъявляемым к UI требованиям соответствует несколько типов UI. Но бывают ситуации, когда ни один из типов UI не обеспечивает полностью все требования. В этом случае необходимо рассмотреть возможность создания нескольких разных типов UI, которые будут совместно использовать бизнес-логику. Примером этому может служить приложение для call-центра, некоторые из возможностей которого предоставляются клиенту для самостоятельного использования через Веб и на мобильных устройствах.

Мобильные приложения могут разрабатываться как тонкое клиентское или насыщенное клиентское приложение. Насыщенные клиентские мобильные приложения могут поддерживать сценарии без подключения или с периодическим подключением. Веб- или тонкие клиентские мобильные приложения поддерживают только сценарии с подключением. Ограничением при проектировании мобильных приложений могут быть и аппаратные ресурсы.

Насыщенные клиентские приложения обычно являются автономными или сетевыми приложениями с графическим пользовательским интерфейсом, отображающим данные с помощью различных элементов управления, развертываемыми на настольном или портативном компьютере локального пользователя. Эти приложения подходят для сценариев без подключения или с периодическим подключением, поскольку выполняются на клиентском компьютере. Насыщенное клиентское приложение является хорошим выбором, если требуется высокодинамичный UI с малым временем отклика или UI должен обеспечивать

насыщенные функциональность и взаимодействие с пользователем; либо если приложение должно поддерживать как сценарии с подключением, так и сценарии без подключения, использовать ресурсы локальной системы на клиентском компьютере или интегрироваться с другими приложениями на этом компьютере.

Обычно **насыщенные Интернет-приложения** (RIA) – это Веб-приложения с насыщенным графическим пользовательским интерфейсом, выполняющиеся в браузере. Как правило, приложения RIA используются в сценариях с подключением. Используйте RIA, если необходимо обеспечить UI, поддерживающий динамическое взаимодействие с пользователем с малым временем отклика или использующий потоковое мультимедиа и доступный на широком диапазоне устройств и платформ. Эти приложения могут использовать вычислительные мощности клиентского компьютера, но не могут напрямую взаимодействовать с локальными ресурсами системы, такими как веб-камеры¹, или с другими клиентскими приложениями, такими как приложения Microsoft Office.

Веб-приложения поддерживают сценарии с постоянным подключением и могут поддерживать множество разных браузеров, выполняющихся под управлением множества различных операционных систем и на разных платформах. Веб-приложение – замечательный выбор, если UI должен быть стандартизованным, доступным для широчайшего диапазона устройств и платформ и работать только при постоянном подключении к сети. Также Веб-приложения хорошо подходят, если необходимо обеспечить доступность содержимого приложения для поиска средствами поиска в Веб.

Консольные приложения предлагают альтернативный текстовый пользовательский интерфейс и обычно выполняются в командных оболочках, таких как Command window (Интерфейс командной строки) или Power Shell. Такие приложения лучше всего подходят для задач администрирования или разработки и не используются как часть многослойного дизайна приложений.

Шаг 3 – Выбор технологии UI

После того, как вы определились с типом UI для своих компонентов UI, необходимо выбрать соответствующую технологию. Как правило, выбор зависит от выбранного типа UI. Далее рассмотрим, какие технологии подходят для каждого из типов UI:

Пользовательские интерфейсы **мобильных клиентов** могут быть реализованы с использованием следующих технологий создания пользовательского интерфейса:

- **Microsoft .NET Compact Framework.** Это версия Microsoft .NET Framework, разработанная специально для мобильных устройств. Используйте эту технологию для мобильных приложений, которые должны выполняться на устройствах без гарантированного подключения к сети.
- **ASP.NET для мобильных устройств.** Это версия ASP.NET, разработанная специально для мобильных устройств. ASP.NET для мобильных приложений может размещаться

¹ Начиная с Silverlight 4.0, RIA приложения могут работать с камерой и микрофоном (прим. научного редактора).

на сервере Internet Information Services (IIS)¹. Используйте эту технологию для мобильных Веб-приложений, если требуется поддерживать широкий диапазон мобильных устройств и браузеров и можно рассчитывать на постоянное подключение к сети.

- **Silverlight для мобильных устройств.** Для этой версии Silverlight-клиента требуется, чтобы на мобильном устройстве был установлен подключаемый модуль Silverlight. Используйте эту технологию, чтобы портировать существующие Silverlight-приложения на мобильные устройства, или если желаете создавать более насыщенные UI, чем обеспечивают другие технологии. (На момент написания данного руководства эта технология заявлена, но еще не выпущена).

Пользовательские интерфейсы **насыщенных клиентов** могут быть реализованы с использованием следующих технологий представления:

- **Windows Presentation Foundation (WPF).** Приложения WPF поддерживают более широкие графические возможности, такие как 2-D и 3-D графика, независимость от разрешения экрана, расширенная поддержка документов и полиграфического оформления, анимация с временной шкалой, потоковое аудио и видео и векторная графика. WPF использует расширяемый язык разметки приложений (Extensible Application Markup Language, XAML) для описания UI, связывания данных и событий. WPF также включает расширенные возможности связывания данных и описания шаблонов. WPF-приложения отделяют визуальные аспекты UI от базовой логики управления, обеспечивая тем самым поддержку совместной работы разработчика и дизайнера: разработчики могут сосредоточиться на бизнес-логике, тогда как дизайнеры занимаются внешним видом. Используйте эту технологию для создания насыщенных медиа и интерактивных пользовательских интерфейсов.
- **Windows Forms.** Windows Forms является частью .NET Framework с момента ее появления и идеально подходит для бизнес-приложений. Даже несмотря на существование возможностей Windows Presentation Foundation (WPF), Windows Forms будет идеальным решением для приложений, к которым не предъявляются никакие особые требования по насыщенным медиа или интерактивным возможностям, или если группа разработки уже имеет богатый опыт работы с Windows Forms.
- **Windows Forms с элементами WPF.** Этот подход позволяет использовать преимущества, предоставляемые элементами управления WPF, для создания более мощных UI. WPF можно добавлять в существующее приложение Windows Forms, возможно, как этап постепенного перехода к реализации полностью на WPF. Используйте этот подход для введения насыщенных медиа и интерактивных возможностей в существующие приложения, но не забывайте, что элементы управления WPF лучше всего работают на мощных клиентских компьютерах.

¹ Информационные Интернет-службы (прим. переводчика).

- **WPF с элементами Windows Forms.** Этот подход позволяет дополнить WPF элементами управления Windows Forms, предоставляющими функциональность, не обеспечиваемую WPF. Добавлять элементы управления Windows Forms в UI можно с помощью элемента **WindowsFormsHost** из сборки **WindowsFormsIntegration**. Используйте этот подход, если в пользовательском интерфейсе WPF необходимы элементы управления Windows Forms, но не забывайте о некоторых ограничениях и сложностях, связанных с перекрытием элементов управления, фокусом интерфейса и методиками формирования визуального представления, используемыми разными технологиями.
- **XAML Browser Application (XBAP), использующее WPF.** Данная технология позволяет размещать WPF-приложение в изолированной программной среде в Microsoft Internet Explorer или Mozilla Firefox для Windows. В отличие от Silverlight, в данном случае, доступны все возможности инфраструктуры WPF лишь с некоторыми ограничениями относительно доступа к системным ресурсам из частично доверяемой изолированной программной среды. XBAP требует, чтобы на клиентском компьютере была установлена Windows Vista или .NET Framework 3.5 и подключаемый модуль браузера XBAP. Применяйте XBAP, если имеете готовое WPF-приложение, которое требуется развернуть в Веб, или если хотите использовать насыщенные возможности WPF для создания визуального представления и UI, недоступные в Silverlight.

Пользовательские интерфейсы **насыщенных Интернет-приложений** могут быть реализованы с использованием следующих технологий представления:

- **Silverlight.** Это оптимизированный для работы в браузере аналог WPF, не зависящий от платформы и браузера. По сравнению с XBAP, Silverlight меньше и быстрее устанавливается. Благодаря своему небольшому размеру и поддержке разных платформ Silverlight является хорошим выбором для графических приложений, не требующих полной поддержки графических возможностей WPF¹, или в случае, когда необходимо избежать установки приложения на клиенте.
- **Silverlight с AJAX.** Silverlight поддерживает Асинхронный JavaScript и XML (Asynchronous JavaScript and XML, AJAX) и предоставляет свою объектную модель для доступа из JavaScript, размещенного в Веб-странице. Эту возможность можно использовать для обеспечения взаимодействия между компонентами Веб-страницы и приложением Silverlight.

Пользовательские интерфейсы **Веб-приложений** могут быть реализованы с использованием следующих технологий формирования представления:

- **ASP.NET Web Forms.** Это фундаментальная технология проектирования и реализации UI для Веб-приложений .NET. Приложение ASP.NET Web Forms должно быть установлено только на Веб-сервере, на клиентский компьютер никакие компоненты устанавливать не надо. Используйте эту технологию для Веб-

¹ Например, поддержки 3D-графики (прим. научного редактора).

приложений, в которых не требуются дополнительные возможности, предоставляемые описываемыми в данном разделе AJAX, Silverlight, MVC или Dynamic Data.

- **ASP.NET Web Forms с AJAX.** Применяйте AJAX с ASP.NET Web Forms для асинхронной обработки запросов между сервером и клиентом, что сократит время отклика, обеспечит более насыщенное взаимодействие с пользователем и сократит количество обращений к серверу. AJAX входит в состав ASP.NET в .NET Framework версии 3.5 и последующих.
- **ASP.NET Web Forms с элементами управления Silverlight.** Добавив элементы управления Silverlight в готовое приложение ASP.NET, его можно расширить более насыщенными возможностями создания визуального представления и взаимодействия с пользователем. И это избавит от необходимости создавать совершенно новое Silverlight-приложение. Такой подход хорош для внедрения насыщенного медиа-содержимого Silverlight в существующее Веб-приложение. Элементы управления Silverlight и включающая их Веб-страница могут взаимодействовать на стороне клиента с помощью JavaScript.
- **ASP.NET MVC.** Эта технология позволяет использовать ASP.NET для создания приложений на базе шаблона Model-View-Controller (MVC). Используйте данную технологию, если требуется поддерживать разработку через тестирование и обеспечить четкое разделение функциональности обработки UI и формирования визуального представления UI. Такой подход также поможет получить полный контроль над формированием HTML и избежать смешения данных представления с кодом логики обработки.
- **ASP.NET Dynamic Data.** Эта технология позволяет создавать управляемые данными ASP.NET-приложения, использующие модель данных Language-Integrated Query (LINQ) to Entities. Применяйте ее, если необходима модель быстрой разработки управляемых данными LOB-приложений, основанных на простом формировании шаблонов, но при этом поддерживающих полную настройку.

Пользовательские интерфейсы **консольных приложений** могут быть реализованы с использованием следующих технологий представления:

- **Консольные приложения** – это текстовые приложения, которые могут выполняться в командных оболочках и формировать вывод в стандартную консоль вывода и консоль ошибок. Обычно такие приложения принимают все данные в момент запуска и выполняются без сопровождения.
- **Командлеты Power Shell.** Power Shell – это оболочка командной строки и среда написания сценариев для обеспечения полного управления и автоматизации задач администрирования системы и приложения. Командлеты – это специализированные расширения среды Power Shell, обеспечивающие более глубокую интеграцию с языком Power Shell.

Более подробно перечисленные здесь технологии рассматриваются в приложении В, «[Матрица технологий слоя представления](#)».

Шаг 4 – Проектирование компонентов представления

Следующий шаг после выбора технологии реализации UI – проектирование компонентов UI и компонентов логики представления. Могут использоваться следующие типы компонентов представления:

- [Компоненты пользовательского интерфейса](#)
- [Компоненты логики представления](#)
- [Компоненты модели представления](#)

Эти компоненты поддерживают разделение функциональных областей в рамках слоя представления и часто используются для реализации шаблонов раздельного представления, таких как MVP (Model-View-Presenter) или MVC (Model-View-Controller), через разделение задач обработки UI на три роли: Модель, Представление и Контроллер/Презентатор. Такое разделение функциональности слоя представления повышает удобство обслуживания, тестируемость и возможности повторного использования. Применение абстрактных шаблонов, таких как внедрение зависимостей, также способствует упрощению тестирования логики представления.

Общие принципы проектирования компонентов и компоненты, обычно используемые в разных слоях приложения, более подробно рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)».

Компоненты пользовательского интерфейса

Компоненты UI – это визуальные элементы, отображающие данные пользователю и принимающие пользовательский ввод. В рамках отдельного слоя представления их обычно называют Представлениями (Views). При проектировании компонентов UI руководствуйтесь следующими рекомендациями:

- Разбейте страницы или окна на отдельные пользовательские элементы управления, чтобы упростить и обеспечить возможность повторного использования этих элементов управления. Выбирайте соответствующие компоненты UI и используйте преимущества возможностей связывания данных элементов управления, применяемых в UI.
- Избегайте создания иерархий наследования пользовательских элементов управления и страниц, чтобы обеспечить возможность повторного использования кода. Отдавайте предпочтение композиции, а не наследованию, и создавайте компоненты логики представления, пригодные для повторного использования.
- Создавайте специализированные элементы управления, только если это необходимо для специализированного отображения или сбора данных. Если видите, что имеющиеся требования к UI не могут быть реализованы стандартными элементами управления, прежде чем создавать собственные специализированные

элементы управления, рассмотрите возможность приобретения готового набора элементов управления. При создании специализированных элементов управления старайтесь расширять существующие элементы управления, а не создавать элементы управления с нуля. Расширяйте существующие элементы управления путем подключения к ним поведения, а не наследования от них. Реализуйте поддержку дизайнера для специализированных элементов управления, чтобы разработчикам было проще работать с ними.

Компоненты логики представления

Компоненты логики представления занимаются невизуальными аспектами пользовательского интерфейса, к которым обычно относится проверка, ответ на действия пользователя, взаимодействие компонентов UI и координация взаимодействий с пользователем. Компоненты логики представления необходимы не всегда; создавайте их, только если собираетесь выполнять в слое представления большой объем обработки, которая должна быть отделена от компонентов UI, или если хотите обеспечить более благоприятные условия для модульного тестирования логики представления. При проектировании компонентов логики представления руководствуйтесь следующими рекомендациями:

- Если UI требует сложной обработки или должен обмениваться данными с другими слоями, используйте компоненты логики представления для отделения этой обработки от компонентов UI.
 - Используйте компоненты логики представления для хранения состояния, относящегося к UI, но не характерного для конкретной реализации. Старайтесь не включать в компоненты логики представления бизнес-логику или бизнес-правила, кроме валидации ввода и данных. Также избегайте реализации в компонентах логики представления логики формирования визуального представления UI или специализированной логики UI.
 - С помощью компонентов логики представления обеспечьте согласованное состояние пользовательского интерфейса при восстановлении приложения после сбоя или ошибки.
 - Там, где UI требует поддержки сложного рабочего процесса, создавайте отдельные компоненты рабочего процесса, использующие такую систему управления рабочим процессом, как Windows Workflow Foundation, или реализуйте собственный механизм в бизнес-слое приложения. Более подробно эти вопросы рассматриваются в главе 14, «[Проектирование компонентов рабочего процесса](#)».
-

Компоненты модели представления

Компоненты модели представления представляют данные, поступающие с бизнес-слоя, в формате, доступном для использования UI и компонентами логики представления. Обычно модели представляют данные, и поэтому используют компоненты доступа к данным и, возможно, компоненты бизнес-слоя для сбора этих данных. Если модель также инкапсулирует бизнес-логику, ее обычно называют сущностью представления. Компоненты модели

представления могут, к примеру, агрегировать данные из множества источников, преобразовывать данные для обеспечения удобства их отображения в UI, реализовывать логику проверки и помогать в представлении бизнес-логики и состояния в рамках слоя представления. Обычно они используются для реализации шаблонов раздельного представления, таких как MVP or MVC. При проектировании компонентов модели представления руководствуйтесь следующими рекомендациями:

- Определитесь, нужны ли вам компоненты модели представления. Обычно модели слоя представления используются для отображения специальных данных или форматов слоя представления или в случае применения шаблона раздельного представления, такого как MVP или MVC.
- Работая с элементами управления с привязкой к данным, проектируйте или выбирайте соответствующие компоненты модели представления, которые можно легко связать с элементами управления UI. При использовании в качестве формата компонента модели представления специальных объектов, коллекций или наборов данных обеспечьте реализацию ими соответствующих интерфейсов и событий для поддержки привязки данных.
- При выполнении валидации данных в слое представления размещайте код валидации в компонентах модели представления. Но также продумайте преимущества использования кода или библиотек для централизованной валидации.
- Рассмотрите требования сериализации для данных, передаваемых в компоненты модели представления, если эти данные будут передаваться по сети или сохраняться на жестком диске клиента.

Также необходимо выбрать подходящий тип данных для компонентов модели представления и сущностей представления. Этот выбор определяется требованиями, предъявляемыми к приложению, и ограничениями, налагаемыми инфраструктурой и возможностями разработки. Начните с выбора формата данных слоя представления и примите решение о том, будут ли компоненты также инкапсулировать бизнес-логику и состояние. Далее необходимо принять решение о том, как будут представляться данные в пользовательском интерфейсе. Существуют такие общие форматы представления данных:

- **Собственный класс.** Используйте собственный класс, если необходимо представлять данные как сложный объект, проецируемый непосредственно на бизнес-сущности. Например, можно создать собственный объект Order, который будет представлять данные заказа. Также собственный класс может использоваться для инкапсуляции бизнес-логики и состояния и осуществления проверки в слое представления или реализации собственных свойств.
- **Array (Массив) и Collection (Коллекция).** Используйте массив или коллекцию, если требуется выполнить привязку данных к элементам управления, таким как окно списка или выпадающий список, в которых используются значения одного столбца.

- **DataSet (Набор данных) и DataTable (Таблица данных).** Используйте DataSet или DataTable при работе с простыми табличными данными и элементами управления с привязкой к данным, такими как таблица, окно списка и выпадающий список.
- **Typed Dataset (Типизированный набор данных).** Используйте Typed DataSet, если хотите реализовать тесное связывание с бизнес-сущностями, чтобы избежать возникновения несогласованности из-за изменений базы данных.
- **XML.** Этот формат полезен при работе с Веб-клиентом, когда данные могут быть встроены в Веб-страницу или извлекаться через Веб-сервис или HTTP-запрос. Выбирайте XML при работе с такими элементами управления, как дерево или таблица. Также XML легко сохранять, сериализовать и передавать по каналам связи.
- **DataReader (Модуль чтения данных).** Используйте DataReader в сценариях с постоянным подключением для извлечения данных в режиме только для чтения и только для пересылки. DataReader обеспечивает эффективный способ для последовательной обработки данных, поступающих из базы данных, или для извлечения больших объемов данных. Однако он очень тесно связывает логику со схемой базы данных, что, как правило, не рекомендуется.

Сущности представления

Компоненты модели представления должны по возможности инкапсулировать и данные, поступающие с бизнес-слоя, и бизнес-логику, и поведение. Это позволяет обеспечить непротиворечивость и корректность данных в слое представления и способствует улучшению качества взаимодействия с пользователем.

В некоторых случаях в роли компонентов модели представления могут выступать бизнес-сущности бизнес-слоя, используемые напрямую слоем представления. В других случаях компоненты модели представления могут представлять подмножество компонентов бизнес-сущностей, в частности, разработанных для поддержки слоя представления приложения. Например, в них могут храниться данные в формате, более удобном для использования UI и компонентами логики представления. Такие компоненты иногда называют сущностями представления.

Если бизнес-слой и слой представления располагаются на клиенте, что является типовым сценарием для насыщенных клиентских приложений, бизнес-сущности обычно используются напрямую из бизнес-слоя. Однако если необходимо сохранять или обрабатывать бизнес-данные в формате или способом, отличным от формата и поведения, предоставляемыми бизнес-сущностями бизнес-слоя, можно рассмотреть возможность применения сущностей представления.

Если бизнес-слой и слой представления располагаются на разных уровнях, использование бизнес-сущностей уровнем представления может быть реализовано через их сериализацию и передачу по сети с помощью объектов передачи данных с последующим их восстановлением в виде экземпляров бизнес-сущностей на уровне представления. Или можно восстанавливать данные как сущности представления, если требуемый формат и поведения отличаются от используемых бизнес-сущностями. Этот сценарий продемонстрирован на рис. 1.

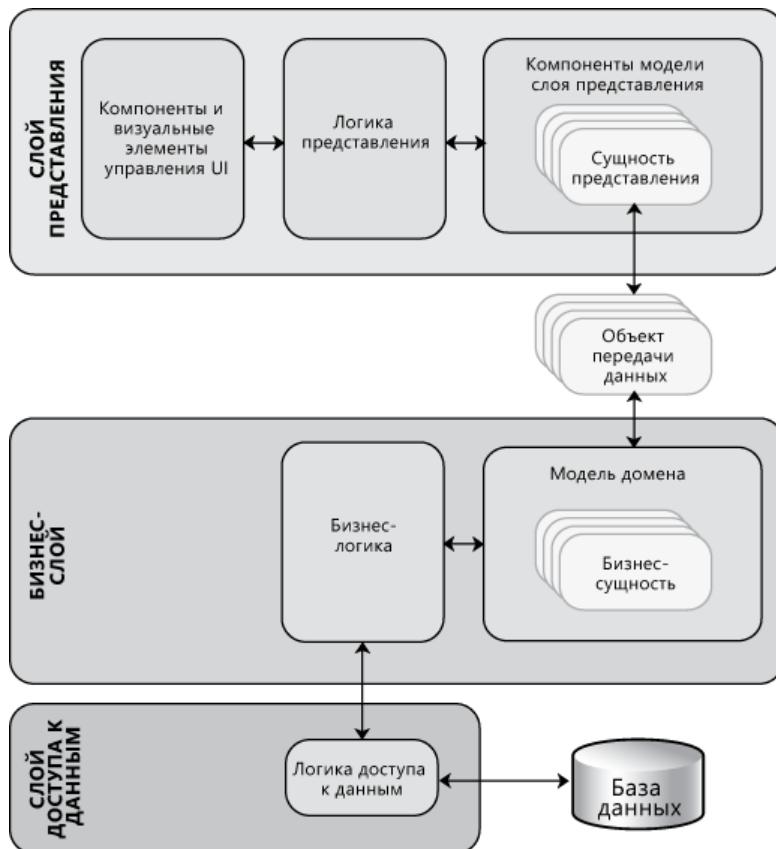


Рис. 11

Компоненты модели представления и сущности представления могут быть полезны при размещении слоя представления и бизнес-слоя на разных уровнях

Шаг 5 – Определение требований к привязке данных

Привязка данных обеспечивает возможность создания связи между элементами управления пользовательского интерфейса и данными или логическими компонентами приложения.

Привязка данных позволяет отображать и взаимодействовать с данными баз данных, а также данными других структур, таких как массивы и коллекции. Привязка данных – это мост между целью привязки (обычно это элемент управления пользовательского интерфейса) и источником привязки (обычно это структура данных, модель или компонент логики представления).

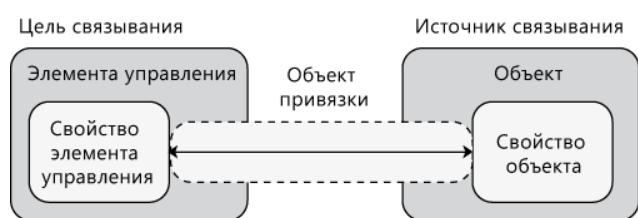


Рис. 12

Объекты, используемые при привязке данных

Как показано на рис. 2, обычно в привязке данных участвуют четыре элемента, взаимодействующих друг с другом для обновления свойств связанных объектов значениями, предоставляемыми источником привязки. Элементы управления с привязкой к данным – это

элементы управления, связанные с источниками данных. Например, элемент управления **DataGrid** связан с коллекцией объектов. Привязка данных часто используется с шаблонами раздельного представления для связывания компонентов UI (Представлений) с презентаторами или контроллерами (компоненты логики представления) либо с моделью слоя представления или компонентами сущностей.

Поддержка привязки данных и ее реализация в каждой технологии UI разная. В общем, большинство технологий UI позволяют выполнять привязку элементов управления к объектам и спискам объектов. Но некоторые специальные технологии привязки данных могут потребовать реализации в источниках данных определенных интерфейсов и событий для обеспечения полной поддержки привязки данных, например, интерфейса **INotifyPropertyChanged** (Уведомление об изменении свойства) в WPF или **IBindingList** (Список привязки) в Windows Forms. При использовании шаблона раздельного представления логика представления и компоненты данных должны гарантированно поддерживать необходимые интерфейсы или события, чтобы обеспечить простоту привязки элементов управления UI к ним.

Обычно используются два типа привязки:

- **Односторонняя привязка.** Изменения свойства источника приводят к автоматическому обновлению целевого свойства, но изменения целевого свойства не распространяются на исходное свойство. Такой тип привязки подходит для неявно доступных только для чтения элементов управления. Примером такой односторонней привязки могут быть биржевые сводки. Если нет необходимости отслеживать изменения целевого свойства, использование односторонней привязки позволит избежать ненужных издержек.
- **Двухсторонняя привязка.** Изменения любого из свойств, исходного либо целевого, приводят к автоматическому обновлению второго свойства. Такой тип привязки подходит для редактируемых форм или других полностью интерактивных сценариев UI. Многие редактируемые элементы управления в Windows Forms, ASP.NET и WPF поддерживают двухстороннюю привязку, так что изменения источника данных отражаются в элементе управления UI, и изменения в элементе управления UI отражаются в источнике данных.

Шаг 6 – Выработка стратегии обработки ошибок

Компоненты UI являются внешней границей приложения, и поэтому должны реализовывать соответствующую стратегию обработки ошибок для обеспечения стабильности приложения и положительного впечатления при взаимодействии с пользователем. При проектировании стратегии обработки ошибок рассмотрите следующие варианты:

- **Стратегия централизованной обработки исключений.** Обработка исключений и ошибок относится к сквозной функциональности. Она должна реализовываться в отдельных компонентах, которые обеспечивают централизацию этой функциональности и делают ее доступной во всех слоях приложения. Это также упрощает обслуживание и способствует повторному использованию.

- **Протоколирование исключений.** Очень важно протоколировать ошибки на границах системы, чтобы служба поддержки могла выявлять и диагностировать их. Это важно для компонентов представления, но может создавать большие сложности для кода, выполняющегося на клиентских компьютерах. Будьте осторожны и тщательно выбирайте методы протоколирования информации личного порядка (Personally Identifiable Information, PII) или конфиденциальных данных и обратите особое внимание на размер и размещение журнала.
- **Вывод на экран понятных пользователю сообщений.** При использовании этой стратегии в случае возникновения ошибки на экран выводится понятное пользователю сообщение с указанием причины ошибки и описанием, как ее можно исправить. Например, ошибки проверки данных должны отображаться так, чтобы было понятно, какие данные являются ошибочными и почему. В этом сообщении также указывается, как пользователь может исправить или ввести действительные данные.
- **Разрешение повторной попытки.** При использовании этой стратегии на экран выводится понятное пользователю сообщение, объясняющее причину ошибки и предлагающее пользователю повторить операцию. Такая стратегия полезна, если ошибки формируются из-за возникновения временных исключительных ситуаций, таких как недоступность ресурса или истечение времени ожидания сети.
- **Вывод на экран универсальных сообщений.** Если в приложении возникает непредвиденная ошибка, данные ошибки необходимо запротоколировать, но для пользователя вывести только универсальное сообщение. Предоставьте пользователю уникальный код ошибки, который может быть представлен группе технической поддержки. Эта стратегия полезна при возникновении непредвиденных исключений. Как правило, в случае возникновения непредвиденного исключения рекомендуется закрыть приложение, чтобы предотвратить повреждение данных или риски безопасности.

Более подробно методики обработки исключений обсуждаются в главе 17, «[Сквозная функциональность](#)». Enterprise Library, обеспечивающая полезные возможности для реализации стратегий обработки исключения, рассматривается в приложении F, «[Enterprise Library от patterns & practices](#)».

Шаг 7 – Определение стратегии валидации

Эффективная стратегия валидации пользовательского ввода поможет фильтровать нежелательные или злонамеренные данные и будет способствовать повышению защищенности приложения. Как правило, валидация ввода осуществляется слоем представления, тогда как проверка на соответствие бизнес-правилам проводится компонентами бизнес-слоя. При проектировании стратегии проверки, прежде всего, необходимо определить все вводимые данные, подлежащие проверке. Например, ввод от Веб-клиента в поля формы, параметры (такие как данные операций GET и POST и строки запросов), скрытые поля и состояние представления (view state) подлежат проверке. В общем, проверяться должны все данные, поступающие из источников, не имеющих доверия.

Для приложений, имеющих компоненты и на стороне клиента, и на стороне сервера, таких как приложения RIA или насыщенные клиентские приложения, вызывающие сервисы на сервере приложений, кроме всех проверок на клиенте, должна проводиться дополнительная проверка на сервере. Но для повышения удобства использования и по соображениям производительности некоторые из проверок можно продублировать на клиенте. Проверка на клиенте позволяет обеспечивать пользователям быструю обратную связь в случаях ввода ими некорректных данных. Это может сохранить время и полосу пропускания, но не забывайте, что злоумышленники могут обойти любую реализованную на клиенте проверку.

Определившись с данными, подлежащими проверке, выберите методики проверки для них. Самыми распространенными методиками проверки являются:

- **Прием заведомо допустимого** (*Список разрешенного ввода* или позитивная проверка). Принимаются только данные, удовлетворяющие заданным критериям, все остальные данные отклоняются.
- **Отклонение заведомо недопустимого** (*Список запрещенного ввода* или негативная проверка). Принимаются данные, не содержащие известный набор символов или значений.
- **Очистка.** Известные плохие символы или значения устраняются или преобразовываются с целью сделать ввод безопасным.

В общем, следует принимать заведомо допустимые значения (*Список разрешенного ввода*), а не пытаться выявить все возможные недействительные или злонамеренные значения, которые должны быть отклонены. Если невозможно полностью определить список известных допустимых значений, можно дополнить проверку частичным списком известных недопустимых значений и/или проводить очистку в качестве второй линии защиты.

Разные технологии представления используют разные подходы к проверке и информированию пользователя о проблемах. В WPF, к примеру, используются конвертеры и объекты правил проверки, часто подключаемые с помощью XAML, тогда как Windows Forms обеспечивает события проверки и привязки.

Более подробно методики валидации рассматриваются в главе 17, «[Сквозная функциональность](#)». Enterprise Library, обеспечивающая полезные возможности для проверки объектов и данных, как на стороне сервера, так и на стороне клиента, обсуждается в приложении F, «[Enterprise Library от patterns & practices](#)».

Предложения patterns & practices

Узнать о дополнительных предложениях группы Microsoft patterns & practices можно из следующих источников:

- **Composite Client Application Guidance for WPF**¹ (Руководство по проектированию составных клиентских приложений для WPF) как для настольных, так и для Silverlight-приложений; поможет в создании модульных приложений. Больше

¹ Также известно как Prism (прим. научного редактора).

информации можно найти в статье «*Composite Client Application Guidance*» по адресу <http://msdn.microsoft.com/en-us/library/cc707819.aspx>.

- **Enterprise Library** включает наборы блоков приложений, реализующих сквозную функциональность. Больше информации можно найти в статье «*Enterprise Library*» по адресу <http://msdn.microsoft.com/en-us/library/cc467894.aspx>.
 - **Software Factories** (Фабрики ПО) ускоряют разработку определенных типов приложений, таких как смарт-клиенты, WPF-приложения и Веб-сервисы. Больше информации можно найти в статье «*patterns & practices: by Application Type*» (*patterns & practices: по типу приложения*) по адресу <http://msdn.microsoft.com/en-gb/practices/bb969054.aspx>.
 - **Unity Application Block** (Блок Unity) как для корпоративных, так и для Silverlight-сценариев; обеспечивает средства для реализации внедрения зависимостей, обнаружения сервисов и обращения управления. Больше информации можно найти в статье «*Unity Application Block*» по адресу <http://msdn.microsoft.com/en-us/library/dd203101.aspx>.
-

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Design Guidelines for Web Applications*» (Руководство по проектированию Веб-приложений) по адресу <http://msdn.microsoft.com/en-us/library/ms978618.aspx>.
 - «*Data Binding Overview*» (Обзор привязки данных) по адресу <http://msdn.microsoft.com/en-us/library/ms752347.aspx>.
 - «*Design Guidelines for Exceptions*» (Руководство по проектированию обработки исключений) по адресу <http://msdn.microsoft.com/en-us/library/ms229014%28VS.80%29.aspx>.
-

12

Проектирование компонентов бизнес-слоя

Обзор

Проектирование компонентов бизнес-слоя является важной задачей: их неудачный дизайн, скорее всего, впоследствии приведет к созданию кода, который будет сложно обслуживать или расширять. При проектировании и реализации приложений используется несколько типов компонентов бизнес-слоя. К этим компонентам относятся компоненты бизнес-логики, бизнес-сущности, компоненты бизнес-процесса или рабочего процесса и служебные или вспомогательные компоненты. Данная глава начинается с обзора разных типов компонентов бизнес-слоя, используемых в большинстве приложений, с акцентированием внимания на компонентах бизнес-логики. Мы покажем, как разные аспекты проектирования приложения, требования к транзакциям и правила обработки влияют на выбираемый дизайн. Разобравшись с требованиями, можно сосредоточиться на шаблонах проектирования, обеспечивающих эти требования.

Шаг 1 – Выбор компонентов бизнес-слоя, которые будут использоваться в приложении

Для реализации бизнес логики в бизнес-слое может понадобиться создать или использовать разные типы компонентов. Цель этого этапа – выявить эти компоненты и выбрать необходимые для приложения. Следующие рекомендации помогут принять решение о том, какие компоненты использовать:

- Используйте **компоненты бизнес-логики** для инкапсуляции бизнес-логики и состояния приложения. Бизнес-логика – это логика приложения, занимающаяся вопросами реализации бизнес-правил и поведения приложения и обеспечением общей согласованности процессов, таких как валидация данных. Компоненты бизнес-логики должны быть легко тестируемыми и не зависеть от слоев представления и доступа к данным приложения.
- Используйте **бизнес-сущности** как часть подхода моделирования предметной области для инкапсуляции бизнес-логики и состояния в компоненты, представляющие реальные бизнес-сущности предметной области, такие как

продукты и заказы, с которыми должно работать создаваемое приложение. Более подробно бизнес-сущности рассматриваются в главе 13, «[Проектирование бизнес-сущностей](#)».

- Используйте **компоненты рабочего процесса**, если приложение должно поддерживать многошаговые процессы, выполняемые в определенном порядке; использует бизнес-правила, требующие взаимодействия между многими компонентами бизнес-логики; или вы желаете изменить поведение приложения, обновляя рабочий процесс по мере доработки приложения или изменения требования. Также рассмотрите возможность использования компонентов рабочего процесса, если приложение должно реализовывать динамическое поведение на основании бизнес-правил. В этом случае сохраняйте правила в обработчике правил. Для реализации компонентов рабочего процесса применяйте Windows Workflow Foundation. В качестве альтернативного варианта можно рассмотреть серверную среду интеграции, такую как BizTalk Server, если приложение должно обрабатывать многошаговый процесс, зависящий от внешних ресурсов, или включает процесс, который должен выполняться как длительная транзакция. Более подробно компоненты рабочего процесса рассматриваются в главе 14, «[Проектирование компонентов рабочего процесса](#)». Сервисы интеграции обсуждаются в приложении D, «[Матрица технологий интеграции](#)».

Шаг 2 – Принятие ключевых решений по компонентам бизнес-слоя

То, какие компоненты бизнес-слоя будут использоваться для обработки запросов, определяет общий дизайн и тип создаваемого приложения. Например, компоненты бизнес-слоя Веб-приложения обычно работают с основанными на сообщениях запросами, тогда как приложение Windows Forms обычно взаимодействует с компонентами бизнес-слоя напрямую с помощью основанных на событиях запросов. Кроме того, существуют и другие факторы, которые необходимо учесть при работе с разными типами приложений. Некоторые из этих факторов являются общими для различных типов, тогда как некоторые характерны лишь для конкретного типа приложений. Рассмотрим ключевые решения, которые должны быть приняты при проектировании компонентов бизнес-слоя:

- **Размещение.** Компоненты бизнес-слоя будут размещаться на клиенте, на сервере приложений или и там, и там? Размещайте часть или все компоненты бизнес-слоя на клиенте, если создаете изолированный насыщенный клиент или насыщенное Интернет-приложение (RIA), если желаете улучшить производительность, либо если используете при проектировании бизнес-сущностей модель предметной области. Размещайте часть или все компоненты бизнес-слоя на сервере приложений, если общая бизнес-логика должна поддерживать множество типов клиентов, если компоненты бизнес-слоя требуют доступа к ресурсам, недоступным с клиента, или по соображениям безопасности.
- **Связывание.** Как компоненты представления будут взаимодействовать с компонентами бизнес-слоя? Должно ли использоваться тесное связывание, при

котором компоненты представления напрямую взаимодействуют с компонентами бизнес-слоя, или слабое связывание, когда применяется абстракция, скрывающая детали компонентов бизнес-слоя? Для простоты в насыщенном клиентском приложении или RIA, в котором оба набора компонентов располагаются на клиенте, можно использовать тесное связывание между компонентами представления и бизнес-слоя, но слабое связывание между этими компонентами обеспечит лучшую тестируемость и гибкость. Если компоненты бизнес-слоя насыщенного клиентского приложения или RIA располагаются на сервере приложений или Веб-сервере, спроектируйте интерфейс сервиса, чтобы обеспечить предельно слабое связывание.

- **Взаимодействие.** Если компоненты бизнес-слоя и слоя представления размещаются на одном уровне, используйте компонентные взаимодействия через события и методы, что обеспечивает максимальную производительность. Однако реализуйте интерфейс сервиса и используйте взаимодействия посредством обмена сообщениями между слоем представления и компонентами бизнес-слоя, если компоненты бизнес-слоя и Веб-сервер располагаются на разных уровнях; если разрабатывается Веб-приложение со слабым связыванием между слоем представления и бизнес-слоем; или при наличии насыщенного клиента или приложения RIA. Если насыщенное клиентское приложение или RIA подключаются к серверу приложений или Веб-серверу лишь время от времени, необходимо внимательно подойти к вопросу проектирования интерфейса сервиса, который должен обеспечивать повторную синхронизацию клиента при подключении.

При реализации взаимодействия посредством сообщений продумайте, как будут обрабатываться повторяющиеся запросы и обеспечиваться гарантированная доставка сообщений. **Идемпотентность** (способность игнорировать дублирующиеся запросы) важна для сервисных приложений; основанных на сообщениях приложений, которые используют такую систему обмена сообщениями, как Microsoft Message Queuing; или для Веб-приложений, в которых долго выполняющиеся процессы могут привести к попыткам пользователя выполнить одно действие несколько раз. **Гарантированная доставка** необходима для основанных на сообщениях приложений, которые используют такую систему обмена сообщениями, как Microsoft Message Queuing; сервисов, применяющих маршрутизаторы сообщений между клиентом и сервисом; или сервисов, поддерживающих операции типа «отправил и забыл», при которых клиент отправляет сообщение, не ожидая ответа на него. Также не забывайте, что кэшированные сообщения, сохраняемые в ожидании обработки, могут устаревать.

Шаг 3 – Выбор соответствующей поддержки транзакций

Компоненты бизнес-слоя отвечают за координирование и управление всеми транзакциями, которые могут потребоваться в бизнес-слое. Но, прежде всего, необходимо убедиться в необходимости поддержки транзакций. Транзакции гарантируют, что наборы действий над одним или более диспетчерами ресурсов, такими как базы данных или очереди сообщений, выполняются единым блоком независимо от других транзакций. Если хотя бы одно из действий набора дает сбой, для всех остальных действий должен быть выполнен откат, чтобы обеспечить согласованность состояния системы. Например, имеется операция, которая

обновляет три разные таблицы, использующие множество компонентов бизнес-логики. Если два из этих обновлений выполняются успешно, но одно дает сбой, источник данных оказывается в несогласованном состоянии, т.е. содержит некорректные данные, от которых могут зависеть другие операции. Доступны следующие варианты реализации транзакций:

- **System.Transactions** использует компоненты бизнес-логики для запуска и управления транзакциями. Было введено в версии 2.0 .NET Framework вместе с легковесным диспетчером транзакций (Lightweight Transaction Manager, LTM), используется с диспетчерами недолгосрочных ресурсов или одним диспетчером долгосрочных ресурсов. Этот подход требует явного использования объекта типа **TransactionScope** (Область действия транзакции) и может расширять область действия транзакции и выполнять делегирование к распределенному координатору транзакций (Distributed Transaction Coordinator, DTC) в случае, если в транзакции участвует несколько диспетчеров долгосрочных ресурсов. Используйте **System.Transactions** при реализации поддержки транзакций в создаваемом приложений, если имеются транзакции, охватывающие несколько диспетчеров недолгосрочных ресурсов.
- **Транзакции WCF** были представлены в версии 3.0 .NET Framework и базируются на функциональности **System.Transactions**. Они обеспечивают декларативный подход к управлению транзакциями, который реализуется посредством ряда атрибутов и свойств, таких как **TransactionScopeRequired** (Необходима область действия транзакции), **TransactionAutoComplete** (Автоматическое завершение транзакции) и **TransactionFlow** (Поток транзакции). Используйте транзакции WCF, если требуется поддерживать транзакции при взаимодействии с сервисами WCF. Однако рассмотрите возможность применения декларативного описания транзакций, вместо того чтобы использовать код для управления транзакциями.
- **Транзакции ADO.NET** появились в версии 1.0 .NET Framework, требуют применения компонентов бизнес-логики для запуска и управления транзакциями. Они используют явную модель программирования, когда разработчики должны реализовывать управление нераспределенными транзакциями в коде. Используйте транзакции ADO.NET при расширении приложения, которое уже использует транзакции ADO.NET; или если используете для доступа к базе данных ADO.NET-поставщиков, и транзакции осуществляются только с одним ресурсом. ADO.NET 2.0 и последующие версии дополнительно поддерживают распределенные транзакции, использующие возможности **System.Transactions**, описанные выше.
- Транзакции **базы данных** используются для реализации функциональности управления транзакциями, которая может быть включена в хранимые процедуры, что также может упростить дизайн бизнес-процесса. Если транзакции запускаются компонентами бизнес-логики, транзакция базы данных будет входить в состав транзакции, созданной бизнес-компонентом. Используйте транзакции базы данных при разработке хранимых процедур, инкапсулирующих все изменения, которые должны выполняться транзакцией; или при наличии множества приложений, использующих одинаковые хранимые процедуры, когда требования к транзакциям могут быть инкапсулированы в эти хранимые процедуры.

На забывайте, что при использовании распределенных транзакций может увеличиваться связанность между подсистемами системы. Транзакции, включающие удаленные системы, скорее всего, повлекут снижение производительности из-за увеличения сетевого трафика. Транзакции – ресурсоемкий процесс, поэтому они должны выполняться быстро, в противном случае чрезмерно долгая блокировка ресурсов может привести к истечению времени ожидания или взаимным блокировкам.

Участвовать в транзакциях могут только сервисы с высоким уровнем доверия, потому что участие в транзакции позволяет внешним сервисам блокировать внутренние ресурсы. При использовании сервисов для выполнения бизнес-процессов создавайте атомарные транзакции только в крайних случаях, когда этого никак нельзя избежать.

Шаг 4 – Выработка стратегии обработки бизнес-правил

Обработка бизнес-правил может быть одним из наиболее сложных аспектов проектирования приложения. Общей рекомендацией является реализация бизнес-правил в рамках бизнес-слоя. Однако где именно в бизнес-слое это должно происходить? Это может быть бизнес-логика или компоненты рабочего процесса, обработчик бизнес-правил или дизайн модели предметной области, инкапсулирующий правила в модели. Рассмотрим возможные варианты обработки бизнес-правил:

- **Компоненты бизнес-логики** могут использоваться для обработки простых или очень сложных правил в зависимости от шаблона проектирования, применяемого для их реализации. Используйте компоненты бизнес-логики для задач или операций с документами в Веб-приложениях или сервисах, если не реализуете модель предметной области при проектировании бизнес-сущностей, или используете бизнес-правила из внешнего источника.
- **Компоненты рабочего процесса** используются, если необходимо отделить бизнес-правила от бизнес-сущностей, или если применяемые бизнес-сущности не поддерживают инкапсуляцию бизнес-правил, или если взаимодействие множества бизнес-сущностей управляется инкапсулированной бизнес-логикой.
- **Обработчики бизнес-правил** обеспечивают возможность задавать и изменять правила без участия разработчиков, но при этом усложняют и добавляют издержки в приложения, поэтому должны использоваться только в случае необходимости. Иначе говоря, применяйте обработчик правил при наличии правил, которые должны будут корректироваться на основании разных факторов, связанных с приложением. Используйте обработчик бизнес-правил при наличии часто меняющихся бизнес-правил, т.е. таких, которые будут меняться регулярно; для поддержки возможности настройки и гибкости; или если хотите предоставить бизнес-пользователям возможность управлять и обновлять правила. Убедитесь, что пользователям предоставляются только те правила, которые подлежат изменениям, и что изменять правила, критические с точки зрения корректности поведения бизнес-логики, могут только авторизованные пользователи.

- **Проектирование модели предметной области** может использоваться для инкапсуляции бизнес-правил в бизнес-сущности. Но модель предметной области может быть сложно правильно реализовать, кроме того, она имеет тенденцию сосредотачиваться на одном конкретном срезе или контексте. Инкапсулируйте правила в модель предметной области, если имеете насыщенное клиентское приложение или RIA, в котором части бизнес-логики развернуты на клиенте, и сущности модели предметной области инициализируются и сохраняются в памяти; или если имеете модель предметной области, которая может сохраняться в состоянии сеанса, ассоциированном с Веб-приложениями или приложениями сервисов. При размещении частей модели предметной области на клиенте необходимо продублировать ее на сервере, чтобы применить правила и поведение и обеспечить безопасность и возможность обслуживания.
-

Шаг 5 – Выбор шаблонов, соответствующих требованиям

Поведенческие шаблоны создаются на базе наблюдений за поведением системы в действии и выявлении повторяющихся процессов. Для компонентов бизнес-слоя обычно используются поведенческие шаблоны проектирования, иначе говоря, шаблоны, основной задачей которых является реализация поведения приложения на уровне дизайна. Была проделана большая работа по выявлению и описанию шаблонов поведения, имеющих место в приложениях разных типов и на разных уровнях дизайна приложения. Изучить все описанные шаблоны просто невозможно, но необходимо хорошо разбираться в разных типах шаблонов и уметь находить в сценарии поведение, которое может быть описано шаблоном. В следующей таблице приведены шаблоны, которые обычно используются с компонентами бизнес-слоя.

Шаблон	Рекомендации
<i>Adapter</i> (Адаптер)	Обеспечивает возможность совместной работы классов с несовместимыми интерфейсами, позволяя разработчикам реализовывать наборы полиморфных классов, обеспечивающих альтернативные реализации существующего класса.
<i>Command</i> (Команда)	Рекомендуется для насыщенных клиентских приложений с меню, панелями инструментов и реализациями клавишных комбинаций быстрого вызова, которые используются для выполнения одних и тех же команд из разных компонентов. Также может использоваться для реализации команд с шаблоном Supervising Presenter.
<i>Chain of Responsibility</i> (Цепочка обязанностей)	Объединяет обработчики запросов так, что каждый обработчик проверяет запрос и либо обрабатывает его, либо передает следующему обработчику. Альтернативой этому шаблону являются выражения «if, then, else» с возможностью обработки сложных бизнес-правил.
<i>Decorator</i> (Декоратор)	Расширяет поведение объекта во время выполнения, добавляя или изменяя операции, которые будут осуществляться при обработке запроса. Требует общего интерфейса, реализовываемого классами декоратора, которые могут объединяться для обработки сложных бизнес-правил.
<i>Dependency Injection</i> (Внедрение зависимостей)	Создает и заполняет члены (поля и свойства) объектов, используя отдельный класс, который обычно создает эти зависимости во время выполнения на основании конфигурационных файлов. Конфигурационные файлы описывают

	контейнеры, определяющие сопоставление или регистрация типов объектов. Сопоставление и регистрация объектов может также выполняться в коде приложения. Обеспечивает гибкий подход к изменению поведения и реализации сложных бизнес-правил.
<i>Façade</i> (Фасад)	Обеспечивает слабо детализированные операции, унифицирующие результаты, поступающие от множества компонентов бизнес-логики. Обычно реализуется как удаленный фасад для интерфейсов на основе сообщений бизнес-слоя и используется для обеспечения слабого связывания между слоем представления и бизнес-слоем.
<i>Factory</i> (Фабрика)	Создает экземпляры объектов без указания конкретного типа. Требует наличия объектов, которые реализуют общий интерфейс или расширяют общий базовый класс.
<i>Transaction Script</i> (Сценарий транзакции)	Рекомендуется для базовых CRUD-операций с минимальным набором бизнес-правил. Компоненты сценария транзакции также инициируют транзакции. Это означает, что все операции, осуществляемые компонентом, должны представлять неделимую единицу работы. При использовании этого шаблона компоненты бизнес-логики взаимодействуют с другими компонентами бизнес-слоя и компонентами данных для завершения операции.

Этот список представляет многие общие шаблоны, используемые с компонентами бизнес-слоя, но он далеко не полный. Основная задача при выборе шаблона – убедиться, что он соответствует сценарию и не усложняет приложение больше, чем требуется.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- Более подробно проектирование компонентов бизнес-слоя рассматривается в статье «*Application Architecture for .NET: Designing Applications and Services*» (Архитектура приложений для .NET: проектирование приложений и сервисов) <http://msdn.microsoft.com/en-us/library/ms954595.aspx> .
- Производительность бизнес-слоев и компонентов более подробно рассматривается в следующих источниках:
 - «*Architecture and Design Review of a .NET Application for Performance and Scalability*» (Обзор архитектуры и дизайна .NET-приложения с точки зрения производительности и масштабируемости) по адресу <http://msdn.microsoft.com/en-us/library/ms998544.aspx>.
 - «*Design Guidelines for Application Performance*» (Рекомендации по производительности приложений) по адресу <http://msdn.microsoft.com/en-us/library/ms998541.aspx>.
- Реализация транзакций в компонентах бизнес-слоя более подробно рассматривается в следующих источниках:

- «*Introducing System.Transactions in the .NET Framework 2.0*» (Представляем System.Transactions in the .NET Framework 2.0) по адресу <http://msdn.microsoft.com/en-us/library/ms973865.aspx>.
 - «*Transactions in WCF*» (Транзакции в WCF) по адресу <http://msdn.microsoft.com/en-us/library/ms730266.aspx>.
 - «*Transaction Processing in .NET 3.5*» (Обработка транзакций в .NET 3.5) по адресу <http://msdn.microsoft.com/en-us/library/w97s6fw4.aspx>.
 - Реализация рабочего процесса в компонентах бизнес-слоя более подробно рассматривается в следующих источниках:
 - «*Introduction to the Windows Workflow Foundation Rules Engine*» (Введение в обработчик правил Windows Workflow Foundation) по адресу <http://msdn.microsoft.com/en-us/library/aa480193.aspx>.
 - «*Windows Workflow Foundation*» по адресу <http://msdn.microsoft.com/en-us/library/ms735967.aspx>.
-

13

Проектирование бизнес-сущностей

Обзор

Бизнес-сущности хранят значения данных и предоставляют их через свойства. Они содержат и управляют бизнес-данными, которые используются приложением, и обеспечивают программный доступ с сохранением состояния к бизнес-данным и связанной функциональности. Бизнес-сущности также должны проводить проверку содержащихся в них данных и инкапсулировать бизнес-логику для обеспечения непротиворечивости и реализации бизнес-правил и необходимого поведения. Таким образом, правильное проектирование или выбор бизнес-сущностей жизненно важны для обеспечения наилучшей производительности и эффективности бизнес-слоя.

Данная глава поможет разобраться с проектированием компонентов бизнес-сущностей. Она начинается с рассмотрения разных форматов данных и использования данных в приложении. После этого демонстрируется, как выбранный формат определяет возможные пути реализации бизнес-правил. Наконец, представляются варианты дизайна для собственных объектов и рассматривается, как реализовывается поддержка сериализации для разных форматов данных.

Общие принципы проектирования компонентов и компоненты, обычно используемые в слоях приложения, более подробно рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)».

Шаг 1 – Выбор способа представления сущностей

На этом этапе рассматриваются разные способы представления бизнес-сущностей и преимущества и недостатки каждого из них, что поможет сделать правильный выбор представления для конкретного сценария. Перечислим наиболее распространенные форматы:

- **Собственные бизнес-объекты.** Это объекты общеязыковой среды выполнения (common language runtime, CLR), описывающие сущности системы. Для создания этих объектов может использоваться технология объектно-реляционного сопоставления (object/relational mapping, O/RM), такая как ADO.NET Entity Framework (EF) или NHibernate (больше сведений можно найти в разделе

[Дополнительные источники](#) в конце данной главы), или их можно создавать вручную. Собственные бизнес-объекты подходят в случаях, когда требуется инкапсулировать сложные бизнес-правила или поведение вместе со связанными с ними данными. Если требуется выполнять доступ к бизнес-объектам через границы **AppDomain** (Домен приложения), процесса или физические границы, можно реализовать слой сервисов, который будет обеспечивать доступ посредством объектов передачи данных (Data Transfer Objects, DTO) и операций обновления или редактирования бизнес-объектов.

- **DataSet** или **DataTable**. Объекты **DataSet** – это разновидность базы данных в памяти, которая обычно очень близко соответствует фактической схеме базы данных. Объекты **DataSet**, как правило, используются только, если не используется механизм О/RM-сопоставления и создается объектно-ориентированное приложение, в котором данные логики приложения очень близко соответствуют схеме базы данных. **DataSet** не может расширяться для инкапсуляции бизнес-логики или бизнес-правил. Несмотря на то, что **DataSet** могут быть сериализованы в XML, к ним не должен предоставляться доступ через границы процесса или сервиса.
- **XML**. Это основанный на стандартах формат для организации структурированных данных. XML обычно используется для представления бизнес-сущностей, только если этого требует слой представления или если логика должна работать с содержимым, исходя из его схемы. Например, система маршрутизации сообщений, логика которой направляет сообщения на основании некоторых известных элементов XML-документа. Не забывайте, что использование и обработка XML может требовать больших объемов памяти.

Шаг 2 – Выбор дизайна бизнес-сущностей

Если принято решение о том, что собственные объекты обеспечат наилучшее представление бизнес-сущностей, следующим шагом является проектирование этих объектов. Подход, применяемый к проектированию собственных объектов, зависит от сорта объекта, который планируется использовать. Например, сущности модели предметной области требуют глубокого анализа предметной области, тогда как сущности модуля таблицы требуют понимания схемы базы данных. Рассмотрим общие подходы к проектированию при использовании бизнес-объектов:

- **Модель предметной области (Domain Model)** – объектно-ориентированный шаблон проектирования. Цель проектирования предметной области – определение бизнес-объектов, которые представляют реальные сущности предметной области. При использовании модели предметной области бизнес-сущности и сущности предметной области включают и поведение, и структуру. Иначе говоря, бизнес-правила и отношения инкапсулированы в модели предметной области. Проектирование предметной области требует глубокого анализа предметной области и, как правило, не сопоставляется с реляционными моделями, используемыми большинством баз данных. Применяйте модель предметной области, если предметная область содержит сложные бизнес-правила, если

создается насыщенный клиент, и модель предметной области может инициализироваться и удерживаться в памяти. Модель предметной области не может применяться при работе с бизнес-слоем, не сохраняющим состояние, который требует инициализации модели предметной области при каждом запросе. Более подробно модель предметной области и проектирование на основе предметной области рассматриваются в разделе «[Проектирование на основе предметной области](#)» далее в данной главе.

- **Модуль таблицы (Table Module)** – это объектно-ориентированный шаблон проектирования. Цель проектирования модуля таблицы – определение сущностей на основании таблиц или представлений базы данных. Операции, используемые для доступа к базе данных и заполнения сущностей модуля таблицы, обычно инкапсулированы в сущности. Однако для осуществления операций с базой данных и заполнения сущностей модуля таблицы могут также использоваться компоненты доступа к данным. Применяйте подход с модулем таблицы, если таблицы или представления базы данных достаточно точно представляют бизнес-сущности, используемые приложением, или если бизнес-логика и операции касаются одной таблицы или представления.
- **Специальные XML-объекты (Custom XML objects)** представляют десериализованные XML-данные, которые могут обрабатываться кодом приложения. Объекты являются экземплярами классов, определяемыми атрибутами, которые сопоставляют свойства классов с элементами и атрибутами XML-структуры. Microsoft .NET Framework предоставляет компоненты, которые могут использоваться для десериализации XML-данных в объекты и сериализации объектов в XML. Используйте специальные XML-объекты, если данные, с которыми вы работаете, уже поступают в XML-формате (например, XML-файлы или операции базы данных, возвращающие XML как результирующее множество); если требуется формировать XML-данные из источника не XML-данных; или при работе с документами, доступными только для чтения.

При использовании собственных объектов не обязательно, чтобы проектирование всех бизнес-сущностей выполнялось по одной схеме. Например, для одного из аспектов приложения со сложными правилами может потребоваться проектирование на основе модели предметной области, а все остальное приложение может использовать XML-объекты, модуль таблицы или объекты предметной области.

Шаг 3 – Определение механизмов сериализации

Должно быть принято решение о способе передачи бизнес-сущностей через границы. В большинстве случаев для передачи данных через физические границы, такие как AppDomain, процесс и границы интерфейса сервисов, требуется сериализация данных. Также данные можно сериализовать при пересечении логических границ, но при этом не забывайте о производительности. Рассмотрите следующие варианты передачи бизнес-сущностей:

- **Предоставление прямого доступа к сериализуемым бизнес-сущностям только в случае необходимости.** Если бизнес-сущности используются другим слоем приложения, располагающимся на том же уровне, самый простой подход – предоставить бизнес-сущности для прямого доступа без сериализации. Однако недостатком такого подхода является установление зависимости между потребителями бизнес-сущностей и реализацией бизнес-сущностей. Поэтому применение такого подхода рекомендуется только в том случае, если имеется возможность напрямую управлять потребителями бизнес-сущностей, и нет необходимости в удаленном доступе к бизнес-сущностям с других уровней.
- **Преобразование бизнес-сущностей в сериализуемые объекты передачи данных.** Чтобы устраниТЬ зависимость потребителей данных от внутренней реализации бизнес-слоя, используйте преобразование бизнес-сущностей в специальные сериализуемые объекты передачи данных. Объект передачи данных (Data Transfer Object, DTO) – это шаблон проектирования, используемый для упаковки множества структур данных в одну структуру для передачи через границы. Объекты передачи данных также полезны, если потребители бизнес-сущностей используют другое представление или модель данных, например, уровень представления. Этот подход позволяет менять внутреннюю реализацию бизнес-слоя без влияния на потребителей данных и упрощает реализацию контроля версии интерфейсов. Его применение рекомендуется при потреблении данных внешними клиентами.
- **Предоставление XML для прямого доступа.** В некоторых случаях бизнес-сущности могут сериализовываться и предоставляться как XML. .NET Framework обеспечивает широкую поддержку сериализации XML-данных. Атрибуты бизнес-сущностей преимущественно используются для управления сериализацией в XML.

Более подробно схемы данных для интерфейсов сервисов рассматриваются в главе 9, «[Рекомендации по проектированию слоя сервисов](#)». Взаимодействию слоев и уровней посвящена глава 18, «[Взаимодействие и обмен сообщениями](#)».

Проектирование на основе предметной области

Проектирование на основе предметной области (Domain Driven Design, DDD) – это объектно-ориентированный подход к проектированию программного обеспечения на основе предметной области, ее элементов и поведения и отношений между ними. Цель такого проектирования – создание программных систем, которые являются реализацией базовой предметной области, через описание модели предметной области, представленной на языке специалистов этой предметной области. Модель предметной области можно рассматривать как инфраструктуру, из которой впоследствии могут быть выведены решения.

Для применения DDD необходимо четко понимать предметную область, которую предполагается моделировать, или иметь способности для овладения такими знаниями. При создании модели предметной области группа разработки нередко работает в сотрудничестве со специалистами в данной области. Архитекторы, разработчики и специалисты в рассматриваемой области обладают разной подготовкой и во многих ситуациях будут

использовать разные языки для описания своих целей, желаний и требований. Однако в рамках DDD вся группа договаривается использовать единый язык, ориентированный на предметную область и исключающий все технические жаргонизмы.

В качестве ядра ПО выступает модель предметной области, которая является прямой проекцией этого общего языка; с ее помощью путем анализа языка группа быстро находит пробелы в ПО. Создание общего языка – это не просто упражнение по получению сведений от специалистов и их применению. Довольно часто в группах возникают проблемы с обменом информацией не только по причине непонимания языка предметной области, но также и из-за неопределенности языка самого по себе. Процесс DDD имеет целью не только реализацию используемого языка, но также улучшение и уточнение языка предметной области. Это, в свою очередь, положительно сказывается на создаваемом ПО, поскольку модель является прямой проекцией языка предметной области.

Модель предметной области представляется с помощью сущностей, объектов-значений, сводных корней, хранилищ и сервисов предметной области, которые организовываются в крупные области ответственности, называемые Ограничеными контекстами (Bounded Contexts).

Сущности – это объекты модели предметной области, имеющие уникальный идентификатор, который остается неизменным при изменении состояния ПО. Сущности инкапсулируют и состояние, и поведение. Примером сущности может быть объект Customer, представляющий и сохраняющий состояние определенного клиента и реализующий операции, которые могут выполняться с этим клиентом.

Объекты-значения – это объекты модели предметной области, используемые для описания определенных аспектов предметной области. Они не имеют уникального идентификатора и неизменны. Примером объекта-значения является Transaction Amount (Сумма транзакции) или Customer Address (Адрес клиента).

Сводные корни – это сущности, логически группирующие взаимосвязанные дочерние сущности или объекты-значения, управляют доступом к ним и координируют взаимодействия между ними.

Хранилища отвечают за извлечение и хранение сводных корней, как правило, с использованием инфраструктуры объектно-реляционного сопоставления (O/RM).

Сервисы предметной области представляют операции, действия или бизнес-процессы и обеспечивают функциональность, используемую другими объектами модели предметной области. В некоторых случаях определенная функциональность или аспект предметной области не может быть сопоставлен ни с одним объектом, имеющим определенный жизненный цикл или идентификатор; такая функциональность может быть объявлена как сервис предметной области. В качестве примера сервиса предметной области можно привести сервис установления цен каталога товаров системы электронной торговли.

Чтобы сделать модель строгой и полезной языковой конструкцией, как правило, приходится интенсивно использовать изоляцию и инкапсуляцию в рамках модели предметной области. Это может обусловить относительную дороговизну системы, основанной на DDD. Несмотря на то, что DDD обеспечивает массу преимуществ с технической точки зрения, таких как удобство в

обслуживании, эта схема должна применяться лишь для сложных предметных областей, для которых процессы моделирования и лингвистического анализа обеспечивают безусловные преимущества при обмене сложной для понимания информацией и формулировании общего видения предметной области.

Обзор методик проектирования на основе предметной области представлен в статье «*Domain Driven Design Quickly*» по адресу <http://www.infoq.com/minibooks/domain-driven-design-quickly>. Также полезными будут книга Эрика Эванса (Eric Evans) «*Domain-Driven Design: Tackling Complexity in the Heart of Software*» (Addison-Wesley, ISBN: 0-321-12521-5) и книга Джимми Нильсона (Jimmy Nilsson) «*Применение DDD и шаблонов проектирования*» (Вильямс, ISBN 978-5-8459-1296-1, 0-321-26820-2).

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- Более подробно шаблоны, используемые для проектирования бизнес-сущностей, рассматриваются в статье «*Enterprise Solution Patterns Using Microsoft .NET*» по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.
- Проектированию бизнес-сущностей посвящена статья «*Integration Patterns*», которую можно найти по адресу <http://msdn.microsoft.com/en-us/library/ms978729.aspx>.
- Больше информации о проектировании на основе предметной области можно найти в следующих источниках:
 - «*An Introduction To Domain-Driven Design*» по адресу <http://msdn.microsoft.com/en-us/magazine/dd419654.aspx>.
 - «*Domain Driven Design and Development in Practice*» по адресу <http://www.infoq.com/articles/ddd-in-practice>.
- Более подробно шаблоны, применяемые для проектирования бизнес-слоя, рассматриваются в статье «*Service Orientation Patterns*» (Сервисно-ориентированные шаблоны) по адресу <http://msdn.microsoft.com/en-us/library/aa532436.aspx>.
- ADO.NET Entity Framework подробно рассматривается в статье «*The ADO.NET Entity Framework Overview*» (Обзор ADO.NET Entity Framework) по адресу [http://msdn.microsoft.com/en-us/library/aa697427\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx).
- Больше сведений о проектировании бизнес-сущностей с использованием Microsoft Dynamics можно найти в статье «*Business Entities*» (Бизнес-сущности) по адресу <http://msdn.microsoft.com/en-us/library/ms940455.aspx>.
- Больше сведений о моделировании бизнес-сущностей с использованием Microsoft Dynamics можно найти в статье «*Modeling Entities*» (Моделирование сущностей) по адресу <http://msdn.microsoft.com/en-us/library/aa475207.aspx>.

- Более подробно использование бизнес-сущностей в Office Business Applications (ОВА) рассказывает статья «*Building Office Business Applications*» (Создание офисных бизнес-приложений) по адресу <http://msdn.microsoft.com/en-us/library/bb266337.aspx>.
 - Инфраструктуре с открытым исходным кодом NHibernate посвящен сайт «*NHibernate Forge*» (Кузница NHibernate), который можно найти по адресу <http://nhforge.org/Default.aspx>.
-

14

Проектирование компонентов рабочего процесса

Обзор

Во многих сценариях задачи пользователя должны осуществляться в определенном порядке на основании выполнения определенных этапов или удовлетворять ряду базовых бизнес-правил. Компоненты рабочего процесса используются для инкапсуляции задач и для согласования шагов, необходимых для их выполнения. Компоненты рабочего процесса также могут поддерживать задачи, зависящие от обрабатываемых данных, таких как данные, вводимые пользователем или динамическими бизнес-правилами, которые определяют бизнес-процесс.

Данная глава обсуждает разные сценарии и предлагает руководство по проектированию компонентов рабочего процесса. Она начинается с рассмотрения того, как реальные сценарии проецируются на ключевые сценарии организации рабочего процесса; это поможет сделать правильный выбор стиля рабочего процесса для создаваемого приложения. Далее демонстрируется, как требования и правила определяют доступные варианты реализации компонентов рабочего процесса. И, в заключение, приводятся рекомендации по проектированию компонентов рабочего процесса с учетом различных доступных вариантов.

Общие принципы проектирования и более подробное описание компонентов, обычно используемых в слоях приложения, можно найти в главе 10, «[Рекомендации по проектированию компонентов](#)».

Шаг 1 – Выбор стиля рабочего процесса на основании сценариев

Существует три основных типа стилей рабочего процесса: последовательный, конечный автомат и управляемый данными. При последовательном рабочем процессе выполнение задач состоит из определенного набора этапов. В конечном автомате действия определены как набор состояний и событий, которые обуславливают переходы из одного состояния в другое. В управляемом данными рабочем процессе действия выполняются на основании информации, ассоциированной с данными. Таким образом, первым делом при проектировании компонентов рабочего процесса необходимо понять, какой рабочий процесс

требуется поддерживать. Рассмотрим рекомендации по выбору одного из этих трех базовых стилей рабочего процесса:

- **Последовательный рабочий процесс.** Такой рабочий процесс управляет последовательностью действий и принимает решение о том, какой из этапов будет выполнен следующим. Несмотря на то, что последовательный рабочий процесс может включать ветви и циклы, он предсказуем. Используйте последовательные рабочие процессы, если для реализации определенной задачи требуется выполнять серии заранее определенных шагов; или для таких сценариев как управление системами, координация операций между компаниями и обработка бизнес-правил.
- **Конечный автомат.** Такой рабочий процесс переходит в заданное состояние и ожидает возникновения определенных событий для перехода в другое состояние. Используйте конечный автомат для управляемых событиями сценариев, сценариев пользовательских интерфейсов, таких как интерфейс мастера, или систем обработки заказов, в которых предпринимаемые шаги и процессы зависят от данных заказа.
- **Управляемый данными рабочий процесс¹.** В таком рабочем процессе данные документа определяют действия, выполняемые рабочим процессом. Этот стиль подходит для таких задач, как процесс утверждения документов.

Шаг 2 – Выбор способа разработки

Для создания рабочих процессов можно использовать код, языки разметки или их сочетание. Используемый подход зависит от требований, предъявляемых к способу разработки в создаваемом решении. Выбор способа разработки также зависит от того, как приложение будет упаковываться и распространяться. Возможны такие варианты:

- **Только код.** Выбирайте этот вариант, если рабочий процесс не будет сильно меняться со временем; если имеются сложные бизнес-правила, которые тяжело выразить средствами языков разметки; если группе разработки привычнее писать управляемый код, а не создавать разметку в визуальных дизайнерах; или если требуется разрабатывать новые типы рабочих процессов, что невозможно сделать с помощью разметки. Также рабочие процессы, созданные с использованием только кода, просто интегрировать в систему управления исходным кодом.
- **Разделение кода².** Выбирайте этот вариант, если имеются сложные бизнес-правила, инкапсулированные компонентами бизнес-слоя; или если требуется предоставить пользователям или администраторам возможность изменять некоторые аспекты рабочего процесса с помощью дизайнеров рабочих процессов.

¹ Так же известны, как рабочие процессы на базе политик (прим. научного редактора).

² То есть используется и язык разметки, и код (прим. научного редактора).

- **Разметка.** Выбирайте этот вариант, если предполагается более частое изменение рабочего процесса в будущем; если ассоциированные с рабочим процессом бизнес-правила можно без труда выразить средствами языков разметки; если нет необходимости создавать новые типы рабочих процессов; если необходимо обеспечить гибкость для обновления модели рабочего процесса без повторной сборки используемого моделью программного кода, реализующего рабочий процесс.
-

Шаг 3 – Определение стратегии обработки правил

На данный момент уже сделан выбор относительно стиля рабочего процесса и способа разработки рабочих процессов. Следующий шаг – принятие решения о том, как рабочий процесс будет обрабатывать бизнес-правила. Доступные варианты определяются сложностью, стабильностью бизнес-правил и требованиями к управлению, с ними связанными. При выборе стратегии обработки бизнес-правил в компонентах рабочего процесса следует учитывать следующие факторы:

- **Если правила сложные,** для их разработки необходимо использовать только код или разделение кода. Правила могут реализовываться и инкапсулироваться в компонентах бизнес-слоя с возможностью координирования их выполнения рабочим процессом.
- **Если правила подвержены изменениям**, т.е. простые или управляемые данными, для них следует применять разметку. Однако если правила управляются внешней системой, такой как обработчик бизнес-правил, применяйте только код или разработку с разделением кода.
- **Если правила будут контролироваться бизнес-пользователями, администраторами или аналитиками**, выбирайте решение с использованием языков разметки, что обеспечит визуальный дизайнер или другое средство редактирования правил, или решение, поддерживающее предметно-ориентированный язык программирования (Domain Specific Language, DSL). Однако если правила управляются внешней системой, такой как подсистема управления бизнес-правилами, применяйте разработку с разделением кода.

Шаг 4 – Выбор решения для рабочего процесса

Выбрав стиль рабочего процесса, способ разработки и требования по обработке правил, можно переходить к выбору решения для рабочего процесса, который зависит от возможностей, обеспечиваемых каждым решением. На платформе Microsoft доступны следующие технологии:

- **Windows Workflow Foundation (WF).** WF обеспечивает ориентированное на разработчика решение для создания последовательного рабочего процесса, конечного автомата или управляемого данными рабочего процесса. WF поддерживает разработку с помощью только кода, с разделением кода и с

использованием языков разметки. Поддержка дизайнера доступна в Visual Studio 2005 через расширения и непосредственно в Visual Studio 2008 и последующих версиях. WF включает дополнительные средства для безопасного, надежного, транзакционного обмена данными, отслеживания действий, широкий выбор средств передачи и кодирования передаваемых данных, а также обеспечивает поддержку длительных рабочих процессов, которые могут сохраняться между выключениями и повторными запусками системы.

- **Workflow Services.** Workflow Services (Сервисы рабочего процесса) обеспечивают интеграцию Windows Communication Foundation (WCF) и Windows Workflow Foundation (WF) для обеспечения рабочих процессов WCF-возможностями. Начиная с Microsoft .NET Framework 3.5, WCF расширена для обеспечения поддержки рабочих процессов, предоставляемых как сервисы, и возможности вызова сервисов из рабочих процессов. Кроме того, Microsoft Visual Studio 2008 включает новые шаблоны и инструментальные средства, поддерживающие сервисы рабочего процесса.
- **Microsoft Office SharePoint Services (MOSS).** MOSS¹ – платформа для управления информацией и координации совместной деятельности, обеспечивающая поддержку рабочих процессов на основе технологий WF. MOSS обеспечивает решение для рабочего процесса, управляемого оператором, и совместной деятельности в контексте сервера Microsoft Office SharePoint® Server. Используя Веб-интерфейс, можно определять рабочие процессы для визирования документов, связанных с элементами списка SharePoint; использование дизайнера SharePoint или Windows Workflow Designer (Дизайнер Windows рабочих процессов) в Visual Studio позволяет определять условные и управляемые данными рабочие процессы. Для настройки рабочих процессов может использоваться объектная модель WF в Visual Studio. Однако MOSS подходит, только если бизнес-слой взаимодействует лишь с одним сайтом SharePoint и не требует доступа к данным других сайтов.
- **BizTalk Server.** Сервер BizTalk поддерживает последовательные рабочие процессы, конечные автоматы и управляемые данными рабочие процессы, а также разработку с разделением кода и применением языков разметки. Он обеспечивает возможность обмена электронными документами между компаниями с использованием форматов Electronic Data Interchange (EDI)² и/или XML и включает мощные возможности оркестровки для проектирования и выполнения длительных, тесно связанных бизнес-процессов и рабочих процессов с возможностями надежного хранения и пересылки сообщений. BizTalk интегрируется с гетерогенными приложениями и системами через адаптеры и обеспечивает обработчик бизнес-правил и мониторинг деловой активности (Business Activity Monitoring). Если требуется взаимодействовать с системами не-Microsoft,

¹ Сервисы Microsoft Office SharePoint (прим. переводчика).

² Электронный обмен данными (прим. переводчика).

выполнять EDI-операции или реализовывать шаблоны Enterprise Service Bus (ESB), используйте комплект инструментов ESB Toolkit для BizTalk Server.

Шаг 5 – Проектирование компонентов бизнес-слоя для поддержки рабочего процесса

Общей рекомендацией является создание рабочих процессов, состоящих из многошаговых или длительных процессов, реализованных в отдельных компонентах, и обеспечить обработку всех сбоев рабочих процессов через предоставление соответствующих исключений. При проектировании бизнес-процессов необходимо учесть вызовы методов, не требующие ответа или с длительным временем ответа. Если компонент должен выполнять заданный набор этапов последовательно и синхронно, используйте конвейерный шаблон. Если этапы могут выполняться асинхронно в любом порядке, используйте событийный шаблон.

Следующие разделы помогут разобраться с проектированием рабочих процессов с применением технологий, предлагаемых платформой Microsoft.

Windows Workflow Foundation

Для Windows Workflow Foundation (WF) можно разрабатывать следующие бизнес-компоненты: собственные рабочие процессы, действия и объекты состояний, а также собственные сервисы. То, какие компоненты понадобятся, зависит от стиля рабочего процесса и способа разработки. Далее описывается процесс создания трех основных типов рабочих процессов, собственных сервисов и разметки рабочего процесса с использованием WF:

- При проектировании **последовательных рабочих процессов** описываются или используются существующие классы **Activity** (Действие) (только код или разделение кода), определяются классы рабочего процесса (только код) и определяются компоненты бизнес-процесса, взаимодействующие с компонентами рабочего процесса (только код).
- При проектировании **конечных автоматов** описываются классы состояний, используемые для представления разных состояний процесса (только код или разделение кода), описываются или используются существующие события, запускающие изменения состояний (только код или разделение кода), описываются или используются существующие классы Activity, управляющие переходами состояний (только код или разделение кода), описываются классы рабочего процесса (только код) и определяются компоненты бизнес-процесса, взаимодействующие с компонентами рабочего процесса (только код).
- При проектировании **управляемых данными рабочих процессов** описываются или используются существующие классы Activity (только код или разделение кода), описываются или используются существующие классы **Condition** (Условие) для взаимодействия с поставщиками данных (только код или разделение кода), описываются специальные классы рабочего процесса (только код) и определяются компоненты бизнес-процесса, взаимодействующие с компонентами рабочего процесса (только код).

- При проектировании **собственных сервисов** описываются или используются существующие классы Activity для взаимодействия с сервисом, определяется интерфейс сервиса, поддерживающий необходимые операции, на базе проверенных практик проектируется сервис и выбирается соответствующий хост для сервиса (IIS, Workflow Appliance Software (WAS)¹ или **WorkflowServiceHost** (Хост сервиса рабочего процесса)).
- При проектировании **разметки рабочего процесса** может использоваться дизайнер Visual Studio (доступный как расширение Visual Studio 2005 и входящий в состав Visual Studio 2008 и последующих версий) или дизайнер SharePoint Designer для построения рабочих процессов на базе списков SharePoint. В качестве альтернативы разметка, связанная с продуктом стороннего производителя, может создаваться в дизайнере стороннего производителя, или можно вручную написать код разметки, используя соответствующий синтаксис XAML.

Сервер BizTalk

BizTalk может поддерживать разработку и с разделением кода, и с применением языков разметки. При использовании BizTalk может понадобиться спроектировать компоненты рабочего процесса, используемые в оркестровке BizTalk. Примерами таких компонентов рабочего процесса являются адаптеры и коннекторы. Также может понадобиться создать сервисы, обеспечивающие операции, необходимые рабочему процессу, или спроектировать бизнес-компоненты, обрабатывающие запросы для рабочих процессов BizTalk.

BizTalk может использоваться и без написания специальных компонентов, т.е. позволяет применить разработку с использованием разметки. Иначе говоря, если необходимо выполнять только простые операции, возможности сервера BizTalk прекрасно подойдут для преобразования сообщений и описания функций. Далее описывается процесс создания рабочих процессов с использованием BizTalk:

- При проектировании компонентов рабочего процесса для BizTalk определяется класс, реализующий соответствующий интерфейс, и затем этот класс регистрируется в COM.
- При проектировании компонентов бизнес-слоя для BizTalk определяются классы, поддерживающие необходимые операции. В случае необходимости, в компонентах бизнес-слоя можно запускать атомарные транзакции, вызываемые оркестровкой. Бизнес-слой должен проектироваться с использованием проверенных практик для обеспечения поддержки необходимых операций.
- При проектировании специальных сервисов описываются или используются существующие классы BizTalk для взаимодействия с сервисом, определяется интерфейс сервиса, поддерживающий необходимые операции; при проектировании сервиса используются проверенные практики и выбирается соответствующий хост для его размещения (IIS или WAS).

¹ ПО, содержащее рабочий процесс (прим. переводчика).

На рис. 1 представлена совместная работа всех этих компонентов для обеспечения поддержки рабочего процесса BizTalk.

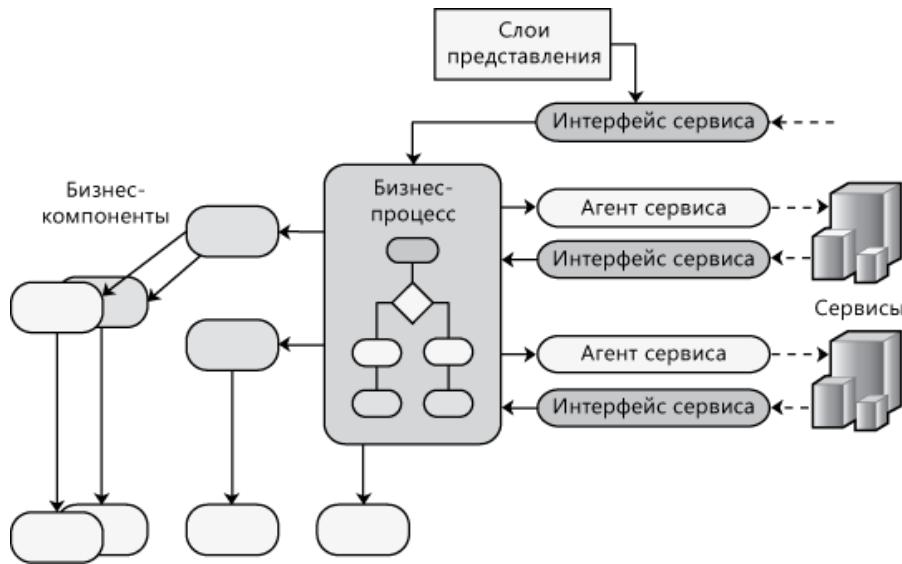


Рис. 13

Совместная работа компонентов для поддержки рабочего процесса BizTalk.

BizTalk с ESB

Комплект инструментов Microsoft Enterprise Service Bus (ESB) Toolkit расширяет BizTalk возможностями для создания подключенных сервисно-ориентированных корпоративных приложений. Комплект инструментов ESB Toolkit включает компоненты, поддерживающие и реализующие среду обмена сообщениями, упрощая тем самым построение основанных на сообщениях корпоративных приложений. Комплект инструментов предоставляет следующие компоненты:

- **Веб-сервисы ESB.** Обеспечивают основные возможности Microsoft ESB Toolkit. Предоставляются следующие сервисы:
 - Маршрутизирующие Веб-сервисы (Itinerary on-ramp Web services), принимающие внешние сообщения и отсылающие их для дальнейшей обработки обработки.
 - Веб-сервис преобразования адресов (Resolver Web service), позволяющий внешним приложениям вызывать инфраструктуру преобразования адресов (Resolver Framework) для поиска конечных точек ESB на основании механизмов разрешения, поддерживаемых инфраструктурой преобразования адресов, таких как политики бизнес-правил, регистрации UDDI, статический вызов, интерфейс **WS-MetadataExchange** (Обмен метаданными) и на основании содержимого сообщения.
 - Веб-сервис преобразования (Transformation Web service) обеспечивает функции для преобразования содержимого сообщения и выполнения бизнес-требований. Преобразованиям может подвергаться

непосредственно входящее сообщение или сообщения, извлекаемые из базы данных **MessageBox** (Хранилище сообщений) BizTalk.

- Веб-сервис обработки исключений (Exception Handling Web service) принимает сообщения об исключениях из внешних источников и публикует их в Инфраструктуре управления исключениями ESB (ESB Exception Management Framework). Оттуда конвейер обработки сбоев будет нормализовать, отслеживать и публиковать сообщение об исключении в Портале управления ESB (ESB Management Portal).
 - Веб-сервис UDDI позволяет приложениям и пользователям выполнять поиск конечных точек по имени сервиса, поставщику услуг или категории деловой активности; также с его помощью приложения и пользователи могут управлять поставщиками услуг, сервисами и категориями, хранящимися в хранилище UDDI.
 - Веб-сервис обслуживания BizTalk предоставляет сведения о хостах BizTalk, оркестровке, приложениях и состоянии.
- **Портал управления ESB.** Обеспечивает такие возможности как отслеживание исключений и сбоев, повторная передача сообщений, предупреждения и уведомления, интеграция с UDDI, составление отчетов и аналитика, возможности настройки.
 - **Компоненты конвейера взаимодействий ESB.** К ним относятся Сервис обмена сообщениями Java (Java Messaging Service, JMS) и компоненты пространств имен для использования в конвейерах BizTalk.
 - **Инфраструктура управления исключениями.** Может перехватывать исключения от подсистем обмена сообщениями и оркестровки BizTalk и формировать сообщения о сбоях.
 - **Инфраструктура поставщика преобразования адресов и адаптера ESB.** Реализует подключаемую и настраиваемую архитектуру для динамически разрешаемых конечных точек и трансформаций, а также для маршрутизации сообщений.
 - **Обработка маршрутов.** Этот механизм обеспечивает возможность упрощенного динамического описания, передачи и выполнения множества вызовов сервисов или маршрутизации/преобразования запросов.
 - **Примеры приложений ESB.** Демонстрируют применение комплекта инструментов Microsoft ESB Toolkit, показывая пути использования предоставляемых им возможностей в собственных приложениях SOA и ESB.

Совместное использование Windows Workflow Foundation и BizTalk

Во многих ситуациях Windows Workflow Foundation (WF) или BizTalk по отдельности не могут обеспечить полную поддержку необходимых рабочих процессов. В этом случае в одном приложении можно использовать необходимые функции обоих решений для рабочих процессов. Сочетайте WF и BizTalk:

- если хотите реализовать с помощью только кода рабочий процесс бизнес-правил с использованием компонентов WF, взаимодействующий с подсистемой управления бизнес-правилами BizTalk;
- если имеются существующие рабочие процессы WF, которые должны вызываться из системы оркестровки BizTalk;
- при создании рабочего процесса SharePoint, который должен выполнять оркестровку BizTalk;
- если рабочий процесс WF должен интегрироваться с гетерогенными или устаревшими системами.

Дополнительные источники

Электронная версия списка используемых источников по технологиям проектирования рабочих процессов доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Introduction to Programming Windows Workflow Foundation*» (Введение в программирование Windows Workflow Foundation) по адресу <http://msdn.microsoft.com/en-us/library/ms734696.aspx>
- «*Microsoft BizTalk ESB Toolkit*» (Комплект инструментов Microsoft BizTalk ESB) по адресу <http://msdn.microsoft.com/en-us/library/dd897973.aspx>

15

Проектирование компонентов слоя доступа к данным

Обзор

Компонентами слоя доступа к данным являются компоненты, обеспечивающие функциональность доступа к данным, размещаемым в системе, и компоненты агентов сервисов, обеспечивающие функциональность доступа к данным, которые предоставляются другими серверными системами через Веб-сервисы. Кроме того, слой доступа к данным также может включать компоненты, обеспечивающие вспомогательные функции и утилиты.

Эта глава описывает основные этапы проектирования компонентов данных. Первый шаг – выявление ограничений связанных данных, к которым предполагается выполнять доступ, что поможет выбрать соответствующую технологию доступа к данным. Следующий шаг – выбор стратегии сопоставления и затем определение подхода к реализации доступа к данным, куда входит определение используемых бизнес-сущностей и их формата. После этого можно принять решение о том, как будет выполняться подключение компонентов доступа к данным и источнику данных. Наконец, вырабатывается стратегия обработки ошибок для управления исключениями источника данных.

Шаг 1 – Выбор технологии доступа к данным

При выборе технологии доступа к данным необходимо учесть тип данных, с которыми предполагается работать, и то, как эти данные будут обрабатываться в приложении. Для каждого конкретного сценария есть наиболее подходящие технологии. Чтобы правильно выбрать технологию доступа к данным, соответствующую сценариям создаваемого приложения, руководствуйтесь следующими рекомендациями:

- **ADO.NET Entity Framework¹.** Используйте ADO.NET Entity Framework (EF), если хотите создать модель данных и соотнести ее с реляционной базой данных; соотнести один класс с множеством таблиц, используя наследование; или выполнять запросы к реляционным хранилищам, не входящим в семейство продуктов Microsoft SQL Server. EF подойдет, если имеется объектная модель, которую необходимо

¹ Инфраструктура сущностей ADO.NET (прим. переводчика).

соответствии с реляционной моделью, используя гибкую схему, и необходима гибкость, обеспечивающая возможность отделения схемы сопоставления от объектной модели. При использовании EF также рассмотрите возможность применения:

- **LINQ to Entities.** Используйте LINQ to Entities, если необходимо выполнять запросы через строго типизированные сущности или запрашивать реляционные данные, используя синтаксис LINQ.
- **ADO.NET Data Services Framework**¹. ADO.NET Data Services построена на базе EF и позволяет предоставлять части модели сущностей (Entity Model) через REST-интерфейс. Используйте ADO.NET Data Services Framework, если разрабатываете RIA или n-уровневое насыщенное клиентское приложение и хотите выполнять доступ к данным через ресурсно-ориентированный интерфейс сервиса.
- **ADO.NET Core**². Используйте ADO.NET Core, если для обеспечения полного управления доступом к данным в приложении необходим низкоуровневый API; если хотите использовать уже сделанные инвестиции в ADO.NET-решения; если используете традиционную логику доступа к данным. ADO.NET Core подойдет, если нет необходимости в дополнительной функциональности, предлагаемой другими технологиями доступа к данным, или если разрабатываемое приложение должно поддерживать сценарии доступа к данным без постоянного подключения.
- **ADO.NET Sync Services**³. Используйте ADO.NET Sync Services при проектировании приложения, которое должно поддерживать сценарии без постоянного подключения или требует синхронизации баз данных.
- **LINQ to XML.** Используйте LINQ to XML, если приложение работает с XML-данными, запросы к которым необходимо выполнять, применяя синтаксис LINQ.

Более подробно технологии доступа к данным, предлагаемые платформой Microsoft, рассматриваются в приложении С, «[Матрица технологий слоя доступа к данным](#)».

Шаг 2 – Принятие решения о методе извлечения и хранения бизнес-объектов источника данных

Определившись с требованиями источника данных, необходимо выбрать стратегию заполнения бизнес-объектов или бизнес-сущностей данными из хранилища данных и сохранения данных бизнес-объектов или бизнес-сущностей в хранилище данных. Обычно интерфейсы объектно-ориентированной модели данных и реляционного хранилища данных не согласованы, что порой усложняет передачу данных между ними. Существует ряд подходов к решению этой проблемы, которые отличаются между собой используемыми типами данных, структурой, технологиями обеспечения транзакций и способами обработки данных. Самые

¹ Инфраструктура сервисов данных ADO.NET (прим. переводчика).

² Базовый ADO.NET (прим. переводчика).

³ Сервисы синхронизации ADO.NET (прим. переводчика).

распространенные подходы используют инструменты и инфраструктуры объектно-реляционного сопоставления (Object/Relational Mapping, O/RM). Используемый в приложении тип сущности является основным фактором при принятии решения о способе сопоставления сущностей со структурами источника данных. Следующие рекомендации помогут выбрать технику извлечения и сохранения бизнес-объектов в хранилище данных:

- Используйте инфраструктуру O/RM, обеспечивающую преобразования между сущностями предметной области и базы данных. При работе в среде «greenfield», когда вы имеете полный контроль над схемой базы данных, инструмент O/RM может обеспечить формирование схемы для поддержки объектной модели и сопоставления сущностей базы данных и предметной области. При работе в среде «brownfield», где вам приходится использовать предлагаемую схему базы данных, инструмент O/RM поможет сопоставить модель предметной области и реляционную модель.
- В объектно-ориентированном проектировании обычно используется модель предметной области, шаблон, основанный на моделировании сущностей соответственно объектам предметной области. Методики проектирования на основе предметной области подробно рассматриваются в главе 13, [«Проектирование бизнес-сущностей»](#).
- Правильно сгруппируйте сущности, чтобы достичь высокого уровня связности. Для этого может понадобиться ввести в модель предметной области дополнительные объекты и сгруппировать взаимосвязанные сущности в сводные корни.
- При работе с Веб-приложениями или сервисами группируйте сущности и обеспечивайте опции для частичной загрузки сущностей предметной области только необходимыми данными. Это сократит использование ресурсов за счет того, что не придется удерживать в памяти инициализированные модели предметной области для каждого пользователя, и позволит приложениям справляться с более высокой нагрузкой пользователей.

Шаг 3 – Выбор способа подключения к источнику данных

Зная, как компоненты доступа к данным сопоставляются с источником данных, можно принять решение о том, как будет выполняться подключение к источнику данных, реализовываться защита пользовательских учетных данных и выполняться транзакции. Рекомендации в следующих разделах помогут правильно выбрать подход:

- [Подключения](#)
- [Пул подключений](#)
- [Транзакции и параллелизм](#)

Подключения

Подключения к источникам данных – это фундаментальная часть слоя доступа к данным. Слой доступа к данным должен координировать все подключения к источнику данных, используя для этого инфраструктуру доступа к данным. На создание и управление подключениями приходится расходовать ценные ресурсы и в слое доступа к данным, и в самом источнике данных. Следующие рекомендации помогут выбрать соответствующую технику подключения к источникам данных:

- Открывайте подключения к источнику данных как можно позже и закрывайте их как можно раньше. Это обеспечит блокировку ресурсов лишь на короткие промежутки времени и сделает их более доступными для других процессов. Для малоизменяющихся данных используйте оптимистическую блокировку, это снизит издержки на блокировку строк базы данных, включая затраты на подключение, которое должно оставаться открытым в течение блокировки.
- Там где это возможно, осуществляйте транзакции через одно подключение. Это позволит использовать возможности транзакций ADO.NET без внешних сервисов координации распределенных транзакций.
- Используйте пул подключений и оптимизируйте производительность на основании результатов нагружочных тестов. Рассмотрите возможность настройки уровней изоляции подключений для запросов к данным. В приложении с высокими требованиями к пропускной способности некоторые операции с данными могут выполняться на более низких уровнях изоляции, чем остальные операции транзакции. Сочетание уровней изоляции может иметь негативное влияние на согласованность данных, поэтому этот вариант необходимо тщательно анализировать для каждого конкретного случая в отдельности.
- По соображениям безопасности избегайте использования системных или пользовательских DSN для хранения данных подключения.
- Предусмотрите логику повторного подключения для случаев разрыва соединения с источником данных или его закрытия по истечении времени ожидания.
- По возможности используйте пакетные команды, что позволит сократить количество обращений к серверу базы данных.

Другой важный аспект, который необходимо учесть – требования к безопасности в связи с доступом к источнику данных. Иначе говоря, необходимо продумать, как источник данных будет аутентифицировать компоненты доступа к данным, и каковы будут требования к авторизации. Следующие рекомендации обеспечат проектирование безопасного подхода для подключения к источникам данных:

- Предпочтительнее использовать аутентификацию Windows, а не аутентификацию SQL Server. При работе с Microsoft SQL Server используйте аутентификацию Windows с доверенной подсистемой.

- При использовании аутентификации SQL применяйте учетные записи с надежными паролями; с помощью ролей базы данных ограничьте права доступа каждой учетной записи в рамках SQL Server; добавьте ACL во все файлы, используемые для хранения строк подключения; и шифруйте строки подключения в конфигурационных файлах.
- Используйте учетные записи с наименьшими правами доступа к базе данных и требуйте от вызывающей стороны предоставлять слою данных идентификационные данные для целей аудита.
- Не храните пароли для проверки пользователей в базе данных, ни в виде открытого текста, ни в зашифрованном виде. Храните хеши паролей с шумом (случайные разряды, используемые как один из параметров в функции хеширования).
- При использовании SQL-выражений для доступа к источнику данных четко обозначьте границы доверия и применяйте параметризованные запросы, а не конкатенацию строк. Это обеспечит защиту от атак через внедрение SQL-кода.
- Защитите конфиденциальные данные, передаваемые по сети к и от SQL Server. Не забывайте, что аутентификация Windows обеспечивает защиту учетных данных, но не данных приложения. Для защиты данных в канале передачи используйте протоколы IPSec или SSL.

Пул подключений

Пул подключений обеспечивает возможность повторного использования приложением подключения из пула или создания нового подключения и добавления его в пул в случае недоступности подходящего подключения. Когда приложение закрывает подключение, оно возвращается в пул, и базовое подключение остается открытым. Это означает, что ADO.NET не надо создавать новое подключение к источнику данных, и каждый раз открывать его заново. Хотя использование пула открытых подключений потребляет ресурсы, это обеспечивает сокращение задержек при доступе к данным и повышает эффективность выполнения приложения в случае доступности подходящих соединений из пула. Рассмотрим другие вопросы использования пула подключений:

- Чтобы максимально повысить эффективность пула подключений, используйте модель безопасности доверенная подсистема и по возможности избегайте олицетворения. Использование минимального числа учетных записей повышает вероятность повторного использования подключения из пула и сокращает шансы переполнения пула подключений. Если каждый вызов использует разные учетные данные, ADO.NET приходится каждый раз создавать новое подключение.
- Подключения, которые остаются открытыми в течение длительного периода времени, могут удерживать ресурсы на сервере. Обычная причина этого – раннее открытие подключений и позднее их закрытие (например, когда подключение не закрывается явно и не удаляется до тех пор, пока не выходит за рамки области действия).

- Подключения могут оставаться открытыми в течение длительного времени при использовании объектов **DataReader**, которые являются действительными только пока подключение открыто.
-

Транзакции и параллелизм

При наличии в приложении ответственных операций используйте транзакции для их выполнения. Транзакции позволяют выполнять связанные действия с базой данных как единую операцию и гарантировать тем самым целостность базы данных. Транзакция считается завершенной, если все входящие в нее действия выполнены, после этого внесенные в ходе этой транзакции изменения базы данных становятся постоянными. Транзакции поддерживают отмену (откат) действий в случае возникновения ошибки, что помогает сохранить целостность данных в базе данных. Следующие рекомендации помогут при проектировании транзакций:

- При проектировании доступа к одному источнику данных по возможности используйте транзакции на базе подключения. При использовании создаваемых вручную или явных транзакций реализуйте транзакцию в хранимой процедуре. Если не можете использовать транзакции, реализуйте компенсационные методы для возвращения хранилища данных в предыдущее состояние.
- При использовании длительных атомарных транзакций избегайте слишком долгого удержания блокировок. В подобных сценариях лучше использовать компенсационные блокировки. Если для завершения транзакции требуется длительное время, используйте асинхронные транзакции, осуществляющие обратный вызов клиента по завершении. Также для параллельно выполняющихся приложений, осуществляющих большое число транзакций, используйте технологию MARS (множество активных результирующих множеств), это позволит избежать потенциальных взаимоблокировок.
- Если вероятность возникновения конфликта данных из-за их одновременного изменения несколькими пользователями низка (например, когда пользователи, преимущественно, добавляют данные или редактируют разные строки), используйте оптимистическую блокировку, при которой действительным считается последнее обновление. Если вероятность возникновения конфликта данных из-за их одновременного изменения несколькими пользователями высока (например, когда пользователи, преимущественно, редактируют одни и те же строки), используйте пессимистическую блокировку, при которой обновление может применяться только к последней версии данных. Также учтите вопросы параллельной обработки при доступе к статическим данным приложения или при использовании потоков для осуществления асинхронных операций. Статические данные по природе своей не являются потокобезопасными, т.е. изменения, вносимые в такие данные в одном потоке, будут оказывать влияние на другие потоки, использующие эти же данные.
- Транзакции должны быть максимально короткими, это обеспечит самые короткие блокировки и улучшит условия параллельной работы. Однако не следует забывать, что короткие и простые транзакции могут привести к созданию слишком

детализированного интерфейса, которому для завершения одной операции понадобится делать множество вызовов.

- Используйте соответствующий уровень изоляции. Необходимо найти баланс между непротиворечивостью данных и конкуренцией за ресурсы. Более высокий уровень изоляции обеспечит более высокую непротиворечивость данных за счет общего снижения возможностей для параллельной обработки. Более низкий уровень изоляции, снижая конкуренцию за ресурсы, улучшает производительность ценой потери непротиворечивости данных.

Существует три общих типа поддержки транзакций:

- Классы пространства имен **System.Transactions** обеспечивают поддержку явных и неявных транзакций как часть .NET Framework. Используйте **System.Transactions** при разработке нового приложения, требующего поддержку транзакций, или при наличии транзакций, охватывающих несколько диспетчеров недолгосрочных ресурсов. Для реализации большинства транзакций рекомендуется использовать явную модель, которую обеспечивает объект **TransactionScope** пространства имен **System.Transactions**. Хотя неявные транзакции не настолько быстрые, как созданные вручную, или явные, но их проще создавать, и они обеспечивают решения промежуточного уровня, гибкие и более простые в обслуживании. Если не желаете использовать неявную модель для транзакций, можно реализовать создание транзакций вручную, используя класс **Transaction** пространства имен **System.Transactions**.
- **Транзакции ADO.NET**, использующие единственное подключение к базе данных. Это наиболее эффективный подход для управляемых клиентом транзакций с одним хранилищем данных. Выбирайте транзакции ADO.NET, если расширяете приложение, уже использующее транзакции ADO.NET; если используете поставщиков ADO.NET для доступа к базе данных и транзакции выполняются только к одной базе данных; или если развертываете приложение в среде, не поддерживающей версию 2.0 .NET Framework. Команды ADO.NET обеспечивают начало, фиксацию и откат операций, осуществляемых в рамках транзакции.
- **Транзакции T-SQL (базы данных)**, управляемые командами, выполняемыми в базе данных. Это наиболее эффективный способ реализации управляемых сервером транзакций с одним хранилищем данных, при котором база данных контролирует все аспекты транзакции. Используйте транзакции базы данных при разработке хранимых процедур, инкапсулирующих все изменения, которые должны быть выполнены транзакцией, или при наличии множества приложений, использующих одни и те же хранимые процедуры, когда требования транзакции могут быть инкапсульированы в хранимые процедуры.

Шаг 4 – Выработка стратегий обработки ошибок источника данных

На данном этапе должна быть выработана общая стратегия обработки ошибок источников данных. Все исключения, связанные с источниками данных, должны перехватываться слоем доступа к данным. Исключения, касающиеся самих данных, а также ошибки доступа к источнику данных и истечения времени ожидания, должны обрабатываться в этом слое и передаваться в другие слои, только если эти сбои оказывают влияние на время отклика или функциональность приложения. Рекомендации, приведенные в следующих разделах, помогут правильно выбрать соответствующий подход:

- [Исключения](#)
 - [Логика повтора попыток](#)
 - [Истечение времени ожидания](#)
-

Исключения

Стратегия централизованного управления исключениями обеспечит единообразие при обработке исключений. Обработка исключений относится к сквозной функциональности, поэтому эту логику рекомендуется реализовывать в отдельных компонентах, которые могут использоваться совместно слоями и уровнями приложения. Особое внимание необходимо уделить исключениям, распространяющимся через границы доверия и на другие слои или уровни, и необрабатываемым исключениям, чтобы они не приводили к нарушению надежности приложения или раскрытию конфиденциальных данных приложения. Следующий подход поможет при проектировании стратегии обработки исключений:

- Определите, какие исключения должны перехватываться и обрабатываться слоем доступа к данным. Проверки на наличие взаимоблокировок, проблем с подключениями и нежестких блокировок обычно могут проводиться в рамках слоя данных.
 - Рассмотрите возможность реализации повторных попыток для операций, в которых могут возникать ошибки, связанные с источником данных или с истечением времени ожидания, но только в случаях, когда это безопасно.
 - Разрабатывайте соответствующую стратегию распространения исключений. Например, обеспечьте возможность передачи исключений в граничные слои, где они могут быть запротоколированы и преобразованы соответствующим образом для передачи в следующий слой.
 - Выработайте соответствующую стратегию протоколирования и уведомления о критических ошибках и исключениях, обеспечивая сокрытие конфиденциальных данных.
 - Используйте существующие инструменты, такие как Enterprise Library от группы patterns & practices, для реализации единообразной стратегии обработки и управления исключениями.
-

Логика повтора попыток

Предусмотрите логику повтора попыток для обработки ошибок, возникающих при переходе на другой ресурс в случае сбоя сервера или базы данных. Логика повтора должна перехватывать все ошибки, возникающие при подключении к базе данных или выполнении команд (запросов или транзакций). Причин формирования ошибки может быть множество. При возникновении ошибки компонент данных должен восстановить подключение, закрыв существующие подключения и создав новое, и затем повторить команды, давшие сбой, если это необходимо. Повторные попытки должны выполняться лишь определенное количество раз, после чего, в случае их неудачи, попытки выполнить команды прекращаются, и возвращается исключение. Все запросы и любые последующие повторные попытки должны выполнятся асинхронно, это обеспечит невозможность создания ситуации, когда приложение не отвечает.

Истечение времени ожидания

Очень важно правильно выбрать время ожидания для подключения и команды. Задание времени ожидания для подключения или команды, превышающего время ожидания клиента (например, в случае со временем ожидания запроса Веб-приложения, браузера или Веб-сервера), может привести к тому, что время ожидания запроса клиента истечет до того, как подключение к базе данных будет открыто. Задание недостаточного времени ожидания приведет к тому, что обработчик ошибок начнет выполнять логику повтора попыток. Если время ожидания истекает во время выполнения транзакции, в случае использования пула подключений ресурсы базы данных могут остаться заблокированными после закрытия подключения. В таких случаях, чтобы закрытое подключение не возвращалось в пул, оно должно удаляться. Это обеспечивает откат транзакции и высвобождение ресурсов базы данных.

Шаг 5 – Проектирование объектов агентов сервисов (необязательный)

Агенты сервисов – это объекты, которые управляют семантикой взаимодействия с внешними сервисами, изолируют приложение от специфических особенностей взаимодействия с разными сервисами и обеспечивают дополнительные сервисы, такие как сопоставление формата данных, предоставляемого сервисом, и формата, требуемого приложением. Они также могут реализовывать кэширование и поддержку работы в автономном режиме или неустойчивого подключения. Выполните разработку объектов агентов сервисов в следующей последовательности:

1. Используйте соответствующий инструмент для добавления ссылки на сервис. Это обеспечит формирование прокси-классов и классов данных, представляющих контракт данных сервиса.
2. Определите, как сервис будет использоваться в приложении. Для большинства приложений агент сервиса выступает в роли уровня абстракции между бизнес-слоем и удаленным сервисом и может обеспечивать единообразный интерфейс независимо от формата данных. В небольших приложениях слой представления может выполнять доступ к агенту сервиса напрямую.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- «*.NET Data Access Architecture Guide*» по адресу <http://msdn.microsoft.com/en-us/library/ms978510.aspx>.
 - «*Data Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/ms998446.aspx>.
 - «*Designing Data Tier Components and Passing Data Through Tiers*» по адресу <http://msdn.microsoft.com/en-us/library/ms978496.aspx>.
-

16

Показатели качества

Обзор

Показатели качества – это общие факторы, оказывающие влияние на поведение во время выполнения, дизайн системы и взаимодействие с пользователем. Они представляют функциональные области, потенциально влияющие на все приложение со всеми его слоями и уровнями. Некоторые из этих показателей касаются общего дизайна системы, тогда как другие относятся только ко времени выполнения, времени проектирования или связаны только с вопросами взаимодействия с пользователем. Степень, с которой приложение реализует желаемое сочетание показателей качества, таких как удобство и простота использования, производительность, надежность и безопасность, свидетельствует об успешности дизайна и общем качестве программного приложения.

При выполнении любого из требований показателей качества во время проектирования приложений важно учесть, какое влияние это может оказать на другие требования. Необходимо проанализировать соотношение выгод и потерь для совокупности множества показателей качества. Важность или приоритетность каждого из показателей качества в разных системах разная; например, возможность взаимодействия с другими системами, как правило, не так важна для коробочного приложения индивидуального использования, чем для бизнес-системы (LOB).

В данной главе перечислены и описаны показатели качества, которые должны быть учтены при проектировании приложения. Чтобы работать с данной главой максимально эффективно, с помощью приведенной ниже таблицы разберитесь, как показатели качества сопоставляются с факторами качества системы и приложения, и затем ознакомьтесь с описаниями всех показателей качества. Затем перейдите к разделам с основными рекомендациями по каждому из показателей качества, чтобы понять, как каждый из них может влиять на дизайн, и какие решения необходимо принять для решения этих вопросов. Имейте в виду, что приводимый здесь список показателей качества не полный, но он является хорошим стартом и поможет научиться задавать правильные вопросы об архитектуре.

Общие показатели качества

В следующей таблице описываются показатели качества, рассматриваемые в данной главе. Они разбиты на четыре категории, касающиеся качества дизайна, качеств времени

выполнения, системы и взаимодействия с пользователем. Используйте данную таблицу, чтобы понять, какое значение имеет каждый из атрибутов с точки зрения дизайна приложения.

Категория	Показатель качества	Описание
Качества дизайна	Концептуальная целостность	Концептуальная целостность определяет согласованность и связность дизайна в целом. Сюда относится и то, как спроектированы компоненты или модули, и такие факторы как стиль написания кода и именование переменных.
	Удобство и простота обслуживания	Удобство и простота обслуживания – это способность системы изменяться. Это касается изменения компонентов, сервисов, функций и интерфейсов при добавлении или изменении функциональности, исправлении ошибок и реализации новых бизнес-требований.
	Возможность повторного использования	Возможность повторного использования определяет пригодность компонентов и подсистем к использованию в других приложениях и сценариях. Возможность повторного использования обеспечивает снижение дублирования компонентов и также сокращение времени, затрачиваемого на реализацию.
Качества времени выполнения	Доступность	Доступность определяет, какую часть времени система функциональна и работает. Доступность может быть измерена как процентное соотношение времени простоя системы за заданный промежуток времени. На доступность оказывают влияние ошибки системы, проблемы инфраструктуры, злонамеренные атаки и нагрузка системы.
	Возможность взаимодействия	Возможность взаимодействия – это способность системы или разных систем успешно работать через взаимодействие и обмен данными с другими внешними системами, созданными и выполняемыми внешними сторонами. Способная к взаимодействию система упрощает обмен и повторное использование данных, как внутри, так и вне ее границ.
	Управляемость	Управляемость определяет, насколько просто системным администраторам управлять приложением, как правило, посредством достаточного и полезного инструментария, предоставляемого для использования в системах мониторинга, а также для отладки и настройки производительности.

	<i>Производительность</i>	Производительность – это показатель, характеризующий скорость, с которой система выполняет любое действие в заданный промежуток времени. Производительность измеряется в показателях задержки или пропускной способности. Задержка – это время, необходимое для ответа на любое событие. Пропускная способность – это число событий, имеющих место в заданный промежуток времени.
	<i>Надежность</i>	Надежность – это способность системы сохранять работоспособность в течение некоторого времени. Надежность определяется как вероятность того, что система сможет выполнять предусмотренные функции в течение заданного промежутка времени.
	<i>Масштабируемость</i>	Масштабируемость – это способность системы справляться с увеличением нагрузки без влияния на ее производительность или способность легко расширяться.
	<i>Безопасность</i>	Безопасность – это способность системы предотвращать злонамеренные или случайные действия, не предусмотренные при проектировании, или не допускать разглашение или утрату данных. Безопасная система должна защищать ресурсы и предотвращать несанкционированные изменения данных.
Качества системы	<i>Обеспеченность технической поддержкой</i>	Обеспеченность технической поддержкой – это способность системы предоставлять сведения, необходимые для выявления и разрешения проблем при некорректной работе.
	<i>Тестируемость</i>	Тестируемость – это мера того, насколько просто создать критерий проверки для системы и ее компонентов и выполнить эти тесты, чтобы определить, отвечает ли система данному критерию. Хорошая тестируемость означает большую вероятность того, что сбои в системе могут быть своевременно и эффективно изолированы.
Качества взаимодействия с пользователем	<i>Удобство и простота использования</i>	Удобство и простота использования определяет, насколько приложение соответствует требованиям пользователя и потребителя с точки зрения понятности, простоты локализации и глобализации, удобства доступа для пользователей с физическими недостатками и обеспечения хорошего взаимодействия с пользователем в общем.

В следующих разделах все эти показатели качества описываются более подробно. Кроме того, предоставляются рекомендации по ключевым вопросам и решениям, которые должны быть приняты для каждого из них:

- [Доступность](#)
 - [Концептуальная целостность](#)
 - [Возможность взаимодействия](#)
 - [Удобство и простота обслуживания](#)
 - [Управляемость](#)
 - [Производительность](#)
 - [Надежность](#)
 - [Возможность повторного использования](#)
 - [Масштабируемость](#)
 - [Безопасность](#)
 - [Обеспеченность технической поддержкой](#)
 - [Тестируемость](#)
 - [Взаимодействие с пользователем / удобство и простота использования](#)
-

Доступность

Доступность определяет ту долю времени, когда система функциональна и работает. Она может быть измерена в процентном соотношении как доля общего времени простой системы за заданный период. На доступность оказывают влияние ошибки системы, проблемы инфраструктуры, злонамеренные атаки и нагрузка системы. Основные проблемы доступности:

- Физический уровень, такой как сервер базы данных или сервер приложений, может дать сбой или не отвечать, приводя к общему сбою системы. Продумайте реализацию поддержки обработки отказов для уровней системы. Например, с помощью механизмов балансировки сетевой нагрузки (Network Load Balancing) для Веб-серверов распределите нагрузку и не допустите направление запросов на нефункционирующий сервер. Также используйте технологию RAID для смягчения негативного влияния на систему при сбое диска. Рассмотрите возможность размещения резервного узла в другой точке планеты на случай стихийных бедствий, таких как землетрясения или торнадо.
- Атаки типа отказ в обслуживании (Denial of Service, DoS), которые препятствуют доступу к системе авторизованных пользователей, могут быть причиной прерывания операций, если система не в состоянии своевременно справиться с большими нагрузками, что часто происходит из-за недостаточности времени на обработку либо из-за конфигурации сети и перегрузки системы запросами. Чтобы максимально

сократить негативное влияние атак типа DoS, сократите поверхность атаки, выявляйте злонамеренное поведение, используйте инструментарий приложения для предоставления непредусмотренного поведения и реализуйте исчерпывающую проверку данных. Устойчивость системы помогут повысить шаблоны Circuit Breaker (Прерыватель) и Bulkhead (Перемычка).

- Негативно сказаться на доступности может несоответствующее использование ресурсов. Например, слишком ранний запрос и слишком долгое удержание ресурсов приводит к их нехватке и неспособности системы обрабатывать дополнительные параллельные запросы пользователей.
- Дефекты или ошибки приложения могут стать причиной общего сбоя системы. Проектируйте соответствующую обработку исключений, чтобы сократить количество сбоев приложений, приводящих к потере работоспособности системы.
- Частые обновления, такие как обновления безопасности или пользовательского приложения, могут понизить доступность системы. Продумайте стратегию обновлений во время выполнения.
- Сбой сети может привести к недоступности приложения. Продумайте, как будете обеспечивать работу системы при наличии ненадежных сетевых подключений, например, путем создания клиентов с возможностью работы в условиях без постоянного подключения.
- Обозначьте границы доверия в своем приложении и обеспечьте, чтобы подсистемы использовали некоторую форму управления доступом или межсетевой экран, а также реализуйте расширенную проверку данных. Это позволит повысить устойчивость и доступность.

Концептуальная целостность

Концептуальная целостность определяет согласованность и связность дизайна в целом. Сюда относится то, как спроектированы компоненты или модули, а также такие факторы, как стиль написания кода и именования переменных. Связную систему проще обслуживать, поскольку заранее известно, что согласуется с общим дизайном. И, наоборот, система без концептуальной целостности подвержена постоянным изменениям интерфейсов, частым сменам модулей, ей не свойственно единообразие выполнения задач. Основные проблемы концептуальной целостности:

- Смешение разных функциональных областей. Выявите функциональные области и правильно сгруппируйте их в логические слои представления, доступа к данным, сервисов и бизнес-слой.
- Несогласованные или плохо управляемые процессы разработки. Проведите оценку эффективности применения методик Управления жизненным циклом приложения (Application Lifecycle Management, ALM) и используйте испытанные и протестированные инструменты и методики разработки.
- Недостаточный уровень сотрудничества и обмена информацией между разными группами, принимающими участие в жизненном цикле приложения. Реализуйте

процесс разработки, интегрированный с инструментами разработки, для упрощения рабочего процесса, обмена сведениями и совместной работы.

- Недостаточное использование стандартов проектирования и написания кода. Используйте опубликованные рекомендации по стандартам проектирования и написания кода и выполняйте анализ кода в процессе разработки, что обеспечит следование рекомендациям.
- Требования поддержки существующих (устаревших) систем могут препятствовать как реструктуризации, так и переходу на новые платформы или парадигмы. Продумайте возможности ухода от устаревших технологий и изоляции приложения от внешних зависимостей. Например, реализация шаблона проектирования *Gateway* (Шлюз) обеспечит интеграцию с устаревшими системами.

Возможность взаимодействия

Возможность взаимодействия – это способность системы или разных систем успешно работать через взаимодействие и обмен данными с другими внешними системами, созданными и выполняемыми внешними сторонами. Способная к взаимодействию система упрощает обмен и повторное использование данных, как внутри, так и вне ее границ. Протоколы связи, интерфейсы и форматы данных – все это основные средства обеспечения возможности взаимодействия. Еще одним важным аспектом при проектировании способной к взаимодействию системы является стандартизация. Рассмотрим основные проблемы, связанные с обеспечением возможности взаимодействия:

- Взаимодействие с внешними или устаревшими системами, использующими другие форматы данных. Продумайте возможности реализации взаимодействия систем, которые развиваются отдельно или даже могут быть заменены. Например, для связи с внешними или устаревшими системами используйте оркестровку, адаптеры и преобразование данных при их передаче между системами; или примените каноническую модель данных для реализации работы с множеством разных форматов данных.
- Размытие границ, что позволяет артефактам одной системы проникать в другую. Продумайте возможности изоляции систем через использование интерфейсов сервисов и/или сопоставления уровней. Например, для обеспечения поддержки возможности взаимодействия с другими системами предоставьте сервисы с помощью интерфейсов, использующих XML или стандартные типы. Проектируйте связные и слабо связанные компоненты для обеспечения максимальной гибкости, упрощения замены и обеспечения возможности повторного использования.
- Недостаточное следование стандартам. Ознакомьтесь с формальными и реально используемыми стандартами предметной области, в которой работаете, и используйте один из них, а не создавайте что-то новое и особое.

Удобство и простота обслуживания

Удобство и простота обслуживания – способность системы изменяться. Это касается изменения компонентов, сервисов, функций и интерфейсов при добавлении или изменении функциональности, исправлении ошибок и реализации новых бизнес-требований. Удобство и простота обслуживания также может влиять на время, необходимое на восстановление работоспособности системы после поломки или снятия с эксплуатации для обновления. Улучшение удобства и простоты обслуживания системы может повысить доступность и снизить влияние дефектов времени выполнения. Удобство и простота обслуживания приложения часто является функцией всех параметров качества в целом, но существует ряд ключевых факторов, которые могут напрямую влиять на удобство и простоту обслуживания:

- Слишком большая зависимость между компонентами и слоями и несоответствующее связывание с конкретными классами усложняет замену, обновления и внесение изменений и может привести к тому, что изменение отдельных классов будет влиять на систему в целом. Правильно разделяйте системы на слои или функциональные области, которые четко обозначают функциональность UI, бизнес-процессов и доступа к данным системы. Реализуйте зависимости слоев с помощью абстракций (таких как абстрактные классы и интерфейсы), а не конкретных классов, и максимально сократите количество зависимостей между компонентами и слоями.
- Использование прямого взаимодействия является препятствием при изменении физического развертывания компонентов и слоев. Правильно выбирайте модель, формат и протокол связи. Проектируя интерфейсы, обеспечивающие возможность использования подключаемых модулей или адаптеров для улучшения гибкости и расширяемости, создавайте подключаемую архитектуру, которую легко обновлять, обслуживать и тестировать.
- Использование собственных реализаций функций, таких как аутентификация и авторизация, препятствует повторному использованию и затрудняет обслуживание. Чтобы избежать этого, максимально используйте встроенные функции и возможности платформы.
- Несвязность кода логики компонентов и сегментов усложняет их обслуживание и замену и создает ненужные зависимости от других компонентов. Проектируйте связные и слабо связанные компоненты, чтобы обеспечить максимальную гибкость, упростить их замену и повторное использование.
- Объемный, трудно поддающийся управлению, хрупкий или слишком сложный код, затрудненная реструктуризация из-за требований обратной совместимости. Проектируйте системы, правильно разделяя их на слои или функциональные области, которые четко обозначают функциональность UI, бизнес-процессов и доступа к данным системы. Продумайте, как будете реализовывать внесение изменений в бизнес-процессы и динамические бизнес-правила, возможно, через использование подсистемы управления бизнес-процессами, если предполагается их изменение. Реализуйте правила с помощью бизнес-компонентов, если предполагается изменение только значений бизнес-правил. Используйте внешний источник, такой как обработчик бизнес-правил, если предполагается изменять правила принятия бизнес-решений.

- Существующий код не имеет комплекта автоматизированных регрессивных тестов. Инвестируйте в автоматизацию тестов при разработке системы. Это окупится при проверке функциональности системы и в виде документации с описанием функциональности различных частей системы и их совместной работы.
 - Недостаток документации может затруднять использование, управление и обновления в будущем. Обеспечьте документацию, которая, как минимум, описывает общую структуру приложения.
-

Управляемость

Управляемость определяет, насколько просто системным администраторам управлять приложением, как правило, посредством достаточного и полезного инструментария, предоставляемого для использования в системах мониторинга, а также для отладки и настройки производительности. Обеспечьте такой инструментарий при проектировании приложения. Рассмотрим основные проблемы управляемости:

- Недостаток хороших механизмов мониторинга, трассировки и диагностики. Создайте полноценную модель, определяющую существенные изменения состояния, которые могут оказывать влияние на производительность приложения, и используйте эту модель для выработки требований по инstrumentированию управления. Реализуйте инструментарий, такой как счетчики событий и производительности, который выявляет изменения состояния и предоставляет эти изменения через стандартные системы, такие как журналы событий (Event Logs), файлы трассировки или инструментарий управления Windows (Windows Management Instrumentation, WMI). Отслеживайте и собирайте достаточное количество сведений об ошибках и изменениях состояния для обеспечения точного мониторинга, отладки и управления. Также рассмотрите возможность создания пакетов управления, которые могут использоваться администраторами в средах мониторинга для управления приложением.
 - Отсутствие возможности настройки во время выполнения. Продумайте, как можно обеспечить изменение поведения системы на основании требований операционной среды, таких как изменение инфраструктуры или развертывания.
 - Отсутствие инструментов поиска и устранения неисправностей. Рассмотрите возможность включения кода для создания снимка состояния системы, который может использоваться для поиска и устранения неисправностей, и включения специального инструментария для обеспечения подробных операционных и функциональных отчетов. Реализуйте сбор данных протоколирования и аудита, которые могут быть полезны для обслуживания и отладки, таких как детали запросов или выходные данные модулей и вызовы других систем и сервисов.
-

Производительность

Производительность – это показатель, характеризующий скорость, с какой система выполняет любое действие в заданный промежуток времени. Производительность измеряется в

показателях задержки или пропускной способности. Задержка – это время, необходимое для ответа на любое событие. Пропускная способность – это число событий, имеющих место в заданный промежуток времени. Производительность приложения может напрямую влиять на его масштабируемость, а недостаточная масштабируемость может негативно сказываться на производительности. Повышение производительности приложения часто приводит к улучшению его масштабируемости за счет снижения вероятности конкуренции за совместно используемые ресурсы. К факторам, влияющим на производительность системы, относятся запрос конкретного действия и ответ системы на этот запрос. Рассмотрим основные проблемы, связанные с производительностью:

- Увеличение времени ответа клиента, снижение пропускной способности и повышенное использование ресурсов сервера. Обеспечьте соответствующую структуру приложения и развертывайте его в системе или системах, предоставляющих достаточное количество ресурсов. Если необходимо обеспечить взаимодействие между процессами или через границы уровней, используйте слабо детализированные интерфейсы, требующие минимального числа вызовов (предпочтительнее всего один) для выполнения определенной задачи, и рассмотрите возможность использования асинхронного взаимодействия.
- Повышенное потребление памяти, что приводит к снижению производительности, увеличению промахов кэша (невозможность найти необходимые данные в кэше) и повышению количества обращений к хранилищу данных. Убедитесь в эффективности и правильности используемой стратегии кэширования.
- Повышенный объем обработки на сервере базы данных, что приводит к снижению пропускной способности. Убедитесь, что выбраны эффективные типы транзакций, блокировок, соответствующие подходы к организации потоковой обработки и очередей. Использование эффективных запросов позволит максимально сократить негативное влияние на производительность и избежать выбора всех данных, когда требуется отображать только часть из них. Неправильное проектирование работы с базой данных приводит к ненужной повышенной нагрузке на сервер базы данных, невозможности реализовать требования по производительности и повышению денежных затрат.
- Повышенная нагрузка на сеть, что приводит к увеличению времени ответа и повышению нагрузки и на клиент, и на сервер. Правильный выбор соответствующего механизма удаленного взаимодействия обеспечит высокопроизводительное взаимодействие между уровнями. Страйтесь сократить число транзакций через физические границы и максимально сократить объем передаваемых по сети данных. Работайте в пакетном режиме, чтобы сократить количество вызовов по сети.

Надежность

Надежность – это способность системы сохранять работоспособность в течение некоторого времени. Надежность определяется как вероятность того, что система сможет выполнять предусмотренные функции в течение заданного промежутка времени. Рассмотрим основные проблемы надежности:

- Система дает сбой или не отвечает. Определите возможности для выявления сбоев и автоматического запуска обработки отказов или перенаправления нагрузки на запасную или резервную систему. Также реализуйте код, обеспечивающий использование альтернативных систем при выявлении определенного числа неудачных запросов к текущей системе.
- Неустойчивый вывод. Реализуйте инструментарий, такой как счетчики событий и производительности, для выявления низкой производительности или сбоев запросов к внешним системам с последующим предоставлением этих сведений через стандартные системы, такие как журналы событий, файлы трассировки или WMI. Протоколируйте данные производительности и аудита для вызовов, выполняемых к другим системам и сервисам.
- Система дает сбой из-за недоступности внешних ресурсов, таких как системы, сети и базы данных. Определите способы обработки ненадежности внешних систем, сбоев связи и транзакций. Продумайте, как можно реализовать очередь ожидающих выполнения запросов на время пребывания системы в автономном режиме. Реализуйте системы передачи данных с промежуточным хранением или системы связи посредством сообщений с кэшированием, обеспечивающие возможность сохранения запросов, когда целевая система недоступна, и их воспроизведения при восстановлении подключения. Использование Windows Message Queuing или BizTalk Server обеспечит надежный механизм одноразовой доставки для асинхронных запросов.

Возможность повторного использования

Возможность повторного использования определяет пригодность компонентов и подсистем к использованию в других приложениях и сценариях для добавления новой функциональности с внесением незначительных изменений или вообще без таковых. Возможность повторного использования обеспечивает снижение дублирования компонентов, а также сокращение времени, затрачиваемого на реализацию. Выявление общих атрибутов в различных компонентах – первый шаг к созданию небольших компонентов, пригодных для повторного использования в больших системах. Рассмотрим основные проблемы, связанные с возможностью повторного использования:

- Применение разного кода или компонентов для достижения того же результата в разных частях программы; например, дублирование логики во многих компонентах и дублирование логики во многих слоях или подсистемах. Проанализируйте дизайн приложения для выявления общей функциональности и реализуйте ее в отдельных компонентах, которые можно будет использовать повторно. Определитесь со сквозной функциональностью, такой как проверка, протоколирование и аутентификация, и реализуйте эти функции как отдельные компоненты.
- Использование множества аналогичных методов для реализации похожих задач. Разнообразить поведение метода помогут параметры.
- Использование нескольких систем для реализации одной и той же возможности или функции, вместо совместного или повторного использования функциональности

другой системы, множества систем или разных подсистем приложения. Предоставляйте функциональность компонентов, слоев и подсистем через интерфейсы сервисов, которые могут использоваться другими слоями и системами. Применяйте независимые от используемой платформы типы данных и структуры, с которыми могут работать разные платформы.

Масштабируемость

Масштабируемость – это способность системы справляться с увеличением нагрузки без влияния на производительность системы или способность системы легко расширяться. Существует два метода улучшения масштабируемости: вертикальное масштабирование (scale up) и горизонтальное масштабирование (scale out). Для обеспечения вертикального масштабирования в систему добавляется больше ресурсов, таких как ЦП, память и дисковое пространство. Для обеспечения горизонтального масштабирования в ферме, выполняющей приложение и распределяющей нагрузку, используется большее количество компьютеров. Рассмотрим основные проблемы масштабируемости:

- Приложения не могут справляться с увеличивающейся нагрузкой. Проектируйте слои и уровни, обеспечивая их масштабируемость. Продумайте, как выбранный дизайн может влиять на способность приложения и базы данных масштабироваться в вертикальном или горизонтальном направлении в случае необходимости. Логические слои можно разместить на одном физическом уровне, что позволит сократить количество необходимых серверов, повышая при этом возможности распределения нагрузки и обработки отказов. Реализуйте распределение данных по нескольким серверам баз данных, что обеспечит лучшие возможности для вертикального масштабирования и гибкое размещение подмножеств данных. Избегайте использования компонентов и подсистем, сохраняющих состояние, для снижения привязки к конкретному серверу.
 - Пользователи ощущают задержку при обращениях к системе и увеличение продолжительности времени выполнения задач. Продумайте, как обеспечить обработку всплесков трафика и нагрузки. Реализуйте код для использования дополнительных или альтернативных систем при достижении предельной загруженности сервиса или определенного числа ожидающих обработки запросов к текущей системе.
 - Система не может обработать очередь задач, сформированную в период повышенной загруженности, за время, когда нагрузка понижена. Реализуйте системы передачи данных с промежуточным хранением или системы связи посредством сообщений с кэшированием, обеспечивающие возможность сохранения запросов, когда целевая система недоступна, и их воспроизведения при подключении системы к сети.
-

Безопасность

Безопасность – это способность системы предотвращать злонамеренные или случайные действия, не предусмотренные при проектировании, или не допускать разглашение или утрату данных. Повышение безопасности ведет также к повышению надежности системы за счет

снижения вероятности успеха атак и их негативного влияния на работу системы. Безопасная система должна защищать ресурсы и предотвращать несанкционированный доступ или изменение данных. Безопасность системы определяют такие факторы, как конфиденциальность, целостность и доступность. Для обеспечения безопасности системы используются аутентификация, шифрование, аудит и протоколирование. Рассмотрим основные проблемы безопасности:

- Подделка удостоверения пользователя. Избежать подделки удостоверения пользователя позволяют аутентификация и авторизация. Определите границы доверия и проводите аутентификацию и авторизацию пользователей, пересекающих эти границы.
- Нанесение ущерба злонамеренным вводом, таким как внедрение SQL-кода или межсайтовая атака внедрением сценариев. Защититься от таких атак поможет проверка длины, диапазона, формата и типа всего ввода с использованием принципов ограничения, отклонения и очистки. Выполните кодирование всех выходных данных, отображаемых пользователю.
- Повреждение или подделка данных. Используйте разделение для анонимных, идентифицированных и аутентифицированных пользователей на сайте и примените инструментарий для протоколирования и реализации поведения, которое может отслеживаться. Также пользуйтесь защищенными транспортными каналами, шифруйте и подписывайте передаваемые по сети конфиденциальные данные.
- Отказ пользователя от выполненных действий. Используйте инструментарий для аудита и протоколирования всех взаимодействий пользователя для наиболее важных операций приложения.
- Разглашение сведений и утрата конфиденциальных данных. При проектировании всех аспектов приложения обеспечьте невозможность доступа или разглашения конфиденциальных данных системы или приложения.
- Перебои в обслуживании из-за отказа в обслуживании (DoS). Сократите время ожидания сеансов и реализуйте код или аппаратные средства для выявления и предотвращения подобных атак.

Обеспеченность технической поддержкой

Обеспеченность технической поддержкой – это способность системы предоставлять сведения, необходимые для выявления и разрешения проблем при некорректной работе. Рассмотрим основные проблемы, связанные с обеспеченностью технической поддержкой:

- Недостаток диагностических сведений. Примите решение о том, как будет выполняться наблюдение за работой системы и производительностью. Используйте программное обеспечение системного мониторинга, такое как Microsoft System Center.
- Отсутствие инструментов диагностики и устранения неисправностей. Включите код для создания снимка состояния системы, который может использоваться для диагностики неисправностей, и специальный инструментарий, который сможет обеспечить подробные операционные и функциональные отчеты. Предусмотрите сбор данных

протоколирования и аудита, таких как детали запросов или выходные данные модулей и вызовы других систем и сервисов, которые могут быть полезны для обслуживания и отладки.

- Отсутствие трассировки. С помощью обычных компонентов обеспечьте поддержку трассировки в коде, возможно, посредством методик аспектно-ориентированного программирования (Aspect Oriented Programming, AOP) или внедрения зависимостей. Включите трассировку в Веб-приложениях для выявления ошибок.
- Отсутствие качественного мониторинга. Создайте хорошую модель, определяющую существенные изменения состояния, которые могут влиять на производительность приложения, и используйте эту модель для определения требований по инструментированию управления. Реализуйте инструментарий, такой как счетчики событий и производительности, которые выявляют изменения состояния и предоставляют эти изменения через стандартные системы, такие как журналы событий, файлы трассировки или Инструментарий управления Windows (WMI). Отслеживайте и собирайте достаточное количество сведений об ошибках и изменениях состояния для обеспечения точного мониторинга, отладки и управления. Также рассмотрите возможность создания пакетов управления, которые могут использоваться администраторами в средах мониторинга для управления приложением.

Тестируемость

Тестируемость – это мера того, насколько просто создать критерий проверки для системы и ее компонентов и выполнить эти тесты, чтобы определить, отвечает ли система данному критерию. Тестируемость означает, что сбои в системе могут быть своевременно и эффективно изолированы. Рассмотрим основные проблемы, связанные с тестируемостью:

- Невозможность согласованного тестирования сложных приложений с множеством связей в логике работы, вероятно, из-за невозможности автоматизированного или детального тестирования приложения, имеющего монолитный дизайн. Обеспечивайте тестируемость, создавая модульные системы. Обеспечивайте инструментарий или реализуйте точки внедрения для тестирования, механизмы для отладки вывода и способы простого задания ввода. Проектируйте высокосвязанные и слабосвязанные компоненты, которые могут тестироваться отдельно от остальной системы.
- Отсутствие планирования тестов. Начинайте тестирование в цикле разработки как можно раньше. Используйте при тестировании фиктивные объекты и создавайте простые структурированные решения для тестирования.
- Недостаточное тестовое покрытие, как для ручных, так и для автоматизированных тестов. Продумайте возможности автоматизации тестов взаимодействия с пользователем и то, как обеспечить максимальное покрытие тестами кода и требований.
- Несогласованность входных данных и результатов; несоответствие результатов вводимым данным и неполное покрытие результатами предметной области вывода

даже в случае предоставления всех известных вариантов ввода. Продумайте, как упростить задание и понимание входных и выходных данных системы для упрощения создания вариантов тестирования.

Взаимодействие с пользователем /удобство и простота использования

Интерфейсы приложений должны проектироваться с учетом требований пользователя и потребителя, так чтобы они были просты в использовании, могли быть локализованы и глобализованы, были доступны для пользователей с физическими недостатками и обеспечивали хорошее взаимодействие с пользователем в общем. Рассмотрим основные проблемы взаимодействия с пользователем:

- Для выполнения задачи требуется слишком много действий (и нажатий клавиш). Обеспечивайте максимальную простоту использования при проектировании логики работы с окнами и вводом данных, а также сценариев взаимодействия с пользователем.
 - Неверная последовательность выполнения шагов в многошаговых интерфейсах. По возможности используйте рабочие процессы для упрощения многошаговых операций.
 - Неправильная группировка элементов данных и элементов управления. Правильно выбирайте типы элементов управления (такие как группы переключателей и кнопки-флажки) и компонуйте элементы управления и содержимое, используя общепринятые шаблоны проектирования UI.
 - Плохая обратная связь с пользователем, особенно для ошибок и исключений, и в случаях, когда приложение не отвечает. Реализуйте технологии и техники, обеспечивающие максимальное взаимодействие с пользователем, такие как Asynchronous JavaScript and XML (AJAX)¹ на Веб-страницах и проверка ввода на стороне клиента. Для фоновых задач и таких задач, как заполнение элементов управления данными или выполнение длительных задач, применяйте асинхронные методики.
-

Дополнительные источники

Электронная версия списка используемых источников по реализации и аудиту показателей качества доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Implementing System-Quality Attributes*» (Реализация показателей качества системы) по адресу <http://msdn.microsoft.com/en-us/library/bb402962.aspx>.
- «*Software Architecture in the New Economy*» (Архитектура ПО в новых экономических условиях) по адресу <http://msdn.microsoft.com/en-us/library/cc168642.aspx>.
- «*Quality-Attribute Auditing: The What, Why, and How*» (Аудит показателей качества: что, как и почему) по адресу <http://msdn.microsoft.com/en-us/library/bb508961.aspx>.
- Майкл К. Физерс. Эффективная работа с унаследованным кодом. Вильямс, 2009.

¹ Асинхронный JavaScript и XML (прим. переводчика).

- Кайл Бейли (Kyle Baley) и Дональд Белчам (Donald Belcham). *Brownfield Application Development in .NET* (Разработка приложений Brownfield в .NET). Manning Publications Co, 2008.
 - Майкл Найгард (Michael Nygard). *Release It!: Design and Deploy Production-Ready Software* (Проектирование и развертывание готового к производственной эксплуатации ПО). Pragmatic Bookshelf, 2007.
-

17

Сквозная функциональность

Обзор

В большинстве проектируемых приложений имеется общая функциональность, которую нельзя отнести к конкретному слою или уровню. Обычно это такие операции как аутентификация, авторизация, кэширование, связь, управление исключениями, протоколирование и инструментирование, а также валидация. Эти функции называют *сквозной функциональностью* (*crosscutting concerns*), потому что они оказывают влияние на все приложение и, по возможности, должны реализовываться централизованно. Например, если код, формирующий записи журнала и выполняющий запись в журналы приложения, разбросан по разным слоям и уровням, в случае изменения требований, связанных с этими вопросами (например, перенесение журнала в другой каталог), придется выискивать и обновлять соответствующий код по всему приложению. Но если код протоколирования централизован, изменить поведение, можно изменив код лишь в одном месте.

Данная глава поможет понять роль сквозной функциональности в приложении и найти области, в которых эта функциональность используется. В ней также представлены основные проблемы, с которыми придется столкнуться при проектировании сквозной функциональности. Существует несколько разных подходов к реализации этой функциональности, начиная от общих библиотек, таких как Enterprise Library группы patterns & practices, до методов аспектно-ориентированного программирования (Aspect Oriented Programming, AOP), использующих метаданные для внедрения кода сквозной функциональности непосредственно в откомпилированный вывод или во время выполнения.

Общие принципы проектирования

Следующие рекомендации помогут понять основные факторы, оказывающие влияние на сквозную функциональность:

- Проанализируйте все функции в каждом слое и найдите те из них, которые могут быть выделены в общие компоненты, возможно, даже компоненты общего назначения, настраиваемые в зависимости от конкретных требований каждого слоя приложения. Скорее всего, эти компоненты можно будет использовать и в других приложениях.
- В зависимости от физического распределения компонентов и слоев приложения, возможно, понадобится установить компоненты сквозной функциональности на

нескольких физических уровнях. Но несмотря на это, преимущества от возможности повторного использования и сокращения времени и затрат на разработку сохраняются.

- Используйте шаблон Dependency Injection для внедрения экземпляров компонентов сквозной функциональности в приложение на основании данных конфигурации. Это позволяет без труда изменять используемые в каждой подсистеме компоненты сквозной функциональности без необходимости повторной компиляции и развертывания приложения. Библиотека Unity группы patterns & practices обеспечивает полную поддержку шаблона Dependency Injection. К другим популярным библиотекам Dependency Injection относятся StructureMap, Ninject и Castle Windsor (больше информации по этому вопросу можно найти в разделе [Дополнительные источники](#) в конце данной главы).
- Сократить время разработки позволит использование библиотек компонентов сторонних производителей, предоставляющих легкоconfigурируемые компоненты. Одним из примеров такой библиотеки является Enterprise Library группы patterns & practices, содержащая блоки приложений, которые облегчат реализацию функций кэширования, обработки исключений, аутентификации и авторизации, протоколирования, валидации и шифрования. Она также включает механизмы, реализующие контейнер внедрения политик и зависимостей, которые упрощают реализацию решений для ряда аспектов сквозной функциональности. Более подробно Enterprise Library рассматривается в приложении F, «[Enterprise Library от patterns & practices](#)». Другой популярной библиотекой является Castle Project (больше информации по этому вопросу можно найти в разделе [Дополнительные источники](#) в конце данной главы).
- Используйте методики аспектно-ориентированного программирования (AOP), что поможет внедрить сквозную функциональность в приложение без реализации явных вызовов в коде. Библиотека Unity и Enterprise Library Policy Injection Application Block (Блок внедрения политик библиотеки Enterprise Library) группы patterns & practices поддерживают этот подход. Другими примерами являются библиотеки Castle Windsor и PostSharp (больше информации по этому вопросу можно найти в разделе [Дополнительные источники](#) в конце данной главы).

Специальные вопросы проектирования

В следующих разделах перечислены основные области, которые следует рассмотреть при разработке архитектуры, и приведены рекомендации, которые помогут избежать обычных проблем в каждой из этих областей:

- [Аутентификация](#)
- [Авторизация](#)
- [Кэширование](#)
- [Сетевое взаимодействие](#)

- [Управление конфигурацией](#)
 - [Управление исключениями](#)
 - [Протоколирование и инструментирование](#)
 - [Управление состоянием](#)
 - [Валидация](#)
-

Аутентификация

Проектирование эффективной стратегии аутентификации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак с подделкой пакетов, атак перебором по словарю, перехватом сеансов и других типов атак. При проектировании стратегии аутентификации руководствуйтесь следующими рекомендациями:

- Определите границы доверия и проводите аутентификацию пользователей и вызовов на границах доверия. Учтите, что может понадобиться аутентифицировать вызовы, как клиента, так и сервера (взаимная аутентификация).
 - Обеспечьте использование надежных паролей или парольных фраз.
 - При наличии множества систем в рамках приложения, или если пользователи должны иметь возможность доступа ко многим приложениям, используя одни и те же учетные данные, применяйте стратегию единой регистрации.
 - Не передавайте пароли по сети и не храните их в базе данных или хранилище данных в открытом виде. Храните хеш пароля.
-

Больше сведений о разработке стратегии аутентификации и методиках ее реализации можно найти в разделе «[Дополнительные источники](#)» в конце данной главы.

Авторизация

Проектирование эффективной стратегии авторизации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для разглашения сведений, повреждения или подделки данных и несанкционированного получения прав. При проектировании стратегии авторизации руководствуйтесь следующими рекомендациями:

- Определите границы доверия и проводите авторизацию пользователей и вызовов на границах доверия.
- Защитите ресурсы, проводя авторизацию вызывающей стороны на основании ее удостоверения, групп или ролей. Обеспечьте минимальное дробление, по возможности ограничивая количество используемых ролей.
- Применяйте авторизацию на основании ролей для бизнес-решений. При авторизации на основании ролей все пользователи распределяются по группам (ролям), что позволяет задавать права доступа для роли, а не для каждого

пользователя в отдельности. Это упрощает управление, поскольку администратору приходится работать лишь с небольшим набором ролей, а не со всеми пользователями системы.

- Применяйте авторизацию на базе ресурсов для аудита системы. При авторизации на базе ресурсов права доступа определяются в самом ресурсе. Например, список управления доступом (ACL) ресурса Windows использует удостоверение исходногозывающего для определения его прав доступа к ресурсу. При использовании авторизации на базе ресурсов в WCF необходимо выполнить олицетворение исходногозывающего через клиента или слой представления, через слой сервисов WCF и для кода бизнес-логики, выполняющего доступ к ресурсу.
- Используйте авторизацию на основании утверждений, если требуется поддерживать интегрированную авторизацию на базе сочетания данных, таких как удостоверение, роль, разрешения, права и т.д. Авторизация на основании утверждений обеспечивает дополнительные уровни абстракции, что упрощает отделение правил авторизации от механизма авторизации и аутентификации. Например, пользователь может быть аутентифицирован по сертификату или имени пользователя/паролю, после чего набор этих утверждений передается в сервис для определения прав доступа к ресурсу.

Больше сведений о разработке стратегии авторизации и методиках ее реализации можно найти в разделе «[Дополнительные источники](#)» в конце данной главы.

Кэширование

Кэширование может улучшить производительность и время отклика приложения. Однако неправильно спроектированная стратегия кэширования может негативно сказаться на этих показателях. Кэширование должно применяться для оптимизации поиска используемых данных, сокращения количества обращений к сети и предотвращения ненужной или повторной обработки. При реализации кэширования необходимо принять решение о том, когда загружать кэшированные данные, а также как и когда удалять устаревшие кэшированные данные. Предварительная асинхронная загрузка в кэш часто используемых данных или применение пакетной обработки помогут избежать задержек на стороне клиента. При проектировании стратегии кэширования руководствуйтесь следующими рекомендациями:

- Выберите подходящее размещение для кэша. Если приложение развертывается на Веб-ферме, избегайте применения локальных кэшей, для которых необходима синхронизация. В этом случае рекомендуется использовать систему управления транзакционными ресурсами, такую как Microsoft® SQL Server®, или продукт, поддерживающий распределенное кэширование, такой как технология Memcached производства компании Danga Interactive или механизм кэширования Velocity от компании Microsoft (больше информации по этому вопросу можно найти в разделе [Дополнительные источники](#) в конце данной главы).
- При работе с кэшем в памяти применяйте кэширование данных в готовом к использованию виде. Например, кэшируйте не просто необработанные данные

базы данных, а используйте специализированные объекты необходимые приложению. Реализуйте кэширование в памяти с помощью Microsoft Velocity.

- Не кэшируйте часто изменяющиеся данные и незашифрованные конфиденциальные данные.
- Не полагайтесь на кэшированные данные, они могут быть удалены. Реализуйте механизм обработки сбоев кэша, возможно, путем повторной загрузки элемента из источника.
- Будьте особенно осторожны при работе с кэшем из нескольких потоков. В случае использования множества потоков для обеспечения непротиворечивости данных убедитесь, что любой доступ к кэшу является потокобезопасным.

Более подробно разработка стратегии кэширования рассматривается в разделе «[Этапы проектирования стратегии кэширования](#)» далее в этой главе.

Сетевое взаимодействие

Средства связи обеспечивают взаимодействие компонентов разных слоев и уровней. Выбор механизма связи зависит от сценариев развертывания, которые должно поддерживать ваше приложение. При проектировании стратегии связи руководствуйтесь следующими рекомендациями

- Используйте взаимодействие посредством обмена сообщениями при пересечении физических границ или границ процесса; и взаимодействие объектов внутри процесса (при пересечении только логических границ). Для сокращения числа циклов запрос-ответ и повышения производительности взаимодействия через физические границы и границы процесса, проектируйте обобщенные интерфейсы, взаимодействующие не так часто, но передающие большие объемы данных при каждом взаимодействии. Тем не мене, там где это необходимо, предоставляемые детализированные интерфейсы, которые будут использоваться вызовами внутри процесса, и заключайте эти вызовы в обобщенный фасад, который будет использоваться процессами, выполняющими доступ к нему через физические границы или границы процессов.
- Если порядок получения сообщений не имеет значения, и сообщения не зависят друг от друга, используйте асинхронное взаимодействие. Это поможет избежать блокировок обработки или потоков UI.
- Используйте механизм Microsoft Message Queuing, обеспечивающий помещение сообщений в очередь для их отложенной доставки в случае сбоя системы или разрыва подключения. Message Queuing может осуществлять транзакционную доставку сообщений и поддерживает гарантированную одноразовую доставку.
- Выбирайте соответствующий транспортный протокол, такой как HTTP для связи через Интернет и TCP для связи по внутренней сети. Продумайте соответствующие схемы обмена сообщениями, то, будете ли вы использовать взаимодействие с установлением подключения или без установления подключения, поддержку

гарантий надежности (такие как соглашения на уровне сервиса) и механизм аутентификации.

- Обеспечьте защиту сообщений и конфиденциальных данных при передаче, используя шифрование, цифровые сертификаты и функции безопасности каналов.

Более подробно проектирование стратегии взаимодействия рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Управление конфигурацией

Правильный выбор механизма управления конфигурацией имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для различных атак. Также неправильный механизм управления конфигурацией может обуславливать издержки на администрирование приложения. При проектировании стратегии управления конфигурацией руководствуйтесь следующими рекомендациями:

- Тщательно продумывайте, какие параметры должны быть конфигурируемыми извне. Убедитесь в реальной прикладной целесообразности каждого настраиваемого параметра и обеспечьте минимум параметров конфигурации, необходимый для выполнения этих требований. Излишняя настраиваемость может усложнить систему и сделать ее уязвимой из-за брешей безопасности и неправильного функционирования, обусловленных неверной конфигурацией.
- Примите решение о том, будут ли сведения о конфигурации храниться централизованно и загружаться или применяться к пользователям при запуске (например, через Active Directory Group Policy¹). Продумайте, как обеспечите ограничение доступа к сведениям о конфигурации. Используйте менее привилегированный процесс и учетные записи сервиса и шифруйте конфиденциальные данные в хранилище конфигурации.
- Распределите элементы конфигурации по логическим разделам на основании того, являются ли они настройками пользователя, приложения или среды. Это упростит разделение конфигурации, если потребуется поддерживать разные настройки для разных наборов пользователей или множества сред.
- При наличии множества уровней в приложении распределите элементы конфигурации по логическим разделам. Если сервер приложений выполняется на Веб-ферме, определите, какая часть конфигурации является общей, и какая часть применяется исключительно к компьютеру, на котором выполняется приложение. После этого выберите соответствующее хранилище конфигурации для каждого раздела.
- Предоставьте отдельный административный UI для редактирования конфигурационных данных.

¹ Групповая политика Active Directory (прим. переводчика).

Управление исключениями

Проектирование эффективной стратегии управления исключениями имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. Неправильный выбор стратегии очень усложнит диагностирование и решение проблем приложения, сделает его уязвимым для атак типа отказ в обслуживании (DoS), а также может привести к разглашению конфиденциальных и важных сведений. Формирование и обработка исключений является ресурсоемким процессом, поэтому важно, чтобы при проектировании были также учтены вопросы производительности. Хорошим подходом является проектирование централизованного механизма управления исключениями для приложения и предоставление точек доступа к системе управления исключениями (таких как события WMI) для обеспечения поддержки систем мониторинга уровня предприятия, таких как Microsoft System Center. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Проектируйте соответствующую стратегию распространения исключений, которая обеспечивает обертывание или замену исключений или внесение необходимых дополнительных данных. Например, позвольте исключениям распространяться вверх к граничным слоям, где они могут быть запротоколированы и преобразованы должным образом для передачи в следующий слой. Применяйте контекстные идентификаторы, это позволит находить взаимосвязанные исключения в разных слоях при проведении анализа основных причин ошибок и сбоев. Также убедитесь, что в дизайне предусмотрены необрабатываемые исключения. Перехватывайте внутренние исключения, только если можете обработать их или должны добавить некоторые данные. Не используйте исключения для управления логикой приложения.
- Обеспечьте, чтобы приложение не оставалось в нестабильном состоянии после сбоя, и чтобы исключения не приводили к разглашению конфиденциальных данных или сведений о процессе. Если не можете гарантированно обеспечить корректное восстановление после сбоя, позвольте приложению завершиться с необработанным исключением; это лучше, чем оно будет продолжать выполнение в неизвестном и, возможно, поврежденном состоянии.
- Выработайте соответствующую стратегию протоколирования и уведомления для критических ошибок и исключений, обеспечивая сохранение достаточно детальных сведений об исключении. Это позволит техническому персоналу восстановить сценарий, который привел к исключению. Однако не предоставляйте конфиденциальные данные в сообщениях об исключениях и файлах журнала.

Более подробно проектирование стратегии управления исключениями рассматривается в разделе «[Этапы проектирования стратегии управления исключениями](#)» далее в этой главе.

Протоколирование и инструментирование

Проектирование эффективной стратегии протоколирования и инструментирования имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, пользователи смогут безнаказанно отказываться от своих действий. Также

файлы журналов могут пригодиться для доказательства правонарушений в случае судебного разбирательства. Аудит и протоколирование действий во всех слоях приложения необходимы, потому что могут помочь обнаружить подозрительные действия и обеспечить раннее выявление серьезных атак. Аудит считается наиболее достоверным, если данные аудита формируются непосредственно в момент доступа к ресурсу и именно теми процедурами, которые выполняют доступ к ресурсу. Инструментирование можно реализовать, используя счетчики производительности и события, обеспечивающие администраторов сведениями о состоянии, производительности и работоспособности приложения. При проектировании стратегии протоколирования и инструментирования руководствуйтесь следующими рекомендациями:

- Проектируйте централизованный механизм протоколирования и инструментирования, обеспечивающий перехват критически важных для системы и бизнеса событий. Избегайте слишком детализированного протоколирования и инструментирования, но предусмотрите дополнительные функции протоколирования и инструментирования, настраиваемые во время выполнения, для получения дополнительных данных и для помощи при отладке.
- Создавайте политики безопасного управления файлами журнала. Не храните конфиденциальные данные в файлах журнала и защищайте их от неавторизованного доступа. Продумайте, как обеспечить безопасный доступ и передачу данных аудита и протоколирования между слоями приложения, и обеспечьте сдерживание и правильную обработку сбоев протоколирования.
- Сделайте свои приемники журнала, или слушатели трассировки, настраиваемыми, чтобы обеспечить возможность их изменения во время выполнения соответственно требованиям инфраструктуры развертывания. В реализации протоколирования и инструментирования в приложении очень помогут такие библиотеки, как Enterprise Library группы patterns & practices. Среди популярных библиотек можно также упомянуть NLog и log4net (больше информации по этому вопросу можно найти в разделе [Дополнительные источники](#) в конце данной главы)

Более подробно протоколирование и инструментирование рассматриваются в разделе «[Этапы проектирования стратегии управления исключениями](#)» далее в этой главе.

Управление состоянием

Управление состоянием – это вопросы, связанные с хранением данных, представляющих состояние компонента, операции или этапа процесса. Для хранения данных состояния могут использоваться разные форматы и хранилища. Механизм управления состоянием может оказывать влияние на производительность приложения; сохранение даже небольших объемов данных состояния может неблагоприятно сказываться на производительности приложения и его способности масштабироваться. Сохраняться должны только необходимые данные, и вы должны понимать, какие варианты управления состоянием допустимы. При проектировании стратегии управления состоянием руководствуйтесь следующими рекомендациями:

- Сохраняйте минимальный объем данных состояния.

- Если для сохранения или совместного использования данные состояния должны передаваться через границы процессов и сетей, обеспечьте их сериализуемость.
- Правильно выбирайте хранилище состояния. Хранение состояния в процессе или в памяти обеспечит наилучшую производительность, но эта техника может использоваться, только если состояние не должно сохраняться между повторными запусками процесса или системы. Если хотите, чтобы данные состояния были доступны после перезапуска процесса или системы, сохраняйте их на локальном диске или локальном SQL Server. Если состояние является критически важным аспектом приложения, или если данные состояния должны использоваться совместно несколькими компьютерами, храните состояние централизованно, например, на выделенном SQL Server.

Валидация

Проектирование эффективного механизма валидации имеет большое значение с точки зрения обеспечения удобства и простоты использования и надежности приложения, в противном случае, оно может остаться открытым для несогласованности данных и нарушений бизнес-правил, а также обеспечивать неудовлетворительный уровень взаимодействия с пользователем. Кроме того, приложение может оказаться уязвимым к таким угрозам безопасности, как межсайтовые атаки внедрением сценариев, атаки типа внедрение SQL-кода, переполнение буфера и другие атаки посредством входных данных. Четкого и исчерпывающего определения действительного или злонамеренного ввода нет. Возможные риски, обуславливаемые злонамеренным вводом, зависят также от того, как приложение использует ввод. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- По возможности реализуйте позитивную валидацию с использованием списков разрешенного ввода. Намного проще расширить список разрешенного ввода, чем пытаться определить все варианты некорректного ввода.
- Для проверки безопасности не полагайтесь только на проверку на стороне клиента. Используйте проверку на стороне клиента для обеспечения обратной связи пользователю и улучшения взаимодействия с пользователем. Всегда проверяйте ввод на наличие неверных или злонамеренных данных на стороне сервера, поскольку проверка на стороне клиента может быть без труда подделана.
- Если необходимо обеспечить повторное использование подхода к валидации, реализуйте его централизованно, вынося функциональность валидации в отдельные компоненты, или обратитесь к библиотекам сторонних производителей, таким как Enterprise Library Validation Block (Блок валидации Enterprise Library) группы patterns & practices. Это позволит создать единый механизм валидации для всех слоев и уровней приложения.
- Ограничивайте, отклоняйте и очищайте пользовательский ввод. Иначе говоря, исходите из предположения о том, что весь пользовательский ввод является злонамеренным. Определите границы доверия и проверяйте длину, формат, тип и диапазон всех вводимых данных, пересекающих границы доверия.

Более подробно проектирование стратегии валидации рассматривается в разделе «[Этапы проектирования стратегии валидации ввода и данных](#)» далее в данной главе.

Этапы проектирования стратегии кэширования

Кэширование может играть решающую роль в повышении производительности.

Исключительно важно спроектировать соответствующую стратегию кэширования, потому что неправильный выбор методик может негативно сказаться на производительности.

Рассматриваемые далее этапы проектирования помогут выработать правильную стратегию кэширования для приложения.

Шаг 1 – Выбор данных, подлежащих кэшированию

В ходе проектирования приложения важно определиться с тем, какие данные годятся для кэширования. Для каждого слоя приложения создайте список данных, которые могут быть кэшированы. Рассмотрите возможность кэширования следующих типов данных:

- **Общие данные приложения.** Рассмотрите возможность кэширования статических данных, которые используются всеми пользователям приложения. Примерами таких данных являются списки продуктов и сведения о продуктах.
- **Относительно статические данные.** Рассмотрите возможность кэширования полностью статических данных или данных, которые меняются нечасто, например, константы или фиксированные значения, считываемые из конфигурационного файла или базы данных.
- **Относительно статические Веб-страницы.** Рассмотрите возможность кэширования вывода Веб-страниц или частей Веб-страниц, которые меняются нечасто.
- **Параметры хранимых процедур и результаты запросов.** Рассмотрите возможность кэширования часто используемых параметров и результатов запросов.

Шаг 2 – Выбор места кэширования данных

При принятии решения о месте кэширования данных обычно необходимо рассмотреть два вопроса: физическое размещение кэша и его логическое размещение.

Физически кэш размещается либо в памяти, либо на диске в файлах или базе данных.

Кэширование в памяти может осуществляться с помощью механизма кэширования ASP.NET, Enterprise Library Caching Application Block или механизма распределенного кэширования в памяти, такого как проект Microsoft под кодовым названием Velocity или технология Memcached от компании Danga Interactive. Размещайте кэш в памяти, если приложение часто использует данные; если кэшированные данные относительно часто меняются, и их приходится довольно часто запрашивать повторно; и если объем кэшированных данных относительно мал. Размещайте кэш в системе или базе данных, если использовать данные из хранилища кэша более эффективно по сравнению с их запросом из исходного хранилища; если кэшированные данные относительно редко меняются; и если сервисы для повторного запроса данных не всегда доступны. Подход с хранением кэша на диске также идеален при большом

объеме кэшированных данных, или если кэшированные данные должны сохраняться при перезапусках процесса или компьютера.

Логическое размещение кэша – это его место в логике приложения. Важно кэшировать данные в максимальной близости к месту их использования. Это обеспечит снижение объема необходимой обработки, сокращение количества обращений к сети и времени отклика приложения и повышение производительности. Принимая решение о логическом размещении кэша данных, руководствуйтесь следующими рекомендациями:

- Кэшируйте на **клиенте** данные, характерные для страницы или пользователя; данные, не содержащие конфиденциальных сведений; и данные небольшого объема.
- Кэшируйте на **прокси-сервере** или **Веб-сервере** (для Веб-приложений) относительно статические страницы, часто запрашиваемые клиентами; страницы, обновляемые с известной периодичностью; или результаты, возвращаемые Веб-сервисами. Также используйте этот подход для страниц, которые могут формировать разный вывод в зависимости от параметров HTTP, и эти параметры меняются нечасто. Это особенно полезно при небольшом диапазоне выходных данных.
- Кэшируйте в **слое представления** относительно статический вывод страниц; небольшие объемы данных, касающиеся предпочтений пользователей для небольших групп пользователей; или если имеются элементы UI, создание которых достаточно ресурсоемко. Также используйте этот подход для ресурсоемких данных, отображаемых пользователю, например, списков продуктов или сведений о продуктах
- Кэшируйте данные в **бизнес-слое**, если необходимо сохранять состояние сервиса, бизнес-процесса или рабочего процесса; или если для обработки запросов от уровня представления требуются относительно статические данные, создание которых достаточно ресурсоемко.
- Кэшируйте данные в **слое доступа к данным** при наличии коллекции входных параметров для часто вызываемых хранимых процедур или небольших объемов необработанных данных, возвращаемых часто выполняемыми запросами. Рассмотрите варианты кэширования для типизированных наборов данных в слое данных.
- Кэшируйте в отдельную таблицу **базы данных** любые данные, для получения которых требуется выполнение достаточно сложного и ресурсоемкого запроса. Этот вариант кэширования поможет также обеспечить повышение производительности, если требуется кэшировать очень большие объемы данных при реализации механизма разделения на страницы.

Шаг 3 – Определение формата кэширования данных

Теперь, определившись с данными, которые необходимо кэшировать, и приняв решение о месторасположении кэша, важно выбрать формат для кэшированных данных. При

кэшировании храните данные в формате, оптимизированном для предполагаемого использования и не требующем дополнительной или повторной обработки или преобразования. Этого правила следует придерживаться, если данные кэшируются в памяти, если кэш не будет использоваться совместно разными процессами или компьютерами, если нет необходимости перемещать кэшированные данные в разные участки памяти, и если приходится кэшировать необработанные данные, такие как объекты **DataSet**, **DataTable** и Веб-страницы.

Если необходимо хранить или передавать кэшированные данные, рассмотрите возможность их сериализации. Сериализация кэшированных данных – хороший выбор для кэширования данных на диск или для хранения состояния сеансов на отдельном сервере или базе данных SQL Server. Это также хороший подход, если необходимо обеспечить совместное использование кэша разными процессами или компьютерами, перемещать кэшированные данные в разные участки памяти или кэшировать собственные объекты. Сериализация может осуществляться с помощью механизма сериализации XML или механизма бинарной сериализации. Механизм сериализации XML подойдет, если определяющим фактором является возможность взаимодействия. Если основной упор делается на производительность, используйте механизм бинарной сериализации.

Шаг 4 – Выработка подходящей стратегии управления кэшем

Необходимо определить соответствующую политику срока действия кэша и сброса кэша. И удаление по истечении срока действия, и сброс данных являются стратегиями удаления кэшированных данных из хранилища кэша. Отличаются они тем, что при сбросе могут удаляться действительные данные для высвобождения памяти под более часто используемые элементы, тогда как удаление данных по истечении срока их действия означает, что эти данные стали недействительными. Проверьте возможности используемой базовой системы кэширования, не все реализации кэша предлагают все возможные варианты.

Стратегия **срока действия кэша** должна обеспечивать, чтобы в кэше находились только действительные данные и элементы. Политика срока действия может использовать как срок действия по времени, так и срок действия по уведомлению:

- При использовании политики срока действия **по времени** кэшированные данные устаревают или становятся недействительными по прошествии определенного интервала времени в абсолютных или относительных показателях. Такую политику рекомендуется применять для часто меняющихся данных, если кэшированные данные регулярно обновляются, или если кэшированные данные остаются действительными лишь в течение определенного промежутка времени или до определенного времени. Выбирая политику срока действия по времени, можно остановиться на политике срока действия по абсолютному времени или по скользящему временному интервалу. Политика срока действия по абсолютному времени позволяет определять время жизни кэшированных данных через задание времени истечения их срока действия. Политика срока действия по скользящему временному интервалу определяет время жизни кэшированных данных путем задания промежутка времени с момента последнего доступа к кэшированным данным, через который они будут считаться устаревшими.

- При использовании политики срока действия **по уведомлению** кэшированные данные устаревают или становятся недействительными на основании уведомлений от внутренних или внешних источников. Такая политика подойдет при работе с нечасто меняющимися кэшированными данными, если кэшированные данные обновляются через непостоянные промежутки времени, или если данные остаются действительными до тех пор, пока не будут изменены внешними или внутренними системами. Обычно в качестве источников уведомлений выступают модули записи файлов на диск, события WMI, уведомления об изменениях в базе данных и операции бизнес-логики. Поступление уведомления означает истечение срока действия или устаревание всех соответствующих элементов кэша.

Стратегия **сброса кэша** должна обеспечивать эффективное использование хранилища, памяти и других ресурсов. Стратегия сброса кэша может быть явной или в результате сборки мусора:

- Для **явного сброса** кэша необходимо задать, когда элемент должен быть удален. Такая политика используется, если требуется поддерживать сценарий удаления поврежденных или устаревших кэшированных данных, если используются хранилища, не поддерживающие сборки мусора, или при работе с кэшем на диске.
- Для **сборки мусора** необходимо определить условия и набор эвристических правил, согласно которым элемент должен быть удален в ходе сборки мусора. Эту политику рекомендуется применять, если требуется автоматически активировать сборку мусора при устаревании ресурсов системы, если требуется обеспечить автоматическое удаление редко используемых или маловажных элементов из кэша, или при работе с кэшем в памяти.

Рассмотрим общие правила сборки мусора:

- Алгоритм **вытеснения по давности использования** (**Least Recently Used**) обеспечивает удаление элементов, которые не использовались в течение наибольшего периода времени.
- Алгоритм **вытеснения по частоте использования** (**Least Frequently Used**) обеспечивает удаление элементов, которые с момента загрузки использовались реже всего.
- При использовании алгоритма **вытеснения по приоритетности** (**Priority**) всем кэшированным элементам присваиваются приоритеты, и сборка мусора выполняется на основании этих приоритетов с сохранением элементов, имеющих более высокий приоритет.

Шаг 5 – Выбор метода загрузки кэшированных данных

Правильный выбор способа наполнения кэша позволит увеличить производительность и сократить время отклика приложения. Принимая решение о том, как будет заполняться кэш, учитывайте, сколько данных должно быть доступно при запуске приложения или при исходной загрузке кэша, а также то, какое влияние это будет иметь на время запуска и производительность приложения. Например, можно выполнять предварительную загрузку

данных в кэш при запуске приложения или извлекать кэшированные данные, только когда они запрашиваются. Загрузка данных в кэш при запуске приложения может сократить время отклика приложения, но при этом увеличить время загрузки. С другой стороны, загрузка данных в кэш только по необходимости способствует сокращению времени запуска приложения, но может увеличить время отклика при первом обращении к этим данным.

При проектировании стратегии заполнения кэша может использоваться упреждающая или реактивная загрузка:

- **Упреждающая загрузка** обеспечивает извлечение всех данных приложения при его запуске и их кэширование на весь период выполнения приложения. Упреждающая загрузка подойдет для относительно статических данных, или если известны заранее частота их обновления, время жизни и размер. Если размер данных неизвестен, их загрузка может привести к истощению ресурсов системы. Используйте этот вариант загрузки также, если в качестве источника кэшированных данных предполагается медленная база данных, или данные извлекаются по медленной сети или из ненадежного Веб-сервиса.
- **Реактивная загрузка** обеспечивает извлечение данных при запросе приложением и их кэширование для запросов в будущем. Реактивная загрузка подойдет для относительно непостоянных данных, когда время жизни кэшированных данных неизвестно, объем кэшированных данных велик и источник данных надежен и всегда доступен.

Этапы проектирования стратегии управления исключениями

Надежная и тщательно продуманная стратегия управления исключениями может упростить дизайн приложения и повысить безопасность и управляемость. Она будет способствовать облегчению задачи по разработке приложения для разработчиков, снижению времени и затрат на разработку. Приведенные далее рекомендации помогут правильно выработать стратегию управления исключениями.

Шаг 1 – Выбор обрабатываемых исключений

При проектировании стратегии управления исключениями для приложения важно определиться с тем, какие исключения требуется обрабатывать. Должны обрабатываться исключения системы или приложения, такие как формируются при попытках доступа к системным ресурсам пользователями, не имеющими для этого разрешений; и системные сбои, возникающие из-за проблем с жестким диском, ЦП или памятью. Также необходимо обозначить обрабатываемые исключения бизнес-логики, т.е. исключения, обусловленные такими действиями, как нарушение бизнес-правил.

Шаг 2 – Выбор стратегии выявления исключений

Создаваемый дизайн должен обеспечивать единообразную обработку исключений по всему приложению. Это сделает приложение более устойчивым к ошибкам, сократиться вероятность возникновения несогласованного состояния. Структурированная обработка исключений

является средством для управления исключениями с помощью блоков try, catch и finally, их своевременного выявления и соответствующего реагирования на них.

Перехватывайте выявленные исключения, только если необходимо собрать данные об исключении для протоколирования, добавить в исключение некоторые значимые данные, очистить ресурсы, используемые в блоке кода, или повторить операцию для выхода из состояния исключения. Не перехватывайте исключения с последующей их передачей вверх по стеку вызовов, если не собираетесь выполнять ни одну из вышеперечисленных задач.

Шаг 3 – Выработка стратегии распространения исключений

Рассмотрим стратегии распространения исключений. В зависимости от требований контекста приложение может (и должно) использовать сочетание любых или всех этих стратегий:

- **Разрешить распространение исключений.** Эта стратегия хороша тем, что не требует сбора данных об исключении для протоколирования, добавления данных в исключение, очистки каких-либо ресурсов в блоке кода или повтора операции для выхода из состояния исключения. Вы просто позволяете исключению распространяться вверх по стеку кода.
- **Перехват и повторное формирование исключений.** При такой стратегии исключение перехватывается, обрабатывается некоторым образом и повторно формируется. Обычно в этом случае данные исключения остаются нетронутыми. Применяйте эту стратегию, если требуется очищать ресурсы, протоколировать данные исключения или выполнить попытку выхода из состояния ошибки.
- **Перехват, обертывание и формирование исключений.** Эта стратегия обеспечивает перехват универсальных исключений с последующей очисткой ресурсов или любой другой соответствующей обработкой. Если не удается обработать ошибку, исключение заключается в другое исключение, более уместное для вызывающей стороны, и после этого формируется новое исключение для обработки кодом, расположенным выше в стеке кода. Применяйте такую стратегию, если хотите сохранить уместность исключения и/или обеспечить дополнительными сведениями код, обрабатывающий исключение.
- **Перехват и аннулирование исключений.** Эта стратегия является нерекомендуемой, но может применяться в особых случаях. Исключение перехватывается, и приложение продолжает выполняться в обычном порядке. В случае необходимости, можно зарегистрировать исключение и выполнить очистку ресурсов. Такая стратегия подходит для исключений системы, не оказывающих влияния на другие операции, такие как исключения, формируемые при заполнении журнала.

Шаг 4 – Выработка стратегии использования собственных исключений

Подумайте, существует ли необходимость в проектировании собственных исключений или достаточно будет стандартных типов исключений .NET Framework. Не используйте собственные исключения, если в иерархии исключений или в .NET Framework уже имеется подходящее исключение. Но создавайте собственные исключения, если приложение должно

выявлять и обрабатывать специальную исключительную ситуацию, чтобы избежать применения условной логики, или если необходимо включить дополнительные данные для выполнения определенного требования.

Если все-таки приходится создавать собственные классы исключений, применяйте в них стандартные конструкторы, включая конструктор сериализации, и обязательно заканчивайте имя класса словом «Exception» (Исключение). Это важно для обеспечения интеграции со стандартным механизмом исключений. Реализуйте собственное исключение путем наследования от подходящего более общего исключения и его специализации соответственно конкретным требованиям.

В общем, при проектировании стратегии управления исключениями вы должны создавать иерархию исключений и организовывать собственные исключения в ее рамках. Это поможет пользователям быстро анализировать и прослеживать возникающие проблемы. В собственных исключениях должен быть указан слой, в котором было сформировано исключение, компонент, в котором, возможно, возникло исключение, и тип сформированного исключения (такой как исключение безопасности, бизнес-логики или системное исключение).

Храните иерархию исключений приложения в отдельной сборке, на которую может ссылаться код приложения. Это поможет централизовать управление и развертывание классов исключений. Также продумайте, как будет выполняться передача исключений через физические границы, границы процессов или AppDomain. Классы .NET Framework Exception поддерживают сериализацию. При проектировании собственных классов исключений обеспечьте, чтобы они также поддерживали сериализацию.

Шаг 5 – Выбор соответствующих данных для сбора

Один из наиболее важных аспектов при обработке исключений – правильный выбор стратегии сбора данных исключения. Перехватываемые сведения должны точно представлять условие возникновения исключения. Также они должны быть значимыми и информативными для аудитории. Можно выделить три категории аудитории: конечные пользователи, разработчики приложения и операторы. На основании сценария и индивидуального контекста установите, кому адресовано исключение.

Для конечных пользователей требуется осмысленное и хорошо представленное описание. При сборе данных исключения для конечных пользователей позаботьтесь, чтобы они были поданы в виде понятного пользователям сообщения, описывающего природу ошибки и предлагающего пути восстановления после такой ошибки, если это возможно. Разработчикам приложения необходимы более подробные сведения, которые помогут в диагностике проблемы.

Разработчикам приложения необходимы данные о точном месте возникновения исключения в коде, а также такие сведения, как тип исключения и состояние системы в момент возникновения исключения. Операторам службы поддержки должна предоставляться соответствующая информация, которая позволит им реагировать соответствующим образом и предпринять необходимые шаги по восстановлению системы после ошибки. При сборе данных исключения для операторов службы поддержки обеспечьте сведения, которые помогут им найти людей, которых они должны уведомить об исключении, и информацию, которую они должны будут передать для решения проблемы.

Независимо от целевой аудитории всегда полезно обеспечивать исчерпывающую информацию об исключении. Сохраняйте эти сведения в файле журнала для последующей проверки и анализа частоты возникновения исключений и их данных. По умолчанию должны фиксироваться, как минимум, дата и время, имя компьютера, источник исключения и тип, сообщение об исключении, данные о стеке и вызовах, доменное имя приложения, имя и версия сборки, ID потока и сведения о пользователе.

Шаг 6 – Выработка стратегии протоколирования исключений

Существует ряд возможных вариантов протоколирования данных исключения. Следующие основные соображения помогут сделать правильный выбор:

- Используйте **Windows Event Log** (Журнал регистрации событий Windows) или **Windows Eventing 6.0**, если приложение развернуто на одном компьютере, если необходимо использовать существующие инструменты для просмотра журнала, или если надежность является основным требованием.
- Используйте **SQL Database**, если приложение развернуто на ферме или в кластере, если необходимо обеспечить централизованное протоколирование, или если требуется гибкость в структурировании и протоколировании данных исключения.
- Используйте **собственный файл журнала**, если приложение развернуто на одном компьютере, если необходима абсолютная гибкость выбора формата протоколирования, или если хотите просто и легко реализовать хранилище журнала регистрации. Контролируйте размер файла журнала, периодически очищая или архивируя журнал, чтобы он не разрастался в размерах.
- Выбирайте **Message Queuing** в качестве механизма доставки для передачи сообщений об исключениях в точку их окончательного назначения, если надежность является главным требованием, если приложения развернуты на ферме или в кластере, или если требуется централизовать протоколирование.

В любом приложении может использоваться несколько из представленных вариантов в зависимости от сценария и политики обработки исключений. Например, исключения безопасности могут протоколироваться в Security Event Log, а исключения бизнес-логики – в базу данных.

Шаг 7 – Выбор стратегии уведомления об исключениях

При выработке стратегии управления исключениями необходимо также принять решение о стратегии уведомления. Часто в корпоративных приложениях только управления исключениями и протоколирования не достаточно. Необходимо предусмотреть также уведомления, чтобы обеспечить информирование о возникающих исключениях администраторов и операторов. Для этого могут использоваться такие технологии, как события WMI, электронные письма SMTP, текстовые сообщения SMS или другие системы уведомления.

Рассмотрите возможность использования внешних механизмов уведомления, таких как системы мониторинга журналов или среды сторонних производителей, которые выявляют в данных журнала условия возникновения ошибки и формируют соответствующие уведомления.

Такой подход рекомендуется, если требуется отделить систему мониторинга и уведомления от кода приложений, оставив в приложениях только код протоколирования. Но если вы хотите обеспечить немедленное уведомление без использования внешних систем мониторинга, можно ввести в приложение специализированные механизмы уведомления.

Шаг 8 – Принятие решения об обработке необрабатываемых исключений

Если исключение остается необрабатыванным до последней точки или границы и нет возможности восстановления после исключения перед возвращением управления пользователю, приложение должно обработать это необрабатыванное исключение. Для необрабатываемых исключений требуется собрать необходимые сведения, записать их в файл журнала или аудита, разослать все необходимые для этого исключения уведомления, выполнить всю необходимую очистку и, наконец, передать информацию об ошибке пользователю.

Не раскрывайте всех деталей исключения. Предоставьте пользователю понятное универсальное сообщение об ошибке. Для клиентов без пользовательского интерфейса, таких как Веб-сервисы, вместо детального исключения можно сформировать универсальное исключение. Это предотвратит возможность разглашения данных системы конечному пользователю.

Реализуйте стратегию управления исключениями, протоколирования и уведомления с помощью Exception Handling Application Block (Блок обработки исключений) и Logging Application Block (Блок протоколирования), созданных группой patterns & practices. Exception Handling Application Block поддерживает ряд вариантов обработки исключений. Logging Application Block может принимать, форматировать и отправлять сообщения и уведомления журнала в различные журналы и другие точки назначения. Более подробно эти вопросы рассматриваются в приложении F, «[Enterprise Library от patterns & practices](#)».

Этапы проектирования стратегии валидации ввода и данных

Приведенные далее рекомендации помогут правильно выработать стратегию валидации для приложения. При проектировании валидации ввода и данных для приложения, прежде всего, необходимо обозначить границы доверия и ключевые сценарии того, когда должна проводиться валидация данных. Далее определяются данные, подлежащие валидации, и место, где эта валидация должна выполняться. Также необходимо найти, как обеспечить возможность повторного использования стратегии валидации при ее реализации. И, наконец, определить соответствующую стратегию валидации для приложения.

Шаг 1 – Определение границ доверия

Границы доверия разделяют данные из доверенного источника и данные, к которым нет доверия. Данные, находящиеся с одной стороны границы доверия, являются доверяемыми, и с другой стороны границы располагаются не имеющие доверия данные. Чтобы понять, что придется проверять, необходимо, прежде всего, выявить данные, пересекающие границы доверия. Используйте валидацию ввода и данных на каждой границе доверия, чтобы сократить угрозы безопасности, такие как межсайтовая атака внедрением сценариев и

внедрение кода. Примерами границ доверия могут служить пограничный межсетевой экран, граница между Веб-сервером и сервером базы данных, граница между приложением и сервисом стороннего производителя.

Определите, какие системы и подсистемы будут взаимодействовать с приложением, и внешние границы системы, пересекаемые при записи данных в файлы на сервере, вызове сервера базы данных или вызове Веб-сервиса. Обозначьте точки входа на границах доверия и точки выхода, в которых выполняется запись данных из клиентского ввода или из недоверяемых источников, таких как совместно используемые базы данных.

Шаг 2 – Определение ключевых сценариев

После того, как границы доверия в приложении обозначены, необходимо определиться с ключевыми сценариями, в которых требуется валидация данных. Все вводимые пользователем данные до тех пор, пока не пройдут валидацию, должны рассматриваться как злонамеренные. Например, в слое представления Веб-приложения подлежат проверке поля форм, строки запросов и скрытые поля; параметры запросов GET и POST; загруженные данные (злонамеренные пользователи могут перехватывать HTTP-запросы и изменять содержимое); и cookies (которые располагаются на клиентском компьютере и могут быть изменены).

В бизнес-слое ограничения на данные налагаются бизнес-правила. Любое нарушение этих правил рассматривается как ошибка валидации, и бизнес-слой должен сформировать исключение для представления этого нарушения. При использовании подсистемы управления правилами или рабочим процессом необходимо обеспечить им проверку результатов каждого правила на основании данных, требуемых этим правилом, и выводов, полученных в результате проверки предыдущих правил.

Шаг 3 – Выбор места валидации

На этом этапе определяемся с местом проведения валидации: на клиенте или на сервере, и на клиенте. Никогда не полагайтесь только на валидацию на стороне клиента. Используйте ее для обеспечения более интерактивного UI, но всегда реализуйте также и валидацию на стороне сервера, чтобы безопасно проверить данные в рамках границ доверия. Валидация данных и бизнес правил на клиенте может сократить количество циклов запрос-ответ к серверу и улучшить взаимодействие с пользователем. Для Веб-приложения браузер клиента должен поддерживать DHTML и JavaScript, в идеальном варианте реализованный в отдельном файле .js для обеспечения возможности повторного использования и кэширования браузером. Самый простой подход в Веб-приложении – применение элементов управления валидацией ASP.NET. Это набор серверных элементов управления, которые могут проводить валидацию данных на стороне клиента, и также будут автоматически проверять их на стороне сервера.

В Веб-приложении проверка данных и бизнес-правил на стороне сервера может быть реализована с использованием элементов управления валидации ASP.NET. Также, как альтернативный вариант для Веб- и других типов приложений, используйте Validation Application Block (Блок валидации) от группы patterns & practices. Validation Application Block поможет создать логику валидации, пригодную для повторного использования в разных слоях. Validation Application Block может применяться в приложениях Windows Forms, ASP.NET и WPF.

Подробно Validation Application Block рассматривается в приложении F, «[Enterprise Library от patterns & practices](#)».

Шаг 4 – Выработка стратегий валидации

Рассмотрим общие стратегии валидации данных:

- **Прием заведомо допустимого** (*Список разрешенного ввода* или позитивная проверка). Принимаются только данные, удовлетворяющие заданным критериям, все остальные данные отклоняются. По возможности, придерживайтесь этой стратегии, поскольку она обеспечивает наивысшую безопасность.
- **Отклонение заведомо недопустимого** (*Список запрещенного ввода* или негативная проверка). Принимаются данные, не отвечающие заданному критерию (например, не содержащие известный набор символов). Применяйте эту стратегию с осторожностью и только как вторую линию защиты, поскольку очень сложно создать исчерпывающий список критериев для всех известных недействительных вводов.
- **Очистка.** Известные плохие символы или значения устраняются или преобразовываются с целью сделать ввод безопасным. Как и подход со списком запрещенного ввода (негативная проверка), применяйте эту стратегию с осторожностью и только как вторую линию защиты, поскольку очень сложно создать исчерпывающий список критериев для всех известных недействительных вводов.

Шаблоны проектирования

Основные шаблоны проектирования, касающиеся сквозной функциональности, организованы по категориям и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Кэширование	Cache Dependency (<i>Кэш с зависимостью</i>). Использует внешние данные для определения состояния данных, хранящихся в кэше. Page Cache (<i>Кэш страницы</i>). Улучшает время ответа динамических Веб-страниц, доступ к которым осуществляется довольно часто, но сами они меняются реже и потребляют большое количество ресурсов системы для воссоздания.
Сетевое взаимодействие	Intercepting Filter (<i>Перехватывающий фильтр</i>). Цепочка пригодных для компоновки фильтров (независимые модули), реализующих обычные задачи предварительной и последующей обработки при запросе Веб-страницы. Pipes and Filters (<i>Каналы и фильтры</i>). Выполняет маршрутизацию сообщений по каналам и фильтрам, которые могут изменять или проверять сообщение при его прохождении по каналу. Service Interface (<i>Интерфейс сервиса</i>). Программный интерфейс, который может использоваться другими системами для

	взаимодействия с сервисом.
--	----------------------------

Более подробно шаблоны Page Cache, Intercepting Filter и Service Interface рассматриваются в статье «*Enterprise Solution Patterns Using Microsoft .NET*» по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.

Более подробно шаблон Pipes and Filters рассматривается в статье «*Integration Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/ms978729.aspx>.

Предложения группы patterns & practices

Узнать о дополнительных предложениях группы Microsoft patterns & practices можно из следующих источников:

- **Enterprise Library** включает наборы блоков приложений, использование которых упростит реализацию типовых задач, таких как кэширование, обработка исключений, валидация, протоколирование, шифрование, управление учетными данными, а также средства для реализации таких шаблонов, как Inversion of Control (Инверсия управления) и Dependency Injection. Подробнее рассматривается в статье «*Microsoft Enterprise Library*» по адресу <http://msdn2.microsoft.com/en-us/library/cc467894.aspx>.
 - **Unity Application Block** легковесный расширяемый контейнер внесения зависимостей, который будет очень полезен при создании слабо связанных приложений. Более подробно рассматривается в статье «*Unity Application Block*» по адресу <http://msdn.microsoft.com/en-us/library/cc468366.aspx>.
-

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

Больше сведений по аутентификации и авторизации можно найти в следующих статьях:

- «*Authorization*» по адресу <http://msdn.microsoft.com/en-us/library/cc949059.aspx>.
- «*Authorization In WCF-Based Services*» (Авторизация в WCF-сервисах) по адресу <http://msdn.microsoft.com/en-us/magazine/cc948343.aspx>.
- «*Designing Application-Managed Authorization*» (Проектирование управляемой приложением авторизации) по адресу <http://msdn.microsoft.com/en-us/library/ms954586.aspx>.
- «*Enterprise Authorization Strategy*» (Корпоративная стратегия авторизации) по адресу <http://msdn.microsoft.com/en-us/library/bb417064.aspx>.
- «*Federated Identity: Scenarios, Architecture, and Implementation*» (Интегрированная идентификация: сценарии, архитектура и реализация) по адресу <http://msdn.microsoft.com/en-us/library/aa479079.aspx>.

- «*Guidance on Patterns & Practices: Security*» (Руководство Patterns & Practices: безопасность) по адресу <http://msdn.microsoft.com/en-us/library/ms954624.aspx>.
- «*Trusted Subsystem Design*» (Проектирование доверенных подсистем) по адресу <http://msdn.microsoft.com/en-us/library/aa905320.aspx>.

Дополнительную информацию по все остальным темам, затрагиваемым в данной главе, можно найти в следующих статьях:

- «*Caching Architecture Guide for .NET Framework Applications*» (Руководство по архитектуре кэширования для приложений .NET Framework) по адресу <http://msdn.microsoft.com/en-us/library/ms978498.aspx>.
- «*Cohesion and Coupling*» (Связность и связанность) по адресу <http://msdn.microsoft.com/en-us/magazine/cc947917.aspx>.
- Duffy, Joe. Concurrent Programming on Windows¹. Addison-Wesley 2009.
- «*Enterprise Solution Patterns Using Microsoft .NET*» по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.
- «*Exception Management Architecture Guide*» (Руководство по архитектуре управления исключениями) по адресу <http://msdn.microsoft.com/en-us/library/ms954599.aspx>.
- «*Integration Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/ms978729.aspx>.
- «*Microsoft Project Code Named Velocity*» по адресу <http://msdn.microsoft.com/en-us/data/cc655792.aspx>.

Сведения о некоторых популярных библиотеках и инфраструктурах сторонник производителей, которые могут быть полезны при реализации сквозной функциональности, можно найти в следующих источниках:

- Castle Project по адресу <http://www.castleproject.org/index.html>.
- Ninject по адресу <http://ninject.org/>.
- PostSharp по адресу <http://www.postsharp.org/>.
- StructureMap по адресу <http://structuremap.sourceforge.net/Default.htm>.
- memcached по адресу <http://www.danga.com/memcached/>.
- NLog по адресу <http://www.nlog-project.org/>.
- log4net по адресу <http://logging.apache.org/log4net/>.

¹ Джо Даффи, «Параллельное программирование в Windows»

18

Взаимодействие и обмен сообщениями

Обзор

Одним из ключевых факторов, оказывающим влияние на дизайн приложения, особенно распределенного приложения, является дизайн инфраструктуры связи для каждой части приложения. Компоненты должны взаимодействовать, например, передавать пользовательский ввод в бизнес-слой и затем обновлять хранилище данных через слой доступа к данным. Когда компоненты физически располагаются на одном уровне, между ними возможно прямое взаимодействие. Однако при развертывании компонентов и слоев физически на разных серверах и клиентских компьютерах, как это происходит в большинстве сценариев, необходимо продумать, как обеспечить эффективное и надежное взаимодействие компонентов этих уровней.

Как правило, приходится выбирать между прямым взаимодействием (таким как вызов методов между компонентами) и связью посредством обмена сообщениями. В связи посредством обмена сообщениями множество преимуществ, таких как возможность разделения компонентов. Уменьшение связанности компонентов не только улучшает обслуживаемость, но также может обеспечить гибкость, что упростит изменение стратегии развертывания в будущем. Однако в связи с обменом сообщениями возникают такие проблемы, как производительность, надежность и безопасность.

В данной главе представлены рекомендации, которые помогут правильно выбрать подход к реализации связи; понять, как добиться наилучших результатов от выбранного подхода, и предугадать возможные проблемы в области безопасности и надежности. Однако основная часть этой главы посвящена проектированию подходящего механизма связи посредством обмена сообщениями. Здесь приводятся рекомендации по асинхронному и синхронному взаимодействию, выбору формата данных, обеспечению производительности, безопасности, возможности взаимодействия и выбору технологии реализации.

Общие принципы проектирования

При проектировании стратегии связи для приложения необходимо учесть, как влияет на производительность взаимодействие между слоями и уровнями. Поскольку любое взаимодействие через логические или физические границы повышает издержки на обработку,

проектирование должно обеспечить эффективную связь за счет сокращения обращений к сети и уменьшению объемов данных, передаваемых по сети. Разрабатывая стратегию связи, руководствуйтесь следующими рекомендациями:

- **Продумайте стратегии связи при пересечении границ.** Продумайте все границы и то, как они влияют на производительность связи. Например, процесс компьютера, компьютер и переходы между управляемым и неуправляемым кодом – все это границы, которые могут пересекаться при взаимодействии компонентов приложения или внешних сервисов и приложений.
- **Используйте связь посредством обмена сообщениями при пересечении границ процесса.** Для обеспечения максимальной производительности используйте Windows Communication Foundation (WCF) и TCP или протокол именованных каналов.
- **Используйте связь посредством обмена сообщениями при пересечении физических границ.** Используйте WCF для организации связи через физические границы с удаленными компьютерами. Используйте Microsoft Message Queuing для обеспечения надежной одноразовой доставки сообщений.
- **Обеспечьте максимальную производительность и скорость ответа при доступе к удаленным слоям.** При взаимодействии с удаленными слоями используйте слабо детализированные методы взаимодействия посредством обмена сообщениями для сокращения требуемого объема информационного обмена, а также по возможности применяйте асинхронное взаимодействие, чтобы избежать блокировки UI или перехода его в состояние «не отвечает».
- **Для форматов данных, передаваемых через границы, учтите возможности сериализации.** Если необходимо обеспечить возможность взаимодействия с другими системами, используйте сериализацию XML. Не забывайте, что сериализация XML обуславливает дополнительные издержки. Если производительность является основным фактором, используйте бинарную сериализацию, потому что она более производительна и размер получаемых в результате сериализованных данных меньше, чем у их XML-эквивалента.
- **Обеспечьте защиту сообщений и конфиденциальных данных при взаимодействии.** Используйте шифрование, цифровые сертификаты и средства обеспечения безопасности канала.
- **Реализуйте механизмы, обеспечивающие идемпотентность и коммутативность.** Обеспечьте, чтобы код приложения мог выявлять и обрабатывать сообщения, поступающие более одного раза (идемпотентность), и множество сообщений, поступающих в произвольном порядке (коммутативность).

Рекомендации по реализации связи посредством обмена сообщениями

Связь посредством обмена сообщениями позволяет предоставлять сервис вызывающим сторонам через интерфейс сервиса, доступный для вызова клиентами путем передачи XML-сообщений по транспортному каналу. Как правило, вызовы посредством обмена сообщениями выполняются с удаленных клиентов, но интерфейсы сервисов, основанные на сообщениях, могут поддерживать и локальные вызовы. Связь посредством обмена сообщениями лучше всего подходит для следующих сценариев:

- Реализация бизнес-системы, представляющей промежуточное звено в долгосрочных инвестициях; например, при создании сервиса, который будет предоставлен для использования партнерами в течение длительного времени.
- Реализация крупномасштабных систем, которые должны предлагать высокую доступность или взаимодействовать через ненадежные сети. В этом случае механизм хранения и передачи сообщений может обеспечить более высокую надежность.
- Создание сервиса, который требуется изолировать от остальных используемых им и использующих его сервисов. Применение интерфейсов сервисов на основе сообщений, раскрывающих детали реализации интерфейса клиентам, упрощает использование сервиса любыми клиентами без необходимости создания специальных реализаций для отдельных клиентов.
- Работа с реальными бизнес-процессами, использующими асинхронную модель.

При использовании связи на основе сообщений руководствуйтесь следующими рекомендациями:

- Предусмотрите то, что подключение будет доступно не всегда, и что может понадобиться сохранять сообщения с последующей их передачей, когда подключение становится доступным.
- Продумайте, как обрабатывать ситуацию, когда ответ на сообщение не поступил. Для сохранения состояния взаимодействия бизнес-логика может протоколировать отправленные сообщения для последующей обработки на случай, если ответ не получен.
- Используйте подтверждения о получении для обеспечения правильной последовательности передачи сообщений.
- Используйте стандартные протоколы, такие как HTTP для связи через Интернет и TCP для связи по внутренним сетям. Реализуйте собственный механизм связи только в случае, если ни одна стандартная комбинация конечной точки, протокола и формата не отвечает заданным требованиям.
- Если время ответа на сообщения является критичным для взаимодействия, используйте синхронную модель программирования, при которой клиенты

ожидают ответа на каждое сообщение. Или, если клиенты могут продолжать выполнение, пока ожидают ответ, используйте асинхронную модель.

При проектировании стратегии связи посредством обмена сообщениями также необходимо учесть специальные вопросы, которые могут оказывать влияние на стабильность, возможность повторного использования, производительность и общий успех дизайна. Более подробно эти вопросы рассматриваются в следующих разделах:

- [Сравнение асинхронного и синхронного взаимодействия](#)
 - [Связывание и связность](#)
 - [Форматы данных](#)
 - [Возможность взаимодействия с другими системами](#)
 - [Производительность](#)
 - [Управление состоянием](#)
-

Сравнение асинхронного и синхронного взаимодействия

Делая выбор между синхронным и асинхронным стилем взаимодействия, учитывайте основные преимущества и недостатки. Синхронное взаимодействие лучше подходит для сценариев, в которых должен быть гарантирован порядок поступления вызовов, или если требуется ожидать ответа на вызов. Асинхронное взаимодействие применяется в сценариях, в которых большое значение имеет время отклика или невозможно гарантировать доступность точки назначения. Выбирая между синхронным и асинхронным взаимодействием, руководствуйтесь следующими рекомендациями:

- Для обеспечения максимальной производительности, слабого связывания и сокращения издержек системы используйте модель асинхронного взаимодействия. Если некоторые клиенты могут выполнять только синхронные вызовы, заключите существующие асинхронные методы сервиса в компонент, осуществляющий синхронное взаимодействие с клиентом.
 - Если необходимо обеспечить гарантированный порядок выполнения операций, или при использовании операций, зависящих от результатов выполнения предыдущих операций, используйте синхронную модель.
 - Для асинхронных вызовов внутри процесса используйте средства платформы (такие как версии **Begin** и **End** методов и обратные вызовы) для реализации асинхронных вызовов методов. Для асинхронных вызовов вне процесса, таких как вызовы через физические уровни и границы, используйте обмен сообщениями или асинхронные запросы сервисов.
-

Если вы выбрали асинхронное взаимодействие и не можете гарантировать возможность сетевого подключения или доступность точки назначения, избежать утери сообщений поможет механизм сохранения сообщений с последующей доставкой. При использовании стратегии сохранения с последующей отправкой организовывайте сохранение сообщений на

случай недоступности системы или сети в локальных кэшах. В качестве альтернативы можно применить Message Queuing. Этот механизм в случае недоступности или сбоя системы или сети обеспечит помещение сообщений в очередь для их последующей доставки. Message Queuing может осуществлять транзакционную доставку сообщений и поддерживает надежную одноразовую доставку. Если необходимо обеспечить взаимодействие с другими системами и платформами на уровне предприятия или осуществлять обмен электронными данными, используйте BizTalk Server. Он обеспечит надежный механизм доставки.

Связывание и связность

Применение методов взаимодействия, налагающих взаимозависимости между распределенными частями приложения, приведет к созданию тесно связанного приложения. Слабо связанное приложение использует методы, выдевающие минимальный набор требований для реализации взаимодействия. При проектировании связывания и связности руководствуйтесь следующими рекомендациями:

- Для обеспечения слабой связанности используйте основанную на сообщениях технологию, такую как ASP.NET Web services (ASMX) или WCF, и данные с самоописанием и общепринятые протоколы, такие как HTTP, REST и SOAP.
 - Для обеспечения связности включайте в интерфейсы только методы, касающиеся их назначения и функциональной области.
-

Форматы данных

Для передачи данных между уровнями чаще всего используются скалярные значения, XML, объекты **DataSets** и собственные объекты. В следующей таблице представлены ключевые факторы, оказывающие влияние на выбор типа данных.

Тип	Факторы
Скалярные значения	Требуется обеспечить встроенную поддержку сериализации. Позволяют обрабатывать возможные изменения схемы. Скалярные значения обусловливают тесное связывание, которое потребует изменения сигнатур методов, таким образом, оказывая влияние на вызывающий код.
XML	Необходимо слабое связывание, при котором вызывающая сторона должна знать только о данных, определяющих бизнес-сущность, и схеме, обеспечивающей метаданные для бизнес-сущности. Необходимо поддерживать разные типы вызывающих сторон, включая клиентов сторонних производителей. Требуется обеспечить встроенную поддержку сериализации.
<i>DataSet</i>	Необходимо обеспечить поддержку сложных структур данных. Необходимо обрабатывать наборы и сложные отношения. Необходимо отслеживать изменения данных DataSet . Требуется обеспечить встроенную поддержку сериализации.
Собственные объекты	Необходимо обеспечить поддержку сложных структур данных. Взаимодействие осуществляется с компонентами, которым известен тип объекта.

	Желательна поддержка бинарной сериализации для обеспечения лучшей производительности.
--	---

При выборе формата данных для канала связи руководствуйтесь следующими рекомендациями:

- Если приложение работает, главным образом, с данными экземпляра, использование простых значений обеспечит лучшую производительность. Применение простых типов значений позволит сократить исходные затраты на разработку, но обусловит тесное связывание, которое может привести к повышению затрат на обслуживание в случае изменения типов в будущем.
- Применение XML может потребовать дополнительного предварительного описания схемы, но обеспечит слабое связывание, что может обусловить сокращение затрат на обслуживание и повысить возможности взаимодействия в будущем (например, если понадобится дополнительно предоставить интерфейс новым поддерживающим XML вызывающим сторонам).
- Объекты **DataSet** прекрасно подойдут для составных типов данных, особенно если они заполняются непосредственно из базы данных. Однако важно понимать, что объекты **DataSet** также включают сведения о схеме и состоянии, которые увеличивают общий объем передаваемых по сети данных, и специфика их формата может ограничить возможности взаимодействия с другими системами. Используйте объекты **DataSet**, если приложение, главным образом, работает с множествами данных и требует такой функциональности, как сортировка, поиск и привязка данных.
- Собственные объекты применяются в случаях, когда ни один из других вариантов не удовлетворяет требованиям, или при взаимодействии с компонентами, которые ожидают использования собственного объекта. Такие объекты обусловливают меньшие издержки, чем объекты **DataSet**, и поддерживают и бинарную, и XML-сериализацию. Собственные объекты, используемые для передачи данных по каналам связи, являются объектами передачи данных (data transfer objects, DTO), в которых содержатся данные, извлеченные из бизнес-сущностей.
- Обеспечьте, чтобы сведения о типе не утрачивались в процессе передачи данных. Точность воспроизведения типа обеспечивает бинарная сериализация. Ее рекомендуется использовать для передачи объектов между клиентом и сервером. Однако такой подход означает необходимость реализации более строгой системы контроля версий для интерфейсов. Стандартная сериализация XML обеспечивает сериализацию только открытых свойств и полей и не сохраняет типы.

Возможность взаимодействия с другими системами

Основными факторами, оказывающими влияние на возможность взаимодействия приложений и компонентов, являются доступность подходящих каналов связи и форматов и протоколов,

понятных участникам взаимодействия. Предлагаем следующие рекомендации по повышению возможности взаимодействия:

- Для обеспечения возможности взаимодействия с широкой номенклатурой платформ и устройств используйте стандартные протоколы и форматы данных, такие как HTTP и XML. Не забывайте, что выбор протокола может оказывать влияние на доступность целевых клиентов. Например, целевая система может быть защищена межсетевыми экранами, блокирующими некоторые протоколы.
- Продумайте вопросы управления версиями для интерфейсов и контрактов. Необходимость в изменении сервисов может возникнуть из-за изменения бизнес-требований, технологических требований или по другим причинам. В случаях, когда изменения приводят к несовместимости интерфейса, контракта сообщений или данных, создавайте новую версию, которая будет доступна клиентам наряду со старой версией. Старую версию смогут использовать клиенты, не нуждающиеся в новой функциональности, предоставляемой новым интерфейсом. Более подробно эти вопросы рассматриваются в статье «*Service Versioning*» (Контроль версий сервисов) по адресу <http://msdn.microsoft.com/en-us/library/ms731060.aspx>.
- Выбор формата данных может влиять на возможность взаимодействия. Например, целевые системы могут не понимать типы, характерные для той или иной платформы, или по-разному выполнять обработку и сериализацию типов.
- Принимаемые решения по шифрованию и дешифрованию могут влиять на возможность взаимодействия. Например, некоторые алгоритмы шифрования/десифрования доступны не во всех системах.

Производительность

Дизайн интерфейсов и используемые форматы данных также оказывают существенное влияние на производительность приложения, особенно при пересечении границ процесса или физических границ. В то время как для других аспектов, таких как обеспечение возможности взаимодействия, может потребоваться применение специальных интерфейсов и форматов данных, улучшить производительность взаимодействия между разными слоями или уровнями приложения можно с помощью определенных методик. При проектировании производительности руководствуйтесь следующими рекомендациями:

- По возможности, не передавайте лишние данные в удаленные методы и максимально сократите объем передаваемых по сети данных. Это обеспечит сокращение задержки сети и издержек на сериализацию. Кроме того, для взаимодействия между процессами и компьютерами избегайте применения детализированных интерфейсов, потому что в этом случае клиент вынужден выполнять множество вызовов метода для осуществления одной логической единицы работы. Применяйте шаблон Façade, он обеспечит слабо детализированную оболочку для существующих детализированных интерфейсов.
- Используйте DTO-объекты для передачи данных единым элементом, вместо того чтобы передавать отдельные типы данных по одному.

- Если производительность сериализации критична для приложения, применяйте специальные классы с бинарной сериализацией.
 - Если для обеспечения возможности взаимодействия требуется применение XML, используйте структуры, основанные на атрибутах, а не на элементах.
-

Управление состоянием

Для сторон взаимодействия в приложении может понадобиться сохранять состояние на протяжении множества запросов. Принимая решение о реализации управления состоянием, руководствуйтесь следующими рекомендациями:

- Сохраняйте состояние между вызовами только в случае крайней необходимости. Для сохранения состояния требуются ресурсы, что может повлиять на производительность приложения и ограничить возможные варианты развертывания.
 - При использовании в компоненте или сервисе модели программирования с сохранением состояния храните данные состояния с использованием механизмов длительного хранения и выполняйте доступ к этим данным с помощью маркера.
 - При проектировании ASMX-сервиса для хранения состояния используйте класс **ApplicationContext** (Контекст приложения), потому что он обеспечивает доступ к стандартным хранилищам данных области определения приложения и сеанса.
 - При проектировании WCF-сервиса используйте расширяемые объекты, предоставляемые платформой для управления состоянием. Эти расширяемые объекты обеспечивают возможность сохранять состояние в различных областях определения, таких как хост сервиса, контекст экземпляра сервиса и контекст операции. Обратите внимание, что все эти состояния удерживаются в памяти и не хранятся длительное время. Если требуется обеспечить длительное сохранение состояния, можно использовать механизм длительного хранения, введенный в .NET Framework 3.5, или реализовать собственное решение.
-

Контрактно-ориентированное проектирование

Традиционно разработчики создавали сервисы, применяя подход *сначала код* (*code first*), при котором проектирование сервиса выполняется на основании требований и предоставляется интерфейс, соответствующий коду и требованиям. Однако подход *сначала контракт* (*contract first*) приобретает всю большую популярность, поскольку обеспечивает снижение несовместимости, которая возможна между несопоставимыми системами и широкой номенклатурой клиентов.

Контрактно-ориентированное проектирование – это процесс проектирования контракта сервиса с точки зрения данных, сообщений и интерфейса, которые он будет предоставлять, с последующим формированием кода интерфейса сервиса на основании этого контракта. После этого в файле выделенного кода интерфейса сервиса можно реализовать необходимые операции. Это позволяет в самом начале процесса сосредоточиться на формате сообщений и

используемых ими типах данных, что способствует повышению возможности взаимодействия и совместимости.

При проектировании интерфейса можно пользоваться средствами моделирования, например, Web Service Software Factory: Modeling Edition (<http://msdn.microsoft.com/servicefactory/>) группы patterns & practices. Или можно создать дизайн интерфейса, используя XML, XSD и схемы, а затем с помощью такого инструмента, как WSDL.exe, с ключом /server сгенерировать описание интерфейса. Технологии с применением шины сообщений, такие как Microsoft BizTalk Server, способствуют использованию принципов контрактно-ориентированного проектирования.

При контрактно-ориентированном проектировании необходимо помнить следующие принципы:

- Работа с XML-схемами и типами данных означает, что вы не оперируете и не можете оперировать категориями типов данных, характерных для той или иной платформы. Это может усложнить описание интерфейса, но обеспечит максимальную возможность взаимодействия и совместимость. Там где требуется применение сложных структур данных, составляйте их из простых и стандартных XML-типов, которые могут использоваться любыми клиентами.
- Рассмотрите платформы, клиентов и системы, которые могут взаимодействовать с сервисом. Учтите все возможные ограничения по применяемым типам данных или форматам.
- Применяйте специальные инструменты для проектирования контрактов сервиса. Это может существенно упростить и ускорить процесс.
- По возможности в процессе проектирования контракта тесно сотрудничайте с заинтересованными сторонами. Они могут выдвигать особые требования или предложения, которые упростят использование контракта, обеспечат его более широкое распространение и повысят возможности повторного использования.

Более подробно контрактно-ориентированное проектирование рассматривается в статье «*Contract-First Service Development*» (Контрактно-ориентированная разработка сервисов) по адресу <http://msdn.microsoft.com/en-us/magazine/cc163800.aspx>.

Вопросы безопасности

Применение стратегии безопасной связи обеспечит защиту конфиденциальных данных от чтения, повреждения и подделки при передаче по сети и, в случае необходимости, гарантирует подлинность вызывающей стороны. Существует две основные области обеспечения безопасности при взаимодействии: безопасность на транспортном уровне и безопасность на уровне сообщения. Максимальную защиту обеспечит сочетание этих двух методик.

Безопасность на транспортном уровне

Безопасность на транспортном уровне используется для обеспечения сквозной безопасности между двумя конечными точками при передаче учетных данных и утверждений пользователя. Защита канала предотвращает возможный злонамеренный доступ ко всем сообщениям, передаваемым по каналу. Общими подходами обеспечения безопасности на транспортном уровне являются применение шифрования по протоколам Secure Sockets Layer (SSL) и Internet Protocol Security (IPSec). Принимая решение об использовании безопасности на транспортном уровне, руководствуйтесь следующими соображениями:

- Для обеспечения безопасности на транспортном уровне используются общие отраслевые стандарты, что создает хорошие условия для взаимодействия и обычно ускоряет шифрование и создание цифровых подписей, поскольку все выполняется на более низких уровнях, иногда даже на уровне сетевого оборудования. Однако в этом случае поддерживается ограниченный набор учетных данных и утверждений по сравнению с безопасностью на уровне сообщения.
- Если взаимодействия между сервисом и потребителем осуществляется без посредников, может использоваться безопасность на транспортном уровне. Если сообщения проходят через одного или более посредников, следует применять безопасность на уровне сообщения, поскольку, в противном случае, сообщение дешифруется и повторно шифруется на каждом посреднике, что представляет угрозу безопасности.
- Безопасность на транспортном уровне является хорошим методом защиты взаимодействия между клиентом и сервисом, располагающимися в частной сети, такой как интранет.

Безопасность на уровне сообщения

Безопасность на уровне сообщения может использоваться с любым транспортным протоколом. Содержимое отдельных сообщений должно защищаться при передаче по каналу за пределы собственной безопасной сети и даже в рамках собственной сети для очень важного содержимого. Общими подходами к обеспечению безопасности на уровне сообщения являются шифрование и цифровые подписи. Принимая решение об использовании безопасности на уровне сообщения, руководствуйтесь следующими соображениями

- Реализуйте безопасность на уровне сообщения для конфиденциальных сообщений, передаваемых за пределы защищенной сети, как например, для сервисов, предоставляемых через Интернет. Но не забывайте, что безопасность на уровне сообщения, как правило, имеет большее негативное влияние на производительность, чем безопасность на транспортном уровне. Для повышения общей производительности можно использовать частичное или выборочное шифрование и подпись сообщений.
- При наличии посредников во взаимодействии клиента и сервиса используйте для конфиденциальных сообщений безопасность на уровне сообщения, потому что это гарантирует сквозную защиту. Серверы-посредники при получении сообщения

разрывают подключение SSL или IPSec и затем создают новое подключение SSL или IPSec для передачи этого сообщения следующему серверу. Поэтому, если сообщения не защищены, существует риск доступа к ним на промежуточном сервере.

Технологии

Платформа Microsoft предлагает две технологии обмена сообщениями: Windows Communication Foundation (WCF) и ASP.NET Web Services (ASMX). В следующих разделах будут представлены возможности каждой из них и рекомендации по выбору наиболее подходящей для конкретных сценариев.

Технология WCF

WCF обеспечивает полноценный механизм реализации сервисов для широкого диапазона ситуаций и обеспечивает возможность точного регулирования конфигурации и содержимого сервисов. Следующие рекомендации помогут понять возможности использования WCF:

- Используйте WCF в следующих ситуациях:
 - Для взаимодействия с Веб-сервисами, когда необходимо обеспечить возможность взаимодействия с другими платформами, которые также поддерживают SOAP, такими как серверы приложений на базе J2EE.
 - Для взаимодействия с Веб-сервисами, использующими не SOAP-сообщения, например, приложениями, работающими с такими форматами, как Really Simple Syndication (RSS).
 - Для взаимодействия посредством SOAP-сообщений и двоичного кодирования для структур данных, когда и сервер, и клиент используют WCF.
 - Для создания сервисов REST Singleton и Collection Services, ATOM Feed и Publishing Protocol Services, а также HTTP Plain XML Services.
 - Используйте WS-MetadataExchange в SOAP-запросах для получения подробных сведений о сервисе, таких как описание на языке описания Веб-сервисов (Web Services Description Language, WSDL) и политики.
 - Используйте WS-Security для реализации аутентификации, целостности данных, конфиденциальности данных и других функций безопасности.
 - Используйте WS-Reliable Messaging для реализации надежной сквозной связи даже при наличии одного или более промежуточных Веб-сервисов.
 - Используйте WS-Coordination для координации двухфазного подтверждения транзакции в контексте взаимодействий Веб-сервисов.
-

WCF поддерживает несколько разных протоколов связи:

- Для сервисов, доступ к которым осуществляется через Интернет, используйте протокол HTTP.
 - Для сервисов, доступ к которым осуществляется в рамках частной сети, используйте протокол TCP.
 - Для сервисов, доступ к которым осуществляется с того же компьютера, на котором они размещаются, используйте протокол именованных каналов, поддерживающий совместно используемый буфер или потоковую передачу данных.
-

Технология ASMX

ASMX обеспечивает более простое решение для создания Веб-сервисов на базе ASP.NET и их предоставления через Веб-сервер IIS. ASMX имеет следующие характеристики:

- Обеспечивает возможность доступа через Интернет с использованием только протокола HTTP. По умолчанию использует порт 80, но изменить это не составляет никакого труда.
 - Не поддерживает транзакций координатора распределенных транзакций (Distributed Transaction Coordinator, DTC). Для программирования длительных транзакций необходимо использовать собственные реализации.
 - Поддерживает аутентификацию IIS, роли, хранящиеся как группы Windows для авторизации, олицетворение IIS и ASP.NET и безопасность на транспортном уровне с использованием протокола SSL.
 - Поддерживает технологию конечных точек, реализованную в IIS.
 - Обеспечивает возможность межплатформенного взаимодействия.
-

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Data Transfer and Serialization*» (Передача и сериализация данных) по адресу <http://msdn.microsoft.com/en-us/library/ms730035.aspx>.
- «*Endpoints: Addresses, Bindings, and Contracts*» (Конечные точки: адреса, привязки и контракты) по адресу <http://msdn.microsoft.com/en-us/library/ms733107.aspx>.
- «*Messaging Patterns in Service-Oriented Architecture*» по адресу <http://msdn.microsoft.com/en-us/library/aa480027.aspx>.
- «*Principles of Service Design: Service Versioning*» (Принципы проектирования сервисов: контроль версий) по адресу <http://msdn.microsoft.com/en-us/library/ms954726.aspx>.

- «*Web Service Messaging with Web Services Enhancements 2.0*» (Web Service Messaging с расширениями Web Services Enhancements 2.0) по адресу <http://msdn.microsoft.com/en-us/library/ms996948.aspx>.
 - «*Web Services Protocols Interoperability Guide*» (Руководство по возможности взаимодействия протоколов Веб-сервисов) по адресу <http://msdn.microsoft.com/en-us/library/ms734776.aspx>.
 - «*Windows Communication Foundation Security*» (Безопасность Windows Communication Foundation) по адресу <http://msdn.microsoft.com/en-us/library/ms732362.aspx>.
 - «*XML Web Services Using ASP.NET*» (Веб-сервисы XML с использованием ASP.NET) по адресу <http://msdn.microsoft.com/en-us/library/ba0z6a33.aspx>.
-

19

Физические уровни и развертывание

Обзор

Дизайны архитектуры приложений существуют в виде моделей, документов и сценариев. Однако приложения развертываются в физической среде, ограничения которой могут вносить корректизы в некоторые архитектурные решения. Следовательно, предполагаемые сценарий развертывания и инфраструктура должны рассматриваться как часть процесса проектирования приложения. В данной главе описываются возможные варианты развертывания различных типов приложений, включая распределенное и нераспределенное развертывание, пути масштабирования приложения, а также руководство и подходы к обеспечению производительности, надежности и безопасности. Рассматривая возможные сценарии развертывания приложения как часть процесса его проектирования, вы предотвращаете ситуацию неудачного развертывания приложения или невозможности выполнения тех или иных требований дизайна из-за технических ограничений инфраструктуры.

Выбор стратегии развертывания сопряжен с нахождением компромиссов в дизайне. Они могут быть связаны с ограничениями по использованию протоколов взаимодействия или портов либо особыми топологиями развертывания, не поддерживаемыми целевой инфраструктурой. Выявление ограничений развертывания на ранних этапах проектирования поможет избежать сюрпризов в будущем. Привлекайте к этой работе группы обслуживания сети и инфраструктуры. При выборе стратегии развертывания:

- Изучите целевую физическую инфраструктуру развертывания.
- Исходя из инфраструктуры развертывания, выявите ограничения архитектуры и дизайна.
- Выявите, какое влияние на безопасность и производительность разрабатываемой системы будет оказывать инфраструктура развертывания.

Распределенное и нераспределенное развертывание

При выработке стратегии развертывания, прежде всего, необходимо определиться, какая модель развертывания будет использоваться: распределенное или нераспределенное

развертывание. Если создается простое приложение для применения во внутренних сетях, подойдет нераспределенное развертывание. Для более сложного приложения, которое должно быть оптимизировано для обеспечения масштабируемости и удобства обслуживания, применяйте распределенное развертывание.

Нераспределенное развертывание

При нераспределенном развертывании вся функциональность и слои приложения, кроме функциональности хранения данных, располагаются на одном сервере, как показано на рис. 1.

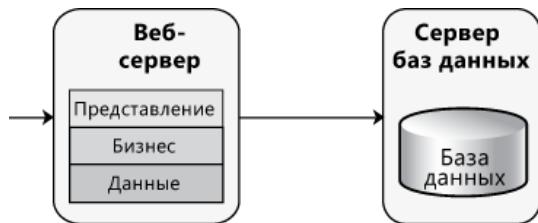


Рис. 14

Нераспределенное развертывание

Преимуществом данного подхода является простота и минимальные требования по количеству необходимых физических серверов. Также обеспечивается наилучшая производительность, поскольку взаимодействие между слоями осуществляется без пересечения физических границ между серверами или кластерами серверов. Но нельзя забывать, что несмотря на сокращение издержек на взаимодействие, использование одного сервера создает другие угрозы производительности. Прежде всего, совместное использование ресурсов всеми слоями приложения. Если один из слоев начинает слишком активно потреблять ресурсы, это негативно сказывается на работе всех остальных слоев. Кроме того, используемые серверы должны быть одинаково конфигурированы и спроектированы с самым строгим соблюдением эксплуатационных требований и должны поддерживать пиковые нагрузки, возникающие при использовании ресурсов системы максимальным числом пользователей. Использование одноуровневой архитектуры снижает общую масштабируемость и удобство обслуживания приложения, поскольку все слои физически располагаются на одном оборудовании.

Распределенное развертывание

При распределенном развертывании все слои приложения располагаются на разных физических уровнях. При многоуровневом развертывании инфраструктура системы организована как набор физических уровней для обеспечения серверных сред, оптимизированных соответственно определенным эксплуатационным требованиям и требованиям по использованию системных ресурсов. Такая модель позволяет распределять слои приложения по разным физическим уровням, как показано в примере на рис. 2.

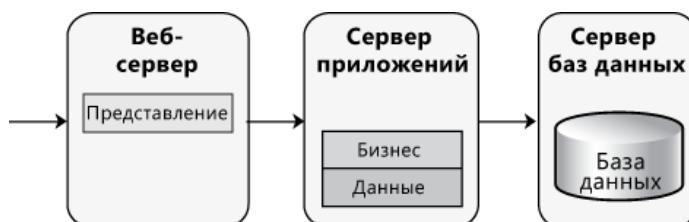


Рис. 15

Распределенное развертывание

Распределенный подход позволяет конфигурировать серверы приложений для размещения на них различных слоев с целью обеспечения максимального соответствия требованиям каждого слоя. Тем не менее, поскольку основным фактором оптимизации развертывания компонентов является сопоставление профиля потребления ресурсов компонента с определенным сервером, прямое проектирование слоев в уровнях часто не является наилучшей стратегией развертывания.

Многоуровневость обеспечивает возможность использования множества инфраструктур. Каждая инфраструктура оптимизируется под определенный набор эксплуатационных требований и требований по использованию системных ресурсов. После этого компоненты могут быть развернуты на уровне, наиболее точно соответствующем требованиям по ресурсам, что обеспечит наилучшие производительность и поведение. Чем больше уровней, тем больше вариантов развертывания имеется для каждого компонента. Распределенное развертывание обеспечивает более гибкую среду, в которой в случае необходимости намного проще реализовать горизонтальное или вертикальное масштабирование. Однако нельзя забывать, что введение большего количества уровней ведет к повышению сложности, увеличению необходимого объема работ по развертыванию и затрат.

Другое основание для добавления уровней – применение особых политик безопасности. Распределенное развертывание позволяет обеспечить более высокую защиту серверам приложений, например, через введение межсетевого экрана между Веб-сервером и серверами приложений и использование разных вариантов аутентификации и авторизации.

Вопросы производительности и проектирования в распределенных средах

Распределение компонентов по уровням может привести к снижению производительности из-за издержек на удаленные вызовы через физические границы. Однако распределение компонентов может улучшить возможности масштабирования, удобство обслуживания, приводя к снижению затрат в долгосрочной перспективе. При проектировании приложения, которое будет выполняться в физически распределенной инфраструктуре, руководствуйтесь следующими рекомендациями:

- При выборе каналов и протоколов связи между уровнями обеспечивайте безопасное взаимодействие между компонентами с минимальным негативным воздействием на производительность. Используйте преимущества асинхронных вызовов, односторонних вызовов или очереди сообщений для максимального сокращения блокировки ресурсов при взаимодействии через физические границы.
- Используйте такие свойства сервисов и операционной системы, как поддержка распределенных транзакций и аутентификация, которые могут упростить дизайн и повысить возможности взаимодействия.
- Применяйте максимально простые интерфейсы компонентов. Детализированные интерфейсы, требующие множества вызовов для выполнения одной задачи, подходят только, если компоненты физически размещаются на одном компьютере. При распределении компонентов по разным компьютерам лучшая

производительность будет достигнута при использовании интерфейсов, обеспечивающих выполнение задачи за один вызов (обобщенные интерфейсы). Тем не менее, если требуется поддерживать как внутрипроцессные вызовы, так и вызовы с других физических уровней, предусмотрите детализированный интерфейс для первого типа вызовов и фасад, который обеспечит обертывание вызовов с других уровней, создавая эффект обобщенного интерфейса для них.

- С помощью специального физического кластера отделите критически важные процессы, выполняющиеся длительное время, от других процессов, которые могут дать сбой, и выработайте стратегию обработки отказов. Например, обычно Веб-серверы предоставляют достаточный объем памяти и вычислительные мощности, но не имеют надежного механизма хранения (такого как зеркальное отображение RAID), который может быть оперативно заменен в случае сбоя оборудования.
- Спланируйте оптимальную стратегию введения дополнительных серверов или ресурсов, что обеспечит повышение производительности и доступности.
- При взаимодействии через физические границы необходимо продумать, как будет реализовано сохранение состояния между уровнями, поскольку это влияет на масштабируемость и производительность. Возможны следующие варианты:
 - **Без сохранения состояния.** Все необходимые данные о состоянии будут предоставлены при обращении к уровню. Такая стратегия обеспечивает лучшую масштабируемость, но часто требует от клиента предоставления данных о состоянии.
 - **С сохранением состояния.** Состояние сохраняется или восстанавливается для каждого клиентского запроса. Для этого требуется больше ресурсов и, следовательно, это негативно сказывается на масштабируемости, но зачастую такой вариант более удобен, поскольку не требует от клиента отслеживания и предоставления данных о состоянии.

Рекомендации по размещению компонентов при распределенном развертывании

При проектировании распределенного развертывания необходимо, прежде всего, распределить имеющиеся логические слои и компоненты по физическим уровням. В большинстве случаев слой представления размещается на клиенте или на Веб-сервере; слой сервисов, бизнес-слой и слой доступа к данным – на сервере приложений; и база данных – на собственном сервере. Но такая схема не является обязательной. При принятии решения о размещении компонентов в распределенной среде руководствуйтесь следующими рекомендациями:

- Распределяйте компоненты только в случае необходимости. Основаниями для реализации распределенного развертывания являются политики безопасности, физические ограничения, совместное использование бизнес-логики и масштабируемость.

- Если бизнес-компоненты используются компонентами представления синхронно, развертывайте их на одном уровне, чтобы обеспечить максимальную производительность и упростить операционное управление.
- Не размещайте компоненты представления и бизнес-компоненты на одном уровне, если аспекты безопасности требуют установления границы доверия между ними. Например, компоненты представления и бизнес-компоненты в насыщенном клиентском приложении можно разделить, разместив компоненты представления на клиенте и бизнес-компоненты на сервере.
- Если аспекты безопасности не требуют установления границы доверия между компонентами агентов сервиса и использующим их кодом, размещайте их на одном уровне.
- По возможности развертывайте вызываемые асинхронно бизнес-компоненты и компоненты рабочего процесса на другом физическом уровне, отдельно от всех остальных слоев приложения.
- Размещайте бизнес-сущности на одном уровне с компонентами, их использующими.

Шаблоны распределенного развертывания

Схемы развертывания, используемые в большинстве решений, могут быть описаны несколькими общими шаблонами. При выборе наилучшего решения развертывания для приложения полезно сначала обозначить общие шаблоны. Полностью разобравшись и поняв разные схемы, можно выбрать наиболее походящую из них, учитывая конкретные сценарии, требования и ограничения безопасности.

Развертывание клиент-сервер

Этот шаблон представляет базовую структуру с двумя основными компонентами: клиент и сервер. При таком сценарии клиент и сервер обычно размещаются на двух разных уровнях. На рис. 3 представлен типовой сценарий для Веб-приложения, в котором клиент взаимодействует с Веб-сервером.

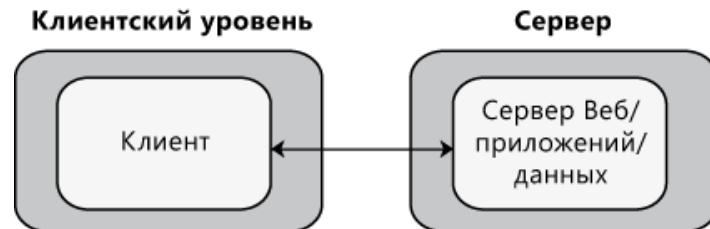


Рис. 16

Типовой сценарий реализации Веб-приложения

Шаблон клиент/сервер рекомендуется применять при разработке клиента, который будет осуществлять доступ к серверу приложений, или при разработке автономного клиента, который будет взаимодействовать с отдельным сервером базы данных.

n-уровневое развертывание

n-уровневый шаблон представляет общую схему развертывания, при которой компоненты приложения развернуты на одном или более серверах. Чаще всего используются 2-уровневый, 3-уровневый или 4-уровневый шаблон, как описывается в следующих разделах. Обычно все компоненты слоя размещаются на одном уровне, но это не является обязательным правилом. Нет требования по абсолютному совпадению слоев и уровней, рабочая нагрузка может быть распределена между несколькими серверами в случае необходимости. Например, разные аспекты бизнес-логики могут располагаться на разных уровнях.

2-уровневое развертывание

По сути, физически это та же компоновка, что и шаблон клиента/сервер. Основное отличие в том, как происходит взаимодействие компонентов уровней. В некоторых случаях, как показано на рис. 4, весь код приложения размещается на клиенте, и база данных выносится на отдельный сервер. Клиент использует хранимые процедуры или минимальную функциональность доступа к данным сервера базы данных.

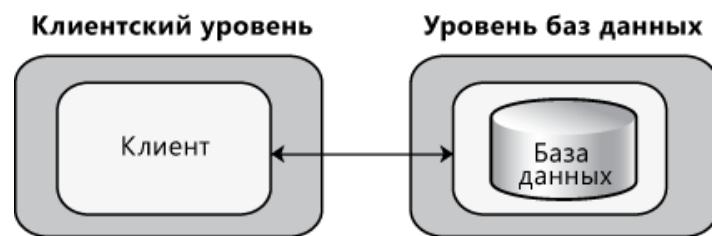


Рис. 17

2-уровневое развертывание с размещением всего кода приложения на клиенте

Используйте 2-уровневый шаблон при разработке клиента, который будет осуществлять доступ к серверу приложений, или при разработке автономного клиента, который будет взаимодействовать с отдельным сервером базы данных.

3-уровневое развертывание

В 3-уровневом дизайне клиент взаимодействует с кодом приложения, развернутым на другом сервере, и сервер приложений взаимодействует с базой данных, размещенной на отдельном сервере, как показано на рис. 5. Этот шаблон применяется для большинства Веб-приложений и Веб-сервисов и подходит для многих общих сценариев. Между клиентом и уровнем Веб/приложений, уровнем Веб/приложений и уровнем базы данных могут устанавливаться межсетевые экраны.



Рис. 18

3-уровневое развертывание с вынесением кода приложения на отдельный уровень

Используйте 3-уровневый шаблон для Интранет-приложения, в котором все серверы размещаются в частной сети, или для Интернет-приложения, для которого требования по обеспечению безопасности не запрещают реализацию бизнес-логики на внешнем Веб-сервере или сервере приложений.

4-уровневое развертывание

В данном сценарии (рис. 6) Веб-сервер и сервер приложений физически разделены. Часто это делают из соображений безопасности, когда Веб-сервер развертывается в пограничной сети и доступен для сервера приложений, размещенного в другой подсети. При таком сценарии между клиентом и Веб-уровнем, а также между Веб-уровнем и уровнем приложений или бизнес-логики могут быть реализованы межсетевые экраны.

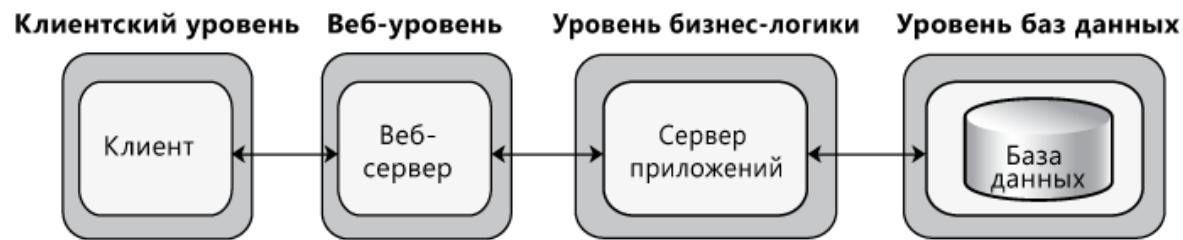


Рис. 19

4-уровневое развертывание, при котором код Веб-приложения и бизнес-логика размещаются на разных уровнях

Используйте 4-уровневый шаблон, если по требованиям безопасности бизнес-логика не может размещаться в пограничной сети, или если имеется код приложения, активно использующий ресурсы сервера, и эту функциональность требуется перенести на другой сервер.

Развертывание веб-приложения

Используйте распределенное развертывание для Веб-приложений, если из соображений безопасности бизнес-логика не может быть развернута на внешнем Веб-сервере.

Взаимодействие посредством обмена сообщений и TCP-протокол с бинарным кодированием обеспечит лучшую производительность для бизнес-слоя. Также необходимо продумать балансировку нагрузки для распределения запросов, так чтобы они обрабатывались разными Веб-серверами, это поможет избежать привязки к конкретному серверу при проектировании масштабируемых Веб-приложений. Используйте в Веб-приложении компоненты без сохранения состояния. Более подробно эти вопросы рассматриваются в разделе «[Шаблоны развертывания для обеспечения наилучшей производительности](#)» далее в этой главе.

Развертывание насыщенного Интернет-приложения

Распределенная архитектура – наиболее подходящий сценарий развертывания в реализациях насыщенных Интернет-приложений (RIA), поскольку позволяет переносить логику представления на клиент. Если бизнес-логика приложения совместно используется другими приложениями, используйте распределенное развертывание. Кроме того, предусмотрите для бизнес-логики интерфейс взаимодействия на основе сообщений.

Развертывание насыщенного клиентского приложения

При n -уровневом развертывании на клиенте могут располагаться логика представления и бизнес-логика или только логика представления. Рис. 7 иллюстрирует вариант, когда на клиенте размещаются логика представления и бизнес-логика.

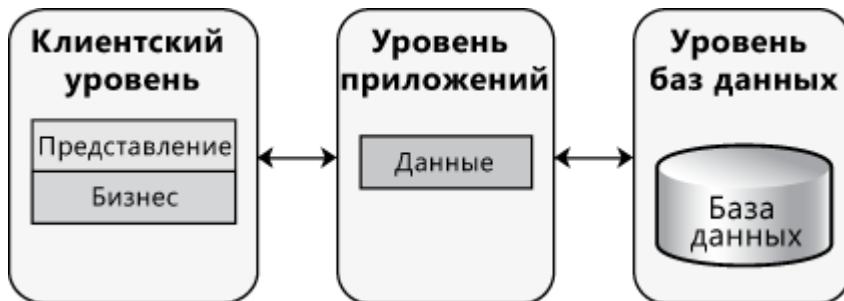


Рис. 20

Насыщенный клиент, бизнес-слой которого размещается на клиентском уровне

На рис. 8 проиллюстрирован вариант размещения бизнес-логики и логики доступа к данным на сервере приложений.

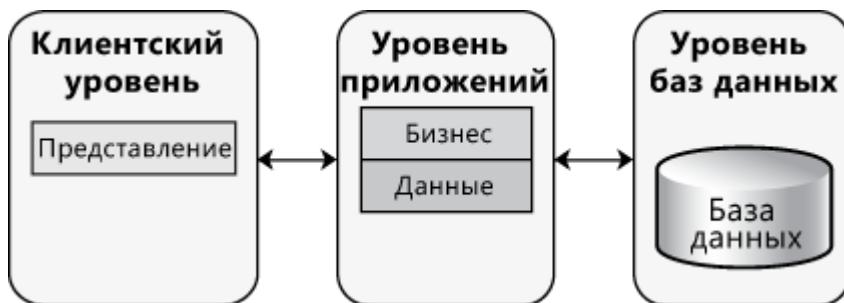


Рис. 21

Насыщенный клиент, бизнес-слой которого размещается на уровне приложений

Шаблоны развертывания для обеспечения наилучшей производительности

Шаблоны развертывания для обеспечения наилучшей производительности представляют проверенные проектные решения типовых проблем, связанных с производительностью. Если требуется обеспечить высокую производительность при развертывании, можно использовать вертикальное или горизонтальное масштабирование. Вертикальное масштабирование подразумевает улучшение оборудования, на котором уже выполняется приложение. Горизонтальное масштабирование подразумевает размещение приложения на множестве физических серверов для распределения нагрузки. По сути, горизонтальное масштабирование – это реализация стратегии балансировки нагрузки. В этой связи часто применяются такие понятия как кластер с балансировкой нагрузки или, для Веб-серверов, Веб-ферма. Эти шаблоны рассматриваются в следующих разделах. Более подробно вопросы выбора горизонтального или вертикального масштабирования обсуждаются в разделе «[Вертикальное и горизонтальное масштабирование](#)» далее в этой главе.

Кластер с балансировкой нагрузки

Сервис или приложение могут быть установлены на нескольких серверах, конфигурированных для разделения рабочей нагрузки, как показано на рис. 9. Такой тип конфигурации называется кластер с балансировкой нагрузки.

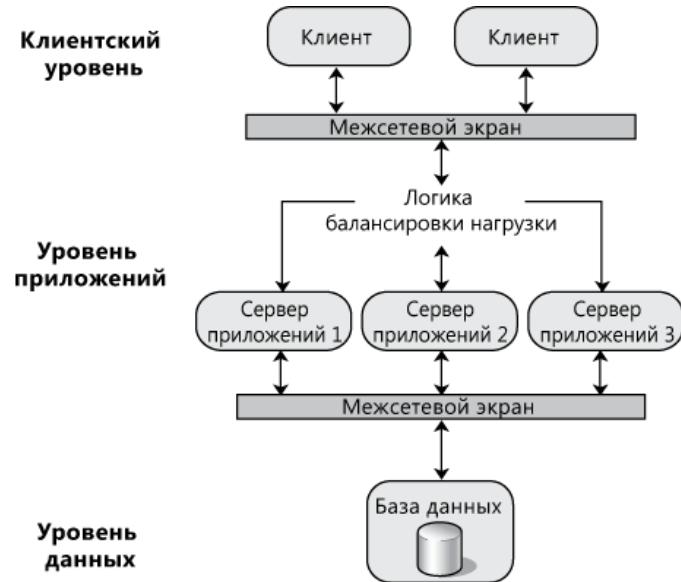


Рис. 9

Кластер с балансировкой нагрузки

Балансировка нагрузки позволяет масштабировать производительность серверных программ, таких как Веб-сервер, через распределение запросов клиента между множеством серверов. Технологии балансировки нагрузки, которые обычно называют подсистемами балансировки нагрузки, принимают поступающие запросы и в случае необходимости перенаправляют их на определенный хост. Хосты с балансировкой нагрузки одновременно отвечают на запросы разных клиентов, даже на множество запросов одного клиента. Например, Веб-браузер может получать разные изображения для одной Веб-страницы с разных хостов кластера. Это позволяет распределять нагрузку, ускоряет обработку и сокращает время ответа.

Технологии маршрутизации позволяют выявлять вышедшие из строя серверы и удалять их из списка маршрутизации, чтобы максимально сократить последствия сбоя. В простых сценариях маршрутизация может быть реализована на основе механизмов циклического обслуживания, когда DNS-сервер последовательно перебирает все адреса имеющихся серверов. Рис. 10 иллюстрирует простую Веб-ферму (кластер Веб-серверов с балансировкой нагрузки), в которой на каждом сервере размещены все уровни приложения, кроме хранилища данных.



Рис. 10

Простая Веб-ферма

Кластеры с балансировкой нагрузки обеспечивают большую масштабируемость и эффективность, если им не приходится отслеживать и сохранять данные между каждым клиентским запросом, иначе говоря, если они реализованы без сохранения состояния. Если требуется отслеживать состояние, вероятно, лучше использовать методики привязки клиента к конкретному серверу и сеансы.

Привязка к конкретному серверу и сеансы пользователей

Приложения могут полагаться на сохранение состояния сеанса между запросами от одного клиента. Веб-серверу, например, может понадобиться отслеживать пользовательские данные между запросами. Веб-ферма может быть конфигурирована так, что все запросы от одного пользователя будут направляться на один и тот же сервер – это называется привязкой к конкретному серверу (*affinity*) – для сохранения состояния, когда эти данные сохраняются в памяти на Веб-сервере. Однако для повышения доступности и надежности на Веб-ферме следует использовать отдельное хранилище для данных о состоянии, это устранит требование привязки к конкретному серверу. Internet Information Services (IIS) 6.0 и последующие его версии можно настроить на работу в режиме Веб-сад, это обеспечит корректную обработку данных о состоянии сеанса в приложении во время его разработки.

В ASP.NET там, где не реализовывается привязка к конкретному серверу, все Веб-серверы должны быть настроены на использование одного ключа и метода шифрования для шифрования ViewState. Там, где система поддерживает эту возможность, для сеансов, использующих шифрование по протоколу Secure Sockets Layer (SSL), должна быть включена привязка к конкретному серверу, либо для SSL-запросов должен применяться отдельный кластер.

Фермы приложений

Бизнес-слой и слой доступа к данным, если они располагаются отдельно от уровня представления, на другом физическом уровне, могут масштабироваться аналогично Веб-серверам и Веб-фермам. Для этого используется ферма приложений. Запросы с уровня представления распределяются между серверами фермы так, чтобы каждый из них был загружен примерно одинаково. Компоненты бизнес-уровня и компоненты уровня доступа к данным можно распределить по разным фермам приложений в зависимости от требований каждого уровня, а также предполагаемой нагрузки и количества пользователей.

Шаблоны развертывания для обеспечения надежности

Шаблоны развертывания для обеспечения надежности представляют проверенные решения типовых проблем, связанных с надежностью. Наиболее распространенным подходом для повышения надежности развертывания является использование отказоустойчивого кластера, который гарантирует доступность приложения даже в случае сбоя одного из серверов.

Отказоустойчивый кластер

Отказоустойчивый кластер – это набор серверов, конфигурированных таким образом, что в случае отказа одного из них, другие серверы принимают на себя его нагрузку и продолжают обработку. На рис. 11 представлен отказоустойчивый кластер.

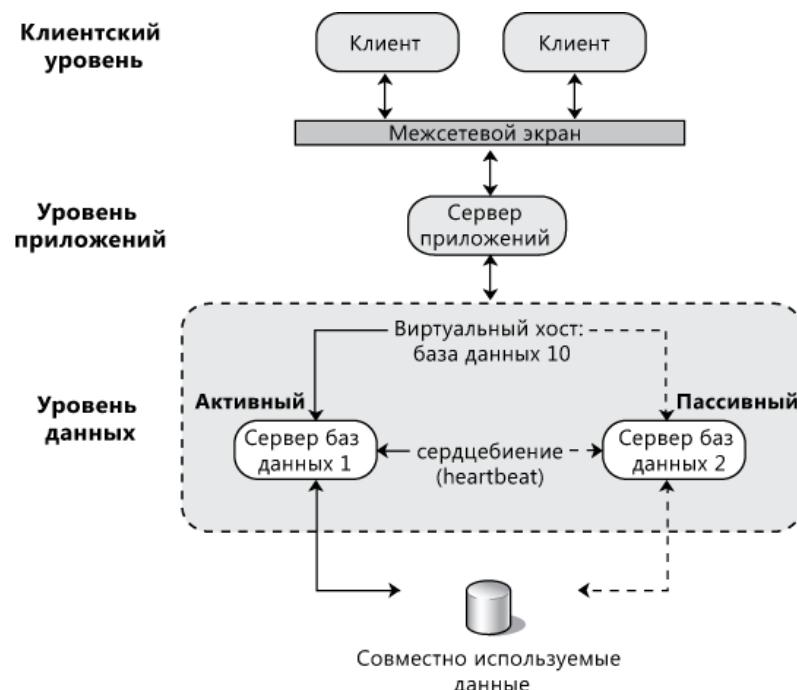


Рис. 22

Отказоустойчивый кластер

Установите приложение или сервис на множество серверов, конфигурированных так, чтобы нагрузка вышедшего из строя сервера перераспределялась между оставшимися. Процесс перевода нагрузки на другой сервер в случае сбоя сервера называют обработкой отказа (failover). Каждый сервер в кластере имеет, по крайней мере, один резервный сервер.

Шаблоны обеспечения безопасности

Шаблоны обеспечения безопасности представляют проверенные решения типовых проблем, связанных с безопасностью. Хорошим подходом для передачи контекста исходной вызывающей стороны на нижние уровни или компоненты приложения является олицетворение/делегирование. Подход с применением доверенной подсистемы подойдет, если требуется реализовывать аутентификацию и авторизацию в вышестоящих компонентах и выполнять доступ к нижестоящему ресурсу с использованием единственного доверенного удостоверения.

Олицетворение/делегирование

В модели олицетворения/делегирования доступ к ресурсам и типам операций (таким как чтение, запись и удаление) контролируется посредством списков управления доступом Windows (Windows Access Control Lists, ACLs) или эквивалентных средств безопасности целевых ресурсов (таких как таблицы и процедуры SQL Server). Доступ к ресурсам пользователями осуществляется с применением их исходных удостоверений через олицетворение, как показано на рис. 12. Нельзя забывать, что в случае применения этого подхода может возникнуть требование о наличии доменной учетной записи, что делает его непривлекательным для некоторых сценариев.

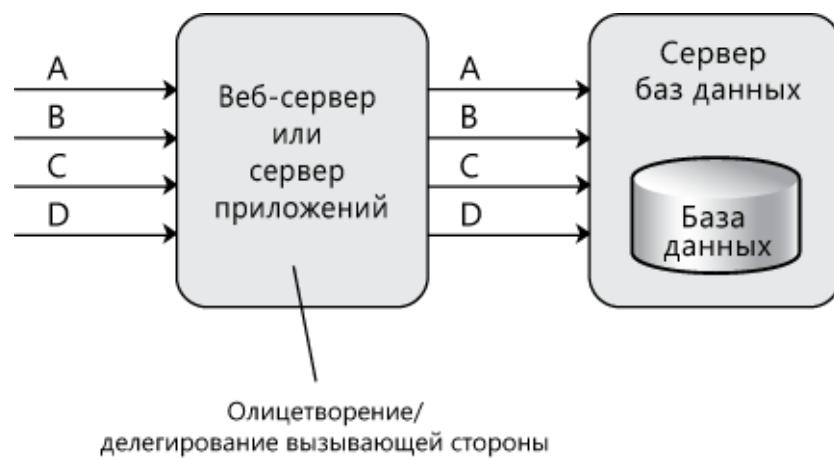


Рис. 23

Модель авторизации с олицетворением/делегированием

Доверенная подсистема

В модели доверенная подсистема (или доверенный сервер) пользователи логически сгруппированы соответственно описанным приложением ролям. Участники определенной роли обладают одинаковыми привилегиями в приложении. Авторизация доступа к операциям (обычно осуществляющегося в виде вызовов методов) происходит на основании принадлежности вызывающей стороны к той или иной роли. При таком основанном на ролях (или операциях) подходе к обеспечению безопасности авторизация доступа к операциям (не сетевым ресурсам) выполняется на основании ролевой принадлежности вызывающей стороны. Роли, определяемые на этапе проектирования приложения, используются как логические контейнеры для группировки пользователей с одинаковыми привилегиями или возможностями в приложении. Сервис промежуточного уровня использует фиксированное удостоверение для доступа к нижестоящим сервисам и ресурсам, как показано на рис. 13.

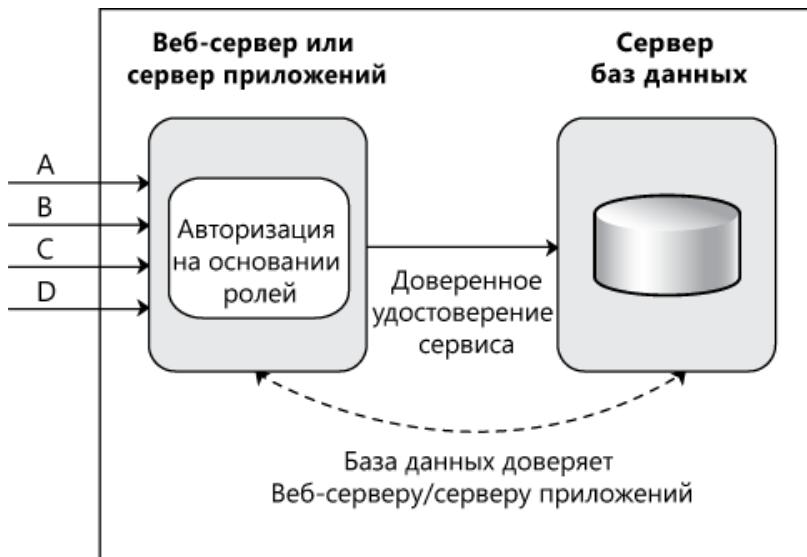


Рис. 24

Модель доверенная подсистема (или доверенный сервер)

Модель с применением множества доверенных удостоверений сервиса

В некоторых ситуациях может потребоваться более одного доверенного удостоверения, например, при наличии двух групп пользователей, одна из которых должна быть авторизована на осуществление операций чтения/записи, а другая – только операций чтения. Использование множества доверенных удостоверений сервиса обеспечивает возможность более детального контроля доступа к ресурсам и аудита без особого влияния на масштабируемость. На рис. 14 показана модель с применением множества доверенных удостоверений сервиса.

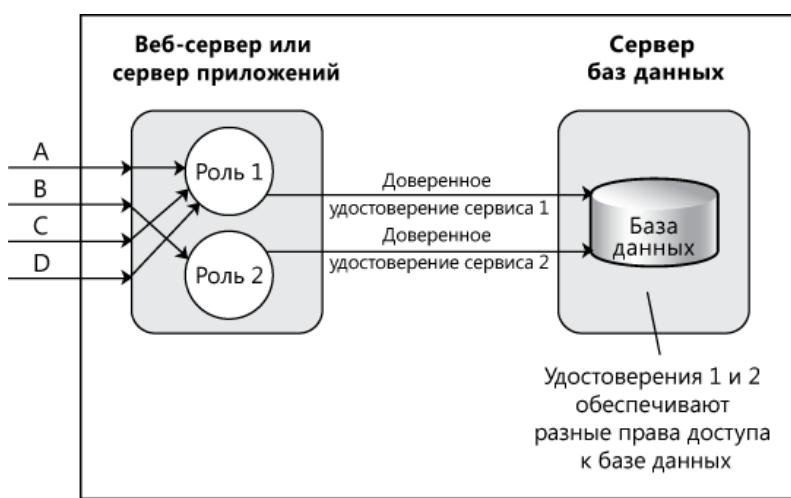


Рис. 25

Модель с применением множества доверенных удостоверений сервиса

Вертикальное и горизонтальное масштабирование

Подход к реализации масштабирования является критически важным аспектом проектирования. Независимо от того, планируется ли выполнять горизонтальное

масштабирование решения с помощью кластера с балансированной нагрузки или секционированной базы данных, дизайн должен обеспечивать поддержку выбранной опции. Существует два основных типа масштабирования: **вертикальное** (большой блок) и **горизонтальное** (больше блоков).

При **вертикальном** масштабировании поддержка повышенной нагрузки обеспечивается через введение в существующие серверы дополнительного оборудования, такого как процессоры, оперативная память и сетевые интерфейсные платы (network interface cards, NIC). Такой простой вариант не добавляет затрат на обслуживание и поддержку, но может быть экономически выгодным лишь до определенного момента. Однако всегда сохраняется вероятность сбоя, что является риском. Кроме того, введение дополнительного оборудования в существующие серверы обеспечивает желаемые результаты не бесконечно, и получение последних 10% расчетной производительности путем наращивания мощностей одного компьютера может быть очень дорогим удовольствием.

Эффективного вертикального масштабирования приложения можно добиться лишь при условии соответствующего вертикального масштабирования базовой инфраструктуры, среди выполнения и архитектуры компьютера. Продумайте, какие ресурсы ограничивают производительность приложения. Например, если это связано с нехваткой памяти или низкой пропускной способностью сети, добавление процессоров ничего не даст.

При **горизонтальном** масштабировании добавляется больше серверов и используются решения с балансировкой нагрузки и кластеризацией. Кроме возможности обработки большей нагрузки, горизонтальное масштабирование смягчает последствия сбоев оборудования. Если один из серверов выходит из строя, другие серверы кластера берут на себя его нагрузку. Например, уровень представления и бизнес-уровень приложения могут размещаться на нескольких Веб-серверах с балансировкой нагрузки, образующих Веб-ферму. Или можно физически отделить бизнес-логику приложения и использовать для нее отдельный средний уровень с балансировкой нагрузки, но при этом размещать уровень представления на внешнем уровне с балансировкой нагрузки. Если приложение имеет ограничения по вводу/выводу и должно поддерживать очень большую базу данных, ее можно распределить по нескольким серверам баз данных. Как правило, способность приложения масштабироваться горизонтально больше зависит от его архитектуры, чем от базовой инфраструктуры.

Вопросы вертикального масштабирования

Вертикальное масштабирование через повышение мощности процессора и увеличение объема памяти может быть экономически эффективным решением. Также при таком подходе не возникает необходимости в дополнительных затратах на управление, как с горизонтальным масштабированием в связи с применением Веб-ферм и кластеризации. Прежде всего, следует рассмотреть варианты вертикального масштабирования и провести тестирование производительности, чтобы убедиться в том, что вертикальное масштабирование решения соответствует заданному критерию масштабирования и обеспечивает приемлемый уровень производительности для требуемого числа одновременно работающих пользователей. Необходимо выработать план масштабирования для системы, который будет отражать перспективы ее роста.

Проектирование с поддержкой горизонтального масштабирования

Если вертикальное масштабирование решения не обеспечивает требуемой масштабируемости из-за достижения предельных показателей для процессора, подсистемы ввода/вывода или памяти, необходимо выполнять горизонтальное масштабирование и вводить дополнительные серверы. Для обеспечения эффективного горизонтального масштабирования приложения при проектировании пользуйтесь следующими практиками:

- **Узкие места идентификации и горизонтального масштабирования.** Часто узким местом являются совместно используемые плохо масштабируемые в вертикальном направлении ресурсы. Например, имеется единственный экземпляр SQL Server, с которым работают множество серверов приложений. В этом случае разделение данных таким образом, чтобы они могли обслуживаться несколькими экземплярами SQL Server, обеспечит возможность горизонтального масштабирования решения. Если существует вероятность того, что узким местом станет сервер базы данных, предусмотрите секционирование данных при проектировании, это избавит от многих проблем в будущем.
- **Слабо связанный и многослойный дизайн.** Слабо связанный многослойный дизайн с четкими интерфейсами, которые могут использоваться удаленно, проще масштабировать горизонтально, чем дизайн, использующий тесно связанные слои с детализированными интерфейсами. Многослойный дизайн будет иметь естественные точки разделения, что делает его идеальным для горизонтального масштабирования в границах уровней. Главное, правильно определить границы. Например, бизнес-логику проще перенести в ферму серверов приложений среднего уровня с балансировкой нагрузки.

Компромиссы и последствия их принятия

Следует учесть аспекты масштабируемости, которые могут быть разными для разных слоев, уровней или типов данных. Выявление необходимых компромиссов позволит увидеть, в каких аспектах имеется гибкость, а в каких нет. В некоторых случаях вертикальное масштабирование с последующим горизонтальным масштабированием с применением Веб-серверов или серверов приложений не является наилучшим подходом. Например, можно установить 8-процессорный сервер, но из соображений экономии, скорее всего, вместо одного большого сервера будут использоваться несколько меньших серверов.

С другой стороны, в определенных ситуациях, в зависимости от роли данных и их использования, вертикальное масштабирование с последующим горизонтальным масштабированием может быть оптимальным подходом для серверов баз данных. Однако возможности балансировки нагрузки и обработки отказов не бесконечны, и количество серверов, которые могут быть охвачены этими процессами, ограничено. Также влияние оказывают и другие аспекты, такие как секционирование базы данных. Кроме технических вопросов и вопросов производительности, нельзя забывать об эксплуатации и управлении и об общей стоимости всей системы.

Как правило, выполняется оптимизация цены и производительности в рамках, налагаемых всеми остальными ограничениями. Например, использование четырех 2-процессорных Веб-

серверов/серверов приложений может быть более оптимальным вариантом с точки зрения цены и производительности по сравнению с использованием двух 4-процессорных серверов. Однако должны быть учтены и другие ограничения, такие как какое максимальное число серверов, которые можно разместить в конкретной инфраструктуре балансировки нагрузки, а также энергопотребление или предоставляемая площадь в data-центре.

Для реализации ферм серверов и для размещения сервисов могут использоваться виртуализированные серверы. Такой подход поможет найти оптимальное соотношение производительности и стоимости, обеспечивая при этом максимальное использование ресурсов и рентабельность инвестиций.

Компоненты без сохранения состояния

Применение компонентов без сохранения состояния (не сохраняющие промежуточного состояния компоненты, которые могут быть реализованы в клиентской части Веб-приложения) означает возможность создания дизайна с лучшими возможностями, как для горизонтального, так и для вертикального масштабирования. Для сознания дизайна без сохранения состояния придется пойти на многие компромиссы, но обеспечиваемые им преимущества с точки зрения масштабируемости, как правило, перевешивают все возможные недостатки.

Секционирование данных и базы данных

Если приложение работает с очень большой базой данных и есть опасения, что операции ввода/вывода станут узким местом системы, заранее предусмотрите секционирование базы данных. Секционирование базы данных на более поздних этапах проектирования обычно требует полной переработки дизайна базы данных и, соответственно, масштабных изменений всего кода приложения. Секционирование обеспечивает несколько преимуществ, включая возможность направления всех запросов к одной секции (таким образом, использование ресурсов ограничивается только одной частью данных) и возможность задействовать множество секций (таким образом, достигаются лучшие возможности одновременной работы и исключительная производительности за счет извлечения данных с множества дисков).

Однако в некоторых ситуациях наличие множества секций может иметь негативные последствия. Например, некоторые операции эффективнее выполнять с данными, сконцентрированными на одном накопителе.

Принимаемые в сценариях развертывания решения о секционировании хранилища данных во многом определяются типом данных. Рассмотрим значимые факторы:

- **Статические справочные данные только для чтения.** Для этого типа данных в целях улучшения производительности и масштабируемости можно без особого труда поддерживать множество копий на разных накопителях, размещаемых в соответствующих местоположениях. Это имеет минимальное влияние на дизайн и обычно определяется соображениями оптимизации. Сведение нескольких логически отдельных и независимых баз данных на один сервер базы данных, даже если это позволяет объем дискового пространства, может быть неудачным решением, и размещение копий ближе к потребителям данных может оказаться в равной степени приемлемым подходом. Однако нельзя забывать, что любое

тиражирование требует применения механизмов обеспечения синхронизации системы.

- **Динамические (часто изменяющиеся) легко секционируемые данные.** Это данные, относящиеся к конкретному пользователю или сеансу, такие как корзина в системе электронной коммерции, где данные пользователя А никак не связаны с данными пользователя В. Управлять такими данными немного сложнее, чем статическими данными только для чтения, но их довольно легко оптимизировать и распределить, поскольку они могут быть секционированы. Нет никаких зависимостей между группами вплоть до отдельных пользователей. Важная особенность этих данных в том, что здесь не выполняется запрос по всем секциям: запрашивается содержимое корзины пользователя А, но не все корзины, включающие определенный товар. Обратите внимание, если последующие запросы могут поступать на другой Веб-сервер или сервер приложений, все эти серверы должны иметь возможность доступа к соответствующей секции.
- **Основные данные.** Это основной случай применения вертикального масштабирования с последующим горизонтальным масштабированием. Как правило, тиражировать данные этого типа нежелательно из-за сложности их синхронизации. Классическое решение для таких данных – вертикальное масштабирование до предельных возможностей (в идеале, сохранение единственного логического экземпляра с соответствующей кластеризацией) и применение секционирования и распределения, только если горизонтальное масштабирование является единственным допустимым вариантом. Прогресс и достижения в технологиях баз данных, такие как распределенные секционированные представления, намного упростили секционирование, тем не менее, оно должно применяться лишь в случае крайней необходимости. Слишком большой размер базы данных редко является определяющим фактором при принятии решения, намного чаще основную роль играют другие соображения, такие как кому принадлежат данные, географическое распределение пользователей, близость к потребителю и доступность.
- **Данные с отложенной синхронизацией.** Некоторые используемые в приложениях данные не требуют немедленной синхронизации или синхронизации вообще. Отличный пример – такие данные онлайн-магазинов, как «С товаром X часто покупают Y и Z». Эти данные извлекаются из основных данных, но не требуют обновления в режиме реального времени. Проектирование стратегий, обеспечивающих перевод данных из основных в секционируемые (динамические) и затем в статические, является ключевым фактором в построении высокомасштабируемых приложений.

Более подробно схемы перемещения и тиражирования данных рассматриваются в статье «*Data Movement Patterns*» (Шаблоны передачи данных) по адресу <http://msdn.microsoft.com/en-us/library/ms998449.aspx>.

Вопросы безопасности сетевой инфраструктуры

Убедитесь в том, что понимаете структуру сети, предоставляемую целевой средой, и понимаете базовые требования безопасности сети, такие как правила фильтрации, ограничения по применяемым портам, поддерживаемые протоколы и т.д. Предлагаем рекомендации по повышению безопасности сети:

- Выясните, как влияют на дизайн и развертывание приложения межсетевые экраны и их политики. Межсетевые экраны должны использоваться для отделения приложений с выходом в Интернет от внутренней сети и для защиты серверов баз данных. Межсетевые экраны разрешают взаимодействие только через соответствующим образом конфигурированные порты и, следовательно, могут блокировать некоторые протоколы и предотвращать использование некоторых опций связи. К ним относится аутентификация, например аутентификация Windows, между Веб-сервером и сервером приложений или базы данных, располагающимся за межсетевым экраном.
- Рассмотрите, какие протоколы, порты и сервисы могут выполнять доступ к внутренним ресурсам с Веб-серверов пограничной сети или из насыщенных клиентских приложений. Определите требуемые дизайном приложения протоколы и порты и проанализируйте потенциальные угрозы, которые могут возникнуть в результате открытия дополнительных портов или использования нестандартных протоколов.
- Обменивайтесь информацией с группой разработки и группой обслуживания сети и записывайте любые предположения о безопасности сети и слоя приложения, а также функциях безопасности, которые будет реализовывать каждый компонент. Это позволит учесть все элементы управления безопасности и политики.
- Обратите внимание и убедитесь в наличии таких средств защиты, как межсетевые экраны, фильтры пакетов и аппаратные системы, которые приложение предполагает получить от сети.
- Рассмотрите возможные последствия изменения конфигурации сети и то, как это влияет на безопасность.

Вопросы управляемости

Решения, принимаемые при развертывании приложения, оказывают влияние на возможности управления и мониторинга приложения. Примите во внимание следующие рекомендации:

- Чтобы избежать дублирования, развертывайте компоненты приложения, используемые многими пользователями, централизованно, например, на ферме серверов или приложений, доступной всем приложениям.
- Храните данные так, чтобы они были доступны средствам архивации и восстановления.

- Компоненты, использующие существующее ПО или оборудование (такое как собственная сеть, к которой можно подключиться только с определенного компьютера), должны физически располагаться на том же компьютере.
 - Разворачивание некоторых библиотек и адаптеров невозможно без дополнительных капиталовложений, плата также может взиматься за каждый ЦП, поэтому централизация этих ресурсов позволит сократить затраты.
 - Сервис, компонент или приложение могут принадлежатьциальному подразделению организации и должны управляться локально.
 - Средства мониторинга, такие как System Center Operations Manager, должны иметь доступ к компьютерам для получения сведений по управлению, и это может влиять на варианты развертывания.
-

Шаблоны проектирования

Основные шаблоны организованы по категориям, таким как Разворачивание, Управляемость, Производительность и надежность и Безопасность, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Разворачивание	<p>Layered Application (Многослойное приложение). Архитектурный шаблон, обеспечивающий организацию системы по слоям.</p> <p>Three-Layered Services Application (Трехслойное сервисное приложение). Архитектурный шаблон, в котором слои проектируются так, чтобы обеспечивать максимальную производительность, и при этом предоставляют сервисы, доступные для использования другими приложениями.</p> <p>Tiered Distribution (Многоуровневое распределение). Архитектурный шаблон, в котором слои дизайна могут распределяться через физические границы.</p> <p>Three-Tiered Distribution (Трехуровневое распределение). Архитектурный шаблон, в котором слои дизайна распределяются по трем физическим уровням.</p> <p>Deployment Plan (План развертывания). Описывает процессы сопоставления логических слоев и физических уровней с учетом ограничений, налагаемых инфраструктурой.</p>
Управляемость	<p>Adapter (Адаптер). Объект, поддерживающий общий интерфейс и транслирующий операции между общим интерфейсом и объектами, реализующими аналогичную функциональность, но использующими другие интерфейсы.</p> <p>Provider (Поставщик). Компоненты, которые предоставляют API, отличающийся от клиентского API, чтобы обеспечить возможность подключения любой специальной реализации. Многие приложения, обеспечивающие инструментарий, предоставляют поставщиков, которые могут использоваться для сбора данных о состоянии и работоспособности</p>

	приложения и системы, в которой размещается приложение.
Производительность и надежность	<p>Server Clustering (Кластеризация сервера). Шаблон развертывания, при котором множество серверов конфигурированы так, что могут разделять рабочую нагрузку, и представляются клиенту как один компьютер или ресурс.</p> <p>Load-balanced Cluster (Кластер с балансировкой нагрузки). Шаблон развертывания, при котором множество серверов конфигурированы так, что могут разделять рабочую нагрузку. Балансировка нагрузки обеспечивает и повышение производительности благодаря распределению работы между множеством серверов, и повышение надежности, поскольку при сбое одного из серверов остальные серверы продолжают обработку.</p> <p>Failover Cluster (Отказоустойчивый кластер). Шаблон развертывания, обеспечивающий уровень инфраструктуры с высоким уровнем доступности для защиты от потери обслуживания из-за сбоя отдельного сервера или размещаемого на нем ПО.</p>
Безопасность	<p>Brokered Authentication (Аутентификация через посредника). Аутентификация на посреднике, который предоставляет маркер для аутентификации при доступе к сервисам или системам.</p> <p>Direct Authentication (Прямая аутентификация). Аутентификация непосредственно на сервисе или системе, к которым выполняется доступ.</p> <p>Impersonation and Delegation (Олицетворение и делегирование). Процесс временного присвоения другого удостоверения для обеспечения возможности использования для доступа к ресурсу другого контекста безопасности или набора учетных данных. При этом учетная запись сервиса может выполнять доступ к удаленному ресурсу от лица другого пользователя.</p> <p>Trusted Subsystem (Доверенная подсистема). Для получения доступа к дополнительным ресурсам приложение ведет себя как доверенная подсистема. Для доступа к ресурсу оно использует собственные учетные данные, а не учетные данные пользователя.</p>

Более подробно шаблоны Layered Application, Three-Layered Services Application, Tiered Distribution, Three-Tiered Distribution и Deployment Plan рассматриваются в статье «*Deployment Patterns*» (Шаблоны развертывания) по адресу <http://msdn.microsoft.com/en-us/library/ms998478.aspx>.

Более подробно шаблон Adapter рассматривается в главе 4, «Структурные шаблоны», книги Эрика Гамма, Ричарда Хельмса, Ральфа Джонсона и Джона Влиссидса «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2006.

Более подробно шаблон Provider рассматривается в материале «*Provider Model Design Pattern and Specification, Part 1*» (Шаблон проектирования модели провайдеров и спецификация, Часть 1) по адресу <http://msdn.microsoft.com/en-us/library/ms972319.aspx>.

Более подробно шаблоны Server Clustering, Load-Balanced Cluster и Failover Cluster рассматриваются в статье «*Performance and Reliability Patterns*» (Шаблоны развертывания для обеспечения производительности и надежности) по адресу <http://msdn.microsoft.com/en-us/library/ms998503.aspx>.

Более подробно шаблоны Brokered Authentication, Direct Authentication, Impersonation and Delegation и Trusted Subsystem рассматриваются в статье «*Web Service Security*» (Безопасность Веб-сервисов) по адресу <http://msdn.microsoft.com/en-us/library/aa480545.aspx>.

Дополнительные источники

Электронная версия списка используемых источников, которые могут быть полезны при проектировании стратегии развертывания, доступна по адресу <http://www.microsoft.com/architectureguide>.

- Более подробно методики авторизации рассматриваются в статье «*Designing Application-Managed Authorization*» по адресу <http://msdn.microsoft.com/en-us/library/ms954586.aspx>.
 - Сценарии развертывания и связанные с ними вопросы рассматриваются в статье «*Deploying .NET Framework-based Applications*» (Развертывание .NET-приложений) по адресу <http://msdn.microsoft.com/en-us/library/ms954585.aspx>.
 - Больше сведений о шаблонах проектирования можно найти в статье «*Enterprise Solution Patterns Using Microsoft .NET*» по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.
-

Архетипы приложений

В данном разделе руководства представлены темы, которые помогут понять возможности, свойства, преимущества и недостатки всех доступных общих типов приложений. Рассмотрение начинается с обзора базовых типов приложений, таких как Веб-приложение, мобильное приложение, насыщенный клиент, сервисы и RIA. В остальных главах каждый из типов описывается более подробно. Также обсуждаются специальные типы приложений, такие как размещаемые сервисы и сервисы в облаке, а также приложения SharePoint и Microsoft Office. Раздел включает следующие главы:

- Глава 20 «[Выбор типа приложения](#)»
 - Глава 21 «[Проектирование Веб-приложений](#)»
 - Глава 22 «[Проектирование насыщенных клиентских приложений](#)»
 - Глава 23 «[Проектирование насыщенных Интернет-приложений](#)»
 - Глава 24 «[Проектирование мобильных приложений](#)»
 - Глава 25 «[Проектирование сервисных приложений](#)»
 - Глава 26 «[Проектирование размещаемых сервисов и сервисов в облаке](#)»
 - Глава 27 «[Проектирование офисных бизнес-приложений](#)»
 - Глава 28 «[Проектирование LOB-приложений SharePoint](#)»
-

20

Выбор типа приложения

Обзор

Данная глава поможет разобраться с типами приложений, рассматриваемыми в этом руководстве, расскажет о влиянии дизайна на выбор того или иного типа приложения и необходимых компромиссах. Ознакомившись с представленной здесь информацией, вы сможете правильно определить тип приложения соответственно своему сценарию и требованиям. В главе кратко рассматриваются пять основных архетипов приложений, и приводятся ссылки на другие главы, где эти вопросы рассматриваются более подробно.

Выдвигаемые требования, ограничения технологии и планируемый тип взаимодействия с пользователем будут определять используемый тип приложения. Например, необходимо заранее продумать, будут ли обслуживаемые клиенты иметь доступ к постоянному сетевому соединению, должно ли предоставляться в Веб-браузере насыщенное медиа-содержимое анонимным пользователям или приложение будет использоваться преимущественно небольшим числом пользователей корпоративной внутренней сети.

Далее приводится обзор архетипов приложений, в котором дается описание каждого типа приложения и типовые сценарии их использования. Предлагаемая таблица поможет сделать осознанный выбор типа приложения на основании предлагаемых им преимуществ и недостатков.

Обзор архетипов приложений

Рассмотрим основные базовые типы приложений:

- **Мобильные приложения.** Приложения этого типа могут разрабатываться как тонкий клиент или насыщенное клиентское приложение. Насыщенные клиентские мобильные приложения могут поддерживать сценарии без постоянного подключения или без подключения вообще. Веб-приложения или тонкие клиентские приложения поддерживают только сценарии с подключением. Ограничением при разработке мобильных приложений могут быть устройства, на которых их предполагается выполнять.
- **Насыщенные клиентские приложения.** Приложения этого типа обычно разрабатываются как самодостаточные приложения с графическим пользовательским интерфейсом, который обеспечивает отображение данных с помощью набора

элементов управления. Насыщенные клиентские приложения могут поддерживать сценарии без подключения или без постоянного подключения, если должны выполнять доступ к удаленным данным или функциональности.

- **Насыщенные Интернет-приложения.** Приложения этого типа могут поддерживать множество платформ и браузеров. Насыщенные Интернет-приложения выполняются в изолированной программной среде браузера, которая ограничивает доступ к некоторым возможностям клиента.
- **Сервисные приложения.** Сервисы предоставляют бизнес-функциональность для совместного использования и позволяют клиентам доступ к ней из локальной или удаленной системы. Вызов операций сервиса осуществляется с помощью сообщений, соответствующих XML-схемам и передаваемых по транспортным каналам. Целью данного типа приложений является обеспечение слабой связанности между клиентом и сервером.
- **Веб-приложения.** Приложения этого типа, как правило, поддерживают сценарии с постоянным подключением и различные браузеры, выполняющиеся в разнообразнейших операционных системах и на разных plataформах.

Существует множество других более специализированных типов приложений. Как правило, эти типы являются специализациями или сочетаниями базовых типов, перечисленных в данном списке.

Типы приложений

В следующей таблице перечислены преимущества и недостатки общих архетипов приложений.

Тип приложения	Преимущества	Недостатки
<i>Мобильные приложения</i>	Поддержка портативных устройств. Доступность и простота использования для мобильных пользователей. Поддержка сценариев без подключения и сценариев без постоянного подключения.	Ограниченные возможности ввода и навигации. Ограниченная область отображения экрана.
<i>Насыщенные клиентские приложения</i>	Возможность использования ресурсов клиента. Лучшее время отклика, насыщенная функциональность UI и улучшенное взаимодействие с пользователем. Очень динамичное взаимодействие с коротким временем отклика. Поддержка сценариев без подключения и сценариев без	Сложность развертывания; при этом широкий выбор вариантов установки, таких как ClickOnce, Windows Installer и XCOPY. Сложности обеспечения совместимости версий. Зависимость от платформы.

	постоянного подключения.	
<i>Насыщенные Интернет-приложения (RIA)</i>	<p>Такие же насыщенные возможности пользовательского интерфейса, как и для насыщенных клиентов.</p> <p>Поддержка насыщенных и потоковых мультимедиа и графики.</p> <p>Простота развертывания с возможностями распределения (насыщенными) такими же, как и для Веб-клиентов.</p> <p>Простота обновления и смены версий.</p> <p>Поддержка различных платформ и браузеров.</p>	<p>Больший объем памяти, занимаемый на клиенте, по сравнению с Веб-приложением.</p> <p>Ограниченнное использование ресурсов клиента по сравнению с насыщенным клиентским приложением.</p> <p>Необходимость развертывания на клиенте подходящей среды выполнения.</p>
<i>Сервисные приложения</i>	<p>Слабо связанное взаимодействие между клиентом и сервером.</p> <p>Могут использоваться различными и невзаимосвязанными приложениями.</p> <p>Поддержка для обеспечения возможности взаимодействия.</p>	<p>Отсутствие поддержки UI.</p> <p>Зависимость от возможности сетевого подключения.</p>
<i>Веб-приложения</i>	<p>Широко доступный и основанный на стандартах UI, поддерживаемый на многих платформах.</p> <p>Простота развертывания и внесения изменений.</p>	<p>Необходимость устойчивого сетевого подключения.</p> <p>Сложно обеспечить насыщенный пользовательский интерфейс.</p>

Каждый тип приложения может быть реализован с использованием одной или более технологий. Выбор технологии будет определяться сценариями и ограничениями технологий, а также возможностями и опытом группы разработки.

В следующих разделах каждый из типов приложений рассматривается более подробно:

- [Мобильное приложение](#)
- [Насыщенное клиентское приложение](#)
- [Насыщенное Интернет-приложение](#)
- [Сервис](#)
- [Веб-приложение](#)

В данном руководстве также рассматриваются более специализированные типы приложений:

- Глава 26 «[Проектирование размещаемых сервисов и сервисов в облаке](#)»
- Глава 27 «[Проектирование офисных бизнес-приложений](#)»
- Глава 28 «[Проектирование LOB-приложений SharePoint](#)»

Мобильное приложение

Мобильное приложение, как правило, структурируется как многослойное приложение, включающее слой пользовательского интерфейса (представления), бизнес-слой и слой доступа к данным (рис. 1).

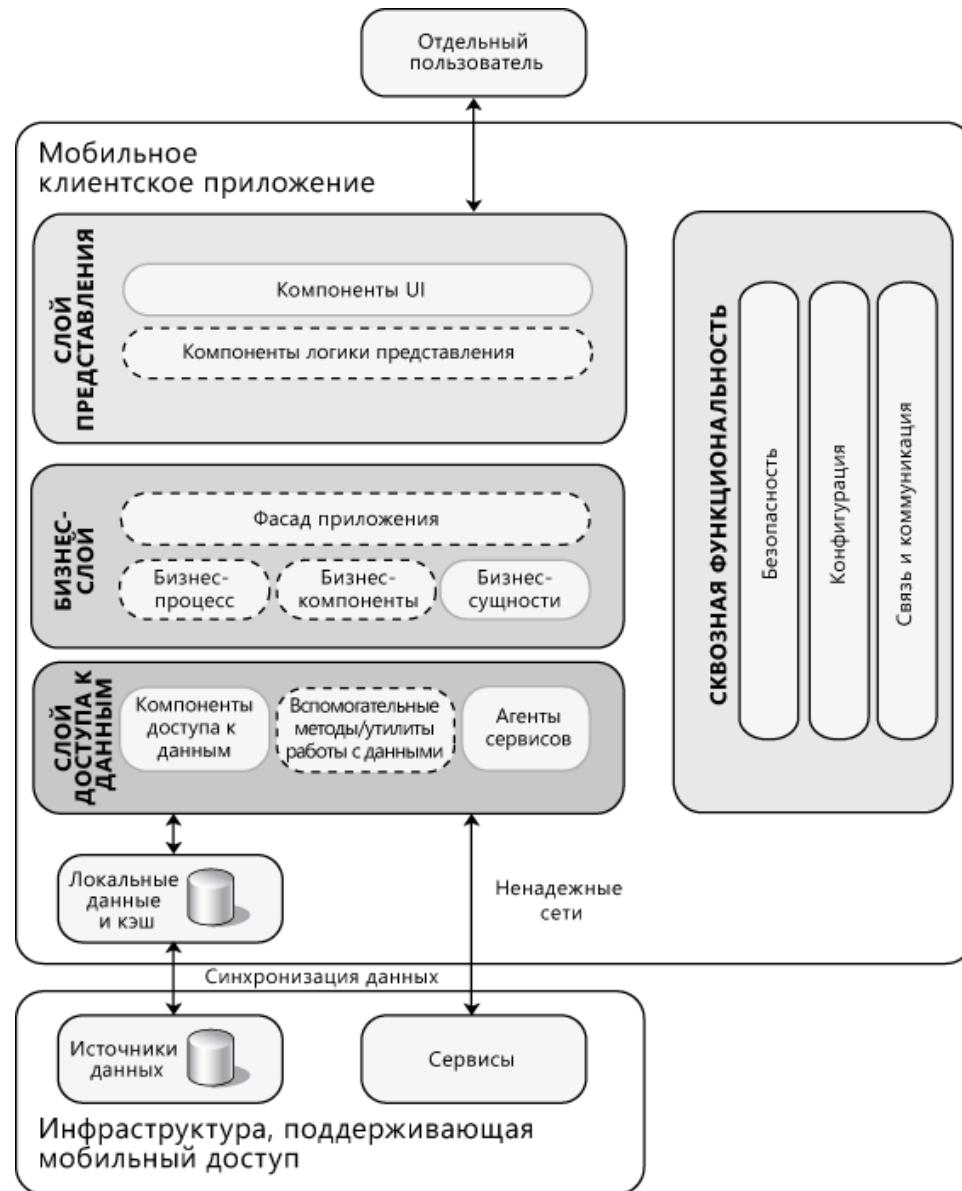


Рис. 26

Типовая структура мобильного приложения

Мобильное приложение может быть тонким Веб-клиентом или насыщенным клиентом. Если создается насыщенный клиент, бизнес-слой и слой доступа к данным, скорее всего, будут располагаться на самом устройстве. Для тонкого клиента бизнес-слой и слой доступа к данным

будут располагаться на сервере. Мобильные приложения обычно реализуют поддержку сценариев без подключения через использование локально кэшированных данных, синхронизация которых выполняется при установлении подключения. Они также могут использовать сервисы, предоставляемые другими приложениями, включая размещаемые сервисы типа S+S (ПО + сервисы) и Веб-сервисы. Часто мобильному клиентскому приложению предоставляется управляемая синхронизация с источником данных и доступ к другим сервисам через специальную серверную инфраструктуру.

Используйте мобильные приложения, если:

- Пользователи зависят от портативных устройств.
- Приложение поддерживает простой UI, подходящий для использования на небольшом экране.
- Приложение должно поддерживать сценарии без подключения или без постоянного подключения. В этом случае более подходящим будет мобильное насыщенное клиентское приложение.
- Приложение должно быть независимым от устройств и может зависеть от возможности сетевого подключения. В этом случае более подходящим будет мобильное Веб-приложение.

Проектированию мобильных приложений посвящена глава 24, «[Проектирование мобильных приложений](#)»

Насыщенное клиентское приложение

Насыщенные клиентские пользовательские интерфейсы могут обеспечить интерактивное, насыщенное взаимодействие с пользователем с минимальным временем отклика для приложений, которые должны выполняться как самодостаточное приложение, в сценариях с подключением, без постоянного подключения и без подключения. Как правило, насыщенное клиентское приложение структурировано как многослойное приложение, включающее слой пользователяского интерфейса (представления), бизнес-слой и слой доступа к данным (рис. 2).

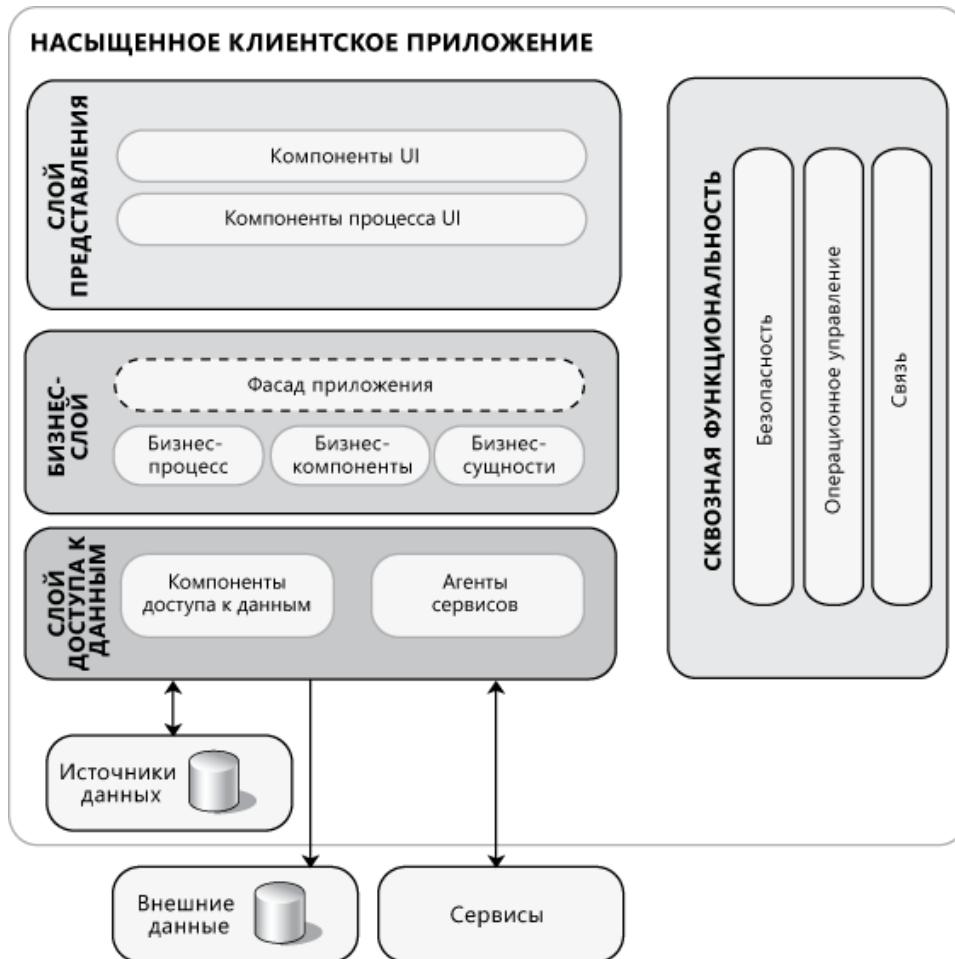


Рис. 27

Типовая структура насыщенного клиентского приложения

Насыщенное клиентское приложение может использовать данные с удаленного сервера, данные, хранящиеся локально, или данные из обоих типов источников. Также оно может потреблять сервисы, предоставляемые другими приложениями, включая размещаемые сервисы типа S+S и Веб-сервисы.

Используйте насыщенные клиентские приложения, если:

- Приложение должно поддерживать сценарии без подключения или без постоянного подключения.
- Приложение будет развертываться на клиентских ПК.
- Приложение должно обеспечивать высокий уровень интерактивности и минимальное время отклика.
- UI приложения должен обеспечивать насыщенную функциональность и взаимодействие с пользователем, но без расширенных графических возможностей или возможностей воспроизведения мультимедиа RIA.
- Приложение должно использовать ресурсы клиентского ПК.

Проектированию насыщенных клиентских приложений посвящена глава 22, «[Проектирование насыщенных клиентских приложений](#)».

Насыщенное Интернет-приложение

Насыщенное Интернет-приложение (RIA) выполняется в браузере в изолированной программной среде. К преимуществам RIA, по сравнению с традиционными Веб-приложениями, относятся более насыщенный пользовательский интерфейс, улучшенное время отклика приложения и эффективность работы с сетью. Как правило, RIA структурировано как многослойное приложение, включающее слой пользовательского интерфейса (представления), бизнес-слой и слой доступа к данным (рис. 3).

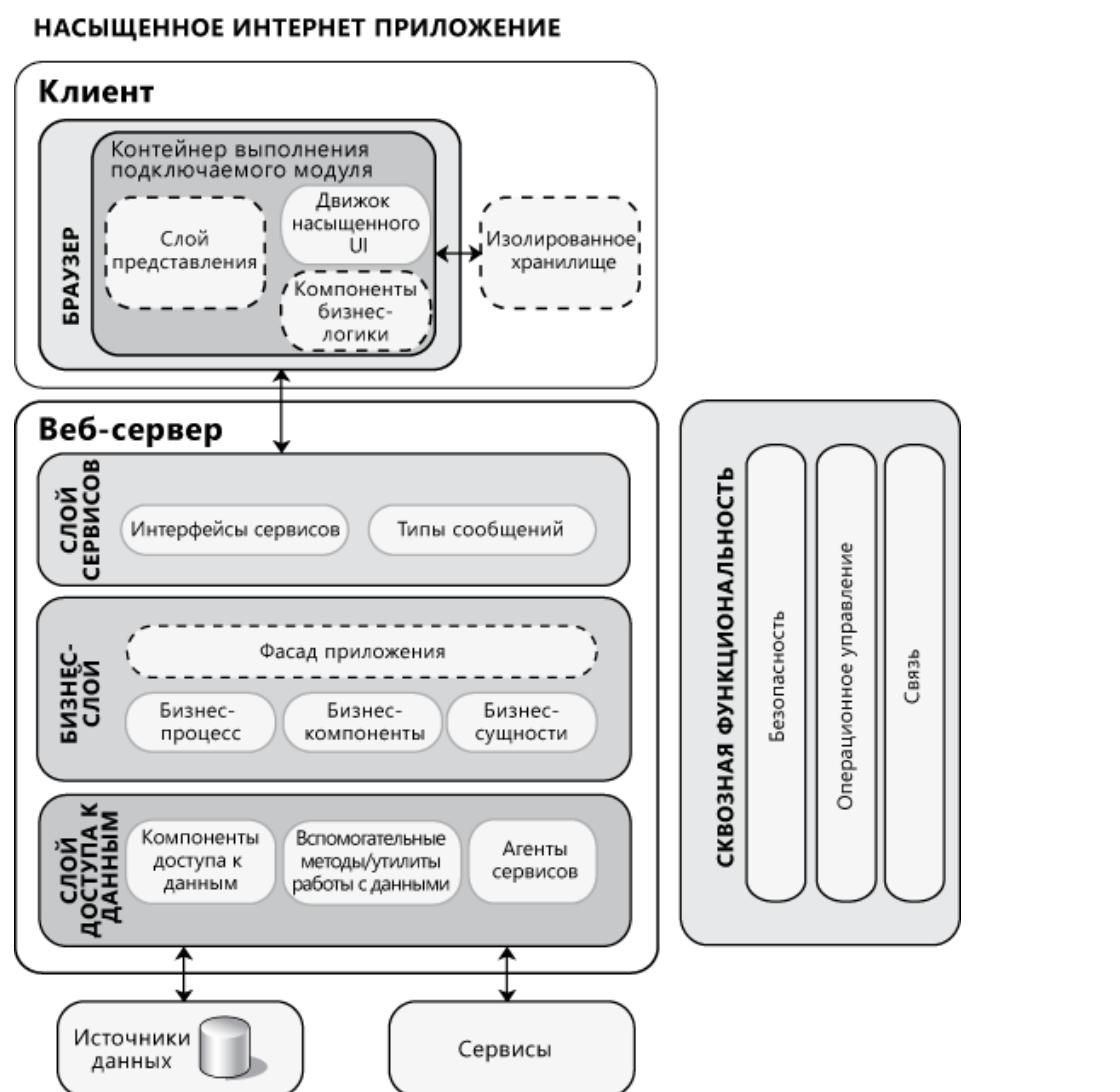


Рис. 28

Типовая структура насыщенного Интернет-приложения

Как правило, RIA-приложения зависят от подключаемого модуля на стороне клиента или размещаемой среды выполнения (такой как среда выполнения XAML или Silverlight).

Подключаемый модуль взаимодействует с удаленными хостами Веб-сервера, которые формируют код и данные, потребляемые клиентским подключаемым модулем или средой выполнения.

Используйте насыщенные Интернет-приложения, если:

- Приложение должно поддерживать насыщенные мультимедиа и обеспечивать представление, насыщенное графическими элементами.
- Приложение должно обеспечивать более насыщенный, интерактивный пользовательский интерфейс с меньшим временем отклика, чем Веб-приложения.
- Использование приложением вычислительных мощностей клиента будет ограничено.
- Использование приложением ресурсов клиента будет ограничено.
- Требуется обеспечить простую модель развертывания в Веб.

Проектированию насыщенных Интернет-приложений посвящена глава 23, «[Проектирование насыщенных Интернет-приложений](#)».

Сервис

В контексте данного руководства *сервис* – это открытый интерфейс, обеспечивающий доступ к единице функциональности. Сервис, фактически, предоставляет программный *сервис* вызывающей стороне, которая потребляет этот сервис. Как правило, сервисное приложение, предоставляющее такие сервисы, структурировано как многослойное приложение, включающее слой сервисов, бизнес-слой и слой доступа к данным (рис. 4).

СЕРВИСЫ

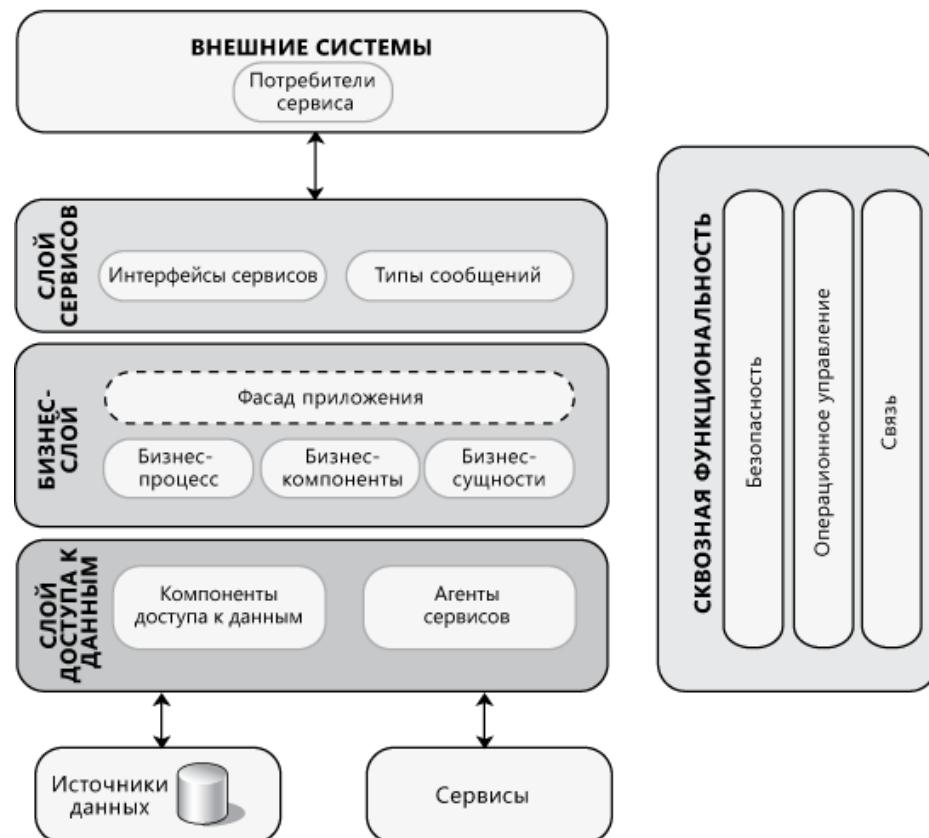


Рис. 29

Типовая структура сервисного приложения

Сервисы слабо связаны и могут сочетаться для обеспечения более сложной функциональности. Сервисы могут быть распределенными, доступ к сервису может осуществляться как удаленно, так и с компьютера, на котором сервис выполняется. Также сервисы ориентируются на обмен сообщениями. Это означает, что интерфейсы описываются WSDL-документом, и операции вызываются с помощью построенных на XML-схемах сообщений, которые передаются по транспортному каналу. Кроме того, благодаря реализации взаимодействия через описание сообщения/интерфейса сервисы поддерживают гетерогенную среду. Если компоненты могут интерпретировать описание сообщения и интерфейса, они могут использовать сервис независимо от собственной базовой технологии.

Используйте сервисные приложения, если:

- Приложение будет предоставлять функциональность, не требующую UI.
- Необходимо обеспечить слабую связь приложения с его клиентами.
- Приложение должно совместно использоваться или потребляться другими внешними приложениями.
- Приложение должно предоставлять функциональность, которая будет потребляться приложениями через Интернет, интранет или на локальном компьютере.

Проектированию сервисов и сервисных приложений посвящена глава 25, «[Проектирование сервисных приложений](#)».

Веб-приложение

Ядро Веб-приложения – его логика на стороне сервера. Эта логика может состоять из множества отдельных слоев. Типовым примером является трехслойная архитектура, включающая слой представления, бизнес-слой и слой доступа к данным (рис. 5).

Веб-приложение

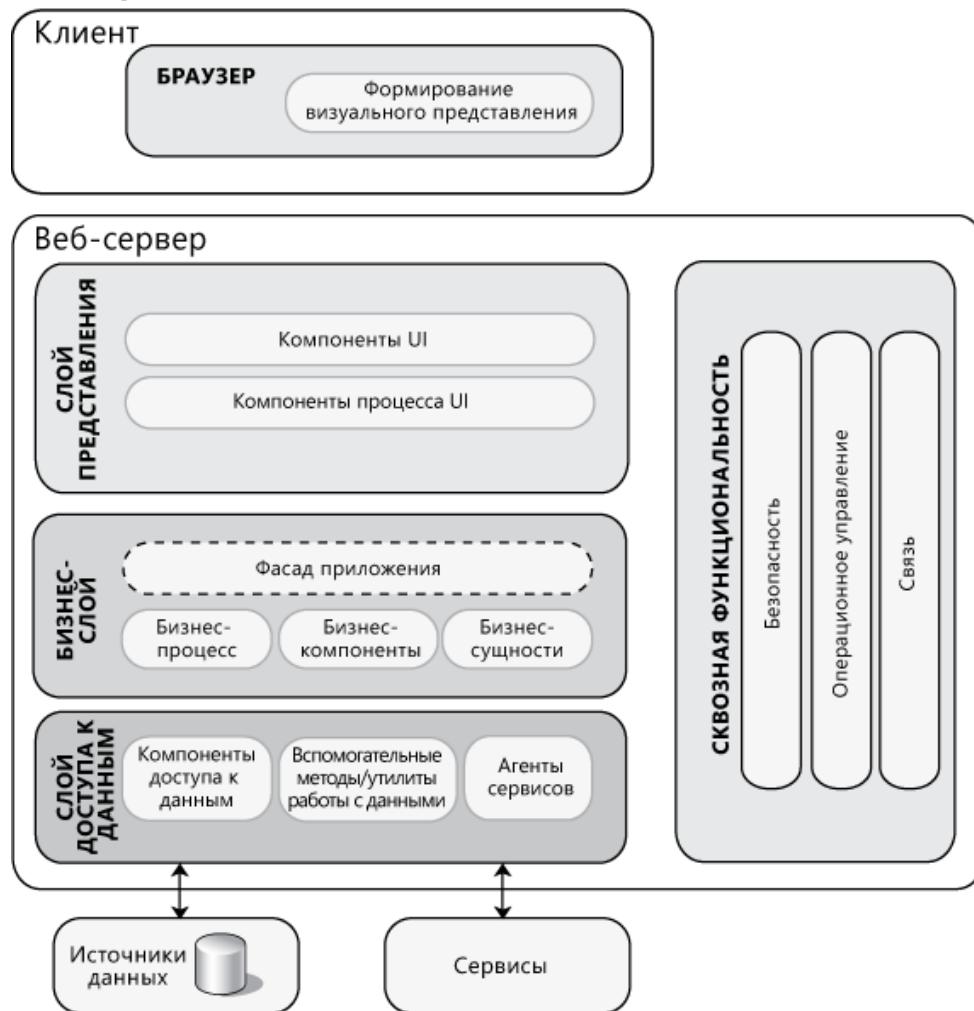


Рис. 30

Типовая структура Веб-приложения

Как правило, Веб-приложение осуществляет доступ к хранилищам данных на удаленном сервере базы данных. Также оно может потреблять сервисы, предоставляемые другими приложениями, включая размещаемые сервисы типа ПО + сервисы и Веб-сервисы.

Используйте Веб-приложения, если:

- Приложению не требуется поддерживать насыщенный UI и мультимедиа, что предлагает насыщенное Интернет-приложение.
- Требуется обеспечить простую модель развертывания в Веб.
- Пользовательский интерфейс должен быть независимым от платформы.
- Приложение должно быть доступным через Интернет.
- Требуется максимально сократить зависимости на стороне клиента и потребление ресурсов, таких как дисковое пространство или вычислительные мощности процессора.

Проектированию Веб-приложений посвящена глава 21, «[Проектирование Веб-приложений](#)».

21

Проектирование Веб-приложений

Обзор

В данной главе рассматриваются общие принципы проектирования и основные атрибуты Веб-приложения. В нее также включены рекомендации по созданию многослойной структуры, по обеспечению наилучшей производительности, безопасности и развертыванию, а также рекомендации по выбору шаблонов и технологий.

Веб-приложение – это приложение, которое может использоваться пользователями через Веб-браузер или специализированный агент пользователя. Браузер создает HTTP-запросы к определенным URL, которые сопоставляются с ресурсами на Веб-сервере. Сервер формирует визуальное представление HTML-страниц, которое может быть отображено браузером, и возвращает их клиенту. Ядро Веб-приложения – это его логика на стороне сервера. Такое приложение может состоять из множества отдельных слоев. Обычным примером является трехслойная архитектура, включающая слой представления, бизнес-слой и слой доступа к данным. Рис. 1 иллюстрирует типовую архитектуру Веб-приложения с чаще всего применяемыми компонентами, сгруппированными по функциональным областям.

Веб-приложение

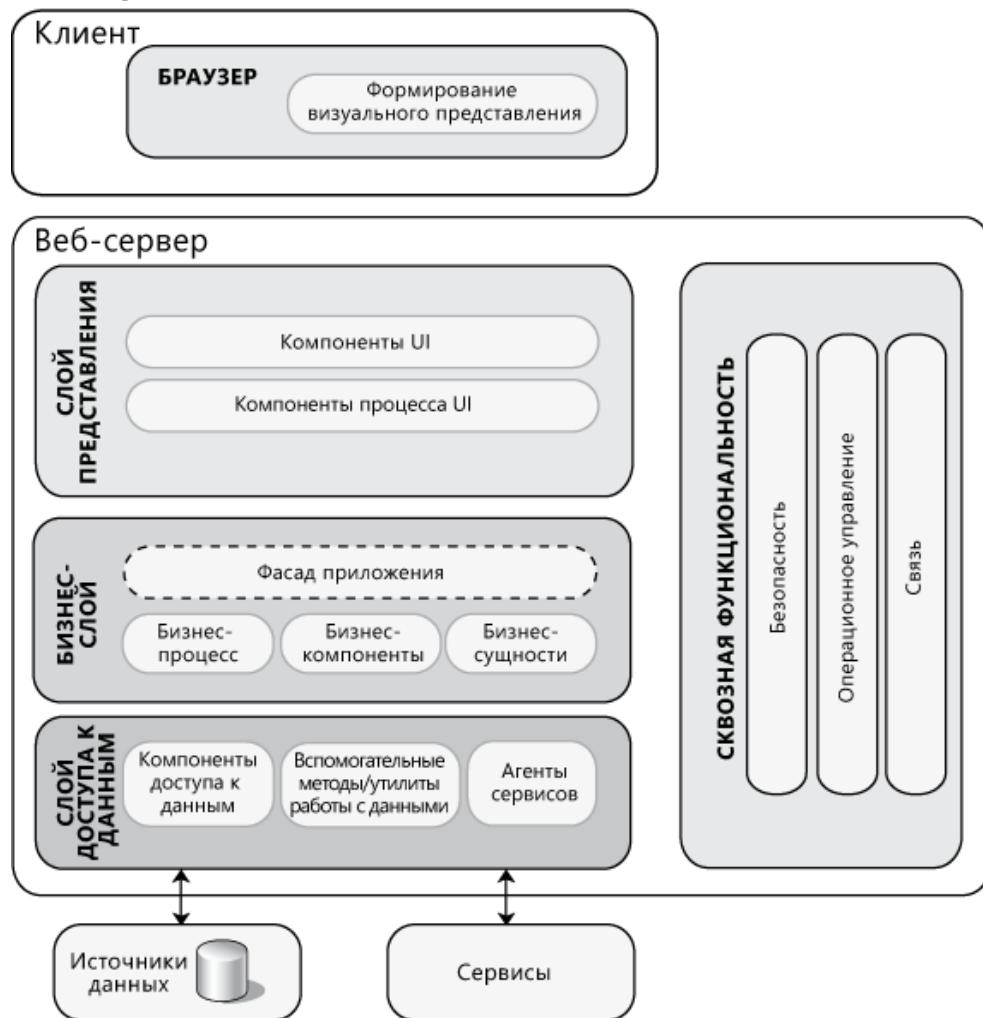


Рис. 31

Типовая структура Веб-приложения

Слой представления обычно включает компоненты UI и логики представления; бизнес-слой – компоненты бизнес-логики, бизнес-процесса и бизнес-сущностей, а также иногда фасад; слой доступа к данным – компоненты доступа к данным и агенты сервисов. Более подробно многослойный дизайн рассматривается в главе 5, «[Рекомендации по проектированию многослойных приложений](#)». Компонентам, используемым в каждом слое, посвящена глава 10, «[Рекомендации по проектированию компонентов](#)».

Общие принципы проектирования

Основной целью архитектора ПО при проектировании Веб-приложения является максимальное упрощение дизайна через разделение задач на функциональные области, обеспечивая при этом безопасность и высокую производительность. Данные рекомендации помогут выполнить все требования и создать условия для эффективной работы приложения в свойственных Веб-приложениям сценариях:

- **Выполните логическое разделение функциональности приложения.** Используйте многослойную структуру для логического разделения приложения на слой представления, бизнес-слой и слой доступа к данным. Это поможет создать

удобный в обслуживании код и позволит отслеживать и оптимизировать производительность каждого слоя в отдельности. Четкое логическое разделение также обеспечивает более широкие возможности масштабирования приложения.

- **Используйте абстракцию для реализации слабого связывания между слоями.** Этот подход можно реализовать путем определения интерфейсных компонентов, таких как фасад с общеизвестными входными и выходными параметрами, который преобразует запросы в формат, понятный компонентам слоя. Кроме того, с помощью интерфейсных типов или абстрактных базовых классов можно определить совместно используемую абстракцию, которая должна быть реализована интерфейсными компонентами.
- **Определитесь с тем, как будет реализовано взаимодействие компонентов друг с другом.** Для этого необходимо понимать сценарии развертывания, которые должно поддерживать приложение. Выясните, должно ли поддерживаться взаимодействие через физические границы или границы процесса, либо все компоненты будут выполняться в рамках одного процесса.
- **Используйте кэширование для сокращения количества сетевых вызовов и обращений к базе данных.** При проектировании Веб-приложения используйте такие техники, как кэширование и буферизация вывода, для сокращения сетевых вызовов между браузером и Веб-сервером и Веб-сервером и нижестоящими серверами. Правильно спроектированная стратегия кэширования, вероятно, единственный наиболее важный с точки зрения производительности аспект дизайна. ASP.NET предоставляет следующие возможности кэширования: кэширование вывода страницы, частичное кэширование страниц и Cache API. Используйте эти возможности при проектировании приложения.
- **Используйте протоколирование и инструментирование.** Необходимо выполнять аудит и протоколирование действий в слоях и уровнях приложения. Журналы регистрации событий могут использоваться для выявления подозрительных действий, что часто обеспечивает раннее обнаружение атак на систему. Не забывайте, что могут возникнуть сложности с регистрацией проблем, возникающих в коде сценариев, выполняющихся в браузере.
- **Продумайте аспекты аутентификации пользователей на границах доверия.** При проектировании приложения необходимо предусмотреть аутентификацию пользователей при пересечении границ доверия, например, при доступе удаленного бизнес-слоя со слоя представления.
- **Не передавайте конфиденциальные данные по сети в виде открытого текста.** Если требуется передавать по сети конфиденциальные данные, такие как пароль или cookie аутентификации, используйте для этого шифрование и подписи данных либо шифрование с использованием протокола Secure Sockets Layer (SSL).
- **Проектируйте Веб-приложение для выполнения под менее привилегированной учетной записью.** Процесс должен иметь ограниченный доступ к файловой системе и другим ресурсам системы. Это позволит максимально сократить возможные

негативные последствия на случай, если злоумышленник попытается взять процесс под свой контроль.

Общим вопросам проектирования посвящена глава 17, «[Сквозная функциональность](#)».

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Обработка запросов приложения](#)
 - [Аутентификация](#)
 - [Авторизация](#)
 - [Кэширование](#)
 - [Управление исключениями](#)
 - [Протоколирование и инструментирование](#)
 - [Навигация](#)
 - [Компоновка страницы](#)
 - [Формирование визуального отображения страницы](#)
 - [Управление сессиями](#)
 - [Валидация](#)
-

Обработка запросов приложения

Существует два основных способа обработки запросов Веб-приложением. При использовании подхода с обратной передачей браузер, преимущественно, взаимодействует с сервером посредством синхронных обращений Web Forms. Популярным альтернативным подходом является применение вызовов сервисов по протоколу REST между браузером и сервером. Оба подхода имеют определенные преимущества и недостатки, и их выбор определит пути реализации аспектов дизайна, обсуждаемых далее.

При выборе стратегии обработки запросов необходимо учесть требуемую степень контроля над UI в приложении, подход к разработке и тестированию, а также требования к производительности и масштабированию.

В подходе с обратной передачей обычно допускается разработка с использованием форм и применяются насыщенные серверные элементы управления, формирующие для браузера визуальное отображение при помощи соответствующего HTML, связанного состояния представления и логики взаимодействия. Используйте этот подход, если необходимо быстро создать Веб-приложение на базе форм.

Подход с применением протокола REST, как правило, дает возможность более детализированного управления UI приложения и обеспечивает большую гибкость с точки зрения навигации, тестируемости и разделения функциональных областей. Используйте этот подход, если для приложения необходима гибкая навигация, тонкое управление UI, если существует вероятность применения альтернативных технологий визуализации UI, или при разработке через тестирование.

Независимо от выбранной стратегии обработки запросов необходимо обеспечить разделение функциональных областей путем реализации логики обработки запросов и логики приложения отдельно от UI. Это обеспечивают несколько шаблонов. Четкое разделение функциональных областей при использовании подхода с обратной передачей Web Forms обеспечит шаблон Model-View-Presenter (MVP) или аналогичные ему шаблоны. Шаблон Model-View-Controller (MVC) обычно используется при обработке REST-запросов.

Также при проектировании стратегии обработки запросов руководствуйтесь следующими рекомендациями:

- Рассмотрите возможность централизации общих этапов пред- и пост-обработки запросов Веб-страницы для повышения возможности повторного использования логики на страницах. Например, разместите общую логику пред- и пост-обработки в специально предусмотренном для этого HTTP-модуле или базовом классе, наследуемом от ASP.NET-класса Page (Страница).
- Выберите соответствующий подход или шаблон для обработки UI. С помощью шаблонов MVC, MVP или аналогичных разделите обработку UI на три роли: модель, представление и контроллер/презентатор. Избегайте смешения в компонентах логики обработки и формирования визуального отображения.
- При проектировании представления для обработки больших объемов данных предоставьте ему доступ к модели через использование шаблона Supervising Presenter (или Supervising Controller), который является разновидностью шаблона MVP. Если в приложении нет необходимости поддерживать состояние представления, и имеется лишь ограниченное число событий управления, используйте шаблон MVC.
- По необходимости применяйте шаблон Intercepting Filter для реализации этапов обработки в виде подключаемых фильтров.
- Обеспечьте защиту всех конфиденциальных данных, передаваемых по сети, особенно по Интернету. Используйте протоколы безопасных каналов, такие как SSL, и предусмотрите шифрование и цифровые подписи для всех особо конфиденциальных данных, передаваемых как по внутренним, так и по внешним сетям.

Аутентификация

Проектирование эффективной стратегии аутентификации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак с подделкой пакетов, атак перебором по словарю, перехватом сеансов и

других типов атак. При проектировании стратегии аутентификации руководствуйтесь следующими рекомендациями:

- Определите границы доверия в рамках слоев Веб-приложения. Это поможет понять, где требуется проводить аутентификацию пользователей.
- Реализуйте практики управления учетными данными, такие как блокировки учетных записей и сроки действия паролей, а также строгие политики паролей, определяющие минимальную длину и сложность пароля.
- По возможности используйте поддерживаемые платформой механизмы аутентификации, такие как аутентификация Windows. Если решено применять аутентификацию с помощью форм, используйте преимущества встроенной поддержки в ASP.NET, не реализуйте собственный механизм аутентификации. Используйте интегрированную аутентификацию или единую регистрацию (SSO), если хотите предоставить пользователям возможность регистрироваться на нескольких сайтах с помощью одного набора учетных данных.
- Если приходится хранить пароли в базе данных, не храните их в виде открытого текста; храните хэш (или хэш с шумом) пароля.

Авторизация

Авторизация определяет, какие задачи может осуществлять аутентифицированный пользователь и к каким ресурсам он имеет доступ. Проектирование эффективной стратегии авторизации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для разглашения сведений, повреждения или подделки данных и несанкционированного получения прав. Глубокая защита – основной принцип безопасности стратегии авторизации приложения. При проектировании стратегии авторизации руководствуйтесь следующими рекомендациями:

- Выполняйте авторизацию пользователей при пересечении ими границ доверия. Используйте авторизацию URL для управления доступом к страницам и каталогам. Выполните доступ к нижестоящим ресурсам реализуйте соответственно модели доверенная подсистема, используя доверенное удостоверение, как описывается в главе 19, «[Физические уровни и развертывание](#)».
- Продумайте, насколько детализированными будут используемые настройки авторизации. Слишком детальная авторизация обусловит слишком высокие издержки на управление; однако меньшая детализация может негативно сказаться на гибкости.
- Применяйте олицетворение и делегирование, чтобы использовать преимущества аудита действий пользователя и тонкого контроля доступа, предоставляемые платформой, но не забывайте о влиянии на производительность и масштабируемость.

Кэширование

Кэширование способствует повышению производительности и сокращению времени отклика приложения. Тем не менее, неправильный выбор стратегии кэширования и плохой дизайн может привести к обратным эффектам. Кэширование позволит оптимизировать поиск данных, сократить количество повторных обращений к сети и избежать ненужной повторной обработки. При реализации кэширования, прежде всего, необходимо принять решение о том, когда будут загружаться данные в кэш. Чтобы избежать задержек на клиенте, кэш должен загружаться асинхронно или с помощью пакетной передачи. При проектировании стратегии кэширования руководствуйтесь следующими рекомендациями:

- По возможности кэшируйте данные в готовом к использованию виде и избегайте кэширования часто меняющихся данных. Кэшируйте конфиденциальные данные только в шифрованном виде.
- Используйте кэширование вывода для кэширования относительно статических страниц. Это значительно улучшает производительность, обеспечивая при этом поддержку изменения на основании передаваемых значений. Если относительно статична только часть страницы, реализуйте частичное кэширование страницы через кэширование пользовательских элементов управления.
- Не кэшируйте, а используйте пул для совместно используемых ресурсов, создание которых требует множества ресурсов, таких как сетевые подключения.

Подробно кэширование рассматривается в главе 17, «[Сквозная функциональность](#)».

Управление исключениями

Проектирование эффективной стратегии управления исключениями имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. Правильная обработка исключений в Веб-страницах предотвращает разглашение конфиденциальных данных об исключениях, повышает надежность приложения и обеспечивает сохранение согласованного состояния приложения в случае возникновения ошибки. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Обеспечьте пользователям понятные сообщения об ошибках, возникающих в приложении, но убедитесь, что не предоставляет конфиденциальные данные на страницах ошибок, в сообщениях об ошибках, файлах журналов и аудита.
- Перехватывайте необрабатываемые исключения и очищайте ресурсы и состояние после возникновения исключения. Проектируйте общий обработчик исключений, отображающий общую страницу ошибок или сообщение об ошибке для всех необрабатываемых исключений. Избегайте использования собственных исключений без крайней необходимости.
- Перехватывайте только те исключения, которые должны обрабатывать; например, чтобы удалить конфиденциальные данные или ввести дополнительные сведения в исключение. Не используйте исключения для управления потоком выполнения приложения.

Подробно управление исключениями рассматривается в главе 17, «[Сквозная функциональность](#)».

Протоколирование и инструментирование

Проектирование эффективной стратегии протоколирования и инструментирования имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. Необходимо выполнять аудит и протоколирование действий всех уровней приложения. Журналы могут использоваться для выявления подозрительных действий, что часто обеспечивает раннее обнаружение атак на систему. Файлы журналов и аудита могут пригодиться для доказательства правонарушений в случае судебного разбирательства, и пользователи уже не смогут безнаказанно отказываться от своих действий. Аудит считается наиболее достоверным, если данные журнала формируются непосредственно в момент доступа к ресурсу той же процедурой, которая выполняет этот доступ. При проектировании стратегии протоколирования и инструментирования руководствуйтесь следующими рекомендациями:

- Выполняйте аудит событий управления пользователями, критических событий системы, критических бизнес-операций и необычных действий.
- Создавайте политики безопасного управления файлами журналов, такие как ограничение доступа к файлам журналов, и предоставляйте пользователям доступ только для записи. Обеспечивайте настраиваемые в ходе разработки и производственной эксплуатации механизмы протоколирования и инструментирования.
- Не храните конфиденциальные данные в файлах журналов или аудита.

Навигация

Разделяйте стратегию навигации и логики обработки. Стратегия навигации должна позволять пользователям без труда перемещаться по экранам или страницам приложения. Обеспечьте единообразное представление навигационных ссылок и элементов управления во всем приложении, чтобы не запутывать пользователя и скрыть сложность приложения. При проектировании стратегии навигации руководствуйтесь следующими рекомендациями:

- При использовании обратной передачи в Web Forms применяйте такие шаблоны проектирования, как MVP, для отделения обработки UI от формирования визуального отображения вывода. Чтобы избежать смешения логики навигации с компонентами пользовательского интерфейса, обрабатывайте навигацию в Презентаторе (Presenter).
- При использовании протокола REST применяйте шаблон MVC, который поможет разнести логику приложения, данные и навигацию в разные компоненты. Типовая реализация MVC-приложения обеспечивает гибкую поддержку навигации через направление запросов в компонент контроллера, который координирует UI и данные приложения.

- Инкапсулируйте навигацию в главной странице¹, чтобы обеспечить ее единообразие по всему приложению. Однако избегайте указания путей перехода в коде приложения. Также обеспечьте, что пользователи могли переходить только к представлениям, для просмотра которых они авторизованы.
- Предусмотрите карту сайта, она поможет пользователям находить необходимые страницы, а поисковым механизмам просматривать сайт в случае необходимости. Используйте в UI визуальные элементы, такие как встроенные ссылки, навигационные меню и «хлебные крошки», чтобы пользователи понимали, где они находятся, что предлагается на сайте и как перемещаться по сайту быстро. Предусмотрите мастера для реализации предсказуемых переходов между формами.

Компоновка страницы

Дизайн приложения должен обеспечивать отделение компоновки страницы от конкретных компонентов UI и логики обработки UI. При выборе стратегии компоновки необходимо учесть, кто будет создавать компоновку: дизайнеры или разработчики. Если этим будут заниматься дизайнеры, выбирайте такой подход к компоновке, который не требует написания кода или использования инструментов разработки. При проектировании стратегии компоновки руководствуйтесь следующими рекомендациями:

- По возможности максимально используйте для компоновки Каскадные таблицы стилей (Cascading Style Sheets, CSS), а не компоновку на базе таблиц. Однако если требуется отображать сетку или если данные представлены в виде таблицы, должна использоваться компоновка на базе таблицы. Не забывайте, что формирование визуального отображения компоновки на базе таблицы может занимать довольно много времени, также сложная компоновка может приводить к различным проблемам.
- Постарайтесь максимально унифицировать компоновку страниц, чтобы обеспечить понятность и простоту работы с ними. Не создавайте большие страницы, выполняющие множество задач, особенно если с каждым запросом обычно выполняется лишь несколько задач. Максимально сокращайте размер страницы, чтобы обеспечить наилучшую производительность и снизить требования по пропускной способности.
- В ASP.NET-приложениях используйте шаблоны страниц для обеспечения общего внешнего вида и поведения для всех страниц и обновления сайта с минимальными трудозатратами. Общие разделы страниц вынесите в отдельные пользовательские элементы управления, это поможет упростить компоновку в целом и обеспечит возможность повторного использования этих элементов управления.
- Использование серверных элементов управления ASP.NET AJAX и библиотеки на стороне клиента ASP.NET AJAX упростит перенос клиентского сценария между

¹ Подразумевается шаблоны в виде Master Page, а не только стартовая страница сайт (прим. научного редактора)

разными браузерами. Также не смешивайте клиентский сценарий с HTML-кодом, это усложняет обслуживание страницы. Размещайте его в отдельных файлах сценария, чтобы обеспечить возможность их кэширования браузером.

- При миграции существующего Веб-приложения используйте на ASP.NET-страницах элементы управления Silverlight, это обеспечит насыщенное взаимодействие с пользователем и максимально сократит необходимость в реорганизации приложения.

Формирование визуального отображения страницы

При проектировании стратегии формирования визуального отображения страницы необходимо обеспечить эффективность этого процесса и максимальное удобство использования интерфейса. При выработке стратегии формирования визуального отображения руководствуйтесь следующими рекомендациями:

- Использование сценария на стороне клиента или ASP.NET AJAX обеспечит лучшее взаимодействие с пользователем и меньшее время отклика за счет снижения числа необходимых обратных передач. Применение собственного клиентского сценария может усложнить тестирование приложений, поскольку разные браузеры и версии реализуют разную поддержку сценариев. Лучше используйте ASP.NET AJAX, который поддерживает большинство популярных браузеров. Не забывайте, что использование любого кода на стороне клиента (включая сценарии, порождаемые встроенными элементами управления ASP.NET) может неблагоприятно сказываться на доступности. Обеспечьте соответствующую поддержку специальных возможностей для специализированных агентов¹ и пользователей с ограниченными возможностями.
- Рассмотрите возможности привязки данных. Например, можно выполнять привязку коллекций, объектов **DataReader**, таблиц **DataSet** и собственных объектов ко многим элементам управления ASP.NET. Используйте технологии разбиения данных на страницы для сокращения проблем с масштабируемостью, обусловленных необходимостью обработки больших объемов данных, и для улучшения производительности и времени отклика.
- Обеспечьте поддержку локализации в компонентах UI.
- Отделите компоненты пользовательского процесса от формирования визуального отображения данных и функций запроса.

Управление сессиями

При проектировании Веб-приложения важно выработать эффективную и безопасную стратегию управления сессиями, это будет иметь большое значение для производительности и надежности. При этом следует учесть такие аспекты, как что сохранять, где сохранять и как

¹ Например, роботы поисковых систем (прим. научного редактора).

долго удерживать сохраненные данные. При проектировании стратегии управления сеансами руководствуйтесь следующими рекомендациями:

- Продумайте, есть ли необходимость сохранять состояние сеанса. Использование состояния сеанса обуславливает дополнительные издержки при каждом запросе страницы.
- Сохраняйте данные сеанса, только если это действительно необходимо, для улучшения производительности используйте сеансы только для чтения или полностью отключайте состояние сеанса.
- Если используется один Веб-сервер, требуется обеспечить оптимальную производительность обработки состояния сеанса и число параллельных сеансов невелико, храните состояние внутри процесса. Однако если воссоздание данных сеанса чрезвычайно ресурсоемко и требуется обеспечить длительное хранение данных в случае перезапуска ASP.NET, используйте сервис состояния сеанса, выполняющийся на локальном Веб-сервере. Для сценариев с использованием множества серверов (Веб-фермы), где требуется централизованное хранение данных сеансов серверов, используйте хранилище состояния SQL Server.
- Если данные состояния хранятся на отдельном сервере, защитите канал связи для передачи состояния сеанса с помощью таких техник, как SSL или IPSec.
- Для снижения затрат на сериализацию отдавайте предпочтение базовым типам при хранении данных сеансов.

Валидация

Эффективная стратегия валидации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. При несоответствующей или недостаточно полной валидации приложение может оказаться уязвимым к таким угрозам безопасности, как межсайтовые атаки внедрением сценариев (cross-site scripting, XSS), атаки типа внедрение SQL-кода, переполнение буфера и другие типы атак посредством входных данных. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- Проверяйте все данные, пересекающие границы доверия приложения. Исходите из предположения, что все управляемые клиентом данные являются злонамеренными и подлежат валидации.
- Стратегия валидации должна обеспечивать ограничение, отклонение и очистку злонамеренного ввода. Проводите проверку длины, диапазона, формата и типа вводимых данных.
- Используйте проверку на стороне клиента для оптимизации взаимодействия с пользователем и сокращения числа обращений к сети, но для обеспечения безопасности всегда выполняйте проверку на сервере.
- Изучите решения, шаблоны проектирования и библиотеки сторонних производителей, которые помогут реализовать централизованное управление и повторное использование правил и кода валидации.

Более подробно методики валидации рассматриваются в главе 17, «[Сквозная функциональность](#)».

Вопросы проектирования слоев

В следующих разделах описываются специальные аспекты, которые необходимо принять во внимание при проектировании слоев многослойного приложения.

Слой представления

Слой представления Веб-приложения обеспечивает отображение UI и взаимодействие с пользователем. Дизайн должен разделять функциональные области, т.е. логика взаимодействия с пользователем должна быть отделена от компонентов UI. В сложных интерфейсах используйте отдельные компоненты UI и компоненты логики представления, и по возможности создавайте компоненты UI на базе стандартных элементов управления Веб. Элементы управления могут быть скомпилированы в сборку для повторного использования в других приложениях или в случае необходимости введения дополнительных функций в существующие серверные элементы управления.

Слой представления Веб-приложения состоит из серверных компонентов (которые формируют визуальное отображение HTML) и клиентских компонентов (браузер или агент пользователя, которые выполняют сценарии и отображают HTML). Как правило, вся логика представления располагается в серверных компонентах, а клиентские компоненты только отображают HTML. Такие клиентские технологии, как AJAX, позволяют выполнять логику на клиенте. Обычно это делается для улучшения взаимодействия с пользователем, но для этого требуются дополнительная разработка и тестирование. Если принято решение выполнять какую-то проверку на стороне клиента, не забудьте продублировать эту же проверку на сервере, поскольку любая валидация на клиенте может быть подделана.

Бизнес-слой

При проектировании бизнес-слоя Веб-приложения необходимо продумать реализацию бизнес-логики и длительных рабочих процессов. Использование выделенного бизнес-слоя, который будет реализовывать бизнес-логику и рабочие процессы, может упростить обслуживание и улучшить тестируемость приложения, а также позволит централизовать и повторно использовать общие функции бизнес-логики. Проектируйте бизнес-сущности, представляющие реальные данные, и используйте их для передачи данных между компонентами.

Проектируйте бизнес-слой без сохранения состояния, это поможет сократить конкуренцию за ресурсы и повысить производительность. Используйте интерфейс взаимодействия на основе сообщений, это замечательный вариант для бизнес-слоя без сохранения состояния для Веб-приложения. В случае осуществления в бизнес-слое критических бизнес-операций используйте транзакции для обеспечения целостности данных и предотвращения их утери.

Слой доступа к данным

В слое доступа к данным Веб-приложения абстрагируйте логику, необходимую для доступа к базе данных. Использование отдельного слоя доступа к данным упрощает конфигурацию и обслуживание приложения, и также скрывает детали базы данных от остальных слоев приложения. Проектируйте объекты сущностей, которые могут заполняться слоем доступа к данным или использоваться для обновления источника данных, и применяйте объекты передачи данных (DTO) для взаимодействия с другими слоями и передачи данных между слоями.

Использование в слое доступа к данным пула подключений позволит сократить число открытых подключений, а пакетные операции помогут сократить количество обращений к базе данных. Слою доступа к данным также может понадобиться работать с внешними сервисами через агенты сервисов. Не забудьте выработать стратегию обработки исключений, обеспечивающую обработку ошибок доступа к данным и передачу исключений на бизнес-слой.

Слой сервисов

Проектируйте отдельный слой сервисов, если планируется развертывать бизнес-слой удаленно или предполагается предоставлять бизнес-логику через Веб-сервис. Отсутствие конкретизации деталей клиентов, которые будут использовать сервисы, обеспечит максимальные возможности их повторного использования. В будущем избегайте внесения изменений, которые могут нарушить интерфейс сервиса для существующих клиентов. Реализация версий интерфейса позволит клиентам подключаться к соответствующей версии интерфейса.

Если бизнес-слой располагается удаленно, проектируйте слабо детализированные методы сервисов, чтобы сократить количество обращений к сети и обеспечить слабое связывание. Также обеспечьте идемпотентность (т.е. возможность обработки ситуации, когда один и тот же запрос поступает несколько раз) и коммутативность (т.е. возможность обработки ситуации, когда сообщения, осуществляющие определенную последовательность задач, поступают в произвольном порядке) сервисов. Не реализуйте бизнес-правила в интерфейсе сервиса, это может создать ненужные зависимости между компонентами и клиентами и усложнить задачу по поддержанию стабильности интерфейса.

Наконец, выполните требования по обеспечению возможности взаимодействия, выбирая соответствующие протоколы и механизмы транспортировки. Например, используйте ASMX для общего и WCF для более детального управления конфигурацией. Примите решение о том, будет ли интерфейс предоставлять методы SOAP, REST или оба типа методов. Более подробно сервисы рассматриваются в главе 9, «[Рекомендации по проектированию слоя сервисов](#)», и главе 18, «[Взаимодействие и обмен сообщениями](#)».

Вопросы тестирования и testability

Тестируемость – это мера того, насколько легко создавать критерии проверки для системы или компонентов и насколько хорошо они подвергаются тестированию на соответствие этим критериям. Учитывать аспекты testability при проектировании архитектуры необходимо

для того, чтобы обеспечить раннюю диагностику проблем и, таким образом, сократить затраты на обслуживание. Для обеспечения тестируемости:

- Четко определяйте ввод и вывод слоев и компонентов приложения уже на этапе проектирования.
- Используйте для слоя представления шаблоны отделения представления, такие как MVC или MVP. Это обеспечит возможность модульного тестирования логики представления.
- Реализуйте бизнес-логику и рабочие процессы в отдельном бизнес-слое, что улучшит тестируемость приложения.
- Проектируйте слабо связанные компоненты, которые можно тестировать по отдельности.
- Выработайте эффективную стратегию протоколирования и трассировки, что позволит выявлять или диагностировать ошибки, которые в противном случае было бы сложно найти. Обеспечьте данные протоколирования и трассировки, которые могут использоваться средствами мониторинга или управления. Это поможет обнаруживать дефектный код при возникновении ошибок. Файлы журнала должны содержать сведения, которые можно использовать для воссоздания проблемы.

Вопросы выбора технологий

В ASP.NET модель ASP.NET Web Forms можно сочетать с широкой номенклатурой других технологий, включая ASP.NET AJAX, ASP.NET MVC, Silverlight и ASP.NET Dynamic Data. Примите во внимание следующие рекомендации:

- Используйте ASP.NET для создания приложений, доступ к которым осуществляется через Веб-браузер, или специализированного агента пользователя.
- Используйте ASP.NET и AJAX для создания приложений, обеспечивающих повышенные возможности взаимодействия с пользователем и фоновую обработку с меньшим количеством перезагрузок страниц.
- Используйте ASP.NET и элементы управления Silverlight для создания приложений, которые включают насыщенное медиа-содержимое и расширенные возможности взаимодействия с пользователем.
- При использовании ASP.NET позаботьтесь о предоставлении шаблонов страниц для реализации единообразного UI для всех страниц.
- Используйте ASP.NET Dynamic Data для создания управляемого данными Веб-приложения, страницы которого формируются на основании модели данных базовой базы данных.
- При использовании разработки через тестирование или в случае необходимости обеспечить детализированное управление UI применяйте шаблон MVC и ASP.NET MVC. Это позволит четко отделить логику приложения и навигации от UI приложения.

Вопросы выбора стратегии развертывания

При развертывании Веб-приложения необходимо учесть то, как размещение слоев и компонентов будет влиять на производительность, масштабируемость и безопасность приложения. Возможно, придется пойти на некоторые компромиссы. В зависимости от бизнес-требований и ограничений инфраструктуры используйте либо распределенное, либо нераспределенное развертывание. Нераспределенное развертывание, как правило, приводит к повышению производительности за счет снижения числа вызовов через физические границы. Однако распределенное развертывание позволит обеспечить лучшую масштабируемость и реализовать защиту каждого слоя в отдельности.

Нераспределенное развертывание

В сценарии нераспределенного развертывания все логически разделенные слои Веб-приложения, кроме базы данных, физически располагаются на одном Веб-сервере. Необходимо продумать, как приложение будет реализовывать параллельную работу множества пользователей и как защитить слои, находящиеся на одном сервере. Этот сценарий представлен на рис. 2.

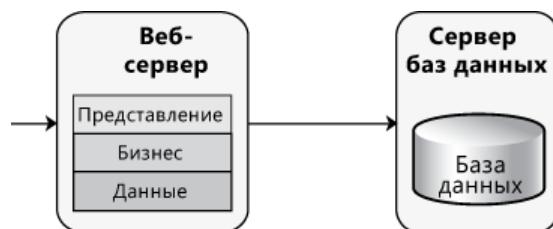


Рис. 32

Нераспределенное развертывание Веб-приложения

Выбирая нераспределенное развертывание, придерживайтесь следующих рекомендаций:

- Используйте нераспределенное развертывание, если Веб-приложение чувствительно к производительности, поскольку локальные вызовы между слоями не имеют столь пагубного влияния на производительность, как удаленные вызовы между уровнями.
- Если с бизнес-логикой будет работать только слой представления и не требуется обеспечивать ее совместное использование другими приложениями, проектируйте для бизнес-слоя компонентный интерфейс.
- Если бизнес-логика и логика представления выполняются в одном процессе, избегайте аутентификации в бизнес-слое.
- Для доступа к базе данных используйте доверенное удостоверение (через модель доверенной подсистемы). Это повысит производительность и масштабируемость приложения.
- Продумайте, как будет реализована защита конфиденциальных данных, передаваемых между Веб-сервером и сервером базы данных.

Распределенное развертывание

В сценарии распределенного развертывания слой представления и бизнес-слой Веб-приложения располагаются на разных уровнях и взаимодействуют удаленно. Как правило, бизнес-слой и слой доступа к данным размещаются на одном сервере. Рис. 2 демонстрирует такой сценарий.

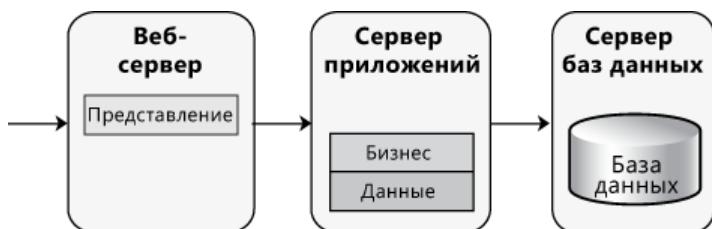


Рис. 3

Распределенное развертывание Веб-приложения

При выборе распределенного развертывания руководствуйтесь следующими рекомендациями:

- Не развертывайте бизнес-слой на отдельном уровне без крайней необходимости; например, чтобы увеличить масштабируемость, или если соображения безопасности запрещают развертывание бизнес-логики на внешнем Веб-сервере.
- Используйте для бизнес-слоя интерфейс взаимодействия на основе сообщений.
- Используйте для взаимодействия с бизнес-слоем TCP-протокол с бинарным кодированием. Это обеспечит лучшую производительность.
- Предусмотрите защиту конфиденциальных данных, передаваемых между разными уровнями.

Балансировка нагрузки

Если Веб-приложение развертывается на множестве серверов, балансировка нагрузки обеспечит распределение запросов и их обработку различными Веб-серверами. Такой подход обеспечит меньшее время отклика, лучшее использование ресурсов и пропускную способность. Этот сценарий продемонстрирован на рис. 4.



Рис. 4

Балансировка нагрузки Веб-приложения

При проектировании балансировки нагрузки для Веб-приложения руководствуйтесь следующими рекомендациями:

- При проектировании Веб-приложения по возможности избегайте привязки к одному серверу, поскольку это может негативно сказаться на возможности масштабирования приложения. Привязка к серверу возникает, когда создаются условия, при которых все запросы одного клиента должны обрабатываться одним сервером. Обычно это происходит при использовании локально обновляемых кэшей, хранения состояния сеансов в процессе или локальных хранилищ состояния сеансов. Если требуется поддерживать привязку к одному серверу, конфигурируйте кластер так, чтобы он обеспечивал маршрутизацию всех запросов от определенного пользователя к одному и тому же серверу.
- Проектируйте для Веб-приложения компоненты без сохранения состояния; например, Веб-интерфейс без промежуточного состояния и без бизнес-компонентов, сохраняющих состояние. Если требуется сохранять состояние пользователей, избегайте применения в Веб-ферме управления сессиями в процессе, если не можете настроить привязку к серверу и гарантировать, что запросы от одного пользователя будут направляться на один и тот же сервер. Лучше используйте внешний сервис хранения состояния или сервер базы данных.
- Используйте Windows Network Load Balancing (NLB)¹ для реализации перенаправления запросов на серверы фермы приложений.
- Используйте кластеризацию для сокращения негативных последствий сбоев оборудования.

¹ Система балансировки сетевой нагрузки Windows (прим. переводчика).

- Если приложению предъявляются высокие требования по вводу/выводу, выполните секционирование базы данных и распределение ее на множество серверов баз данных.

Более подробно шаблоны развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Кэширование, Управление исключениями, Протоколирование и инструментирование, Компоновка страницы, Представление, Обработка запросов и Слой интерфейса сервиса, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Кэширование	Cache Dependency. Использует внешние данные для определения состояния данных, хранящихся в кэше. Page Cache. Улучшает время отклика динамических Веб-страниц, доступ к которым осуществляется довольно часто, но сами они меняются реже и потребляют большое количество ресурсов системы для воссоздания.
Управление исключениями	Exception Shielding. При возникновении исключения предотвращает предоставление сервисом данных о его внутренней реализации.
Протоколирование и инструментирование	Provider. Реализует компонент, который предоставляет API, отличающийся от клиентского API, обеспечивая возможность подключения любой специальной реализации.
Компоновка страницы (UI)	Composite View. Сочетает отдельные представления в композитное представление. Template View. Реализует представление общего шаблона и создает представления на базе этого шаблонного представления. Transform View. Преобразует данные, переданные на уровень представления, в HTML для отображения в UI. Two-Step View. Преобразует модель данных в логическое представление без какого-либо специального форматирования и затем преобразует это логическое представление, добавляя необходимое форматирование.
Представление	Model-View-Controller (Модель-Представление-Контроллер). На основании пользовательского ввода распределяет данные предметной области, представления и действий в три разных класса. Модель управляет поведением и данными предметной области приложения, отвечает на запросы о состоянии (обычно поступающие от Представления) и на команды на изменение состояния (обычно поступающие от Контроллера). Представление управляет отображением данных. Контроллер интерпретирует ввод пользователя, поступающий с устройств ввода, таких как мышь и клавиатура, и дает указания модели и/или представлению о соответствующих изменениях.

	<p>Model-View-Presenter (Модель-Представление-Презентатор). Разделяет обработку запроса на три отдельные роли, где Представление отвечает за обработку пользовательского ввода, Модель – за данные приложения и бизнес-логику, и Презентатор – за логику представления и координирование взаимодействия между Представлением и Моделью.</p> <p>Passive View. Разновидность шаблона MVC. Сокращает представление до необходимого минимума, перенося в контроллер функциональность обработки пользовательского ввода и реализацию обновления представления.</p> <p>Supervising Presenter (или Supervising Controller). Разновидность шаблона MVC, где контроллер отвечает за обработку сложной логики, в частности, согласование представлений, а представление отвечает за простую логику, касающуюся представления.</p>
Обработка запросов	<p>Intercepting Filter. Цепочка пригодных для компоновки фильтров (независимые модули), реализующих обычные задачи предварительной и последующей обработки при запросе Веб-страницы.</p> <p>Page Controller. Принимает ввод из запроса и обрабатывает его для конкретной страницы или действия Веб-сайта.</p> <p>Front Controller. Консолидирует обработку запросов путем направления всех запросов через один объект-обработчик, который можно изменять во время выполнения с помощью декораторов.</p>
Слой интерфейса сервиса	<p>Fascade. Реализует унифицированный интерфейс для набора операций, чтобы обеспечить упрощенный интерфейс и уменьшить связанность систем.</p> <p>Service Interface. Программный интерфейс, который может использоваться другими системами для взаимодействия с сервисом.</p>

Более подробно шаблон Page Cache рассматривается в статье «*Enterprise Solution Patterns Using Microsoft .NET*» по адресу <http://msdn.microsoft.com/en-us/library/ms998469.aspx>.

Более подробно шаблоны Model-View-Controller (MVC), Page Controller, Front Controller, Template View, Transform View и Two-Step View рассматриваются в книге Мартина Фаулера «Архитектура корпоративных приложений». Addison-Wesley, 2002. Или по адресу <http://martinfowler.com/eaaCatalog>.

Более подробно шаблоны Composite View, Supervising Presenter и Presentation Model рассматриваются в статье «*Patterns in the Composite Application Library*» по адресу <http://msdn.microsoft.com/en-us/library/cc707841.aspx>.

Более подробно шаблон Exception Shielding рассматривается в статье «*Useful Patterns for Services*» по адресу <http://msdn.microsoft.com/en-us/library/cc304800.aspx>.

Более подробно шаблон Service Interface рассматривается в статье «*Service Interface*» (Интерфейс сервиса) по адресу <http://msdn.microsoft.com/en-us/library/ms998421.aspx>.

Более подробно шаблон Provider рассматривается в статье «*Provider Model Design Pattern and Specification, Part I*» по адресу <http://msdn.microsoft.com/en-us/library/ms998421.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- Более подробно проектирование и реализация клиентских Веб-приложений рассматривается в статье «*Design and Implementation Guidelines for Web Clients*» по адресу <http://msdn.microsoft.com/en-us/library/ms978605.aspx>.
- Проектированию распределенных Веб-приложений посвящена статья «*Designing Distributed Applications*» (Проектирование распределенных приложений) по адресу [http://msdn.microsoft.com/en-us/library/aa292470\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa292470(VS.71).aspx).
- Вопросы производительности Веб-приложения подробно рассматриваются в статье «*Improving .NET Application Performance and Scalability*» (Улучшение производительности и масштабируемости .NET-приложений) по адресу <http://msdn.microsoft.com/en-us/library/ms998530.aspx>.
- Безопасности Веб-приложений посвящена статья «*Improving Web Application Security: Threats and Countermeasures*» (Повышение безопасности Веб-приложений: угрозы и меры противодействия им) по адресу <http://msdn.microsoft.com/en-us/library/ms994921.aspx>.

22

Проектирование насыщенных клиентских приложений

Обзор

В данной главе обсуждаются основные сценарии использования насыщенных клиентских приложений, их компоненты и важные аспекты проектирования. Также будут рассмотрены сценарии развертывания насыщенных клиентских приложений, основные шаблоны и технологии их проектирования.

Пользовательские интерфейсы насыщенных клиентов могут обеспечивать высокую производительность, интерактивность и насыщенное взаимодействие с пользователем для приложений, которые должны работать как самодостаточные приложения и в сценариях с подключением, без постоянного подключения и без подключения. Windows Forms, Windows Presentation Foundation (WPF) и Microsoft Office Business Application (OBA) – все это доступные среды и инструменты разработки, позволяющие разработчикам быстро и без труда создавать насыщенные клиентские приложения.

Эти технологии могут использоваться для создания как самодостаточных приложений, так и приложений, выполняющихся на клиентском компьютере, но взаимодействующих с сервисами, предоставляемыми другими уровнями (как логическими, так и физическими), и другими приложениями, предоставляющими операции, которые необходимы клиенту. К этим операциям относятся доступ к данным, извлечение данных, поиск, отправка данных в другие системы, резервное копирование и т.д. На рис. 1 представлен общий вид архитектуры типового насыщенного клиента и показаны компоненты, обычно располагающиеся в каждом из слоев.

Насыщенное клиентское приложение

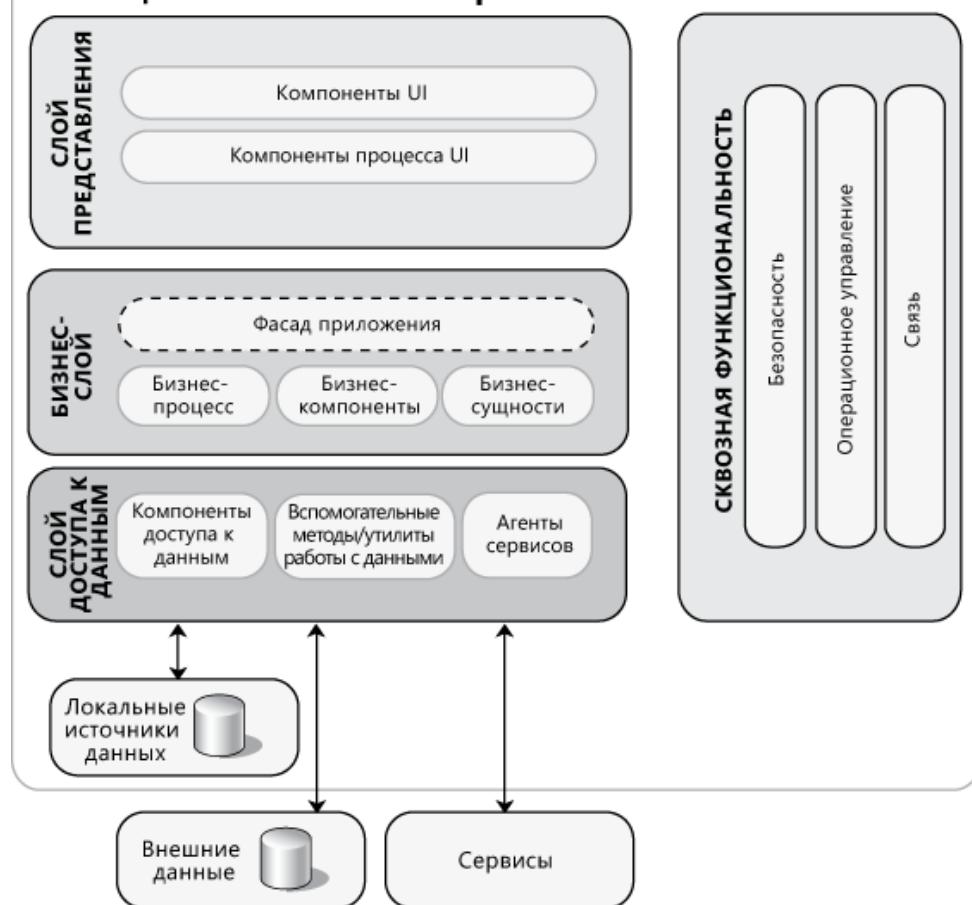


Рис. 33

Общее представление архитектуры типового насыщенного клиентского приложения

Типовое насыщенное клиентское приложение включает три слоя: слой представления, бизнес-слой и слой доступа к данным. Слой представления, как правило, содержит компоненты UI и логики представления; бизнес-слой – компоненты бизнес-логики, бизнес-процесса и бизнес-сущностей; и слой доступа к данным – компоненты доступа к данным и агентов сервисов.

Многослойному дизайну посвящена глава 5, «[Рекомендации по проектированию многослойных приложений](#)». Более подробно компоненты каждого слоя рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)».

Насыщенные клиентские приложения могут быть довольно тонкими приложениями, состоящими, главным образом, из слоя представления, который с помощью сервисов выполняет доступ к удаленному бизнес-уровню, размещаемому на серверах. Примером такого приложения является приложение для ввода данных, передающее все данные на сервер для обработки и хранения.

И наоборот, насыщенные клиентские приложения могут быть очень сложными приложениями, которые осуществляют большую часть обработки самостоятельно и взаимодействуют с другими сервисами и хранилищами данных для получения или отправки данных. Примером такого приложения является ПО на базе Microsoft Excel®, которое выполняет сложные задачи локально, сохраняет состояние и данные локально и взаимодействует с удаленными серверами только для извлечения и обновления связанных

данных. Такие насыщенные клиенты имеют собственные бизнес-слои и слои доступа к данным. Рекомендации по проектированию бизнес-слоев и слоев доступа к данным в таких приложениях аналогичны рекомендациям для всех остальных приложений.

Общие принципы проектирования

Целью архитектора ПО при проектировании насыщенного клиентского приложения является правильный выбор технологии и создание максимально простой структуры за счет распределения задач по разным функциональным областям. Дизайн должен отвечать предъявляемым к приложению требованиям по производительности, безопасности, возможности повторного использования и простоты обслуживания.

При проектировании насыщенных клиентских приложений руководствуйтесь следующими рекомендациями:

- **Выбирайте соответствующую технологию, исходя из требований, предъявляемых к приложению.** К таким технологиям относятся Windows Forms, WPF, XAML Browser Applications (XBAP) и ОВА.
- **Разделяйте логику представления и реализацию интерфейса.** Используйте такие шаблоны проектирования, как Presentation Model и Supervising Presenter (или Supervising Controller), которые отделяют формирование визуального отображения UI от логики UI. Использование отдельных компонентов в приложении сокращает количество зависимостей, упрощает обслуживание и тестирование и создает лучшие условия для повторного использования.
- **Выясните задачи представления и потоки представления.** Это поможет спроектировать все окна и шаги с использованием многооконного подхода или мастера.
- **Спроектируйте подходящий и удобный интерфейс.** Для обеспечения максимальной доступности, удобства и простоты использования учтите такие аспекты, как компоновка, навигация, выбор элементов управления и локализация.
- **Применяйте разделение функциональных областей во всех слоях.** Например, вынесите бизнес-правила и другие задачи, не связанные с представлением, в отдельный бизнес-слой, а код доступа к данным в отдельные компоненты, размещаемые в слое доступа к данным.
- **Используйте существующую общую логику представления.** Библиотеки, содержащие шаблонные, обобщенные функции валидации на стороне клиента, и вспомогательные классы могут использоваться повторно многими приложениями.
- **Обеспечивайте слабое связывание клиента с удаленными сервисами, которые он использует.** Применяйте интерфейс взаимодействия на основании сообщений для взаимодействия с сервисами, размещенными на других уровнях.
- **Избегайте тесного связывания с объектами других слоев.** При взаимодействии с другими слоями приложения используйте абстракцию, предоставляемую общими описаниями интерфейсов, абстрактные базовые классы или обмен сообщениями.

Например, реализация шаблонов Dependency Injection и Inversion of Control может обеспечить абстракцию, которая будет использоваться при взаимодействии между слоями.

- **Сократите количество обращений к сети при доступе к удаленным уровням.**
Применяйте слабо детализированные методы и по возможности выполняйте их асинхронно, чтобы избежать блокировки UI или его перехода в состояние «не отвечает».

Более подробно проектирование бизнес-слоя рассматривается в главе 7, «[Рекомендации по проектированию бизнес-слоя](#)». Проектированию слоя доступа к данным посвящена глава 8, «[Рекомендации по проектированию слоя доступа к данным](#)».

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Бизнес-слой](#)
- [Сетевое взаимодействие](#)
- [Композиция](#)
- [Управление конфигурацией](#)
- [Доступ к данным](#)
- [Управление исключениями](#)
- [Удобство и простота обслуживания](#)
- [Слой представления](#)
- [Управление состоянием](#)
- [Рабочий процесс](#)

Бизнес-слой

Типовой тонкий насыщенный клиент выступает в роли интерфейса бизнес-системы, а бизнес-слой является частью этой бизнес-системы и обычно предоставляется как сервис. Однако в типовом толстом насыщенном клиенте бизнес-слой располагается на самом клиенте. При проектировании бизнес-слоя для насыщенного клиента руководствуйтесь следующими рекомендациями:

- Обозначьте, какие бизнес-слои и интерфейсы сервисов будут использоваться приложением. Если приложение будет работать с удаленными сервисами, импортируйте описания интерфейсов и напишите код доступа к функциям этих

сервисов с использованием интерфейсов. Это поможет уменьшить связанность между клиентом и удаленным бизнес-слоем или сервисами, которые он использует.

- Если бизнес-логика не касается конфиденциальных данных, разместите некоторые бизнес-правила на клиенте, чтобы улучшить производительность UI и клиентского приложения. Если бизнес-логика включает конфиденциальные данные, бизнес-слой должен размещаться на отдельном уровне.
- Продумайте, как клиент будет получать данные, необходимые для выполнения бизнес-правил и другой обработки на стороне клиента, и автоматически обновлять эти данные при их изменении, и как пользователи или администраторы будут обновлять бизнес-правила при изменении требований. Клиент может получать бизнес-правила с удаленного сервера при запуске.

Проектированию бизнес-слоя посвящена глава 7, «[Рекомендации по проектированию бизнес-слоя](#)».

Сетевое взаимодействие

Если бизнес-слой и слой доступа к данным насыщенного клиентского приложения располагаются на удаленном уровне и предоставляются как сервисы, или если насыщенный клиент использует другие удаленные сервисы, он может взаимодействовать с этими сервисами с помощью разнообразных протоколов и методов. Сюда относятся HTTP-запросы, сообщения электронной почты SMTP, SOAP-сообщения Веб-сервисов, DCOM для удаленных компонентов, протоколы удаленного доступа к базе данных и другие стандартные или специальные протоколы связи на базе TCP/IP. Если бизнес-слой и слой доступа к данным располагаются на клиенте, слой представления для взаимодействия с ними может использовать объектные методы. При проектировании стратегии взаимодействия руководствуйтесь следующими рекомендациями:

- При взаимодействии с удаленными сервисами по возможности используйте протокол на основе сообщений. Это обеспечивает более естественную организацию асинхронных вызовов, что позволит избежать блокировок слоя представления, и поддержку конфигураций сервера с балансировкой нагрузки и обработкой отказов. Используйте слабо детализированные интерфейсы взаимодействия, это позволит сократить сетевой трафик и повысить производительность.
- В случае необходимости обеспечьте автономную обработку данных в приложении. Выявляйте и отслеживайте состояние сетевого подключения. В случае отсутствия подключения кэшируйте данные локально и синхронизируйте их при восстановлении подключения. Храните состояние и данные приложения локально в постоянном кэше, это позволит выполнять запуск и выключение/перезапуск приложения без потери данных.
- Для защиты каналов связи используйте IPSec и SSL; конфиденциальные данные защищайте с помощью шифрования. Также применяйте цифровые подписи для выявления повреждения или подделки данных.

- Если приложение должно потреблять или отправлять большие наборы или объемы данных, необходимо учесть потенциальное влияние этих процессов на производительность и сеть. Выбирайте более эффективные протоколы связи, такие как TCP, в которых используются механизмы сжатия, позволяющие сокращать размер данных полезной нагрузки для основанных на сообщениях протоколов, таких как SMTP и SOAP, либо специальные бинарные форматы, если от приложения не требуется поддерживать открытые стандарты связи.

Более подробно взаимодействие клиентов и слоев приложения рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Композиция

Реализация интерфейса с использованием шаблона проектирования Composition (Композиция), когда UI состоит из отдельных модулей или форм, загружаемых динамически во время выполнения, позволяет повысить расширяемость и удобство обслуживания приложения, особенно если оно предоставляет сложный UI, как это обычно бывает во многих бизнес-сценариях. Такой подход удобен, если пользователям необходимо предоставить возможность открывать несколько форм для осуществления определенных задач и различным образом работать с данным). Пользователи могут открывать и закрывать формы по необходимости, благодаря этому повысится производительность приложения и сократятся задержки при запуске. Также продумайте поддержку персонализации интерфейса и предоставьте возможность пользователям изменять компоновку и содержимое соответственно собственным требованиям. При проектировании стратегии композиции руководствуйтесь следующими рекомендациями:

- На основании функциональных спецификаций и требований определите необходимые типы компонентов интерфейса. Это могут быть компоненты Windows Forms, WPF-формы, документы Office, пользовательские элементы управления или собственные модули.
- Если необходимо использовать композицию, выбирайте соответствующий механизм композиции и используйте для композиции представления отдельные модули, доступные для повторного использования. Например, скомпоновать представление из модульных атомарных составных частей поможет шаблон Composite View. Или в качестве альтернативного варианта можно использовать инфраструктуру композиции, такую как Composite Client Application Guidance группы patterns & practices, или встроенные возможности среды разработки, такие как пользовательские элементы управления или панели документов. При этом обеспечивайте минимальные зависимости между компонентами и по возможности используйте шаблоны абстракции, чтобы не усложнять обслуживание приложения. Реализуйте функции для автоматического обновления и контроля версий компонуемых компонентов.
- Если требуется поддерживать взаимодействие между разными формами и компонентами представления, образующими составной интерфейс, реализуйте несвязанное взаимодействие посредством шаблонов Publish/Subscribe или Command. Это позволит снизить связанность между компонентами и улучшить тестируемость.

- Используйте шаблоны и методики привязки данных, предлагаемые выбранной технологией реализации, для упрощения и сокращения количества кода каждой формы, используемой в составном интерфейсе.
 - Реализуйте персонализацию, чтобы пользователи могли настраивать компоновку компонентов интерфейса.
-

Управление конфигурацией

Насыщенные клиентские приложения обычно требуют загрузки конфигурационных данных при запуске, а иногда и во время выполнения. Эти данные могут включать данные сети или подключения, настройки пользователи, бизнес-правила UI или общие настройки отображения и параметры компоновки. Часть или все эти сведения могут храниться локально или загружаться с удаленного сервера при запуске приложения. Также может возникнуть необходимость сохранять изменения данных в ходе выполнения приложения или при завершении выполнения. Это могут быть предпочтения пользователя, параметры компоновки и другие данные UI локального профиля пользователя. При проектировании стратегии управления конфигурацией руководствуйтесь следующими рекомендациями:

- Определите, какие из настраиваемых данных могут изменяться в ходе жизненного цикла приложения; например, местоположения файлов, настройки в среде разработки и в среде производственной эксплуатации, настройки протоколирования, ссылки на сборки и контактная информация для отправки уведомлений. Если требуется, при проектировании приложения обеспечьте функциональность выявления изменений и динамического обновления конфигурации.
 - Выбирайте локальное или централизованное хранение данных. Управляемые пользователем данные (включая данные профиля или личные настройки), как правило, хранятся локально, хотя их можно хранить также централизованно для обеспечения возможности доступа с разных устройств. Глобальные настройки приложения должны храниться централизованно и загружаться локально по соображениям производительности.
 - Определите конфиденциальные конфигурационные данные и реализуйте подходящий механизм их защиты при передаче по сети, локальном хранении и даже при хранении в памяти.
 - Примите во внимание все глобальные политики безопасности и переопределения групповой политики, которые могут оказывать влияние или переопределять локальные настройки.
-

Доступ к данным

Насыщенные клиентские приложения, как правило, выполняют доступ к данным, хранящимся на удаленном сервере, но также и к локальным данным. Процесс доступа к данным часто оказывает существенное влияние на производительность и является наиболее чувствительным фактором в восприятии приложения пользователем с точки зрения удобства использования и времени отклика. Необходимо обеспечивать максимальную производительность процедур

доступа и передачи данных между уровнями. Также при проектировании приложения должны быть учтены типы данных, которые предполагается использовать. Если клиентское приложение не может обрабатывать данные в предоставляемом формате, должен быть реализован механизм трансляции, обеспечивающий их преобразование. Но нельзя забывать, что это негативно скажется на производительности. При проектировании стратегии доступа к данным руководствуйтесь следующими рекомендациями:

- По возможности загружайте данные асинхронно, так чтобы UI не блокировался при их загрузке. Однако необходимо также учесть конфликты, которые могут возникать при попытках пользователя взаимодействовать с данными до того, как они будут полностью загружены, и при проектировании интерфейса предусмотреть механизмы защиты от возникающих в результате этого ошибок.
- Если клиент предполагает потреблять большие объемы данных, разделение данных на фрагменты и их асинхронная загрузка в локальный кэш позволят улучшить производительность. Необходимо спланировать, как будет обрабатываться несогласованность локальных копий и исходных данных. Для этого могут использоваться временные метки или события.
- В сценариях без постоянного подключения отслеживайте подключение и реализуйте механизм диспетчеризации сервисов для поддержки пакетной обработки, чтобы обеспечить пользователям возможность выполнять множество обновлений данных.
- Выработайте стратегию выявления и управления конфликтами параллельной обработки, которые возникают при попытке обновления централизованно хранящихся данных многими пользователями. Рассмотрите возможность применения модели оптимистичной или пессимистичной блокировки.

Более подробно доступ и обработка данных в насыщенных клиентских приложениях рассматривается в разделе «[Вопросы обработки данных](#)» далее в этой главе.

Управление исключениями

Во всех приложениях и сервисах могут возникать ошибки и исключения, поэтому важно реализовать соответствующую стратегию выявления и управления этими ошибками и исключениями. Надежная и хорошо продуманная стратегия управления исключениями может упростить дизайн приложения, повысить безопасность и удобство обслуживания. Также она может способствовать облегчению задачи по созданию приложения и сократить время и затраты на разработку. В насыщенном клиентском приложении, как правило, требуется уведомлять пользователя о возникших проблемах. Кроме того, для всех нетривиальных ошибок UI, таких как сообщения валидации, должно быть предусмотрено протоколирование и формирование исключений, которые могут использоваться операторами и системами мониторинга. Основную сложность при этом обычно представляет сортировка данных журнала или проектирование централизованного приемника данных журнала на сервере, доступ к которому имеют все клиенты. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Выявите ошибки и исключения, которые могут возникать в приложении, и определите, для которых из них требуется только уведомление пользователя. Обычно для таких ошибок, как ошибки валидации, реализуется только локальное уведомление пользователя. Однако такие ошибки, как неоднократная неудачная попытка регистрации или выявление злонамеренных данных, должны протоколироваться, и администраторы должны получать уведомления о них. Все исключения, возникающие в ходе выполнения, и сбои приложения должны протоколироваться, и, если это необходимо, администраторы должны уведомляться о них.
- Выработайте общую стратегию обработки исключений. Она может включать такие действия, как заключение исключений в другие исключения, характерные для приложения или собственные, которые будут содержать дополнительные данные, используемые при разрешении сбоев, или замена исключений для предотвращения раскрытия конфиденциальных данных. Также реализуйте механизм выявления и протоколирования необрабатываемых исключений. В реализации этих задач может быть полезной инфраструктура управления исключениями, такая как разработанная группой patterns & practices Enterprise Library.
- Примите решение о том, как будет реализовано хранение данных исключений, их передача в другие слои приложения в случае необходимости, и как будет выполняться уведомление администраторов. Используйте инструменты или инфраструктуру мониторинга, которая может собирать сведения о событиях с локального компьютера и предоставлять отчет о состоянии приложения администраторам.
- Обеспечьте очистку данных исключений, предоставляемых пользователям, чтобы предотвратить возможность отображения или сохранения в файлах журнала или аудита конфиденциальных данных. В случае необходимости шифруйте данные и передавайте исключения и ошибки на другие уровни приложения по безопасным каналам.
- Перехватывайте только те исключения, которые можете обработать. Например, перехватывайте исключения преобразования данных, которые могут возникать при попытках преобразования значений null. Не используйте исключения для управления бизнес-логикой.

Более подробно обработка исключений рассматривается в главе 17, «[Сквозная функциональность](#)».

Удобство и простота обслуживания

Для всех приложений и компонентов исключительно важно максимально сократить затраты и упростить обслуживание. Необходимо реализовать механизмы, снижающие расходы на обслуживание, например, использовать шаблоны проектирования, обеспечивающие разделение функциональных областей и слабое связывание компонентов. Насыщенные клиентские приложения обычно располагаются на удаленных клиентских компьютерах и, следовательно, их сложнее обслуживать, чем установленные на сервере приложениями. Поэтому необходимо также учесть такие факторы, как развертывание, обновление,

исправление и управление версиями. При проектировании стратегии обслуживания руководствуйтесь следующими рекомендациями:

- Реализуйте подходящий механизм обновления приложения и его компонентов вручную и/или автоматически. Необходимо учесть аспекты управления версиями, чтобы гарантировать непротиворечивость и совместимость версий всех используемых им компонентов.
- Исходя из среды, в которой предполагается использовать приложение, выбирайте соответствующий подход к развертыванию. Например, для открыто доступных приложений может потребоваться программа установки, а для развертывания приложений в закрытой среде можно будет воспользоваться системными инструментами, такими как Microsoft System Center.
- Проектируйте для приложения слабо связанные и по возможности взаимозаменяемые компоненты. Это позволит заменять отдельные компоненты в зависимости от требований, сценариев времени выполнения и требований или предпочтений конкретных пользователей. Также обеспечьте минимальные зависимости между компонентами и слоями, чтобы иметь возможность использовать приложение или отдельные слои и компоненты в различных сценариях.
- Реализуйте для приложения соответствующее протоколирование и аудит. Эти данных будут полезны администраторам и разработчикам при отладке приложения и решении проблем, возникающих во время выполнения.

Более подробно вопросы удобства и простоты обслуживания обсуждаются в главе 16, «[Показатели качества](#)».

Слой представления

Слой представления – это часть приложения, которую видит пользователь и с которой он взаимодействует, поэтому этот слой должен удовлетворять многим требованиям. К ним относятся общие факторы, такие как удобство использования, производительность, дизайн и возможность взаимодействия с пользователем. Неудовлетворительное взаимодействие с пользователем может иметь чрезвычайно негативные последствия для приложения, удачного во всех остальных отношениях. При проектировании приложения важно с самого начала обеспечить привлекательный и интуитивный интерфейс, поскольку на взаимодействие с пользователем оказывают влияние многие аспекты архитектуры приложения. При проектировании представления приложения руководствуйтесь следующими рекомендациями:

- Продумайте, как отделить логику взаимодействия с пользователем от UI, а также от данных, с которыми работает пользователь. Возможно, для этого подойдет шаблон Separated Presentation. Это упростит обновление частей приложения, позволит разработчикам и дизайнерам работать над компонентами независимо друг от друга и улучшит тестируемость.
- Реализуйте гибкие и легко обновляемые стратегии и механизмы управления и навигации. Используйте в приложении популярные шаблоны проектирования, такие

как Command, Publish/Subscribe и Observer, для отделения команд и навигации от компонентов, это позволит улучшить тестируемость.

- Максимально используйте возможности привязки данных, особенно при отображении данных в виде таблиц или множества строк. Это сократит объем необходимого кода и количество ошибок, упростит разработку. Также можно реализовать автоматическую синхронизацию данных в различных представлениях или формах. Используйте двустороннюю привязку, если необходимо обеспечить пользователю возможность обновления данных.
- Продумайте, как будут отображаться документы в интерфейсе в стиле Office либо при отображении содержимого документов или HTML в других элементах UI. Обеспечьте защиту пользователя от недействительного или злонамеренного содержимого, которое может находиться в документах.
- Обеспечьте интернационализацию UI с последующей локализацией для всех географических или культурных сценариев, в которых оно может использоваться. Сюда относятся возможности изменения языка, направления текста и компоновки содержимого на основании конфигурации или автоматического определения культурной принадлежности пользователя. Также обеспечьте необходимые специальные возможности доступа и навигацию.

Вопросы проектирования слоя представления подробно рассматриваются в главе 6, «[Рекомендации по проектированию слоя представления](#)».

Управление состоянием

Управление состоянием занимается вопросами сохранения данных, представляющих состояние компонента, операции или этапа процесса. Данные состояния могут включать пользовательские настройки, конфигурационные данные, данные рабочего процесса, значения бизнес-правил и данные, отображаемые в UI. Приложение должно иметь возможность сохранять эти данные, иметь к ним доступ в случае необходимости и обрабатывать конфликты, перезапуски и изменения состояния подключения. При проектировании стратегии управления состоянием руководствуйтесь следующими рекомендациями:

- Определите, какие данные состояния должно сохранять приложение, а также оцените их размер, частоту изменения и затраты или издержки на обработку восстановления или повторного извлечения этих данных; также выберите механизм управления состоянием, который может обеспечить соответствующую поддержку этому.
- При наличии больших объемов данных состояния храните их локально на диске. Если приложению необходимо, чтобы некоторые данные были доступны при его запуске, используйте механизм постоянного хранения, такой как изолированное хранилище или файл на жестком диске.
- При хранении конфиденциальных данных обеспечьте соответствующий уровень их защиты через использование шифрования и/или цифровых подписей.

- Примите решение о том, насколько детализированными должны быть сохраняемые данные состояния. Например, определите данные состояния, которые относятся ко всем пользователям приложения, и данные, относящиеся только к определенным пользователям или ролям.
-

Рабочий процесс

Некоторые насыщенные клиентские приложения требуют реализации поддержки потока представления или рабочего процесса для обеспечения многошаговых операций или элементов UI в стиле мастера. Эти возможности могут быть реализованы с применением отдельных компонентов или специальных решений, либо можно использовать преимущества такой технологии, как Windows Workflow Foundation (WF). При проектировании рабочего процесса руководствуйтесь следующими рекомендациями:

- Используйте рабочий процесс в бизнес-компонентах для операций, включающих многошаговые или длительные процессы. Реализуйте задачи рабочего процесса и потока представления в отдельных компонентах. Это сокращает количество зависимостей и упрощает замену компонентов в случае изменения требований.
 - Для простых рабочих процессов и потоков представления обычно достаточно собственного кода, основанного на общеизвестных шаблонах, таких Use Case Controller (Контроллер вариантов использования) и ViewFlow (Поток представления). Для более сложных случаев используйте подсистему управления рабочим процессом, такую как WF.
 - Продумайте стратегию перехвата, управления и отображения ошибок в рабочих процессах. Также примите решение об обработке частично завершенных задач и о том, будет ли возможно восстановление и продолжение выполнения задачи после сбоя либо придется перезапускать процесс.
-

Более подробно компоненты рабочего процесса рассматриваются в главе 14, «[Проектирование компонентов рабочего процесса](#)».

Вопросы безопасности

Безопасность охватывает широкий диапазон факторов и имеет решающее значение для всех типов приложений. Проектирование и реализация насыщенных клиентских приложений должны вестись с обязательным учетом вопросов безопасности, а там где эти приложения выступают в роли слоя представления для бизнес-приложений, они должны участвовать в обеспечении безопасности других слоев приложения. Безопасность касается множества аспектов, включая защиту конфиденциальных данных, аутентификацию и авторизацию пользователей, защиту от атак злонамеренного кода и пользователей, аудит и протоколирование событий и действий пользователей. При проектировании стратегии обеспечения безопасности руководствуйтесь следующими рекомендациями:

- Выберите подходящую технологию и подход к аутентификации пользователей, включая поддержку множества пользователей для одного экземпляра насыщенного

клиентского приложения. Необходимо принять решение о том, как и когда регистрировать пользователей, должна ли быть реализована поддержка различных типов пользователей (ролей) с разными правами доступа (например, администраторы и обычные пользователи), и как регистрировать успешные или неудачные попытки входа пользователя в систему. Учитывайте требования по аутентификации в автономном режиме или без подключения, где это необходимо.

- Используйте интегрированную аутентификацию Windows (Windows Integrated Authentication) или интегрированное решение аутентификации, если требуется обеспечить пользователям возможность доступа к множеству приложений по одному удостоверению или с использованием одних и тех же учетных данных. Если нет возможности использовать Windows Integrated Authentication, можно работать с внешним сервисом, предлагающим интегрированную поддержку аутентификации. Если нет возможности использовать внешний сервис, используйте систему на базе сертификатов или создавайте собственное решение для своей организации.
- Учтите требования по валидации пользовательского ввода и вывода из таких источников, как интерфейсы сервисов и других приложений. Возможно, придется создать собственные механизмы проверки или использовать функции валидации применяемой технологии. Среда разработки Microsoft Visual Studio® Windows Forms включает элементы управления валидации. В качестве альтернативы можно использовать инфраструктуру валидации сторонних производителей, такую как Enterprise Library Validation Application Block, которая обеспечивает возможности проверки в UI и бизнес-слое в полном объеме. Независимо от выбранного подхода следует всегда помнить, что валидация данных должна проводиться при любом пересечении ими границ доверия.
- Продумайте, как будет обеспечиваться защита хранящихся в приложении данных, и таких ресурсов, как файлы, кэши и документы, используемые приложением. Шифруйте конфиденциальные данные в случаях, когда есть вероятность их разглашения, и предусмотрите цифровую подпись для предотвращения повреждения или подделки данных. В приложениях с самыми высокими требованиями к безопасности шифруйте часто меняющиеся данные, хранящиеся в памяти. Также не забывайте защищать конфиденциальные данные, передаваемые по сети или каналу связи.
- Продумайте стратегию аудита и протоколирования для приложения и то, какие данные будут включены в эти журналы. Не забывайте защищать конфиденциальные данные в журналах с помощью шифрования. Для наиболее важных данных, уязвимых для повреждения и подделки, предусмотрите цифровые подписи.

Вопросы обработки данных

Серверные приложения могут предоставлять данные приложения через Веб-сервис. Кэширование этих данных на клиенте позволит улучшить производительность и обеспечит возможность работы в автономном режиме. Также насыщенные клиентские приложения могут использовать локальные данные. Данные, используемые насыщенными клиентскими приложениями можно подразделить на две категории:

- **Справочные данные только для чтения.** Это данные, которые меняются нечасто и используются клиентом для справочных целей, такие как, например, каталог продукции. Храните ссылочные данные на клиенте, чтобы сократить объемы данных, которыми обмениваются клиент и сервер и, таким образом, повысить производительность приложения, обеспечить возможности работы в автономном режиме, обеспечить раннюю проверку данных и повысить удобство использования приложения в общем.
- **Изменяющиеся данные.** Это данные, которые могут изменяться как на клиенте, так и на сервере. Сложнее всего обеспечивать параллельную обработку таких данных, когда одновременно многие клиенты могут вносить изменения в одни и те же данные. Необходимо отслеживать все изменения данных, выполняемые клиентами, и обрабатывать обновления на сервере, чтобы не допустить возможных конфликтов.

Кэширование данных

Насыщенные клиенты часто кэшируют локально все данные: и справочные данные только для чтения, и изменяющиеся. Кэширование может улучшить производительность приложения и обеспечить данные, необходимые для работы в автономном режиме. Для обеспечения кэширования насыщенные клиентские приложения должны реализовать некоторую инфраструктуру кэширования, которая может прозрачно реализовывать механизмы кэширования. Самыми распространенными типами кэширования являются:

- **Кратковременное кэширование данных.** Данные хранятся не постоянно, поэтому приложение не может выполняться в автономном режиме.
- **Долгосрочное кэширование данных.** Кэширование данных в постоянном хранилище, таком как изолированное хранилище или локальная файловая система, позволяет приложению работать без подключения к серверу. Насыщенные клиентские приложения должны различать данные, которые уже были успешно синхронизированы на сервере и еще не синхронизированные данные.

Параллельная обработка данных

При одновременном обслуживании множества клиентов возможна ситуация, когда данные на сервере изменяются до того, как выполняется синхронизация изменений, внесенных другим клиентом. Это может приводить к нарушению целостности или несогласованности данных. Поэтому необходимо реализовать механизм, который гарантировал бы соответствующую обработку любых конфликтов данных при их синхронизации, а также непротиворечивость и правильность результирующих данных. Самыми распространенными подходами к реализации параллельной обработки данных являются:

- **Пессимистическая блокировка.** Пессимистическая блокировка используется в случаях, когда риск конфликтов данных велик. Для предупреждения таких конфликтов данный тип блокировки позволяет одному клиенту блокировать данные, предотвращая, таким образом, доступ или изменение этих данных всеми остальными клиентами до тех пор,

пока изменения, вносимые этим клиентом, не будут завершены и подтверждены. Такая схема также известна как пессимистическая блокировка в автономном режиме.

- **Оптимистическая блокировка.** Оптимистическая блокировка используется в случаях, когда риск конфликтов данных невелик. При этом типе блокировки клиент не блокирует данные при обновлении. Для выявления конфликтов данных на сервер передаются исходные данные, и измененные. На сервере исходные данные сравниваются с текущими данными, чтобы выявить, не были ли они обновлены с момента последнего извлечения. Если нет, изменения принимаются; в противном случае, формируется исключение конфликта данных. Такая схема также известна как оптимистическая блокировка в автономном режиме.

ADO.NET **DataSet** обеспечивает клиентам работу с данными в автономном режиме. **DataSets** может отслеживать локальные изменения данных, что упрощает синхронизацию данных с сервером и разрешение конфликтов данных. **DataSets** также может использоваться для объединения данных из разных источников.

Привязка данных

Механизмы привязки данных Windows Forms, WPF и Silverlight поддерживают двустороннюю привязку, что позволяет связывать структуру данных с компонентом UI, отображать текущие значения данных пользователю, обеспечивает возможность пользователю редактировать данные и затем автоматически обновлять их, используя введенные пользователем значения. Привязка данных может применяться для отображения пользователю данных только для чтения, позволяет пользователям обновлять данные в UI и просматривать сложные элементы связанных данных, обеспечивает представления типа мастер/детали и отображение данных из связанных таблиц, благодаря чему в UI могут отображаться понятные пользователю имена элементов данных, а не значения ключей строки данных.

Проектированию слоя доступа к данным посвящена глава 8, «[Рекомендации по проектированию слоя доступа к данным](#)». Более подробно проектирование компонентов слоя доступа к данным для насыщенных клиентских приложений рассматривается в главе 15, «[Проектирование компонентов слоя доступа к данным](#)».

Вопросы автономной работы/работы без постоянного подключения

Считается, что приложение не имеет постоянного подключения, если оно не имеет возможности регулярно взаимодействовать с сервисами или данными по сети. Насыщенные клиентские приложения без постоянного подключения могут выполнять необходимую обработку без подключения к сетевому ресурсу и обновлять сетевые ресурсы в фоновом режиме при установлении подключения.

При проектировании приложений без постоянного подключения отдавайте предпочтение асинхронному взаимодействию и максимально сокращайте или устраняйте сложные взаимодействия с сетевыми данными и сервисами. Это упростит реализацию механизма синхронизации, используемого, когда подключение доступно.

Чтобы иметь возможность работы без подключения, приложение должно реализовывать возможности кэширования данных, которые обеспечивали бы на клиенте все необходимые для продолжения работы в автономном режиме данные. Также требуется предусмотреть механизмы, предотвращающие использование устаревших данных. В общем, необходимо спроектировать механизм передачи данных с промежуточным хранением, при котором сообщения создаются, хранятся в период отсутствия подключения и, в конце концов, отправляются в место назначения при установлении подключения. Самой распространенной реализацией механизма передачи данных с промежуточным хранением является очередь сообщений.

При проектировании сценария без постоянного подключения возможны два подхода:

- **Ориентированный на данные.** При использовании стратегии с ориентированием на данные для приложения локально на клиенте устанавливается система управления реляционной базой данных (relational database management system, RDBMS), и применяются ее встроенные возможности для переноса локальных изменений данных на сервер, выполнения процесса синхронизации, выявления и разрешения любых конфликтов данных.
- **Сервисно-ориентированный.** Приложения, использующие сервисно-ориентированный подход, хранят данные в сообщениях и, если клиент находится в автономном режиме, организуют эти сообщения в очереди. Когда подключение восстанавливается, сообщения, находящиеся в очереди, передаются на сервер для обработки.

Вопросы выбора технологий

Насыщенные клиентские приложения могут быть реализованы с применением нескольких разных технологий. Следующие рекомендации помогут выбрать соответствующую технологию реализации и применить соответствующие шаблоны и системные функции для конфигурации и мониторинга:

- **Выберите подходящую технологию разработки:**
 - Используйте WFP для приложений, которым необходима полная поддержка насыщенных мультимедиа и графики.
 - Используйте Windows Forms, если уже имеете опыт работы с Windows Forms, либо создаете LOB-приложения, которые не требуют насыщенной визуализации и должны выполняться с минимальными требованиями к оборудованию.
 - Используйте XBAP для приложений, которые будут загружаться с Веб-сервера и затем выполняться в браузере.
 - Используйте ОВА для, преимущественно, основанных на документах приложений, либо приложений, которые будут использоваться для составления отчетов.
- **Ознакомьтесь с предложениями patterns & practices, которые могут быть полезны при проектировании и реализации приложения:**

- Используйте Smart Client Software Factory, если решили работать с Windows Forms и проектируете составные интерфейсы.
 - Используйте Composite Client Application Guidance⁵⁵, если решили работать с WPF и/или Silverlight и желаете создавать модульные приложения, которые обычно включают множество окон, насыщенные гибкие сценарии взаимодействия с пользователем и визуализации данных, а также определяемое ролью поведение.
 - Используйте Enterprise Library для реализации решений сквозной функциональности, таких как обработка исключений, кэширование и валидация.
- **Если решено использовать WPF:**
 - Реализуйте шаблон Presentation Model или View Model, чтобы обеспечить возможность модульного тестирования логики UI и упростить изменение стилевого оформления приложения.
 - WPF позволяет подключать дополнительное поведение к существующим реализациям элементов управления. Этот подход является более предпочтительным, чем создание подклассов для элементов управления.
 - **Если требуется обеспечить поддержку удаленного администрирования и мониторинга:**
 - Реализуйте переопределения групповой политики (Group Policy) для конфигурации приложения. Это необходимо для обеспечения соответствия требованиям программы Certified for Windows (Сертифицировано для Windows).
 - Используйте такие технологии, как SNMP и WMI, чтобы обеспечить предоставление сведений об исключениях и состоянии работоспособности.

Вопросы развертывания

Существует несколько вариантов развертывания насыщенных клиентских приложений. Можно создать самодостаточное приложение, для которого вся логика приложения, включая данные, развертывается на клиентском компьютере. Другой вариант – схема клиент/сервер, при которой логика приложения развертывается на клиенте, а данные – на уровне базы данных. Наконец, существует несколько п-уровневых вариантов развертывания, при которых логика приложения размещается на одном или нескольких серверах приложений.

Самодостаточное приложение

Рис. 2 иллюстрирует схему развертывания самодостаточного приложения, когда вся логика приложения и данные размещаются на клиенте.

⁵⁵ Também conhecido como Prism (princípio de editor científico)



Рис. 34

Развертывание насыщенного клиентского приложения как самодостаточного приложения

Развертывание по схеме клиент/сервер

При развертывании по схеме клиент/сервер вся логика приложения располагается на клиенте, и данные размещаются на сервере базы данных, как показано на рис. 3.

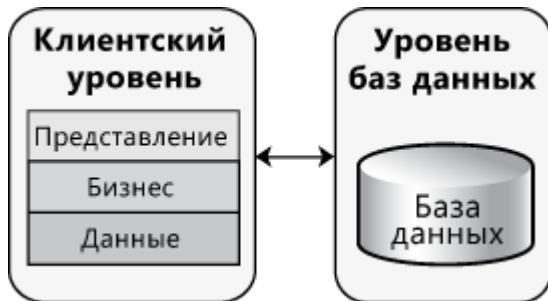


Рис. 35

Развертывание насыщенного клиентского приложения по схеме клиент/сервер

N-уровневое развертывание

При n -уровневом развертывании на клиенте могут размещаться логика представления и бизнес-логика или только логика представления. Рис. 4 иллюстрирует первый вариант.

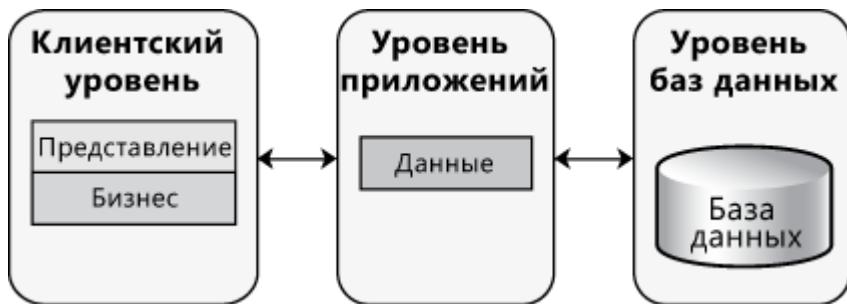


Рис. 36

N-уровневое развертывание, при котором бизнес-слой размещается на клиентском уровне

Рис. 5 иллюстрирует вариант, когда бизнес-логика и логика доступа к данным располагаются на сервере приложений.

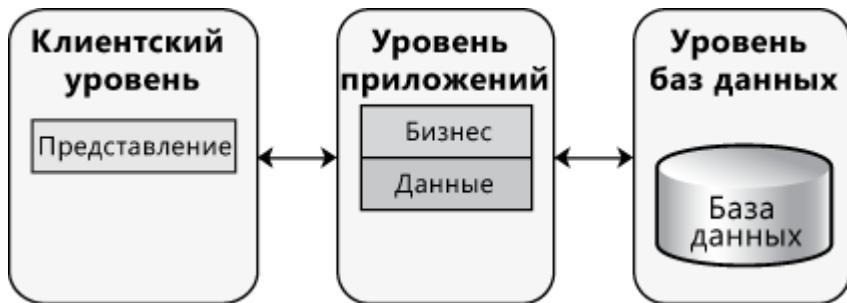


Рис. 37

N- уровневое развертывание, при котором бизнес-слой размещается на уровне приложений

Более подробно стратегии развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

Технологии развертывания

Существует несколько вариантов физического развертывания насыщенного клиентского приложения, каждый из которых обладает определенными преимуществами и недостатками. Необходимо внимательно рассмотреть все варианты, чтобы сделать правильный выбор, удовлетворяющий требованиям целевых инфраструктур, в которых предполагается выполнять приложение. Предлагаются такие варианты развертывания:

- **Развертывание ClickOnce.** Этот подход требует минимального участия пользователя, небольших усилий для разработчика и обеспечивает автоматизированное обновление. Тем не менее, он может использоваться для развертывания только отдельного решения, не являющегося частью большего решения. Этот подход нельзя применять для развертывания дополнительных файлов или разделов реестра; он не обеспечивает пользователю возможность конфигурации установки; и не позволяет создавать фирменный стиль установщика.
- **Развертывание XCOPY.** Если нет необходимости в установке параметров реестра или регистрации компонентов, исполняемый файл можно просто копировать на жесткий диск клиентского компьютера.
- **Пакет Установщик Windows (.MSI).** Это полная программа настройки, которая может устанавливать компоненты, ресурсы, параметры реестра и другие артефакты, необходимые приложению. Для установки самих пакетов MSI пользователь должен иметь права доступа администратора. Существуют решения для распространения приложений в среде предприятия, такие как Microsoft System Center Configuration Manager.
- **Пакет XWAR.** Приложение загружается через браузер и выполняется в ограниченной безопасной среде на компьютере. Обновления могут передаваться на клиент автоматически.

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Сетевое взаимодействие, Композиция, Управление конфигурацией, Управление исключениями,

Представление, Управление состоянием и Рабочий процесс, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Сетевое взаимодействие	<p>Asynchronous Callback. Реализует длительные задачи в отдельном потоке, выполняющемся в фоновом режиме, и обеспечивает потоку функцию для обратного вызова по завершении выполнения задачи.</p> <p>Gateway (Шлюз). Обеспечивает доступ к внешней системе через общий абстрактный интерфейс взаимодействия так, что потребителям нет необходимости понимать внешний интерфейс системы.</p> <p>Service Locator (Локатор сервиса). Централизует поиск распределенных объектов сервисов, обеспечивает централизованную точку управления и выступает в роли кэша, благодаря которому сокращается количество лишних операций поиска.</p> <p>Service Agent and Proxy (Агент и прокси сервиса). Реализует компонент, который может использоваться приложением для доступа к компоненту/сервису, и при этом приложение даже не будет знать, что обращается не к фактическому компоненту/сервису. Этот компонент передает вызовы приложения в удаленный компонент/сервис и возвращает результат приложению. Прокси абстрагирует детали взаимодействия с другими удаленными компонентами, как правило, при использовании сервисов ASMX или WCF.</p> <p>Service Interface. Программный интерфейс, который может использоваться другими системами для взаимодействия с сервисом.</p>
Композиция	<p>Composite View. Сочетает отдельные представления в композитное представление.</p> <p>Template View. Реализует представление общего шаблона и создает представления на базе этого шаблонного представления.</p> <p>Two-Step View. Преобразует модель данных в логическое представление без какого-либо специального форматирования и затем преобразует это логическое представление, добавляя необходимое форматирование.</p> <p>View Helper (Вспомогательный объект представления). Делегирует ответственность за обработку бизнес-данных вспомогательным классам.</p>
Управление конфигурацией	Provider. Реализует компонент, который предоставляет API, отличающийся от клиентского API, обеспечивая возможность подключения любой специальной реализации.
Управление исключениями	Exception Shielding. При возникновении исключения предотвращает предоставление сервисом данных о его внутренней реализации.
Представление	<p>Application Controller. Объект, содержащий всю логику приложения и используемый другими Контроллерами, работающими с Моделью и отображающими соответствующее Представление.</p> <p>Model-View-Presenter. Разделяет обработку запроса на три отдельные роли, где Представление отвечает за обработку пользовательского ввода, Модель – за данные приложения и бизнес-логику, и Презентатора – за логику представления и координирование взаимодействия между Представлением и Моделью.</p> <p>Model-View-ViewModel. Разновидность шаблона Model-View-Controller</p>

	(MVC), приспособленная для современных платформ разработки UI, на которых созданием представления (View) занимаются, главным образом, дизайнеры, а не обычные разработчики Presentation Model (Модель презентации). Распределяет обязанности по визуальному отображению пользовательского интерфейса, реализации состояния представления и поведения между разными классами, которые названы представление и модель презентации, соответственно. Класс представления управляет элементами управления пользовательского интерфейса и инкапсулирует визуальное состояние или поведение, характерное для UI. Класс модели презентации инкапсулирует поведение представления и выступает в роли фасада для базовой модели.
Управление состоянием	Context Object (Объект контекста). Объект, используемый для управления текущим контекстом обработки.
Рабочий процесс	View Flow (Поток представления). Управляет переходами от одного представления к другому на основании состояния приложения или среды, а также условий и ограничений, необходимых для правильной работы приложений. Work Flow (Рабочий процесс). Управляет предопределенным потоком управления в сложном процессно-ориентированном приложении, обеспечивая при этом возможность динамического изменения хода выполнения через решения и структуры ветвления, которые могут изменять маршрутизацию запросов.

Более подробно шаблоны Template View, Transform View и Two-Step View рассматриваются в книге Мартина Фаулера «Архитектура корпоративных приложений». Addison-Wesley, 2002. Или по адресу <http://martinfowler.com/eaaCatalog>.

Более подробно шаблон Provider рассматривается в материале «Provider Model Design Pattern and Specification, Part I» по адресу <http://msdn.microsoft.com/en-us/library/ms998421.aspx>.

Более подробно шаблон Asynchronous Callback рассматривается в статье «Creating a Simplified Asynchronous Call Pattern for Windows Forms Applications» по адресу <http://msdn.microsoft.com/en-us/library/ms996483.aspx>.

Более подробно шаблон Service Interface рассматривается в статье «Service Interface» по адресу <http://msdn.microsoft.com/en-us/library/ms998421.aspx>.

Более подробно шаблон Exception Shielding рассматривается в статье «Useful Patterns for Services» по адресу <http://msdn.microsoft.com/en-us/library/cc304800.aspx>.

Более подробно шаблон Composite View рассматривается в статье «Patterns in the Composite Application Library» по адресу <http://msdn.microsoft.com/en-us/library/dd458924.aspx>.

Более подробно шаблон Presentation Model рассматривается в статье «Presentation Model» (Модель презентации) по адресу <http://msdn.microsoft.com/en-us/library/dd458863.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- Проектированию составных приложений посвящено руководство «*Composite Client Application Guidance*» по адресу <http://msdn.microsoft.com/en-us/library/cc707819.aspx>.
- Более подробно проектирование насыщенных клиентских приложений и клиентских приложений с расширенной логикой рассматривается в руководстве «*Smart Client Architecture and Design Guide*» (Руководство по архитектуре и дизайну клиентских приложений с расширенной логикой) по адресу <http://msdn.microsoft.com/en-us/library/ms998506.aspx>.
- Архитектуры кэширования обсуждаются в статье «*Caching Architecture Guide for .NET Framework Applications*» по адресу <http://msdn.microsoft.com/en-us/library/ms978498.aspx>.
- Сценарии и вопросы развертывания подробно рассматриваются в статье «*Deploying .NET Framework-based Applications*» по адресу <http://msdn.microsoft.com/en-us/library/ms954585.aspx>.

23

Проектирование насыщенных Интернет-приложений

Обзор

В данной главе рассматриваются основные сценарии использования Насыщенных Интернет-приложений (Rich Internet Applications, RIA), их компоненты и важные аспекты проектирования. Даются рекомендации по обеспечению производительности, безопасности и развертыванию, а также основные шаблоны и технологии проектирования RIA.

RIA поддерживают насыщенные графические элементы и сценарии с применением потокового мультимедиа, обеспечивая при этом преимущества развертывания и удобства обслуживания, присущие Веб-приложению. RIA могут выполняться в подключаемом модуле браузера, таком как Microsoft® Silverlight®, в отличие от расширений, использующих код браузера, таких как Asynchronous JavaScript и XML (AJAX). Типовая реализация RIA использует Веб-инфраструктуру в сочетании с клиентским приложением, отвечающим за обработку представления.

Подключаемый модуль обеспечивает библиотечные процедуры для поддержки насыщенной графики, а также контейнер, в целях безопасности ограничивающий доступ к локальным ресурсам. RIA могут выполнять более расширенный и сложный код на стороне клиента, чем обычное Веб-приложение, обеспечивая тем самым возможность сократить нагрузку на Веб-сервер. На рис. 1 представлена типовая структура реализации RIA.

Насыщенное интернет приложение

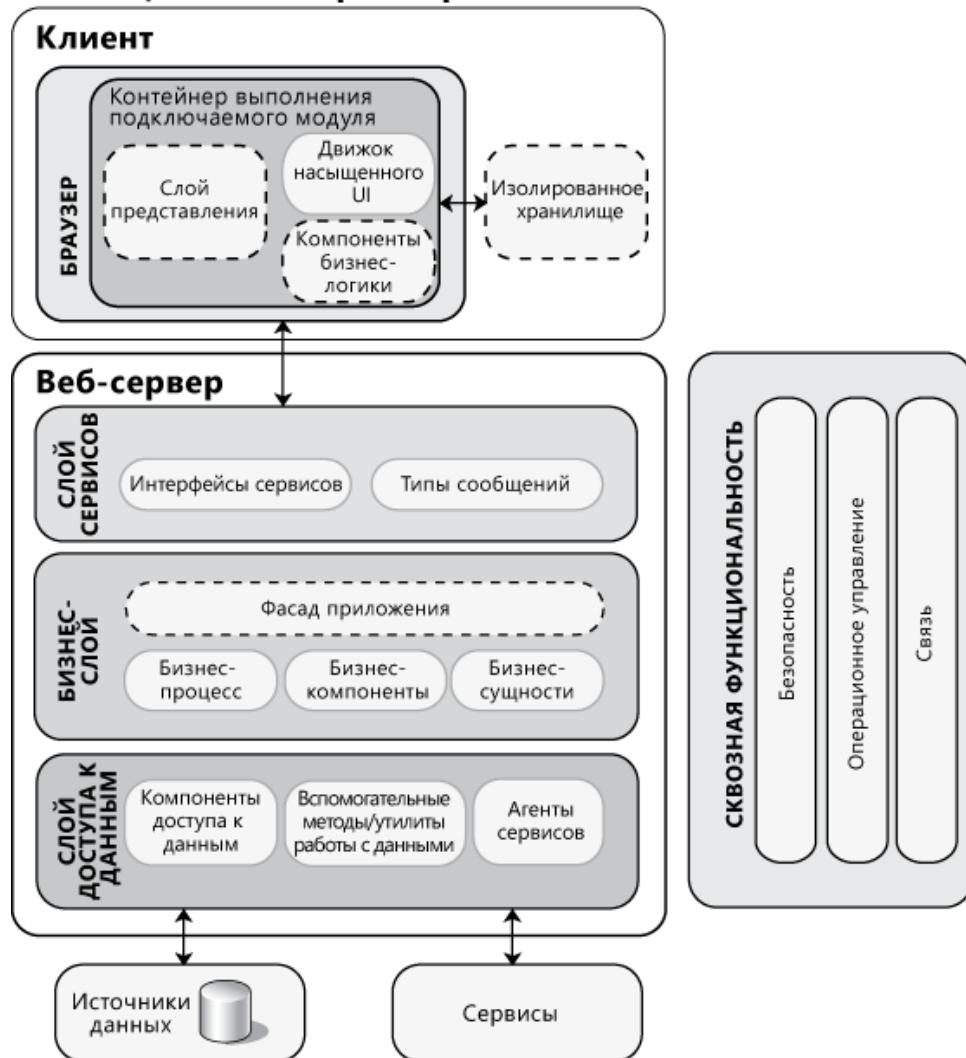


Рис. 38

Архитектура типовой реализации RIA. Пунктиром обозначены необязательные компоненты.

Типовое насыщенное Интернет-приложение включает три слоя: слой представления, бизнес-слой и слой доступа к данным. Слой представления, как правило, содержит компоненты UI и логики представления; бизнес-слой – компоненты бизнес-логики, бизнес-процесса и бизнес-сущностей; и слой доступа к данным – компоненты доступа к данным и агентов сервисов.

В RIA часть бизнес-процессов и даже код доступа к данным часто переносятся на клиент. Поэтому клиентская часть, в зависимости от сценария, может включать некоторую или всю функциональность бизнес-слоя и слоя доступа к данным. На рис. 1 показано, как некоторые бизнес-процессы обычно реализовываются на клиенте.

RIA могут быть как тонкими интерфейсами для серверных бизнес-сервисов, так и сложными приложениями, которые самостоятельно выполняют большую часть процессов и взаимодействуют с сервисами на сервере только для получения или отправки данных.

Следовательно, дизайн и реализация RIA могут быть очень разнообразными. Тем не менее,

существует ряд общих подходов, обеспечивающих создание хорошей архитектуры слоя представления и его взаимодействия с сервисами на сервере. Многие из них основываются на общезвестных шаблонах проектирования, которые способствуют построению приложения из отдельных компонентов. Это обеспечивает сокращение зависимостей, упрощение обслуживания и тестирования, а также более широкие возможности повторного использования.

Многослойному дизайну посвящена глава 5, «[Рекомендации по проектированию многослойных приложений](#)». Компоненты, используемые в каждом слое, более подробно рассматриваются в главе 10, «[Рекомендации по проектированию компонентов](#)».

Общие принципы проектирования

Представленные далее рекомендации касаются нескольких аспектов, которые должны быть учтены при проектировании RIA, и помогут обеспечить эффективную работу приложения в общих для RIA сценариях и соответствие выдвигаемым требованиям:

- **Выбирайте RIA, исходя из предполагаемой аудитории, насыщенного интерфейса и простоты развертывания.** Создавайте RIA, если целевая аудитория использует браузер, поддерживающий RIA. Если часть целевой аудитории работает с браузером, не поддерживающим RIA, рассмотрите возможность ограничения номенклатуры доступных для выбора браузеров только поддерживаемой версией. Если влиять на выбор браузера нет возможности, подумайте, достаточно ли велика неохватываемая аудитория, чтобы выбирать другой тип приложения, такой как Веб-приложение, использующее AJAX. Если клиенты имеют надежное сетевое соединение, развертывать и обслуживать RIA настолько же просто, как и Веб-приложение. Реализации RIA прекрасно подходят для Веб-сценариев, в которых требования, предъявляемые к визуализации, превышают возможности, обеспечиваемые базовым HTML. RIA обеспечивают более устойчивое поведение и требуют менее развернутого тестирования в поддерживаемых браузерах по сравнению с Веб-приложениями, использующими расширенные функции и код. Реализации RIA также идеально подходят для приложений, работающих с потоковыми мультимедиа. Они не так хороши для сверхсложных многостраничных UI.
- **Используйте Веб-инфраструктуру посредством сервисов.** Реализациям RIA необходима такая же инфраструктура, что и Веб-приложениям. Как правило, вся обработка RIA и также взаимодействие с другими сетевыми сервисами (для сохранения данных в базе данных, например) выполняются на клиенте.
- **Используйте вычислительные мощности клиента.** RIA выполняются на клиентском компьютере и могут использовать все доступные на нем вычислительные мощности. По возможности максимально перенесите функциональность на клиент, это позволит улучшить взаимодействие с пользователем. Однако особо важные бизнес-правила должны выполняться на сервере, поскольку проверка на клиенте может быть сфальсифицирована.

- **Обеспечивайте выполнение в безопасной программной среде браузера.** Реализации RIA обладают более высокой безопасностью по умолчанию, поэтому не имеют доступа ко всем ресурсам компьютера, таким как камеры и аппаратное ускорение видео. Доступ к локальной файловой системе ограничен. Локальное хранилище доступно, но с максимальными ограничениями.
- **Определитесь с тем, насколько сложным будет UI.** Продумайте, насколько сложным будет создаваемый UI. Реализации RIA лучше работают при использовании одного окна для всех операций. Возможно применение множества окон, но это требует написания дополнительного кода и обработки переходов между окнами. Пользователи должны иметь возможность без труда перемещаться или останавливаться и возвращаться к соответствующему окну без повторного запуска всего процесса. Для многостраничных UI применяйте методы внешнего связывания. Также управляйте унифицированным указателем ресурса (Uniform Resource Locator, URL), историей и кнопками назад/вперед браузера во избежание сложностей при переходе пользователей между окнами.
- **Используйте сценарии для повышения производительности или сокращения времени отклика приложения.** Проанализируйте общие сценарии приложений и примите решение о разделении и загрузке компонентов приложения, а также о кэшировании данных или переносе бизнес-логики на клиент. Чтобы сократить время загрузки и запуска приложения, разбивайте функциональность на отдельные загружаемые компоненты.
- **Предусмотрите ситуацию отсутствия установленного подключаемого модуля.** Реализации RIA требуют наличия подключаемого модуля браузера, поэтому необходимо предусмотреть сценарий установки подключаемого модуля как часть выполнения приложения. Учтите то, имеют ли ваши клиенты доступ к подключаемому модулю, разрешение на его установку, и захотят ли они его устанавливать. Подумайте над тем, насколько детально будете контролировать процесс установки. Предусмотрите ситуацию, когда пользователи не могут установить подключаемый модуль, с выводом на экран информативного сообщения об ошибке или предоставлением альтернативного пользовательского Веб-интерфейса.

Кроме приведенных выше рекомендаций по реализаций RIA, руководствуйтесь и более общими советами по проектированию насыщенных клиентских приложений (включая мобильные насыщенные клиенты). К общим рекомендациям относятся отделение логики представления от реализации интерфейса, определение задач представления и потоков представления, разделение бизнес-правил и других задач, не связанных с интерфейсом, повторное использование общей логики представления, слабое связывание клиента с используемыми им удаленными сервисами, недопущение тесного связывания с объектами других слоев и сокращение числа обращений к сети при доступе к удаленным слоям. Более подробно эти вопросы рассматриваются в главе 22, «[Проектирование насыщенных клиентских приложений](#)».

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Бизнес-слой](#)
 - [Кэширование](#)
 - [Сетевое взаимодействие](#)
 - [Композиция](#)
 - [Доступ к данным](#)
 - [Управление исключениями](#)
 - [Протоколирование](#)
 - [Мультимедиа и графика](#)
 - [Приложения для мобильных устройств](#)
 - [Портируемость](#)
 - [Представление](#)
 - [Управление состоянием](#)
 - [Валидация](#)
-

Бизнес-слой

В большинстве сценариев RIA-приложения выполняют доступ к данным или сведениям, расположенным вне приложения. Природа этих данных может быть различной, но, по всей вероятности, извлекаться они будут из бизнес-системы. Для обеспечения максимальной производительности и удобства использования часть бизнес-процессов рекомендуется размещать на клиенте. При проектировании взаимодействия бизнес-слоя и слоя сервисов руководствуйтесь следующими рекомендациями:

- Определите, какие бизнес-слои и интерфейсы сервисов будет использовать приложение. Бизнес-слой, располагающийся на клиенте, должен взаимодействовать с интерфейсами сервисов посредством агента сервиса. Агенты сервисов, как правило, можно реализовать путем формирования прокси, используя описание сервиса.
- Если бизнес-логика не работает с конфиденциальными данными, можно разместить некоторые бизнес-правила на клиенте, что улучшит производительность и время отклика приложения. Если все-таки конфиденциальные данные используются, бизнес-логика должна располагаться на сервере приложений.
- Продумайте, как клиент будет получать данные, необходимые для выполнения бизнес-правил и других процессов на стороне клиента, и как будет осуществляться

автоматическое обновление бизнес-правил при изменении требований. Возможно, необходимо будет реализовать механизм, который позволит клиенту при запуске получать сведения о бизнес-правилах с бизнес-слоя.

- Если реализация RIA допускает возможность создания экземпляра без UI, ее можно использовать для реализации бизнес-процессов с применением более структурированных, мощных или знакомых языков программирования (таких как C#), вместо менее гибких поддерживаемых браузерами языков.
- Если бизнес-логика дублируется на клиенте и на сервере, используйте для ее реализации один язык программирования и на клиенте, и на сервере (если реализация RIA допускает это). Это сократит различия между реализациями и обеспечит единообразие при обработке правил. Модели предметной области, которые могут существовать и на стороне сервера, и на стороне клиента, должны быть максимально подобными.
- По соображениям безопасности не размещайте особо важную незашифрованную бизнес-логику на клиенте. Код в загруженных XAP-файлах может быть без труда декомпилирован. Реализуйте важную бизнес-логику на сервере и выполняйте доступ к ней с помощью Веб-сервисов.

Более подробно реализация бизнес-слоя рассматривается в главе 7, «[Рекомендации по проектированию бизнес-слоя](#)».

Кэширование

Как правило, реализации RIA используют обычный механизм кэширования браузера. Разумный подход к кэшированию ресурсов обеспечит повышение производительности приложения. При проектировании стратегии кэширования руководствуйтесь следующими рекомендациями:

- Разделение больших клиентских приложений на меньшие отдельно загружаемые компоненты, обеспечивает возможность кэширования этих компонентов для улучшения производительности и сокращения числа обращений к сети. По возможности избегайте загрузки всего приложения и создания экземпляров всех элементов при запуске. Используйте установку, обновления и пользовательские сценарии для разделения приложения и его модульной загрузки. Например, загружайте при запуске только заглушки и затем динамически догружайте дополнительную функциональность в фоновом режиме. С помощью событий обеспечьте загрузку модулей непосредственно перед тем, как они могут понадобиться.
- Позвольте браузеру кэшировать объекты, изменение которых в течение сеанса маловероятно. Используйте специальное локальное хранилище RIA для данных, изменяющихся в ходе сеанса или данных, которые должны сохраняться между сеансами.

- Во избежание непредвиденных исключений обеспечьте достаточно большое изолированное хранилище данных. Объем хранилища не увеличивается автоматически; это может сделать пользователь по запросу приложения.
-

Более подробно проектирование стратегии кэширования рассматривается в главе 17, «[Сквозная функциональность](#)».

Сетевое взаимодействие

Реализации RIA должны использовать асинхронную модель вызовов для сервисов, это позволит избежать блокировок процессов браузера. Вопросы междоменного взаимодействия, используемых протоколов и эффективности сервисов должны быть учтены при проектировании. Если реализация RIA допускает это, используйте отдельный поток для операций, выполняющихся в фоновом режиме. При проектировании стратегии сетевого взаимодействия руководствуйтесь следующими рекомендациями:

- При наличии продолжительных операций используйте фоновый поток или асинхронное выполнение, это позволит избежать блокировки потока UI.
 - Обеспечьте, чтобы RIA и вызываемые им сервисы использовали совместимые привязки, включающие данные системы безопасности. При аутентификации через сервисы проектируйте сервисы так, чтобы они использовали привязки, поддерживаемые реализацией RIA.
 - Для защиты каналов связи используйте безопасные Интернет-протоколы (Internet Protocol Security, IPSec) и протокол безопасных соединений (Secure Sockets Layer, SSL), для защиты данных – шифрование, и цифровые подписи для выявления повреждения или подделки данных.
 - Для обеспечения доступа RIA-клиента к другим серверам/доменам, не тем, с которых он был загружен, используйте механизм междоменной конфигурации.
 - Используйте механизм двусторонней передачи с Windows Communication Foundation (WCF) для выталкивания данных на клиент, если опрос клиента обуславливает большую нагрузку на сервер, или для выталкивания данных на сервер, если такой вариант намного эффективнее, чем использование сервисов, например, как в игровых сценариях для нескольких участников в режиме реального времени с использованием центрального сервера. Однако не забывайте, что межсетевые экраны и маршрутизаторы могут блокировать некоторые порты и протоколы. Более подробную информацию по этим вопросам можно найти в материалах, приведенных в разделе «[Дополнительные источники](#)» в конце этой главы.
-

Более подробно проектирование сервисов рассматривается в главе 25, «[Проектирование сервисных приложений](#)». Протоколы и механизмы связи обсуждаются в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Композиция

Композиция позволяет создавать более удобные в обслуживании и легко расширяемые приложения, которые не требуют повторной реализации или повторного развертывания всего приложения. Приложение может быть скомпоновано из множества модулей, включающих слабо связанные компоненты. Это позволит расширять приложение через развертывание новых модулей или новых версий модулей; упростит настройку или персонализацию приложения пользователем, либо упростит настройку приложения под определенную роль пользователя или задачу. При проектировании стратегии композиции руководствуйтесь следующими рекомендациями:

- Продумайте, подходит ли композиция для вашего сценария, и, если подходит, выберите наиболее подходящие шаблоны модели композиции. Композиция поможет создать приложения, которые смогут повторно использоваться в разных сценариях с минимальными или вообще без изменений. Однако избегайте дизайнов, вводящих зависимости, что потребует частого повторного развертывания приложения.
- Композиция хорошо подходит для составных приложений, интегрирующих данные и функциональность несопоставимых источников, или в ситуациях, когда необходимо обеспечить пользователю возможность расширять или настраивать приложение.

Доступ к данным

Реализации RIA запрашивают данные с Веб-сервера через сервисы так же, как AJAX-клиент. На клиенте для повышения производительности данные могут быть кэшированы. При проектировании стратегии доступа к данным руководствуйтесь следующими рекомендациями:

- Используйте кэширование на стороне клиента, чтобы сократить число обращений к серверу и обеспечить UI с меньшим временем отклика.
- Фильтрация данных на сервере, а не на клиенте, позволит сократить объем передаваемых по сети данных.

Более подробно проектирование слоя доступа к данным рассматривается в главе 8, «[Рекомендации по проектированию слоя доступа к данным](#)».

Управление исключениями

Надежная и хорошо продуманная стратегия управления исключениями может упростить дизайн приложения и повысить безопасность и удобство обслуживания, а также упростить разработчикам задачу по созданию приложения и снизить время и стоимость разработки. В RIA обычно требуется уведомлять пользователя о возникшей ошибке. Кроме того, для всех нетривиальных ошибок, таких как сообщения валидации, необходимо предусмотреть протоколирование и формирование исключений на сервере для использования операторами и системами мониторинга. Кроме того, реализуйте асинхронные исключения, а также

координирование исключений между кодом клиента и сервера. При проектировании механизма управления исключениями руководствуйтесь следующими рекомендациями:

- Проектируйте как синхронные, так и асинхронные исключения. Используйте блоки `try/catch` для перехватывания исключений в синхронном коде. Обработку исключений для асинхронных вызовов методов выносите в отдельный обработчик, специально предусмотренный для таких исключений; например, в Silverlight таким обработчиком является `OnError`.
- Выработайте стратегию кэширования и обработки необрабатываемых исключений. Необрабатываемые исключения в RIA передаются в браузер. Это позволяет продолжать выполнение после того, как пользователь закроет сообщение об ошибке браузера. По возможности обеспечивайте понятное пользователю сообщение об ошибке. Останавливайте программу, если продолжение выполнения может пагубно сказаться на целостности данных приложения, или если пользователь может ошибочно подумать, что приложение по-прежнему остается в стабильном состоянии.
- Перехватывайте только те внутренние исключения, которые можете обработать. Например, перехватывайте исключения преобразования данных, которые могут возникать при попытке преобразования значений `null`. Не используйте исключения для управления бизнес-логикой.
- Правильно проектируйте стратегию распространения исключений. Например, обеспечьте распространение исключений вверх к граничным слоям, где они могут быть запротоколированы и преобразованы должным образом для передачи на следующий уровень.
- Выработайте соответствующую стратегию протоколирования и уведомления для критических ошибок и исключений, чтобы не допустить разглашения конфиденциальных данных.

Более подробно проектирование стратегии управления исключениями рассматривается в главе 17, «[Сквозная функциональность](#)».

Протоколирование

Реализация протоколирования в целях отладки или аудита может представлять собой довольно сложную задачу в RIA. Например, приложения Silverlight не имеют доступа к клиентской файловой системе, клиентская и серверная части выполняются асинхронно. Обычно для получения полной картины выполнения программы необходимо комбинировать файлы журнала клиента с файлами журнала сервера. При проектировании стратегии протоколирования руководствуйтесь следующими рекомендациями:

- Рассмотрите ограничения компонента протоколирования в реализации RIA. Некоторые реализации RIA выделяют для регистрации данных каждого пользователя отдельный файл, и эти файлы могут храниться в разных местах на диске.

- Выработайте стратегию передачи журналов клиента на сервер для обработки. При диагностике проблем конкретного клиентского компьютера может возникнуть необходимость в объединении журналов разных пользователей этого компьютера. Ведите отдельные журналы для каждого пользователя, а не для компьютера в целом, поскольку на каждом клиентском компьютере могут работать несколько пользователей.
 - При использовании изолированного хранилища для протоколирования предусмотрите предельный размер хранилища и необходимость направления запроса пользователю об увеличении объема хранилища в случае необходимости.
 - Обеспечьте протоколирование критических ошибок и предусмотрите включение протоколирования и передачу журналов на сервер при возникновении исключений.
-

Мультимедиа и графика

Реализации RIA обеспечивают намного более насыщенный интерфейс пользователя и лучшую производительность, чем обычные Веб-приложений. Изучите и примените возможности работы с мультимедиа используемой платформы RIA. Не забывайте, что некоторые возможности платформы RIA могут быть недоступны в автономном проигрывателе мультимедиа. При проектировании возможностей работы с мультимедиа и графикой руководствуйтесь следующими рекомендациями:

- При проектировании обеспечьте воспроизведение потокового мультимедиа и видео в браузере без использования специального проигрывателя. Рекомендуется использовать адаптивную потоковую передачу в сочетании с RIA-клиентами, это позволит корректно и плавно обрабатывать проблемы, возникающие из-за изменяющейся полосы пропускания.
 - Для повышения производительности позиционируйте объекты мультимедиа с привязкой к целочисленным значениям пикселов и представляйте их в исходном размере. Для обеспечения лучшей производительности отрисовки используйте встроенные средства обработки векторной графики.
 - Для особо насыщенного графикой приложения выясните, обеспечивает ли реализация RIA аппаратное ускорение. Если нет, определите минимально допустимую скорость отрисовки. Выработайте план сокращения нагрузки на графический процессор в случае падения скорости ниже допустимого минимума.
 - Принимайте во внимание размер областей отрисовки. Перерисовывайте только те части области, которые на самом деле меняются. Сокращайте области перекрытия, когда в них нет необходимости, для снижения эффекта наложения. Используйте методы профилирования и отладки (например, параметр «**EnableRedrawRegions = true**» в Silverlight) для выявления перерисованных областей. Обратите внимание, что определенные эффекты, такие как размытие, могут обуславливать перерисовку каждого пикселя области. Безоконные прозрачные элементы управления могут также приводить к непредусмотренным перерисовкам и наложению.
-

Приложения для мобильных устройств

Реализации RIA обеспечивают намного более насыщенные пользовательские интерфейсы, чем обычные мобильные приложения. Используйте встроенные возможности работы с мультимедиа той платформы RIA, с которой работаете. При проектировании приложений для работы с мультимедиа и графикой на мобильных устройствах руководствуйтесь следующими рекомендациями:

- Если RIA должно устанавливаться на мобильном клиенте, выясните, доступна ли реализация подключаемого модуля RIA для устройства, которое предполагается поддерживать. Также примите во внимание, что функциональность подключаемого модуля RIA для мобильных устройств может быть ограничена по сравнению с немобильными платформами.
- Страйтесь использовать один или подобный код, а потом уже в случае необходимости создавайте версии кода для конкретных устройств.
- Обеспечьте, чтобы компоновка и реализация UI подходили для небольших экранов мобильных устройств. RIA-приложения работают на мобильных устройствах, но предусмотрите разный код компоновки для каждого типа устройств, это поможет обеспечить корректное отображение UI на экранах разных размеров при проектировании для Windows Mobile.

Более подробно реализация приложений для мобильных устройств рассматривается в главе 24, «[Проектирование мобильных приложений](#)».

Портируемость

Одно из основных преимуществ RIA – портируемость компилированного кода между разными браузерами, операционными системами и платформами. Кроме того, использование одного и того же или подобного исходного кода сокращает время и затраты на разработку и обслуживание, обеспечивая при этом независимость от платформы. При проектировании с обеспечением портируемости руководствуйтесь следующими рекомендациями:

- Полностью используйте встроенные библиотеки кода RIA и проектируйте так, чтобы «однажды написанное выполнялось везде», но будьте готовы создавать различные версии кода в случаях, когда это требуется из-за общей сложности проекта или необходимости принятия компромиссов по обеспечиваемой функциональности.
- Выбирая между RIA и Веб-приложением, не забывайте, что из-за различий браузеров для Веб-приложения потребуется всестороннее тестирование кода ASP.NET и JavaScript. В случае с RIA-приложением вопросами совместимости с различными платформами занимается создатель подключаемого модуля, а не разработчик приложения. Это существенно сокращает затраты на тестирование для каждого сочетания платформы и браузера.
- Если RIA планируется выполнять на множестве платформ, не используйте возможности, доступные только на одной платформе, например, Windows

Integrated Authentication. Основывайте решение на портируемых процедурах и функциях RIA, которые доступны на многих клиентах.

- При создании насыщенных клиентских или RIA-приложений обратите внимание на языки и среды разработки, такие как Composite Client Application Guidance группы patterns & practices, предлагающие решения, подходящие для обеих платформ. Подробнее эти вопросы рассматриваются в статье «*Composite Client Application Guidance*» по адресу <http://msdn.microsoft.com/en-us/library/cc707819.aspx>.

Представление

Поскольку, как правило, RIA-приложения ориентированы на выполнение в браузере, лучше всего они работают при наличии всего одного центрального интерфейса. Для многостраничных приложений требуются механизмы связывания страниц. При проектировании представления руководствуйтесь следующими рекомендациями:

- Используйте шаблоны Separated Presentation для разделения визуального представления от логики представления приложения.
- Максимально используйте преимущества привязки данных для отображения данных, особенно для табличного и многострокового представления данных. Это сократит объем требуемого кода, упростит разработку и уменьшит количество ошибок в коде. Также это может обеспечить автоматическую синхронизацию данных в разных представлениях или формах. Используйте двустороннюю привязку там, где пользователь должен иметь возможность обновлять данные.
- Для многостраничных UI используйте методы внешнего связывания. Это обеспечит уникальную идентификацию и возможность перехода к конкретным страницам приложения.
- Перехватывайте события кнопок вперед и назад браузера. Это позволит избежать нечаянных переходов со страницы. Также рассмотрите возможность управления содержимым адресной строки браузера и списком истории для реализации навигации, как на обычной Веб-странице.

Более подробно реализация слоя представления рассматривается в главе 6, «[Рекомендации по проектированию уровня представления](#)».

Управление состоянием

Состояние приложения может храниться на клиенте в изолированном хранилище. Это удобно для локального сохранения или кэширования состояния между пользовательскими сессиями. Изолированное хранилище управляет не так, как кэш браузера. Приложения, сохраняющие данные в изолированном хранилище, должны либо удалять их напрямую, либо явно указывать пользователю о необходимости удалить эти данные. При проектировании стратегии управления состоянием руководствуйтесь следующими рекомендациями:

- Определите, какие данные состояния должно сохранять приложение, оцените предполагаемый размер необходимого хранилища, частоту изменений, затраты на

обработку или издержки на повторное создание или повторное извлечение данных.

- Чтобы сохранить состояние между сеансами, храните данные состояния на клиенте в изолированном хранилище. Состояние, необходимое для функционирования приложения, всегда должно сохраняться на сервере. Это также обеспечивает пользователям доступ к сохраненному состоянию при входе в систему с другого компьютера.
- При проектировании обеспечьте возможность параллельного выполнения множества сеансов, потому что предотвратить одновременный запуск множества экземпляров невозможно. Обеспечьте в стратегии управления состоянием либо обработку параллелизма, либо механизм, выявляющий и не допускающий нарушение состояния приложения в результате выполнения множества сеансов.

Валидация

Валидация должна осуществляться в коде на клиенте или через сервисы, расположенные на сервере. Если на клиенте требуется нечто большее, чем обычная проверка, выносите логику валидации в отдельную загружаемую сборку. Это упрощает обслуживание правил. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- Используйте валидацию на стороне клиента, чтобы обеспечить лучшее взаимодействие с пользователем, но из соображений безопасности всегда выполняйте проверку и на сервере. Вообще исходите из предположения о том, что все поступающие от клиента данные являются злонамеренными. Сервер должен перепроверять все передаваемые на него данные. Обеспечивайте проверку ввода со всех источников, таких как строки запроса, cookies и элементы управления HTML.
- Проектируйте механизмы валидации, обеспечивающие ограничение, отклонение и очистку данных. Проверяйте длину, диапазон, формат и тип вводимых данных. Обозначьте границы доверия на сервере и проверяйте все данные, пересекающие их.
- Храните правила валидации клиента в изолированном хранилище. Для правил, требующих доступа к ресурсам сервера, продумайте, не будет ли более эффективным использовать отдельный сервис, выполняющий проверку на сервере.
- При наличии на стороне клиента большого объема кода валидации, который может меняться, рассмотрите возможность его размещения в отдельном загружаемом модуле. Это упростит его замену без необходимости повторной загрузки всего RIA-приложения.

Более подробно методики валидации обсуждаются в главе 17, «[Сквозная функциональность](#)».

Вопросы безопасности

Безопасность охватывает широкий диапазон факторов и имеет решающее значение для всех типов приложений. Проектирование и реализация насыщенных Интернет-приложений должны вестись с обязательным учетом вопросов безопасности, а там где эти приложения выступают в роли слоя представления для бизнес-приложений, они должны участвовать в обеспечении безопасности других слоев приложения. Безопасность касается множества аспектов, включая защиту конфиденциальных данных, аутентификацию и авторизацию пользователей, защиту от атак злонамеренного кода и пользователей, аудит и протоколирование событий и действий пользователей.

При проектировании стратегии обеспечения безопасности руководствуйтесь следующими рекомендациями:

- Выберите подходящую технологию и подход к аутентификации пользователей. Необходимо принять решение о том, как и когда регистрировать пользователей, должна ли быть реализована поддержка различных типов пользователей (ролей) с разными правами доступа (например, администраторы и обычные пользователи), и как регистрировать успешные или неудачные попытки входа пользователя в систему.
- Используйте интегрированную аутентификацию Windows (Windows Integrated Authentication), механизм единой регистрации (single sign-on, SSO) или интегрированное решение аутентификации, если требуется обеспечить пользователям возможность доступа к множеству приложений по одному удостоверению или с использованием одних и тех же учетных данных. Если нет возможности использовать Windows Integrated Authentication, можно применить внешний сервис, предлагающий интегрированную поддержку аутентификации. Если нет возможности использовать внешний сервис, используйте систему на базе сертификатов или создавайте собственное решение для своей организации.
- Продумайте необходимость проверки ввода и от пользователя, и из источников, таких как сервисы и интерфейсы других приложений. Можно создавать собственные механизмы проверки или, по возможности, использовать функции валидации, предлагаемые применяемой технологией UI.
- Продумайте стратегию аудита и протоколирования для приложения и то, какие данные будут регистрироваться. Не забывайте защищать конфиденциальные данные в журналах с помощью шифрования. Для наиболее важных данных, уязвимых для повреждения и подделки, можно применять цифровые подписи.

Вопросы обработки данных

Как правило, доступ к данным приложения осуществляется через сетевые сервисы. Кэшируйте эти данные для улучшения производительности и обеспечения возможности их использования в автономном режиме. Данные приложения можно разделить на две основные категории:

- **Справочные данные только для чтения.** Это данные, которые меняются нечасто и используются клиентом для справочных целей, такие как, например, каталог продукции. Храните ссылочные данные на клиенте, чтобы сократить объемы данных,

которыми обмениваются клиент и сервер и, таким образом, повысить производительность приложения, обеспечить возможности работы в автономном режиме, обеспечить раннюю проверку данных и повысить удобство использования приложения в общем.

- **Изменяющиеся данные.** Это данные, которые могут изменяться как на клиенте, так и на сервере. Сложнее всего в насыщенных Интернет-приложениях обеспечивать параллельную обработку таких данных, когда одновременно многие клиенты могут вносить изменения в одни и те же данные. Необходимо отслеживать все изменения данных, выполняемые клиентами, и обрабатывать обновления на сервере, чтобы не допустить возможных конфликтов.

Вопросы выбора технологий

Далее приводится руководство по применению Silverlight и Microsoft Windows Communication Foundation (WCF). На момент написания данного документа были выпущены версии WCF 3.5 и Silverlight 3.0. Данные рекомендации помогут при выборе и реализации соответствующей технологии.

Версии и целевые платформы

- На момент выхода данного руководства Silverlight for Mobile был объявлен и находился в разработке, но еще не был выпущен.
- На текущий момент посредством подключаемого модуля технологию Silverlight поддерживают браузеры Safari, Firefox и Microsoft Internet Explorer. Через эти браузеры Silverlight поддерживает Mac и Windows. В 2008 году было объявлено о поддержке для Windows Mobile. Реализация Silverlight с открытым исходным кодом под названием Moonlight обеспечивает поддержку систем Linux и Unix/X11.
- Silverlight поддерживает языки разработки C#, Iron Python, Iron Ruby и Visual Basic® .NET. Большая часть XAML-кода также может выполняться как на WPF-, так и на Silverlight-хостах.
- В Silverlight 2.0 необходимо реализовывать собственный код валидации ввода и данных. Silverlight 3.0 поддерживает валидацию на основании исключений через привязку данных. Обратитесь к документации, чтобы выяснить, какие варианты валидации доступны в текущей версии.

Безопасность

- Криптографические API .NET доступны в Silverlight и должны применяться при хранении и передаче на сервер конфиденциальных данных, если они еще не зашифрованы с использованием другого механизма шифрования.
- Для каждого зарегистрированного пользователя Silverlight создает отдельный файл журнала. Он не может вести один журнал для всего компьютера.

- Silverlight не скрывает загруженные модули, они могут быть декомпилированы с извлечением логики программы.
-

Сетевое взаимодействие

- Silverlight поддерживает только асинхронное взаимодействие с Веб-сервисами.
 - Silverlight поддерживает только основные HTTP-привязки. WCF в .NET 3.5 поддерживает основные HTTP-привязки, но без включения безопасности по умолчанию. Для защиты взаимодействий сервисов должна быть включена, по крайней мере, безопасность на транспортном уровне.
 - Silverlight поддерживает два формата файлов для вызова сервисов, которые располагаются в разных доменах с источником текущей страницы. Можете использовать либо специальный файл для Silverlight, ClientAccessPolicy.xml, либо файл CrossDomain.xml, совместимый с Adobe Flash. Разместите файл в корневом каталоге сервиса(-ов), к которому(-ым) должен выполнять доступ Silverlight-клиент.
 - Для передачи больших объемов данных с сервера используйте в Silverlight-приложении ADO.NET Data Services.
 - Из-за модели безопасности браузера Silverlight в настоящее время не поддерживает ошибки SOAP, предоставляемые сервисами. Сервисы должны возвращать исключения клиенту, используя другой механизм.
-

Элементы управления

- Silverlight включает элементы управления, специально спроектированные для него. По всей вероятности, также будут доступны дополнительные пакеты элементов управления сторонних производителей.
 - Используйте базоконный режим работы Silverlight, если желаете размещать видимое HTML-содержимое и элементы управления поверх Silverlight-приложения.
 - Silverlight позволяет вводить дополнительное поведение в существующие реализации элементов управления. Используйте этот подход вместо создания подклассов элементов управления.
 - Silverlight выполняет сглаживание всех компонентов UI, поэтому учтите рекомендации по привязке элементов UI к целочисленным значениям пикселов.
-

Хранение

- В качестве локального механизма хранения для Silverlight используется изолированное хранилище клиентского компьютера. Его исходный максимальный размер – 1 МБ. Максимальный размер хранилища не ограничен, но в Silverlight размер хранилища может быть увеличен только пользователем по запросу приложения.
-

Дополнительную информацию можно найти также в материале «*Contrasting Silverlight and WPF*» (Сравнение Silverlight и WPF) по адресу <http://msdn.microsoft.com/en-us/library/dd458872.aspx>.

Вопросы развертывания

Реализации RIA с точки зрения развертывания и удобства обслуживания обеспечивают во многом те же преимущества, что и Веб-приложения. Проектируйте RIA как отдельные модули, которые могут загружаться по отдельности и кэшироваться, чтобы в случае необходимости обеспечить возможность замены одного модуля, а не всего приложения. Обозначайте версии приложения и компонентов, чтобы иметь возможность определять, какую версию выполняют клиенты. При проектировании с учетом аспектов развертывания и удобства обслуживания руководствуйтесь следующими рекомендациями:

- Продумайте, как будут обрабатываться сценарии, когда подключаемый модуль браузера для RIA не установлен.
- Продумайте, как будет осуществляться повторное развертывание модулей в условиях, когда экземпляр приложения продолжает выполняться на клиенте.
- Разделите приложение на логические модули, которые могут кэшироваться по отдельности. Это позволит без труда заменять их, и пользователю при этом не придется повторно загружать все приложение.
- Создавайте версии компонентов.

Установка подключаемого модуля RIA

Продумайте, как будет реализована установка подключаемого модуля браузера для RIA, если он еще не установлен:

- **Инtranет.** По возможности используйте программное обеспечение распространения приложений или групповую политику службы каталогов Microsoft Active Directory® для предварительной установки подключаемого модуля на каждый компьютер организации. Альтернативный вариант – применение клиентского компонента Центра обновления Windows, где Silverlight является необязательным компонентом. Наконец, можно установить подключаемый модуль вручную через браузер; это может сделать пользователь, обладающий правами Администратора на клиентском компьютере.
- **Интернет.** Пользователи должны устанавливать подключаемый модуль вручную, поэтому необходимо обеспечить им соответствующую ссылку для загрузки последней версии подключаемого модуля. Для пользователей Windows Центр обновления Windows предлагает подключаемый модуль как необязательный компонент.
- **Обновления подключаемого модуля.** Как правило, обновления подключаемого модуля разрабатываются с учетом обратной совместимости. Создавая свое приложение, можно ориентироваться на определенную версию подключаемого

модуля, но необходимо предусмотреть план перевода функциональности приложения на новые версии подключаемого модуля браузера по мере их выхода. Для интранет-сценариев переходите на новый подключаемый модуль только после тестирования в нем своего приложения. В Интернет-сценариях учитывайте то, что обновления подключаемого модуля будут выполняться автоматически. Тестируйте свое приложение с бета-версиями подключаемого модуля, чтобы обеспечить безболезненный переход пользователей на новые версии при их выходе.

Распределенное развертывание

Реализации RIA копируют или переносят логику представления на клиент, поэтому наиболее вероятным сценарием их развертывания является распределенная архитектура. При распределенном развертывании RIA логика представления располагается на клиенте; бизнес-слой может находиться на клиенте, на сервере или распределяться между клиентом и сервером; и слой доступа к данным размещается на Веб-сервере или сервере приложений. Как правило, часть бизнес-логики (и даже, возможно, некоторая логика доступа к данным) переносится на клиент для повышения производительности. В этом случае бизнес-слой и слой доступа к данным распределяются между клиентом и сервером приложений, как показано на рис. 2.

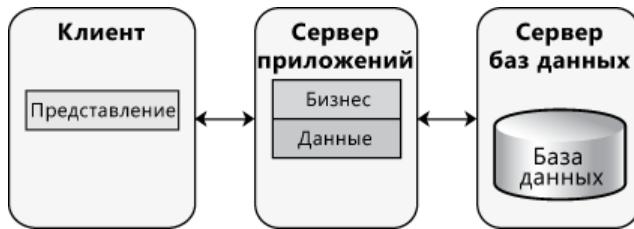


Рис. 39

Распределенное развертывание RIA

При проектировании развертывания для RIA руководствуйтесь следующими рекомендациями:

- Для большого приложения учтите временные затраты при загрузке компонентов RIA на клиенты.
- Если бизнес-логика совместно используется другими приложениями, предоставьте ее как сервис на сервере, чтобы она была доступна всем приложениям.
- Если в приложении используются сокеты или WCF и не используется порт 80, продумайте о влиянии, оказываемом на приложение межсетевыми экранами, которые обычно блокируют другие порты.
- Используйте файл crossdomain.xml, чтобы обеспечить RIA-клиентам возможность доступа к другим доменам в случае необходимости.

Балансировка нагрузки

При развертывании приложения на множестве серверов можно использовать балансировку нагрузки для распределения запросов RIA-клиентов между разными серверами. Это приводит

к сокращению времени отклика, повышает использование ресурсов и увеличивает пропускную способность. На рис. 3 представлен сценарий с балансировкой нагрузки.

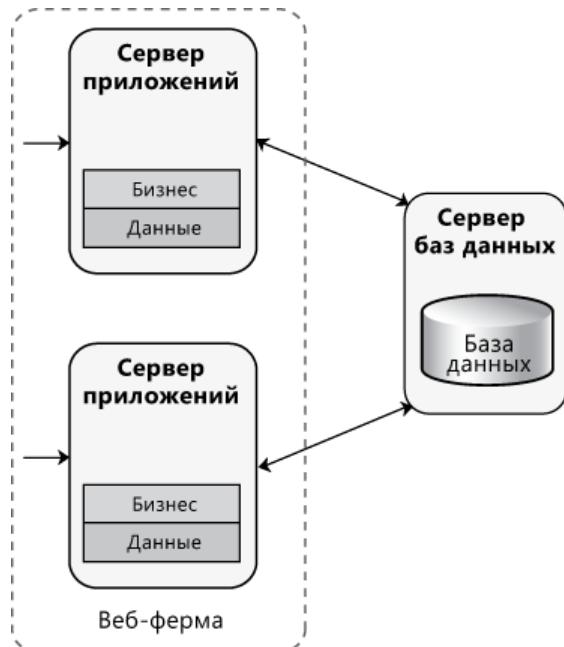


Рис. 40

Развертывание RIA с балансировкой нагрузки

При проектировании приложения с использованием балансировки нагрузки руководствуйтесь следующими рекомендациями:

- Избегайте привязки к конкретному серверу, т.е. ситуации, когда все запросы от определенного клиента обрабатываются одним и тем же сервером. Чаще всего это происходит при использовании кэшей с возможностью локального обновления либо хранилищ состояния сеансов в процессе или локальных хранилищ.
- Страйтесь хранить все данные состояния на клиенте и проектировать бизнес компоненты без сохранения состояния.
- Используйте программное обеспечение балансировки сетевой нагрузки для реализации перенаправления запросов на серверы фермы приложений.

Веб-ферма

Если к RIA-приложению предъявляются довольно высокие требования по обработке бизнес-логики, доступа к данным или данных на сервере приложений, используйте Веб-ферму, которая обеспечит распределение запросов от RIA-клиентов между множеством серверов. Веб-ферма позволяет масштабировать приложение и сокращает влияние, оказываемое сбоями оборудования. Для введения дополнительных серверов для приложения могут использоваться решения с балансировкой нагрузки или кластеризацией. Руководствуйтесь следующими рекомендациями:

- Используйте кластеризацию для снижения влияния сбоев оборудования. Выполните секционирование базы данных на множество серверов баз данных, если к приложению предъявляются высокие требования по вводу/выводу.

- Если требуется поддерживать привязку к конкретному серверу, кэшированные данные или состояние конкретного пользователя, конфигурируйте Веб-ферму так, чтобы обеспечить маршрутизацию всех запросов от одного пользователя к одному и тому же серверу.
- Используйте в Веб-ферме управление сессиями в процессе, только если реализуете привязку к конкретному серверу; в противном случае невозможно гарантировать направление запросов пользователя к одному и тому же серверу. Для сценариев без привязки к конкретному серверу используйте внешний сервис сеанса или сервер базы данных.

Более подробно шаблоны и сценарии развертывания рассматриваются в главе 19, [«Физические уровни и развертывание»](#).

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Слои, Сетевое взаимодействие, Композиция и Представление, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Слои	Service Layer (Слой сервисов). Архитектурный шаблон, в котором интерфейс сервисов и реализация вынесены в отдельный слой.
Сетевое взаимодействие	Asynchronous Callback. Выполняет длительные задачи в отдельном потоке, выполняющемся в фоновом режиме, и обеспечивает потоку функцию для обратного вызова по завершении выполнения задачи. Command. Инкапсулирует обработку запроса в отдельный командный объект с обычным интерфейсом выполнения.
Композиция	Composite View. Сочетает отдельные представления в композитное представление. Inversion of Control. Заполняет все зависимости объектов от других объектов или компонентов, которые должны быть выполнены, прежде чем приложение сможет использовать объект.
Представление	Application Controller. Объект, содержащий всю логику выполнения и используемый другими Контроллерами, работающими с Моделью и отображающими соответствующее Представление. Supervising Presenter (Наблюдающий презентатор). Разделяет дизайн презентации на три разные роли, где Представление отвечает за обработку пользовательского ввода и обменивается данными с компонентом Модели, который инкапсулирует бизнес-данные. Объект Презентатор реализует логику представления и координирует взаимодействия между Представлением и Моделью. Presentation Model. Разновидность шаблона Model-View-Presenter (MVP), предназначенная для современных платформ разработки UI, где созданием Представления занимается дизайнер, а не разработчик.

Более подробно шаблон Composite View рассматривается в статье «*Patterns in the Composite Application Library*» at <http://msdn.microsoft.com/en-us/library/dd458924.aspx>.

Более подробно шаблоны Model-View-Controller (MVC) и Application Controller рассматриваются в книге Мартина Фаулера «Архитектура корпоративных приложений». Addison-Wesley, 2002. Или по адресу <http://martinfowler.com/eaaCatalog>.

Более подробно шаблон Command рассматривается в главе 5, «Поведенческие шаблоны» книги Эрика Гамма, Ричарда Хельма, Ральфа Джонсона и Джона Влиссисда «Приемы объектно-ориентированного проектирования. Паттерны проектирования». Питер, 2006.

Более подробно шаблон Asynchronous Callback рассматривается в статье «*Creating a Simplified Asynchronous Call Pattern for Windows Forms Applications*» по адресу <http://msdn.microsoft.com/en-us/library/ms996483.aspx>.

Более подробно шаблон Service Layer рассматривается в статье «*P of EAA: Service Layer*» (Архитектура корпоративных программных приложений: слой сервисов) по адресу <http://www.martinfowler.com/eaaCatalog/serviceLayer.html>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- Информацию по Silverlight можно найти на официальном Веб-сайте Silverlight по адресу <http://silverlight.net/default.aspx>.
- Об использовании WCF с Silverlight рассказывают статьи «*How to: Build a Duplex Service*» (Как: создать дуплексный сервис) по адресу [http://msdn.microsoft.com/en-us/library/cc645027\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/cc645027(VS.95).aspx) и «*How to: Access a Duplex Service with the Channel Model*» (Как: организовать доступ к дуплексному сервису с использованием модели канала) по адресу [http://msdn.microsoft.com/en-us/library/cc645028\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/cc645028(VS.95).aspx).
- Блоги по Silverlight: блог Бреда Абрамса (Brad Abrams) можно найти по адресу <http://blogs.msdn.com;brada/> и блог Скотта Гатри (Scott Guthrie) можно найти по адресу <http://weblogs.asp.net/Scottgu/>.

24

Проектирование мобильных приложений

Обзор

Данная глава поможет понять, когда и как следует применять мобильные приложения, а также рассмотрит основные вопросы проектирования мобильных приложений. Сюда относятся компоненты мобильного приложения; специальные аспекты мобильных приложений, такие как развертывание, энергопотребление и синхронизация; а также ключевые шаблоны и вопросы выбора технологий.

Как правило, обычное мобильное приложение структурировано как многослойное приложение, состоящее из слоя представления, бизнес-слоя и слоя доступа к данным. При разработке мобильного приложения можно создавать тонкий Веб-клиент или насыщенный клиент. Для насыщенного клиента бизнес-слой и слой сервисов данных, скорее всего, будут располагаться на самом устройстве. Для тонкого клиента все слои будут размещены на сервере. Рис. 1 иллюстрирует типовую архитектуру насыщенного клиентского мобильного приложения, компоненты которого сгруппированы по функциональным областям.

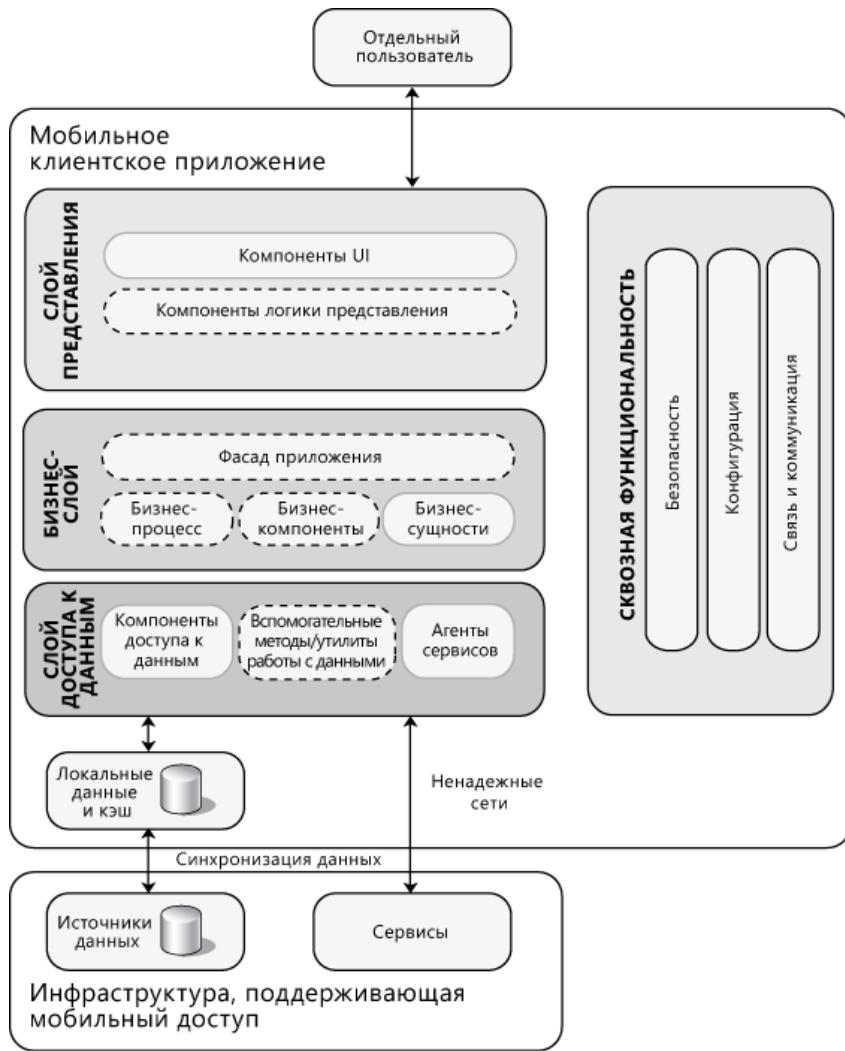


Рис. 41

Типовая структура мобильного приложения

Как правило, в слое представления мобильного приложения располагаются компоненты пользовательского интерфейса и также, возможно, компоненты логики представления. Бизнес-слой, если таковой имеется, обычно включает компоненты бизнес-логики, все компоненты бизнес-процесса и бизнес-сущностей, необходимые приложению, и фасад, если он используется. В слое доступа к данным находятся компоненты доступа к данным и агентов сервисов. Чтобы сократить объем занимаемой приложением памяти устройства, мобильные приложения обычно используют менее жесткие подходы к разделению на слои и меньшее число компонентов. Более подробно многослойный дизайн рассматривается в главе 5, «[Рекомендации по проектированию многослойных приложений](#)». Больше сведений о компонентах каждого слоя можно найти в главе 10, «[Рекомендации по проектированию компонентов](#)».

Общие принципы проектирования

Приведенные далее рекомендации по проектированию касаются различных аспектов, которые должны быть учтены при проектировании мобильного приложения. Они помогут

обеспечить эффективную работу приложения в типовых сценариях мобильных приложений и соответствие всем выдвигаемым требованиям:

- **Определитесь, создается ли насыщенный клиент, тонкий Веб-клиент или насыщенное Интернет-приложение (RIA).** Если создаваемое приложение требует локальной обработки и должно работать в сценарии без постоянного подключения, проектируйте насыщенный клиент. Насыщенное клиентское приложение будет сложнее устанавливать и обслуживать. Если приложение может зависеть от обработки на сервере и будет иметь устойчивое постоянное подключение, создавайте тонкий клиент. Если приложению необходим насыщенный UI, оно имеет только ограниченный доступ к локальным ресурсам и должно быть портируемым на другие платформы, используйте RIA-клиент.
- **Определите, какие типы устройств будут поддерживаться.** При выборе поддерживаемых типов устройств обратите внимание на размер и разрешение экрана, характеристики производительности ЦП, объем памяти и хранилища, а также доступность среды разработки. Кроме того, утите требования пользователей и ограничения организации. Может потребоваться специальное оборудование, такое как глобальная система определения местоположения (global positioning system, GPS) или камера, что может иметь влияние не только на тип приложения, но также на выбор устройства.
- **В случае необходимости учтите сценарии без постоянного подключения и с ограниченной полосой пропускания.** Для автономных мобильных устройств нет необходимости учитывать аспекты подключения¹. Если мобильному приложению требуется возможность сетевого подключения, оно должно обрабатывать сценарии с неустойчивым подключением или без такового. В таких случаях крайне важно спроектировать механизмы кэширования, управления состоянием и доступа к данным в условиях неустойчивого подключения и пакетное взаимодействие, когда подключение доступно. Выбирайте оборудование и программные протоколы, руководствуясь скоростью, энергопотреблением и глубиной детализации, а не только исходя из простоты разработки.
- **Проектируйте UI, подходящий для мобильных устройств, учитывая ограничения платформы.** Чтобы мобильные приложения могли работать в условиях ограничений, налагаемых мобильными устройствами, они должны иметь более простую архитектуру, простой UI, для них должны приниматься особые проектные решения. Учитывайте эти ограничения и не пытайтесь использовать архитектуру или UI настольного или Веб-приложения, а проектируйте для конкретного устройства. Основными ограничениями являются память, время работы батареи, способность адаптироваться к разным размерам или ориентации экрана, безопасность и пропускная способность сети.
- **Создавайте многослойную архитектуру, подходящую для мобильных устройств, которая повышает возможности повторного использования и удобство**

¹ Так как его просто нет (прим. научного редактора).

обслуживания. В зависимости от типа приложения все слои могут располагаться на самом устройстве. Используйте концепцию многослойности, чтобы обеспечить разделение функциональных областей и повысить возможности повторного использования и удобство обслуживания мобильного приложения. Тем не менее, стремитесь к тому, чтобы приложение занимало минимальный объем памяти на устройстве, упрощая дизайн по сравнению с настольным или Веб-приложением.

- **Учитывайте ограничения, налагаемые ресурсами устройства, такие как время работы батареи, объем памяти и частота процессора.** Каждое проектное решение должно учитывать ограничения ЦП, памяти, емкость хранилища и время работы батареи мобильного устройства. Как правило, первостепенным ограничивающим фактором в мобильных устройствах является время работы батареи. Подсветка, чтение и запись в память, беспроводное соединение, специальное оборудование и частота процессора – все эти аспекты оказывают влияние на общее энергопотребление. При недостаточном объеме доступной памяти операционная система Windows Mobile может попросить закрыть приложение или пожертвовать кэшированными данными, что приводит к снижению скорости выполнения программы. Оптимизируйте приложение, чтобы снизить его энергопотребление и объем занимаемой им памяти, не забывая при этом о производительности.

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Аутентификация и авторизация](#)
- [Кэширование](#)
- [Сетевое взаимодействие](#)
- [Управление конфигурацией](#)
- [Доступ к данным](#)
- [Особенности устройств](#)
- [Управление исключениями](#)
- [Протоколирование](#)
- [Портирование приложений](#)
- [Управление энергопотреблением](#)
- [Синхронизация](#)
- [Тестирование](#)
- [Пользовательский интерфейс](#)

- [Валидация](#)
-

Аутентификация и авторизация

Проектирование эффективной стратегии аутентификации и авторизации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. Отсутствие соответствующей аутентификации может сделать приложение уязвимым к неавторизованному использованию. Обычно мобильные устройства проектируются как устройства индивидуального использования и, как правило, не имеют базового профиля пользователя и других средств безопасности, кроме простого пароля. Остальные привычные механизмы обеспечения безопасности настольных приложений также чаще всего отсутствуют.

Возможность обнаружения мобильных устройств через протокол Bluetooth создает для пользователей неожиданные риски. Особые трудности при проектировании мобильного приложения вызывает необходимость реализации сценариев без постоянного подключения. Должны быть учтены все возможные варианты подключения, как проводные, так и беспроводные. При проектировании стратегии аутентификации и авторизации руководствуйтесь следующими рекомендациями:

- Проектируйте аутентификацию и авторизацию как для сценариев с постоянным подключением, так и для сценариев без постоянного подключения; включая синхронизацию по беспроводному соединению, синхронизацию при подключении к компьютеру, обнаружение по Bluetooth, синхронизацию по Виртуальной частной сети (Virtual Private Network, VPN) и локальную синхронизацию карты памяти SD.
 - Учтите то, что у разных устройств могут быть разные программные модели безопасности, что может влиять на авторизацию для доступа к ресурсам.
 - Не делайте предположений о том, что механизмы безопасности, доступные на настольных платформах, будут доступны и на мобильной платформе, даже если используете одни и те же инструменты. Например, в Windows Mobile нет списков управления доступом (ACL) и, следовательно, нет безопасности файлов на уровне операционной системы.
 - Обозначьте границы доверия в рамках слоев мобильного приложения. Например, между клиентом и сервером или сервером и базой данных. Это поможет определить, где и как проводить аутентификацию.
-

Кэширование

Используйте кэширование для улучшения производительности и времени отклика приложения и для обеспечения работоспособности в условиях без сетевого подключения. Кэширование поможет оптимизировать поиск ссылочных данных, избежать повторных обращений к сети и предотвратить ненужную повторную обработку. Принимая решение о том, какие данные кэшировать, учитывайте ограниченность ресурсов устройства; доступный объем хранилища мобильного устройства всегда меньше, чем у настольного компьютера. При проектировании стратегии кэширования руководствуйтесь следующими рекомендациями:

- Обозначьте требования по производительности. Например, определите минимальное время отклика и работы батареи. Протестируйте производительность устройств, которые планируете использовать. У большинства мобильных устройств имеется лишь флэш-память, более медленная, чем память, используемая в настольных компьютерах.
- Проектируйте приложение так, чтобы оно занимало минимальный объем памяти в устройстве. Кэшируйте только жизненно важные для работы приложения данные или те, преобразование которых в готовый к использованию вид требует больших затрат. При проектировании приложения с интенсивным использованием памяти выявите сценарии нехватки памяти и выработайте механизм приоритезации данных для высвобождения памяти в случае необходимости. Однако рассмотрите возможность кэширования любых данных, в том числе и часто меняющихся, для сценариев без постоянного подключения или автономной работы. Приложение должно сохранять работоспособность в условиях недоступности кэшированных данных в сценариях без постоянного подключения или автономной работы.
- Правильно выберите месторасположение кэша. Он может располагаться в устройстве, в мобильном шлюзе или на сервере базы данных. Применяйте для кэширования не память устройства, а SQL Server Compact Edition, поскольку используемая приложением память в ситуациях нехватки памяти может быть очищена.
- Обеспечьте шифрование конфиденциальных данных при кэшировании, особенно при кэшировании данных на съемных носителях.

Более подробно проектирование стратегии кэширования рассматривается в главе 17, «[Сквозная функциональность](#)».

Сетевое взаимодействие

Взаимодействие устройств с компьютером может осуществляться с использованием беспроводной («по воздуху») и проводной связи, и также более специализированных видов связи, таких как Bluetooth или Infrared Data Association (IrDA). При беспроводном взаимодействии предусмотрите меры безопасности для защиты конфиденциальных данных от хищения, повреждения или подделки. При взаимодействии через интерфейсы Веб-сервисов используйте для защиты данных такие механизмы, как стандарты WS-Secure. Не забывайте, что связь с беспроводным устройством является намного менее устойчивой, чем при подключении к компьютеру, и что мобильному приложению, возможно, придется длительное время работать без подключения. При проектировании стратегии связи руководствуйтесь следующими рекомендациями:

- Проектируйте асинхронное взаимодействие с поддержкой потоков, это позволит повысить производительность и удобство использования в сценариях без постоянного подключения. Подключения с ограниченной пропускной способностью, обычно используемые в мобильных устройствах, могут негативно сказаться на производительности и удобстве использования, особенно если они обусловливают блокировку пользовательского интерфейса. Используйте соответствующие протоколы связи и продумайте поведение приложения в ситуации, когда доступны несколько типов подключений. Предоставьте пользователям возможность выбирать

используемое подключение и в случае необходимости выключать связь для сохранения заряда батареи.

- При проектировании приложения для мобильного телефона учтите возможность поступления вызова в ходе передачи данных или выполнения программы. Обеспечьте возможность приостановления и возобновления выполнения или даже завершения приложения в таких случаях.
- Обеспечивайте защиту при взаимодействии с использованием подключений, не имеющих доверия, таких как Веб-сервисы и другие беспроводные методы. Используйте шифрование и цифровые подписи для конфиденциальных данных и обеспечьте защиту данных, передаваемых по VPN. При этом не забывайте о влиянии этих мер обеспечения безопасности на производительность и время работы батареи.
- Если требуется использовать данные из многих источников, взаимодействовать с другими приложениями или работать без подключения, применяйте Веб-сервисы. Обеспечьте эффективное управление подключениями, особенно в условиях ограниченной полосы пропускания.
- Если при взаимодействии с использованием WCF требуется реализовать очередь сообщений, используйте WCF-механизм передачи данных с промежуточным хранением.

Более подробно протоколы и методики связи рассматриваются в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Управление конфигурацией

При проектировании управления конфигурацией устройства проработайте вопросы возвращения устройства в исходное состояние, а также возможности настройки приложения по беспроводному соединению или с компьютера. При проектировании стратегии управления конфигурацией руководствуйтесь следующими рекомендациями:

- Правильно выберите формат конфигурационных данных. Отдавайте предпочтение бинарным форматам, а не XML, чтобы сократить объем занимаемой памяти. Снизить требования к памяти позволит также применение библиотечных процедур сжатия для хранения данных конфигурации и состояния. Шифруйте конфиденциальные данные, хранящиеся в конфигурационных файлах.
- Обеспечьте поддержку восстановления конфигурации после перезагрузки устройства. Продумайте, как будет выполняться синхронизация конфигурационных данных при беспроводной связи и при подключении к компьютеру, а также ознакомьтесь с техниками, используемыми разными производителями для загрузки параметров конфигурации.
- Если для хранения данных предприятия используется Microsoft SQL Server 2005 или 2008 и требуется обеспечить короткий цикл выхода на рынок, используйте репликацию слиянием, обеспечивающую готовым приложением стороннего

производителя¹. Репликация слиянием синхронизирует данные за одну операцию независимо от пропускной способности сети или размера данных.

- Для инфраструктуры Active Directory используйте интерфейс System Center Mobile Device Manager (Диспетчер мобильных устройств), он обеспечит управление групповой конфигурацией, аутентификацией и авторизацией устройств. Требования Mobile Device Manager можно найти в разделе «[Вопросы выбора технологий](#)».

Доступ к данным

Доступ к данным на мобильном устройстве ограничен неустойчивостью сетевых подключений и аппаратными ограничениями самого устройства. При проектировании доступа к данным учитывайте низкую пропускную способность, высокую задержку и неустойчивость подключения. При проектировании стратегии доступа к данным руководствуйтесь следующими рекомендациями:

- Используйте локальную базу данных устройства, которая обеспечивает сервисы синхронизации, такую как SQL Server Compact Edition. Создавайте собственный механизм синхронизации данных, только если стандартные функции синхронизации не отвечают поставленным требованиям.
- Обеспечивайте целостность данных. Файлы, остающиеся открытыми при приостановке работы устройства и сбоях питания, могут обусловить нарушение целостности данных, особенно если данные хранятся на съемном устройстве хранения. Для обеспечения целостности данных в случаях, когда устройства остаются без электропитания или подключения, используйте транзакции в SQL Server Mobile.
- Не делайте допущение о том, что съемное хранилище будет всегда доступным, поскольку пользователь может убрать его в любое время. Проводите проверку на наличие съемного хранилища перед записью в него или использованием **FlushFileBuffers** (Очистка буферов файлов).
- При использовании XML для хранения или передачи данных принимайте во внимание его общий размер и влияние на производительность. Применение XML повышает требования к пропускной способности и размеру локального хранилища. Используйте алгоритмы сжатия или не-XML методы передачи.
- Проектируйте эффективные методы доступа к базе данных и обработки данных, оказывающие минимальное негативное влияние на производительность. Использование типизированных объектов вместо **DataSet** позволит сократить издержки памяти и улучшить производительность. Если предполагается выполнять только чтение данных, используйте объекты **DataReader**. Избегайте операций, требующих интенсивной обработки, таких как навигация по большим наборам данных.

¹ Если в качестве хранилища на мобильном устройстве используется SQL Server Compact Edition, то такая синхронизация обеспечивается встроенными средствами (прим. научного редактора).

Более подробно проектирование слоя доступа к данным рассматривается в главе 8, «[Рекомендации по проектированию слоя доступа к данным](#)».

Особенности устройств

Дизайн и разработка для мобильных устройств уникальна из-за ограничений и разнообразия их аппаратного обеспечения. Мобильное приложение может использоваться на многих устройствах с очень разными параметрами аппаратных средств. При проектировании мобильного приложения необходимо помнить о разнородной среде устройства. Сюда относятся такие факторы, как размеры и ориентация экрана, ограничения объема памяти и хранилища, пропускная способность сети и возможность подключения. Как правило, выбор мобильной операционной системы зависит от типа целевого устройства. При выборе устройства руководствуйтесь следующими рекомендациями:

- Оптимизируйте приложение для устройства, учитывая такие факторы, как размер и ориентация экрана, пропускная способность сети, объем памяти, производительность процессора и другие характеристики аппаратных средств.
- Учитывайте специфические возможности устройства, которые можно использовать для улучшения функциональности приложения, такие как акселерометры, графические процессоры, GPS, осязаемый отклик (прикосновение, усилие и вибрация), компас, камера и устройства чтения отпечатков пальцев.
- При разработке приложения, которое будет использоваться на нескольких устройствах, прежде всего, реализуйте функциональность, общую для всех устройств, а затем уже переходите к настройке кода для выявления и использования характерных для конкретного устройства возможностей в случае их доступности.
- Учитывайте ограниченные ресурсы запоминающих устройств и оптимизируйте приложение для использования минимального объема памяти. При нехватке памяти система может высвобождать кэшированный код на промежуточном языке (intermediate language, IL) для сокращения занимаемого ею объема памяти, возвращаться в режим интерпретации и, таким образом, уменьшать общие темпы выполнения.
- Создавайте модульный код, чтобы упростить извлечение модулей кода из исполняемых файлов. Это решит проблему по сокращению размера исполняемых файлов в условиях ограниченного объема памяти устройства.
- Применение обычных практик программирования может привести к созданию большого количества кода и потреблению большого объема памяти, поэтому используйте упрощенные методы разработки. Например, проанализируйте последствия применения обычных объектно-ориентированных практик, таких как абстрактные базовые классы и инкапсуляция повторяющихся объектов. Используйте отложенную инициализацию, при которой экземпляры объектов создаются только по необходимости.

Управление исключениями

Проектирование эффективной стратегии управления исключениями имеет большое значение с точки зрения безопасности и надежности приложения. Правильная обработка исключений в мобильном приложении обеспечит неразглашение конфиденциальных данных исключения пользователю, повысит надежность приложения и не допустит, чтобы приложение оставалось в несогласованном состоянии после возникновения ошибки. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Проектируйте приложение так, чтобы оно после возникновения ошибки возвращалось в известное стабильное состояние без раскрытия конфиденциальных данных конечному пользователю.
- Перехватывайте исключения, только если можете обработать их, и не используйте исключения для управления логикой выполнения. Спроектируйте глобальный обработчик ошибок для перехвата необрабатываемых исключений.
- Выработайте соответствующую стратегию протоколирования и уведомления, которая будет обеспечивать сохранение достаточного количества сведений об исключениях, но не забывайте при этом об ограничениях объема памяти и хранилища мобильных устройств. Обеспечьте отображение понятных пользователям сообщений об исключениях, не раскрывающих конфиденциальных данных о критических ошибках и исключениях.

Более подробно проектирование стратегии управления исключениями рассматривается в главе 17, «[Сквозная функциональность](#)».

Протоколирование

Мобильные устройства обладают ограниченными ресурсами памяти, поэтому протоколирование и инструментирование должны быть ограничены лишь самыми важными случаями, такими как, например, попытки вторжения в устройство. Если устройство является частью большей инфраструктуры, отслеживайте большинство действий устройства на уровне инфраструктуры. Как правило, аудит считается наиболее достоверным, если данные формируются в момент доступа к ресурсу той же процедурой, которая выполняет доступ. Учтите тот факт, что некоторые журналы могут формироваться на устройстве и должны синхронизироваться с сервером в периоды установления соединения с сетью. При проектировании стратегии протоколирования руководствуйтесь следующими рекомендациями:

- В Windows Mobile нет механизма журнала регистрации событий (Event Log). Используйте механизмы протоколирования сторонних производителей, которые поддерживают .NET Compact Framework, такие как OpenNetCF, NLog или log4Net (более подробную информацию о них можно найти в источниках, указанных в разделе [Дополнительные источники](#) в конце данной главы). Также продумайте, как будет осуществляться доступ к журналам, хранящимся на устройстве.
- При широкомасштабном протоколировании на устройстве предусмотрите сокращенный или сжатый формат для ведения журнала, чтобы снизить требования по

необходимому объему памяти и хранилища. Как альтернативный вариант рассмотрите возможность удаленного протоколирования.

- Для протоколирования и аудита событий используйте такие возможности платформы, как контроль работоспособности на сервере и сервисы мобильного устройства на устройстве. Рассмотрите возможности введения средств удаленного контроля работоспособности, используя стандарт Open Mobile Alliance Device Management¹ (OMA DM).
- Обеспечьте синхронизацию между журналами мобильного устройства и сервера для поддержания возможностей аудита на сервере. При работе с инфраструктурой Active Directory для извлечения журналов с мобильных устройств используйте System Center Mobile Device Manager. Требования Mobile Device Manager приведены в разделе «[Вопросы выбора технологий](#)».
- Храните конфиденциальные данные в файлах журнала и аудита только в случае крайней необходимости. Обеспечьте защиту всех конфиденциальных данных с помощью шифрования.
- Дайте определение, какие действия на устройстве являются необычными или подозрительными, и протоколируйте данные на основании этих сценариев.

Портирование приложений

Часто разработчики пытаются портировать часть или все существующее приложение на мобильное устройство. Определенные типы приложений портировать легче, чем остальные, но очень маловероятно, что приложение удастся перенести прямо, без каких-либо изменений. При проектировании стратегии портирования существующего приложения на мобильное устройство руководствуйтесь следующими рекомендациями:

- Для портирования настольного насыщенного клиентского приложения придется полностью переписать его. Насыщенные клиенты редко подходят для использования на небольших экранах и с ограниченными ресурсами памяти и дискового пространства.
- При портировании Веб-приложения на мобильное устройство придется переработать UI для экрана меньшего размера. Также нельзя забывать об экономном энергопотреблении и возможной высокой стоимости подключения мобильного устройства к сети, поэтому следует учесть ограничения по связи и детализированности интерфейса.
- При портировании RIA-клиента проведите исследование того, какой код портируется без изменений. Конкретные рекомендации по этому вопросу можно найти в разделе «[Вопросы выбора технологий](#)» данной главы.

¹ Стандарт Управления устройствами Открытого сообщества производителей мобильной связи (прим. переводчика).

- Изучите и используйте инструменты для портирования. Например, предлагаются конвертеры Java-в-C++. При преобразовании кода для Smartphone в код для Pocket PC Visual Studio позволяет менять целевую платформу и обеспечивает предупреждения в случае применения характерной для Smartphone функциональности. Также в Visual Studio можно связать проекты для настольного и мобильного приложений, чтобы выяснить, какие их части являются портируемыми.
- Не думайте, что сможете портировать в мобильное приложение собственные элементы управления без изменения. Поддерживаемые API, объем памяти, занимаемый приложением, и поведение UI отличаются в мобильном устройстве. Проводите тестирование элементов управления как можно раньше, чтобы иметь возможность запланировать их переработку или найти альтернативный вариант в случае необходимости.

Управление энергопотреблением

Энергопотребление является основным ограничивающим фактором при проектировании приложений для мобильных устройств. Во всех проектных решениях должна учитываться потребляемая мощность устройства и то, как эти решения влияют на общее время жизни батареи. По возможности используйте устройства, которые способны заряжаться от Universal Serial Bus (USB) или других типов подключений для передачи данных. Изучите протоколы связи с точки зрения их относительной потребляемой мощности. При проектировании стратегии энергопотребления руководствуйтесь следующими рекомендациями:

- Реализуйте профили энергопотребления для повышения производительности, когда устройство подключено к внешнему источнику энергии, и его батарея не заряжается. Предоставьте пользователю возможность отключать функции устройства, которые не используются или не нужны. Типичными примерами такой функциональности являются подсветка экрана, жесткие диски, функции GPS, динамики и беспроводная связь.
- Для увеличения срока жизни батареи, не обновляйте UI, когда приложение выполняется в фоновом режиме.
- Выбирайте протоколы, проектируйте интерфейсы сервисов и пакетную передачу для беспроводного взаимодействия таким образом, чтобы передавать минимально возможное количество байтов. При выборе методов связи учитывайте как потребляемую мощность, так и скорость сети. Также предусмотрите возможность отложить передачу по беспроводному соединению не имеющих первостепенное значение данных до момента подключения устройства к внешнему источнику питания.
- Если предполагается использовать аппаратный протокол связи 3G, учтите, что наряду с намного большей скоростью передачи данных, он также на настоящий момент потребляет намного больше мощности, чем его предшественники, такие как протокол Edge. При использовании 3G обеспечьте пакетное взаимодействие и отключайте связь, когда в ней нет необходимости.

Синхронизация

Продумайте, какой тип синхронизации будете поддерживать: синхронизацию по беспроводному соединению, синхронизацию при подключении к компьютеру или оба типа. Поскольку в синхронизации часто участвуют конфиденциальные данные, реализуйте способы защиты данных синхронизации, особенно при синхронизации по беспроводному соединению. При проектировании синхронизации предусмотрите корректную обработку разрывов подключения, либо отменяя операцию, либо обеспечивая ее возобновление при восстановлении подключения. При репликации слиянием возможна как односторонняя, так и двунаправленная синхронизация. Репликация слиянием является хорошим выбором для инфраструктур, использующих последние версии SQL Server. Используйте Microsoft Sync Framework, которая обеспечивает надежные сервисы синхронизации в разнообразнейших сценариях. При проектировании стратегии синхронизации руководствуйтесь следующими рекомендациями:

- Если пользователи будут выполнять синхронизацию с компьютером, предусмотрите в дизайне синхронизацию при подключении к компьютеру. Если пользователи должны синхронизировать данные, находясь вне офиса, предусмотрите синхронизацию по беспроводному соединению.
- Обеспечьте восстановление приложения при сбросе или прерывании синхронизации. Продумайте, как будут обрабатываться конфликты синхронизации.
- Обеспечьте защиту передачи данных при синхронизации, например, с помощью шифрования и цифровых сертификатов, и используйте безопасные каналы. Особенно внимательно отнеситесь к выбору соответствующей стратегии аутентификации и авторизации при использовании синхронизации по Bluetooth.
- Если требуется поддерживать двунаправленную синхронизацию с SQL Server, используйте синхронизацию через репликацию слиянием. Не забывайте, что синхронизация слиянием обеспечит синхронизацию всех данных множества слияния, для чего может потребоваться дополнительная полоса пропускания, и что может негативно сказаться на производительности.
- Для синхронизации с промежуточным хранением используйте WCF, а не электронную почту или SMS (текстовые сообщения), поскольку WCF обеспечивает гарантированную доставку и превосходно работает в сценариях без постоянного подключения.

Тестирование

Отладка мобильного приложения может быть намного более дорогостоящей, чем отладка аналогичного приложения для компьютера. Затраты на отладку должны быть учтены при принятии решения о том, какие устройства и сколько устройств будет поддерживать создаваемое приложение. Также нельзя забывать, что получить отладочные данные с устройства сложнее, и что эмуляторы устройств не всегда хорошо моделируют аппаратную среду устройства. При проектировании стратегии отладки руководствуйтесь следующими рекомендациями:

- Оцените затраты на отладку при выборе устройств, которые будут поддерживаться приложением. Учитывайте при этом инструментальную поддержку, стоимость устройств (и, возможно, их замены) для начального тестирования и стоимость программных эмуляторов устройств.
- Если имеете доступ к физическому устройству, для которого разрабатываете приложение, выполняйте отладку кода лучше на нем, а не на эмуляторе. Если устройство недоступно, для начального тестирования и отладки используйте эмулятор. Учитывайте, что на эмуляторе код может выполняться медленнее, чем на реальном устройстве.
- Получив физическое устройство, переходите к выполнению кода на устройстве, подключенном к обычному компьютеру. Протестируйте сценарии, когда устройство полностью отключено от любых сетей или подключений, включая сеанс отладки компьютера, и проведите окончательное тестирование, отключив устройство от компьютера. Добавьте временные или постоянные механизмы отладки проблем, возникающих в этом сценарии. Учтите нужды специалистов, которые будут обеспечивать техническую поддержку этого устройства.
- Если вы являетесь производителем оборудования (original equipment manufacturer, OEM), и устройство еще не создано, можно выполнять отладку мобильной программы на выделенном компьютере с процессором x86, работающем под управлением Windows CE. Используйте этот вариант, пока не создано устройство.

Пользовательский интерфейс

При проектировании UI для мобильного приложения не пытайтесь адаптировать или использовать UI настольного приложения. Проектируйте для устройства максимально простой UI, специально предназначенный для рукописного ввода и ограниченных возможностей ввода данных, если это необходимо. Учитывайте тот факт, что мобильное приложение будет выполняться в полноэкранном режиме и сможет отображать одновременно только одно окно, следовательно, операции, приводящие к блокировке, не дадут пользователю взаимодействовать с приложением. При проектировании UI для мобильного приложения руководствуйтесь следующими рекомендациями:

- Проектируйте однооконный полноэкранный UI. Если устройство будет устройством индивидуального использования, выполняющим только основное приложение, используйте полноэкранный режим. Не забывайте, что Windows Mobile не поддерживает этот режим, поэтому придется воспользоваться Windows CE.
- При проектировании приложения учитывайте различные размеры и ориентацию экрана целевых устройств. Также не забывайте об ограничениях, налагаемых небольшим экраном, ограниченным API и сокращенным по сравнению с настольными средами набором элементов управления UI.
- Обеспечивайте удобство использования, предоставляя сенсорный UI или UI с поддержкой стилуса. Располагайте меню и другие элементы управления внизу экрана (с возможностью развертывания вверх в случае необходимости), чтобы предотвратить

загрязнение экрана руками пользователя. Для поддержки сенсорного ввода делайте кнопки достаточно большими и компонуйте элементы управления таким образом, чтобы с UI можно было работать с помощью пальцев или стилуса.

- Обеспечьте визуализацию операций, вызывающих блокировку, например, курсор в виде песочных часов.

Валидация

Используйте валидацию для защиты устройства и приложения и для улучшения удобства использования. Проверка вводимых значений перед их передачей на удаленный сервер может способствовать сокращению количества обращений к сети и улучшению производительности и удобства использования приложения, особенно в сценариях без постоянного подключения или без подключения вообще. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- По возможности проверяйте данные, вводимые пользователем, для предотвращения передачи недействительных данных и сокращения количества обращений к серверу. Это также сокращает время реакции приложения на ввод недействительных значений.
- Проверяйте все данные, получаемые при взаимодействии с компьютером и по беспроводной связи.
- Обеспечьте защиту аппаратным ресурсам, таким как камера и инициирование телефонных вызовов, путем валидации кода и действий, которые автоматически инициируют эти возможности.
- Учитывайте ограниченность ресурсов и производительности устройства, проектируя эффективные механизмы валидации, занимающие минимальный объем памяти на устройстве.

Подробнее методики валидации рассматриваются в главе 17, «[Сквозная функциональность](#)».

Вопросы выбора технологий

В следующих рекомендациях представлены предложения и советы, применимые к типовым сценариям и технологиям мобильных приложений.

Microsoft Silverlight for Mobile

На момент выхода данного руководства Silverlight for Mobile заявлен как продукт, находящийся в стадии разработки, но еще не выпущен. При использовании Silverlight for Mobile руководствуйтесь следующими рекомендациями:

- Используйте Silverlight for Mobile, если желаете создавать приложения, поддерживающие насыщенные мультимедиа и интерактивность, с возможностью выполнения как на мобильном устройстве, так и на настольном компьютере. Код Silverlight 2.0, созданный для выполнения на настольном компьютере в подключаемом модуле Silverlight 2.0, будет выполняться в подключаемом модуле Windows Mobile

Silverlight в последней версии браузера Microsoft Internet Explorer® for Mobile. Но не забывайте о том, что несмотря на возможность использования одного и того же Silverlight-кода для мобильного устройства и для настольного компьютера, для мобильного устройства необходимо учесть другой размер экрана и ограничения ресурсов. Код должен быть оптимизирован для Windows Mobile.

- Если требуется создавать Веб-страницы как для настольных, так и для мобильных платформ, используйте Silverlight for Mobile или обычный ASP.NET/HMTL. Применяйте ASP.NET for Mobile, только если устройство не поддерживает никакие другие альтернативные варианты. Браузеры устройств стали более мощными и теперь могут обрабатывать обычный HTML и ASP.NET, используемый для настольных браузеров, поэтому специальная разработка на ASP.NET для мобильных устройств теряет свою актуальность. В настоящее время ASP.NET for Mobile поддерживает самые разнообразные мобильные устройства благодаря специальным адаптерам разметки и профилям устройств. ASP.NET for Mobile во время выполнения автоматически формирует содержимое соответственно возможностям устройства, но в ней присутствуют издержки, связанные с тестированием и обслуживанием профилей устройств. Поддержка разработки для этих элементов управления включена в Microsoft Visual Studio 2003 и 2005, но не входит в Visual Studio 2008. Поддержка среды выполнения до сих пор доступна, но может быть изъята в будущем. Дополнительные источники по этим вопросам можно найти в разделе «[Дополнительные источники](#)» в конце данной главы.

.NET Compact Framework

При работе с Microsoft .NET Compact Framework руководствуйтесь следующими рекомендациями:

- Если вы знакомы с Microsoft .NET Framework и разрабатываете приложение одновременно для настольной и мобильной платформ, примите во внимание, что .NET Compact Framework является подмножеством библиотеки классов .NET Framework. Она также включает некоторые классы, специально разработанные для Windows Mobile. .NET Compact Framework поддерживает только системы разработки Microsoft Visual Basic® и Microsoft Visual C#®.
- В случае возникновения проблем с построчной трассировкой кода Windows Mobile в отладчике Visual Studio, используйте несколько сеансов отладки. Например, при отладке в одном сеансе машинного и управляемого кода Visual Studio, возможно, не сможет отследить сеанс через границы контекстов. В этом случае придется запустить два экземпляра Visual Studio и отслеживать содержимое между ними вручную.

Windows Mobile

При разработке приложений для Windows Mobile руководствуйтесь следующими общими рекомендациями:

- Если приложение планируется выполнять как под управлением Windows Mobile Professional, так и Windows Mobile Standard, принимайте во внимание отличия модели безопасности в разных версиях Windows Mobile. Код, работающий на одной платформе, может не работать на другой из-за разных моделей безопасности для API. Изучите документацию Windows Mobile для своего устройства и используемой версии. Также ознакомьтесь с источниками, предлагаемыми в разделе «[Дополнительные источники](#)» в конце данной главы.
- Если планируете обслуживать приложение в будущем или модернизируете существующее приложение, убедитесь, что понимаете наследование, схему присваивания имен продуктам и дерево версий операционной системы Windows Mobile. В каждой версии имеются небольшие отличия, которые должны быть учтены при разработке приложения.
 - Windows Mobile является производной выпущенных версий операционной системы Windows CE.
 - Windows Mobile версии 5.x и 6.x основаны на Windows CE версии 5.x.
 - Windows Mobile Pocket PC переименована в Windows Mobile Professional, начиная с версии Windows Mobile 6.0.
 - Windows Mobile Smartphone переименована в Windows Mobile Standard, начиная с версии Windows Mobile 6.0.
 - API Windows Mobile Professional и Windows Mobile Standard несколько отличаются. Например, в Windows Mobile Standard (Smartphone) в ее Compact Framework реализации нет класса Button, потому что вместо него для ввода данных используются программируемые клавиши.
- Для реализации доступа к памяти и файловым структурам всегда используйте API Windows Mobile. Не выполняйте доступ к ним сразу после получения дескриптора любой из структур. Windows CE версия 6.x (и, следовательно, следующая выпущенная версия Windows Mobile) использует виртуализированную модель памяти и модель выполнения процесса, отличную от предыдущих версий. Это означает, что такие структуры, как дескрипторы и указатели файлов, возможно, больше не являются указателями на физическую область памяти. Программы Windows Mobile, использующие эту деталь реализации в версиях 6.x, будут давать сбой при переходе на следующую версию Windows Mobile.
- Mobile Device Manager (MDM) является возможным решением авторизации, отслеживания и сбора журналов с мобильных устройств при использовании инфраструктуры Active Directory. Как и в случае с использованием Windows Mobile 6.1 на управляемых устройствах, для полноценного функционирования MDM на сервере должен быть установлен ряд других продуктов, включая:
 - Windows Mobile 6.1 на устройствах
 - Windows Server Update Service (WSUS) 3.0
 - Windows Mobile Device Management Server

- Enrollment Server
 - Gateway Server
 - Active Directory как часть Windows Server
 - SQL Server 2005 или последующие версии
 - Microsoft Certificate Authority
 - Internet Information Server (IIS) 6.0
 - .NET Framework 2.0 или последующие версии
-

Windows Embedded

При выборе технологии Windows Embedded руководствуйтесь следующими рекомендациями:

- При проектировании ПО для ресивера цифрового телевидения или других мощных устройств используйте Windows Embedded Standard.
 - При проектировании ПО для POS-терминала (устройства пункта обслуживания), такого как банкомат (ATM, терминалы для работы с клиентами или системы кассового самообслуживания) используйте Windows Embedded for Point of Service.
 - При проектировании ПО для устройства с поддержкой GPS или устройства с возможностями навигации, используйте Microsoft Windows Embedded NavReady™. Обратите внимание, что Windows Embedded NavReady 2009 встроена в Windows Mobile 5.0, тогда как в последних версиях Windows Mobile Standard и Professional используется Windows Mobile версия 6.1. Если целью является создание общего кода для NavReady и других устройств, работающих под управлением Windows Mobile, убедитесь, что используете API, доступные на обеих платформах.
-

Вопросы развертывания

Существует множество разных методов развертывания мобильных приложений. Продумывая стратегию развертывания, учитывайте требования пользователей, а также то, как будет обслуживаться приложение. Убедитесь, что обеспечиваете соответствующее управление, администрирование и безопасность развертывания приложения. Рассмотрим возможные сценарии развертывания приложений для устройств, работающих под управлением Windows Mobile (первыми указаны наиболее часто используемые):

- Технология Microsoft Exchange ActiveSync® с использованием файла Windows Installer (MSI).
- По беспроводному соединению с использованием HTTP, SMS или CAB-файлов для обеспечения функциональности установки и запуска.
- С применением Mobile Device Manager, использующим Active Directory для загрузки из файла CAB или MSI.

- Постзагрузка и автоматический запуск, в этом случае специальный пакет компании загружается как часть операционной системы.
- Загрузка с сайта, вручную с использованием SD-карты.

При проектировании стратегии развертывания руководствуйтесь следующими рекомендациями:

- Если требуется предоставить пользователям возможность устанавливать и обновлять приложения удаленно, находясь вне офиса, предусмотрите развертывание по беспроводному соединению.
- При использовании распределения для множества устройств с помощью CAB-файла включите в CAB-файл исполняемые файлы для множества устройств. Обеспечьте выявление необходимого исполняемого файла устройством и удаление всех остальных исполняемых файлов.
- Если приложение широко использует возможности компьютера, применяйте для его развертывания ActiveSync.
- При развертывании базового приложения, выполняющегося под управлением Windows Mobile, используйте механизм постзагрузки для автоматической загрузки приложения сразу после запуска операционной системы Windows Mobile.
- Если приложение будет выполняться только на определенном сайте, и вы хотите вручную управлять распределением, предусмотрите развертывание с использованием карты памяти SD.

Более подробно шаблоны и сценарии развертывания рассматриваются в главе 19, «[Физические уровни и развертывание](#)».

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Кэширование, Сетевое взаимодействие, Доступ к данным, Синхронизация и UI, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Кэширование	Lazy Acquisition (Отложенное получение) . Максимально долго откладывает получение ресурсов для оптимизации использования ресурсов устройства.
Сетевое взаимодействия	Active Object (Активный объект) . Поддерживает асинхронную обработку путем инкапсуляции запроса сервиса и ответа по завершении выполнения сервиса. Communicator (Коммуникатор) . Инкапсулирует внутренние детали взаимодействия в отдельном компоненте, который может обмениваться данными по разным каналам. Entity Translator . Объект, преобразующий типы данных сообщения в бизнес-типы для запросов и выполняющий обратные преобразования для ответов.

	Reliable Sessions. Надежная передача сообщений из конца в конец между источником и точкой назначения, не зависящая от количества или типа посредников между конечными точками
Доступ к данным	<p>Active Record. Включает объект доступа к данным в сущность предметной области.</p> <p>Data Transfer Object (DTO). Объект, в котором сохраняются данные, передаваемые между процессами, что обеспечивает сокращение необходимого числа вызовов методов.</p> <p>Domain Model. Набор бизнес-объектов, представляющих сущности предметной области и отношения между ними.</p> <p>Transaction Script. Организует бизнес-логику каждой транзакции в одну процедуру, обращаясь к базе данных напрямую либо через тонкую оболочку над базой данных.</p>
Синхронизация	Synchronization (Синхронизация). Компонент, установленный на устройство, отслеживает изменения в данных и, когда подключение доступно, обменивается сведениями с компонентом на сервере.
UI	<p>Application Controller. Объект, включающий всю логику выполнения и используемый другими Контроллерами, которые работают с Моделью и отображают соответствующее Представление.</p> <p>Model-View-Controller. Разделяет данные предметной области, представление и действия, основанные на пользовательском вводе, на три отдельные классы. Модель управляет поведением и данными предметной области приложения, отвечает на запросы о состоянии (обычно поступающие от Представления) и реагирует на команды изменить состояние (обычно поступающие от Контроллера). Представление управляет отображением данных. Контроллер интерпретирует пользовательский ввод с мыши и клавиатуры, информируя модель и/или представление о необходимости соответствующих изменений.</p> <p>Model-View-Presenter. Разделяет обработку запросов на три роли, где Представление отвечает за обработку пользовательского ввода, Модель – за данные приложения и бизнес-логику, и Презентатор – за логику представления и координирование взаимодействий между Представлением и Моделью.</p> <p>Pagination (Разбиение на страницы). Разделяет большие объемы содержимого на отдельные страницы для оптимизации ресурсов системы и сокращения места, занимаемого содержимым на экране.</p>

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- Более подробно варианты используемых технологий для Windows Embedded рассматриваются в [Windows Embedded Developer Center](#)
- Больше сведений о фабриках ПО для мобильных устройств можно найти в статье [patterns & practices Mobile Client Software Factory](#)
- Microsoft Sync Framework обсуждается в статье [Microsoft Sync Framework Developer Center](#)

- OpenNETCF.Diagnostics.EventLog в Smart Device Framework рассматривается в материале [Instrumentation for .NET Compact Framework Applications](#) (Инструментирование для приложений .NET Compact Framework)
 - Подробнее об ASP.NET Mobile рассказывает материал [Roadmap for ASP.NET Mobile Development](#) (Дорожная карта для разработки с ASP.NET for Mobile)
 - Более подробно о введении поддержки исходного кода ASP.NET Mobile в Visual Studio 2008 рассказывает статья [Tip/Trick: ASP.NET Mobile Development with Visual Studio 2008](#) (Советы и рекомендации: разработка на ASP.NET Mobile с использованием Visual Studio 2008)
 - Более подробно о разрешениях модели безопасности в Windows Mobile 6.x рассказывается в статье [Security Model for Windows Mobile 5.0 and Windows Mobile 6](#) (Модель безопасности Windows Mobile 5.0 и Windows Mobile 6)
 - Более подробную информацию о сервисах протоколирования Apache «log4Net» можно найти по адресу <http://logging.apache.org/log4net/index.html>.
 - Более подробную информацию о продукте Ярослава Ковальского «NLog» можно найти по адресу <http://www.nlog-project.org/introduction.html>.
 - Больше сведений о сообществе OpenNetCF можно найти по адресу <http://community.opennetcf.com/>.
-

25

Проектирование сервисных приложений

Обзор

Данная глава рассказывает о природе и использовании сервисов, предлагает общие рекомендации для разных сценариев применения сервисов и описывает основные атрибуты сервисов. Также здесь приводятся рекомендации для проектирования слоев в сервисном приложении и ключевые факторы, которые необходимо учесть с точки зрения производительности, безопасности, развертывания, используемых шаблонов, а также обсуждаются вопросы выбора технологий.

Сервис – это открытый интерфейс, обеспечивающий доступ к единице функциональности. Сервисы, буквально, обеспечивают некоторую программную услугу вызывающей стороне, потребляющей этот сервис. Сервисы слабо связаны и для обеспечения более сложной функциональности могут комбинироваться с клиентом или в рамках других сервисов. Сервисы могут быть распределенными, доступ к ним может осуществляться как с удаленного компьютера, так и с компьютера, на котором они выполняются. Сервисы ориентированы на сообщения, т.е. интерфейсы сервисов описываются с помощью Web Services Description Language (WSDL), и операции вызываются с использованием схем сообщений на базе Extensible Markup Language (XML), передаваемых по транспортному каналу. Сервисы поддерживают гетерогенную среду благодаря реализации возможности взаимодействия через описание сообщения/интерфейса. Если компоненты могут интерпретировать сообщение и описание интерфейса, они могут использовать сервис независимо от того, какая технология используется в их основе. На рис. 1 показано общее представление типовой архитектуры сервисного приложения.

Сервисы

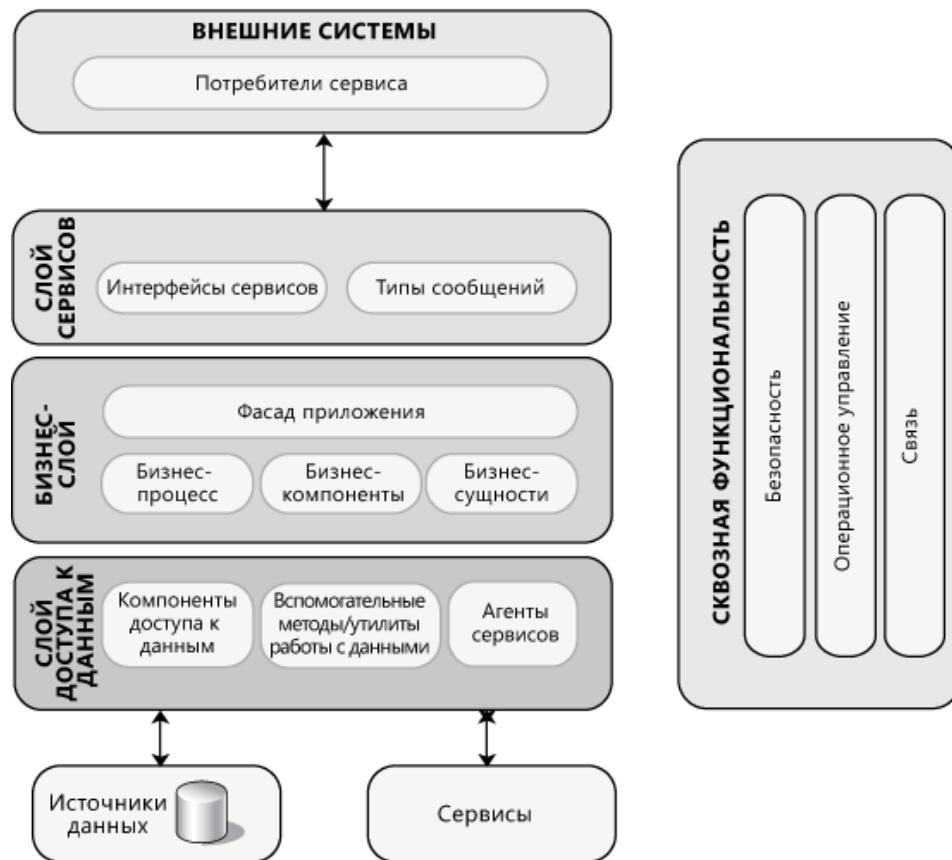


Рис. 42

Типовая архитектура сервисного приложения

Типовое сервисное приложение состоит из трех слоев: слой сервисов, бизнес-слой и слой доступа к данным. Слой сервисов может включать компоненты интерфейсов сервисов, типов сообщений и типов данных; бизнес-слой – компоненты бизнес-логики, бизнес-процесса и бизнес-сущностей; и слой доступа к данным включает компоненты доступа к данным и агентов сервисов. Многослойному дизайну посвящена глава 5, «[Рекомендации по проектированию многослойных приложений](#)». Более подробно о компонентах каждого уровня рассказывает глава 10, «[Рекомендации по проектированию компонентов](#)».

Сервисы являются гибкими по своей природе и могут использоваться в разнообразнейших сценариях и комбинациях. Рассмотрим типовые сценарии:

- **Сервис, предоставляемый через Интернет.** Этот сценарий описывает сервис, используемый рядом клиентов через Интернет. Сюда относятся как сервисы типа «бизнес-бизнес», так и ориентированные на потребителя сервисы. Биржевой Веб-сайт, использующий Веб-сервисы фондовой биржи и предоставляющий котировки акций – один из примеров такого сценария. Решения по аутентификации и авторизации должны приниматься, исходя из границ доверия в Интернете и типов учетных данных. Например, аутентификация по имени пользователя и паролю или применение сертификатов более вероятна в Интернет-сценарии, чем в интранет-сценарии.

- **Сервис, предоставляемый по внутренней сети.** Этот сценарий описывает сервис, используемый по внутренней сети рядом (обычно ограниченным) внутренних или корпоративных клиентов. Система электронного документооборота уровня предприятия – один из примеров такого сценария. Решения по аутентификации и авторизации должны приниматься, исходя из границ доверия во внутренней сети и типов учетных данных. Например, аутентификация Windows с применением Active Directory для хранения пользовательских данных является более вероятным для интранет-сценария, чем для Интернет-сценария.
 - **Сервис, предоставляемый локально на компьютере.** Этот сценарий описывает сервис, потребляемый приложением на локальном компьютере. Решения по защите на транспортном уровне и на уровне сообщений должны приниматься на основании локальных границ доверия и пользователей.
 - **Смешанный сценарий.** Этот сценарий описывает сервис, используемый множеством приложений по Интернету, внутренней сети и/или на локальном компьютере. Сервисное бизнес-приложение (LOB), потребляемое локально насыщенным клиентским приложением и Веб-приложением по Интернету – пример такого сценария.
-

Общие принципы проектирования

При проектировании сервисных приложений необходимо следовать общим рекомендациям, применимым ко всем сервисам. Такими рекомендациями являются проектирование слабо детализированных операций, следование контракту сервиса и готовность к возможному поступлению недействительных запросов или запросов в неверном порядке. Кроме общих рекомендаций, есть специальные рекомендации, которых следует придерживаться при проектировании разных типов сервисов. Например, для Сервисно-ориентированной архитектуры (SOA) необходимо обеспечить стабильность контракта приложения и автономность сервиса. Либо может создаваться приложение, предоставляющее сервисы рабочего процесса, или хранилище операционных данных, обеспечивающее основанный на сервисе интерфейс. При проектировании сервисных приложений руководствуйтесь следующими рекомендациями:

- **Используйте многослойный подход и избегайте тесного связывания слоев.** По возможности распределяйте бизнес-правила и функции доступа к данным по разным компонентам. Обеспечивайте интерфейс бизнес-слоя через абстракцию, которая может быть реализована путем использования открытых интерфейсов объектов, общих описаний интерфейсов, абстрактных базовых классов или через обмен сообщениями. Более подробно абстракция в многослойной архитектуре рассматривается в главе 5, «[Рекомендации по проектированию многослойных приложений](#)».
- **Проектируйте слабо детализированные операции.** При работе с интерфейсом сервиса избегайте использования детализированных вызовов, что может привести к очень низкой производительности. Операции сервиса должны быть слабо детализированными и проецироваться на операции приложения. Применение

шаблона Façade позволит объединять несколько небольших детализированных операций в одну слабо детализированную операцию. Например, для статистических данных необходимо обеспечить операцию, которая будет возвращать все данные за один вызов, чтобы не приходилось делать множество вызовов, каждый из которых будет возвращать лишь подмножество данных.

- **При проектировании контрактов данных обеспечивайте возможность расширения и повторного использования.** Контракты данных должны быть спроектированы с обеспечением возможности их расширения без влияния на потребителей сервиса. Страйтесь создавать используемые сервисом составные типы из стандартных элементов. Слой сервисов должен знать о бизнес-сущностях, используемых бизнес-слоем. Обычно это обеспечивается путем создания для бизнес-сущностей отдельной сборки, которая совместно используется слоем сервисов и бизнес-слоем.
- **При проектировании строго придерживайтесь контракта сервиса.** Слой сервисов должен реализовывать и обеспечивать только ту функциональность, которая оговорена контрактом сервиса, внутренняя реализация и детали сервиса никогда не должны предоставляться потребителям. Также если необходимо изменить контракт сервиса для включения в него новой функциональности, реализованной сервисом, и новые операции и типы не являются обратно совместимыми с существующими контрактами, создавайте новые версии контрактов. Описывайте новые операции, предоставляемые сервисом, в новой версии контракта сервиса и новые типы схем – в новой версии контракта данных. Проектирование контрактов сообщений рассматривается в главе 18, «[Взаимодействие и обмен сообщениями](#)».
- **Проектируйте автономные сервисы.** Сервисы не должны предъявлять никаких требований к их потребителям и не должны делать никаких предположений о потребителе или о том, как он планирует использовать предоставляемый сервис.
- **При проектировании должна быть учтена возможность поступления недействительных запросов.** Никогда не делайте предположения о том, что все поступающие на сервис сообщения будут действительными. Реализуйте логику валидации для проверки всех сообщений на соответствие схемам; и отвергайте или очищайте все недействительные сообщения. Более подробно валидация рассматривается в главе 17, «[Сквозная функциональность](#)». Обеспечьте сервису возможность выявлять и правильно обрабатывать повторные сообщения (идемпотентность) путем реализации широко известных шаблонов или через использование сервисов инфраструктуры. Кроме того, сервис должен уметь обрабатывать сообщения, поступающие в произвольном порядке (коммутативность). Это можно сделать, реализуя дизайн, позволяющий сохранять сообщения и затем обрабатывать их в правильном порядке.
- **Сервисы должны управляться на основе политик и иметь явные границы.** Сервисное приложение должно быть самодостаточным и иметь четкие границы. Доступ к сервису должен быть разрешен только через слой интерфейса сервиса. Сервис должен публиковать политику, описывающую, как потребители могут взаимодействовать с сервисом. Это особенно важно для открытых сервисов, чтобы

обеспечить потребителям возможность проверять политику и требования для взаимодействия.

- **Разделяйте функциональность сервиса и операции инфраструктуры.** Логика сквозной функциональности никогда не должна смешиваться с логикой приложения, поскольку это может привести к реализациям, плохо расширяемым и сложным в обслуживании.
- **Размещайте основные компоненты слоя сервисов в отдельных сборках.** Например, интерфейс, реализация, контракты данных, контракты сервисов, контракты сбоев и трансляторы – каждый из них должен размещаться в собственной сборке.
- **Избегайте использования сервисов данных для предоставления отдельных таблиц базы данных.** Это приведет к слишком детализированным вызовам сервиса и зависимостям между операциями сервиса, что может создать проблемы зависимостей для потребителей сервиса. Кроме того, старайтесь избегать реализации бизнес-правил в сервисах, потому что разные потребители данных будут иметь собственные уникальные срезы доступных данных и правила, и это обусловит ограничения в использовании данных.
- **Обеспечьте при проектировании, чтобы сервисы рабочего процесса использовали интерфейсы, поддерживаемые вашей подсистемой управления рабочим процессом.** Попытки создания собственных интерфейсов могут ограничить типы поддерживаемых операций, при этом потребуется больше усилий для расширения и обслуживания сервисов. Вместо того чтобы добавлять сервисы рабочего процесса в существующее сервисное приложение, спроектируйте автономный сервис, поддерживающий только требования рабочего процесса.

Специальные вопросы проектирования

В следующих разделах предлагаются рекомендации, которые помогут разрешить общие проблемы, возникающие при разработке архитектуры сервисов:

- [Аутентификация](#)
- [Авторизация](#)
- [Бизнес-слой](#)
- [Сетевое взаимодействие](#)
- [Слой доступа к данным](#)
- [Управление исключениями](#)
- [Структура сообщения](#)
- [Конечная точка сообщения](#)
- [Защита сообщений](#)
- [Преобразование сообщений](#)

- [Шаблоны обмена сообщениями](#)
 - [Передача репрезентативного состояния](#)
 - [Слой сервисов](#)
 - [SOAP](#)
 - [Валидация](#)
-

Аутентификация

Стратегия аутентификации сервиса зависит от используемого типа хоста для сервиса. Например, если сервис размещен в Internet Information Services (IIS), можно использовать поддержку аутентификации, предоставляемую IIS. Если сервис размещается с применением Windows Service, должна использоваться аутентификация на основе сообщений или транспорта. При проектировании стратегии аутентификации руководствуйтесь следующими рекомендациями:

- Определите подходящий механизм безопасной аутентификации пользователей, и применяйте аутентификацию при любом пересечении границ доверия. По возможности используйте интегрированные сервисы и механизмы единой регистрации.
 - Учтите последствия применения разных параметров доверия для выполнения кода сервиса.
 - С обычной аутентификацией или при передаче учетных данных в виде простого текста должны использоваться безопасные протоколы, такие как Secure Sockets Layer (SSL). С SOAP-сообщениями применяйте такие механизмы обеспечения безопасности, как Web Services Security¹ (WS-Security).
-

Авторизация

Проектирование эффективной стратегии авторизации имеет большое значение с точки зрения безопасности и надежности сервисного приложения, в противном случае, оно будет уязвимым для разглашения данных, повреждения или подделки данных и несанкционированного получения прав доступа. При проектировании стратегии авторизации руководствуйтесь следующими рекомендациями:

- Задавайте соответствующие права доступа к ресурсам для пользователей, групп и ролей; применяйте детализированную авторизацию при любом пересечении границ доверия. Выполняйте сервисы под учетной записью с минимально необходимыми правами доступа.
- Для защиты URL и файловых ресурсов используйте авторизацию на базе Унифицированного указателя ресурса (Uniform Resource Locator, URL) и/или авторизацию доступа к файлам.

¹ Безопасность Веб-сервисов (прим. переводчика).

- По необходимости ограничивайте доступ к открыто доступным методам сервиса, декларативно описывая разрешения.
-

Бизнес-слой

Бизнес-слой сервисного приложения использует фасад для трансляции операций сервиса в бизнес-операции. Основная цель при проектировании интерфейса сервиса – использовать слабо детализированные операции, которые внутренне могут транслироваться во множество бизнес-операций. Фасад бизнес-слоя отвечает за взаимодействие с соответствующими компонентами бизнес-процесса. При проектировании бизнес-слоя руководствуйтесь следующими рекомендациями:

- Компоненты бизнес-слоя не должны знать о слое сервисов. Бизнес-слой и любой код бизнес-логики не должен иметь зависимостей с кодом слоя сервисов и никогда не должен выполнять код слоя сервисов.
- При поддержке сервисов используйте фасад для бизнес-слоя. Фасад представляет основную точку входа в бизнес-слой. Его задача – прием недетализированных операций и их разложение на множество бизнес-операций. Однако если сервис будет использоваться с локального компьютера или с клиента, который выполняет доступ без пересечения физических границ, можно предоставить и детализированные операции, если это необходимо для клиента.
- Проектируйте бизнес-слой без сохранения состояния. Операции сервиса должны содержать все сведения, включая данные состояния, используемые бизнес-слоем для обработки запроса. Сервис может обрабатывать большое число взаимодействий с потребителями, поэтому попытки сохранять состояние для каждого отдельного потребителя в бизнес-слое могли бы обусловить потребление слишком большого объема ресурсов. Это привело бы к ограничению числа запросов, которые может обрабатывать сервис в любой момент времени.
- Реализуйте все бизнес-сущности в отдельной сборке. Эта сборка представляет разделяемый компонент, который может использоваться и бизнес-слоем, и слоем доступа к данным. Однако обратите внимание, что бизнес-сущности не должны предоставляться через границы сервиса; для передачи данных между сервисами используйте объекты передачи данных (data transfer objects, DTO).

Вопросам реализации бизнес-слоя посвящена глава 7, «[Рекомендации по проектированию бизнес-слоя](#)». Более подробно бизнес-сущности рассматриваются в главе 13, «[Проектирование бизнес-сущностей](#)».

Сетевое взаимодействие

При проектировании стратегии сетевого взаимодействия протокол связи должен выбираться на основании сценария развертывания сервиса. При проектировании стратегии связи руководствуйтесь следующими рекомендациями:

- Если сервис будет развертываться в закрытой сети, можно использовать более эффективный протокол связи Transmission Control Protocol (TCP). Если сервис будет развертываться в публичной сети, следует выбирать протокол Hypertext Transfer Protocol (HTTP).
- Выработайте стратегию обработки ненадежной или неустойчивой связи, возможно, с применением кэширования сообщений с последующей их передачей при возобновлении соединения, а также обработки асинхронных вызовов. Примите решение о том, будет ли связь односторонней или двухсторонней, и есть ли необходимость в применении шаблонов Duplex, Request Response и Request-Reply.
- Используйте динамический подход к формированию URL-адреса конечных точек для обеспечения максимальной гибкости и определите, как будете проверять адреса конечных точек в сообщениях.
- Выберите соответствующие протоколы связи и обеспечьте защиту данных, передаваемых по каналам связи, с помощью шифрования и/или цифровых подписей.

Более подробно протоколы и техники связи рассматриваются в главе 18, «[Взаимодействие и обмен сообщениями](#)».

Слой доступа к данным

Слой доступа к данным сервисного приложения включает функциональность доступа к данным для взаимодействия с внешними источниками данных. Этими источниками могут быть базы данных, другие сервисы, файловая система, списки SharePoint или любые другие приложения, работающие с данными. Согласованность данных является основным условием стабильности и целостности реализации сервиса, неспособность проверить непротиворечивость данных, получаемых сервисом, может привести к размещению в хранилище недействительных данных, к неожиданным исключениям и брешам в системе защиты. Поэтому в реализацию сервиса всегда должны быть включены проверки непротиворечивости данных. При проектировании слоя доступа к данным руководствуйтесь следующими рекомендациями:

- Слой доступа к данным по возможности должен развертываться на одном уровне с бизнес-слоем. Разворачивание слоя доступа к данным на другом физическом уровне потребует сериализации объектов при пересечении ими физических границ.
- Всегда используйте абстракцию при реализации интерфейса слоя доступа к данным. Как правило, это обеспечивается применением шаблонов Data Access или Table Data Gateway, в которых типы входных и выходных данных определены.
- Для каждой таблицы или представления базы данных создайте класс для реализации простых операций Create, Read, Update и Delete (CRUD). Это обеспечивает шаблон Table Module, в котором каждой таблице поставлен в соответствие класс с операциями, взаимодействующими с этой таблицей. Спланируйте обработку транзакций.
- Избегайте применения олицетворения или делегирования для доступа к базе данных, используйте для этого общую сущность, обеспечивая при этом данные удостоверения

пользователя, чтобы процессы протоколирования и аудита могли ассоциировать пользователей с осуществляемыми ими действиями.

Более подробно реализация слоя доступа к данным обсуждается в главе 8, «[Рекомендации по проектированию слоя доступа к данным](#)».

Управление исключениями

Проектирование эффективной стратегии управления исключениями имеет большое значение для обеспечения безопасности и надежности сервисного приложения, в противном случае, оно будет уязвимым для атак Отказа в обслуживании (Denial of Service, DoS) и может допустить разглашение конфиденциальных или критически важных данных. Формирование и обработка исключений является ресурсоемкой операцией, поэтому важно, чтобы стратегия управления исключениями разрабатывалась с учетом влияния на производительность. Хорошим подходом является проектирование централизованного механизма управления исключениями и протоколирования и предоставление контрольных точек, поддерживающих инструментирование и централизованный мониторинг, что облегчит работу системных администраторов. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Убедитесь, что перехватываете необрабатываемые исключения и очищаете ресурсы после возникновения исключений. Не допускайте разглашения конфиденциальных данных в исключениях сервисах, файлах журналов и файлах аудита.
- Перехватывайте только те исключения, которые можете обработать, например, для удаления из них конфиденциальных данных или введения дополнительных данных. Не используйте исключения для управления логикой приложения. Избегайте использования собственных исключений, когда в них нет крайней необходимости.
- Используйте SOAP-элементы Fault или специальные расширения для возвращения данных исключения вызывающей стороне.

Более подробно проектирование стратегии управления исключениями рассматривается в главе 17, «[Сквозная функциональность](#)».

Структура сообщения

Данные, которыми обмениваются сервис и потребитель при взаимодействии, должны быть заключены в сообщение. Формат этого сообщения определяется типами поддерживаемых операций. Например, может происходить обмен документами, выполнение команд или формирование событий. При использовании медленных каналов доставки в сообщение также необходимо включить сведения о сроке действия. При проектировании структуры сообщений руководствуйтесь следующими рекомендациями:

- Выбирайте соответствующие шаблоны для структуры сообщения (такие как Command, Document, Event и Request-Response).
- Разделяйте очень большие объемы данных на меньшие блоки и отправляйте их последовательно.

- Включайте в сообщения с ограничением по времени действия сведения о сроке действия. Сервис должен игнорировать сообщения, срок действия которых истек.
-

Конечная точка сообщения

Конечная точка сообщения представляет подключение, используемое приложением для взаимодействия с сервисом. Реализация интерфейса сервиса обеспечивает конечную точку сообщения. При проектировании реализации сервиса должен быть учтен тип потребляемого сообщения. Кроме того, необходимо предусмотреть ряд сценариев, связанных с обработкой сообщений. При проектировании конечных точек сообщений руководствуйтесь следующими рекомендациями:

- Выбирайте соответствующие шаблоны конечных точек сообщений (такие как Gateway, Mapper (Преобразователь), Competing Consumer и Message Dispatcher).
 - При проектировании предусмотрите сценарии без подключения. Например, для обеспечения гарантированной доставки может понадобиться реализовать кэширование или сохранение сообщений для последующей их доставки. Не допускайте попыток обращения к конечным точкам в условиях отсутствия подключения.
 - Если должны приниматься не все сообщения, реализуйте фильтр для обработки только определенных сообщений.
 - Обеспечьте идемпотентность интерфейса сервиса. Идемпотентность – это ситуация, когда от одного и того же потребителя могут поступать дублирующиеся сообщения, но обрабатываться должно только одно из них. Иначе говоря, идемпотентная конечная точка гарантирует, что из всех дублирующихся сообщений будет обработано только одно, все остальные будут проигнорированы.
 - Обеспечьте коммутативность. Коммутативность – это ситуация, когда сообщения могут поступать в произвольном порядке. Иначе говоря, коммутативная конечная точка гарантирует, что поступающие в произвольном порядке сообщения будут сохранены и впоследствии обработаны в правильном порядке.
-

Захист сообщений

При обмене конфиденциальными данными между сервисом и его потребителем необходимо обеспечить защиту сообщений. Для этого может использоваться защита на транспортном уровне (посредством протоколов IPSec или SSL) или защита на уровне сообщения (посредством шифрования и цифровых подписей). При проектировании стратегии защиты сообщений руководствуйтесь следующими рекомендациями:

- Используйте безопасность на уровне сообщения, если требуется обеспечить сквозную безопасность и существует вероятность наличия между сервисом и вызывающей стороной посредников, таких как серверы и маршрутизаторы. Безопасность на уровне сообщения помогает защитить конфиденциальные данные в сообщениях путем их шифрования. Применение цифровых подписей поможет защититься от отказа

пользователей от действий и повреждения или подделки сообщений. Тем не менее, не забывайте, что все меры по обеспечению безопасности негативно сказываются на производительности.

- Если взаимодействия между сервисом и потребителем осуществляются без посредников, таких как другие серверы и маршрутизаторы, можно использовать защиту на транспортном уровне, такую как протоколы IPSec или SSL. Однако если сообщение проходит через одного или более посредников, всегда должны применяться меры защиты на уровне сообщения. Если используется защита на транспортном уровне, сообщение дешифруется и вновь шифруется на каждом посреднике, что представляет угрозу безопасности.
- Для обеспечения максимальной безопасности используйте оба типа защиты, на транспортном уровне и на уровне сообщения. Защита на транспортном уровне поможет защищать данные заголовков, которые не могут быть зашифрованы при использовании защиты на уровне сообщения.
- При проектировании сервисов для экстрасети или типа «бизнес-бизнес» используйте косвенную аутентификацию на основе сообщений с сертификатами X.509. В сценарии «бизнес-бизнес» сертификат должен быть выпущен коммерческим центром сертификации. Для сервисов экстрасети можно применять сертификаты, выпущенные сервисом сертификации организации.

Преобразование сообщений

При передаче сообщений между сервисом и потребителем существует множество сценариев, когда сообщение требуется преобразовать в формат, понятный потребителю. Обычно это происходит в случаях, когда потребители, не имеющие возможности использовать обмен сообщениями, должны обрабатывать данные, извлекаемые из системы, использующей обмен сообщениями. Для обеспечения доступа к каналу сообщений для таких потребителей могут использоваться адаптеры и трансляторы для преобразования данных сообщения в формат, понятный каждому потребителю. При проектировании стратегии преобразования сообщений руководствуйтесь следующими рекомендациями:

- Выясните, есть ли необходимость в преобразовании. Если такая необходимость имеется, выберите походящие шаблоны для преобразования сообщений. Например, шаблон Normalizer может использоваться для преобразования семантически эквивалентных сообщений в общий формат. Используйте каноническую модель только в случае крайней необходимости.
- Обеспечьте преобразования в соответствующем месте, чтобы избежать повторных обработок и ненужных издержек.
- Определяйте формат сообщения с помощью метаданных и храните эти метаданные во внешнем хранилище.
- Применяйте такие механизмы, как BizTalk Server, который может осуществлять преобразования сообщений для широкой номенклатуры форматов и типов клиентов.

Шаблоны обмена сообщениями

Шаблон обмена сообщениями определяет взаимодействие между сервисом и его потребителем. Это взаимодействие представляет контракт между сервисом и потребителем. Консорциум W3C описывает два шаблона для SOAP-сообщений: Request-Response и SOAP Response. Другая группа стандартизации, OASIS, выработала для сервисов Язык выполнения бизнес-процессов (Business Process Execution Language, BPEL). BPEL определяет процесс для обмена сообщениями бизнес-процесса. Кроме того, другие организации определяют специальные шаблоны обмена сообщениями для обмена сообщениями бизнес-процессов. При проектировании шаблонов обмена сообщениями руководствуйтесь следующими рекомендациями:

- Выбирайте шаблоны, соответствующие имеющимся требованиям и не вводящие дополнительных сложностей. Например, не используйте сложный шаблон обмена сообщениями бизнес-процесса, если достаточно шаблона Request-Response. Для однонаправленного обмена сообщениями используйте шаблон Fire and Forget.
- При использовании техник моделирования бизнес-процесса существует опасность создания шаблонов обмена на основании этапов процесса. Шаблоны должны поддерживать операции, объединяющие этапы процесса.
- Не изобретайте собственные, а применяйте существующие стандарты для шаблонов обмена сообщениями. Это обеспечит создание основанного на стандартах интерфейса, который будет понятен многим пользователям. Иначе говоря, потребители охотнее будут взаимодействовать с основанными на стандартах контрактами, а не изучать и принимать нестандартные контракты.

Передача репрезентативного состояния

Передача репрезентативного состояния (Representational State Transfer, REST) – это архитектурный стиль, основанный на HTTP, работа которого во многом напоминает Веб-приложение. Но вместо пользователя, взаимодействующего с Веб-страницами и перемещающегося по ним, в этом случае приложения взаимодействуют и перемещаются по REST-ресурсам, используя ту же семантику, что и Веб-приложение. В REST ресурс идентифицируется Унифицированным идентификатором ресурса (Uniform Resource Identifier, URI), и действия, которые могут быть осуществлены по отношению к ресурсу, описываются с помощью HTTP-команд, таких как GET, POST, PUT и DELETE. Взаимодействие с REST-сервисом происходит путем выполнения HTTP-операций по отношению к URI, который обычно представлен в форме URL. В результате операции возвращается представление текущего состояния этого ресурса. Кроме того, результат может включать ссылки на другие ресурсы, к которым можно перейти из текущего ресурса.

Наиболее распространенное заблуждение о REST в том, что он подходит только для CRUD-операций с ресурсом. REST может использоваться с любым сервисом, который может быть представлен как конечный автомат. Иначе говоря, если сервис может быть разложен на различимые состояния, такие как извлечен и обновлен, вы можете преобразовать эти состояния в действия и продемонстрировать, как каждое из состояний может привести к одному или более состояниям. Вспомните, как UI Веб-приложения представляет конечный

автомат. При доступе к странице отображаемые данные представляют текущее состояние этих данных. Пользователь может иметь возможность изменять это состояние, обновляя поля формы через операцию POST или переходя к другой странице, используя ссылки, включенные в текущую страницу. Сервис, использующий REST, работает по такой же схеме в том смысле, что приложение может осуществлять операцию GET над ресурсом для получения его текущего состояния, менять состояние ресурса с помощью операции PUT или переходить к другому ресурсу, используя ссылки, предоставляемые текущим ресурсом.

В сервисах, поддерживающих REST, имеется и состояние приложения, и состояние ресурса. Клиент сохраняет состояние всего приложения, тогда как на сервере хранится только состояние ресурса. Каждый запрос клиента к серверу должен включать все необходимые серверу данные для полного понимания данного запроса. Иначе говоря, клиент должен передавать в запросе все данные, касающиеся состояния приложения. Затем, на основании ответов сервера, клиент может принимать решения о том, как изменять состояние ресурса. Передача состояния приложения при каждом запросе обеспечивает возможность масштабирования приложения, поскольку в этом случае можно добавлять множество идентичных Веб-серверов и балансировать нагрузку таким образом, что клиенту нет необходимости привязываться к одному конкретному серверу или состоянию любого совместно используемого приложения.

REST-сервис обладает качествами безопасности и идемпотентности. Безопасность означает способность многократно повторять запрос и получать один и тот же результат без каких-либо побочных эффектов. Под идемпотентностью подразумевается такое поведение, когда многократное повторение вызова приводит к тем же результатам, что и однократный вызов. Эти качества повышают устойчивость к ошибкам и надежность, потому что, даже несмотря на ненадежность HTTP, вы можете безопасно повторять запрос, если сервис не отвечает или возвращает ошибку.

При проектировании REST-ресурсов руководствуйтесь следующими рекомендациями:

- Используйте диаграмму состояния для моделирования и описания ресурсов, которые будет поддерживать REST-сервис. Не используйте состояние сеанса в REST-сервисе.
- Выработайте подход для идентификации ресурсов. Хорошей практикой является использование значимых имен начальных точек и уникальных идентификаторов REST в качестве части общего пути конкретных экземпляров ресурсов. Избегайте включения действий в URI с помощью значений QueryString.
- Определитесь с тем, должны ли поддерживаться несколько реализаций для разных ресурсов. Например, решите, должен ли ресурс поддерживать формат XML, Atom или JavaScript Object Notation (JSON), и сделайте его частью запроса к ресурсу. Ресурс может предоставляться и как (например) <http://www.contoso.com/example.atom>, и как <http://www.contoso.com/example.json>. Не используйте значения QueryString для определения действий с URI. Все действия должны основываться на HTTP-операции, выполняемой по отношению к URI.
- Не злоупотребляйте операцией POST. Хорошей практикой является использование конкретных HTTP-операций, таких как PUT или DELETE, это обеспечит создание дизайна на основе ресурсов и использование унифицированного интерфейса.

- Применение протокола приложений HTTP позволит использовать общую инфраструктуру Веб (кэширование, аутентификацию, общие типы представления данных и т.д.).
 - Обеспечьте безопасность запросов GET в том смысле, чтобы они всегда возвращали один и тот же результат. Сделайте запросы PUT и DELETE идемпотентными, т.е. повторяющиеся идентичные запросы должны приводить к тому же эффекту, что и одиночный запрос.
-

Слой сервисов

Слой сервисов включает компоненты, определяющие и реализующие сервисы для приложения. В частности, этот слой содержит интерфейс сервиса (составленный из контрактов), реализацию сервиса и трансляторы, которые преобразуют внутренние бизнес-сущности в и из внешних контрактов данных. При проектировании слоя сервисов руководствуйтесь следующими рекомендациями:

- Не реализуйте бизнес-правила в слое сервисов. Слой сервисов должен заниматься только управлением запросами к сервису и преобразованием контрактов данных в сущности для использования бизнес-слоем.
 - Доступ к слою сервисов должен определяться политиками. Политики обеспечивают потребителям сервиса способ описания требований к подключению и безопасности, а также других деталей, связанных с взаимодействием с сервисом.
 - Размещайте основные компоненты слоя сервисов в отдельных сборках. Например, интерфейс, реализация, контракты данных, контракты сервисов, контракты сбоев и трансляторы – для каждого из них должна быть создана собственная сборка.
 - Взаимодействие с бизнес-слоем должно осуществляться только через общеизвестный открытый интерфейс. Слой сервисов никогда не должен обращаться к компонентам базовой бизнес-логики.
 - Слой сервисов должен знать, какие бизнес-сущности используются бизнес-слоем. Обычно это обеспечивается созданием отдельной сборки, в которую помещаются все бизнес-сущности, совместно используемые слоем сервисов и бизнес-слоем.
-

SOAP

SOAP – протокол обмена сообщениями, при использовании которого сообщение формируется как XML-конверт, содержащий заголовок и тело. В заголовке могут размещаться сведения, внешние по отношению к осуществляющей сервисом операции. Например, заголовок может включать данные, касающиеся безопасности, транзакции или маршрутизации. Тело содержит контракты, представленные в форме XML-схем, которые определяют сервис и действия, которые он может осуществлять. По сравнению с REST, SOAP обеспечивает большую гибкость в выборе протоколов, позволяя использовать более высокопроизводительные протоколы, такие как TCP. SOAP поддерживает стандарты WS-*, включая безопасность, транзакции и надежность. Безопасность и надежность сообщения гарантирует, что сообщения не только

достигнут точки назначения, но также что они не будут прочитаны или изменены в ходе передачи. Транзакции обеспечивают возможность группировки операций и отката в случае сбоя.

SOAP полезен при взаимодействии типа RPC между сервисами или отдельными слоями приложения. Он исключителен в обеспечении возможности взаимодействия между новыми и устаревшими системами во внутренней сети. Слой сервисов может быть размещен поверх старой системы, делая возможным взаимодействие с системой через API без необходимости ее переработки для предоставления доступа на базе модели REST-ресурсов. SOAP также пригодится в случаях, когда данные активно передаются одной или более системам, которые могут часто менять протоколы связи. Он также может использоваться, если требуется непрозрачно инкапсулировать данные или объекты и затем сохранять или передавать их другой системе.

Если хотите обеспечить масштабируемость приложения, применяя Веб-фермы или балансировку нагрузки, избегайте сохранения состояния сеансов на сервере. Сохранение сеансов на сервере означает, что данный конкретный сервер должен обслуживать клиента в течение всего сеанса или, в случае балансировки нагрузки, должен передавать данные сеанса другому серверу. Передача состояния сеанса между серверами намного усложняет реализацию сценариев обработки отказов и масштабирования.

При проектировании SOAP-сообщений руководствуйтесь следующими рекомендациями:

- В SOAP-конверте SOAP-заголовок является необязательным элементом, тогда как наличие SOAP-тела обязательно. В любых схемах сообщений избегайте использования составных типов.
- Используйте для ошибок SOAP-ошибки, не полагайтесь на стандартный механизм обработки ошибок. При возвращении данных ошибки SOAP-ошибка должна быть единственным дочерним элементом SOAP-тела.
- Чтобы обеспечить обработку блока SOAP-заголовка, задайте атрибуту **mustUnderstand** (должен интерпретироваться) блока значение «true» или «1». Ошибки, возникающие при обработке SOAP-заголовка, должны возвращаться как SOAP-ошибка в элементе SOAP-заголовка.
- Изучите и используйте стандарты WS-*. Эти стандарты обеспечивают единые правила и методы решения проблем, обычно возникающих в архитектуре, использующей обмен сообщениями.

Валидация

Эффективная стратегия валидации ввода и данных является критически важной для обеспечения безопасности приложения. Для него обязательно должны быть определены правила валидации данных, поступающих от потребителей сервиса. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- Проверяйте все данные, поступающие в интерфейс сервиса, включая связанные с сообщением поля данных, и возвращайте информативные сообщения об ошибке в случае сбоя валидации. Для проверки входящих сообщений используйте XML-схемы.

- Проверяйте все вводимые данные на наличие опасного или злонамеренного содержимого и учитывайте то, как предполагается использовать данные. Например, если данные будут использоваться для запросов к базе данных, необходимо обеспечить защиту базы данных от атак внедрением SQL-кода. Это можно реализовать путем применения для доступа к базе данных совместно используемых процедур или параметризованных запросов.
- Выработайте стратегии применения подписей, шифрования и кодирования.
- Определитесь с границами доверия между слоями и где бы то ни было, чтобы обеспечить валидацию данных при любом пересечении этих границ. Однако если валидации, выполняемой в других слоях достаточно, чтобы считать данные пользующимися доверием, повторной проверки таких данных не требуется.

Более подробно методики валидации рассматриваются в главе 17, «[Сквозная функциональность](#)».

Вопросы выбора технологий

При проектировании сервиса примите во внимание следующие вопросы, касающиеся выбора технологий:

- Для простоты используйте ASMX, но только если доступен Веб-сервер, выполняющий IIS.
- Используйте WCF-сервисы, если необходимы дополнительные возможности, такие как надежные сеансы и транзакции, трассировка действий, протоколирование сообщений, счетчики производительности и поддержка множества транспортных протоколов.
- Если при использовании Веб-сервисов ASP.NET требуется обеспечить безопасность на уровне сообщений и передачу двоичных данных, используйте Web Service Extensions (WSE).
- Если решено использовать WCF, продумайте следующее:
 - Если требуется обеспечить возможность взаимодействия с не-WCF или не-Windows клиентами, используйте HTTP-транспорт, основанный на спецификациях SOAP.
 - Если требуется поддерживать клиентов в рамках внутренней сети, используйте протокол TCP и бинарное кодирование сообщений с безопасностью на транспортном уровне и аутентификацией Windows.
 - Для поддержки WCF-клиентов на том же компьютере используйте протокол именованных каналов и бинарное кодирование сообщений.
 - Используйте контракты с явной, а не неявной оболочкой сообщений. Это позволит передавать контракты сообщений на вход и выход операций, следствием чего станет возможность расширять контракты данных, включенные в контракт сообщения, без влияния на контракт сервиса.

Вопросы развертывания

Для проектирования сервисных приложений, как правило, используется многослойный подход, при котором слой интерфейса сервиса, бизнес-слой и слой доступа к данным отделены друг от друга. Для сервисного приложения может применяться распределенное развертывание абсолютно так же, как для любого другого типа приложения. Сервисы могут развертываться на клиенте, на одном сервере, на множестве серверов предприятия. Однако при развертывании сервисного приложения необходимо не забывать о вопросах производительности и безопасности, как в любом распределенном сценарии, и учитывать все ограничения, налагаемые средой производственной эксплуатации. При развертывании сервисного приложения руководствуйтесь следующими рекомендациями:

- По возможности размещайте слой сервисов на одном уровне с бизнес-слоем, это обеспечит лучшую производительность приложения.
- Если сервис физически располагается в одном слое с потребителем сервиса, применяйте для связи именованные каналы или совместно используемую память.
- Если доступ к сервису осуществляется только приложениями локальной сети, используйте протокол связи TCP.
- Конфигурируйте хост сервиса для использования безопасности на транспортном уровне, только если потребители имеют прямой доступ к сервису и запросы не проходят через посредников.
- Включайте трассировку и компиляцию в режиме отладки для всех сервисов только на время разработки и тестирования.

Более подробно шаблоны и сценарии развертывания рассматриваются в главе 19, [«Физические уровни и развертывание»](#).

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Сетевое взаимодействие, Непротиворечивость данных, Структура сообщения, Конечная точка сообщения, Защита сообщений, Преобразование сообщений, REST, Интерфейс сервиса и SOAP, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Сетевое взаимодействие	<p>Duplex. Двунаправленный обмен сообщениями, при котором и сервис, и клиент отправляют сообщения друг другу независимо и без учета того, какой шаблон используется, One-Way (Однонаправленный) или Request-Reply (Запрос-ответ)</p> <p>Fire and Forget. Однонаправленный обмен сообщениями, используемый в случаях, когда нет необходимости ожидать ответа.</p> <p>Reliable Sessions. Надежная передача сообщений из конца в конец между</p>

	<p>источником и точкой назначения, не зависящая от количества или типа посредников между конечными точками.</p> <p>Request Response. Механизм двунаправленного обмена сообщениями, при котором клиент ожидает ответа на каждое отправленное сообщение</p>
Непротиворечивость данных	<p>Atomic Transactions (Атомарные транзакции). Транзакции, область действия которых ограничена одной операцией сервиса.</p> <p>Cross-service Transactions (Межсервисные транзакции). Транзакции, которые могут охватывать множество сервисов.</p> <p>Long-running transactions (Длительные транзакции). Транзакции, являющиеся частью бизнес-процесса.</p>
Структура сообщения	<p>Command Message. Структура сообщения, используемая для поддержки команд.</p> <p>Document Message. Структура, используемая для передачи документов или структуры данных между приложениями.</p> <p>Event Message. Структура, обеспечивающая надежное асинхронное уведомление о событиях между приложениями.</p> <p>Request-Reply. Запрос и отклик передаются по разным каналам.</p>
Конечная точка сообщения	<p>Competing Consumer. Задает несколько потребителей для одной очереди сообщений и заставляет их конкурировать за право обрабатывать сообщения, что позволяет обменивающемуся сообщениями клиенту обрабатывать множество сообщений одновременно</p> <p>Durable Subscriber. Чтобы обеспечить гарантированную доставку в сценарии без подключения, сообщения сохраняются и затем предоставляются для доступа клиенту при подключении к каналу сообщений.</p> <p>Idempotent Receiver. Гарантирует, что сервис обрабатывает сообщение только один раз.</p> <p>Message Dispatcher. Компонент, рассылающий сообщения множеству потребителей.</p> <p>Messaging Gateway. Инкапсулирует вызовы, осуществляемые посредством обмена сообщениями, в один интерфейс, чтобы отделить их от остального кода приложения.</p> <p>Messaging Mapper. Преобразует запросы в бизнес-объекты для входящих сообщений и выполняет обратный процесс для преобразования бизнес-объектов в ответные сообщения.</p> <p>Polling Consumer. Потребитель сервиса, который проверяет канал на наличие сообщений через равные промежутки времени.</p> <p>Service Activator. Сервис, принимающий асинхронные запросы для вызова операций в компонентах бизнес-слоя.</p> <p>Selective Consumer. Потребитель сервиса, использующий фильтры для получения только сообщений, соответствующих определенному критерию.</p> <p>Transactional Client. Клиент, который может реализовать транзакции при взаимодействии с сервисом.</p>
Защита сообщений	<p>Data Confidentiality. Использует шифрование на уровне сообщений для защиты конфиденциальных данных в сообщении.</p> <p>Data Integrity. Гарантированно обеспечивает защиту сообщений от повреждения или подделки при передаче.</p>

	<p>Data Origin Authentication. Проводит проверку источника сообщения как расширенный метод обеспечения целостности данных.</p> <p>Exception Shielding. При возникновении исключения предотвращает разглашение сервисом данных о его внутренней реализации.</p> <p>Federation. Интегрированное представление данных, распределенных по многим сервисам и потребителям.</p> <p>Replay Protection. Обеспечивает идемпотентность сообщения, предотвращая возможность его перехвата и многократного выполнения злоумышленниками.</p> <p>Validation. Проверяет содержимое и значения сообщений для обеспечения защиты сервиса от неправильно сформированного или злонамеренного содержимого.</p>
Преобразование сообщений	<p>Canonical Data Mapper. Использует общий формат данных для осуществления преобразований между двумя несопоставимыми форматами данных.</p> <p>Claim Check. Извлекает данные из постоянного хранилища по необходимости.</p> <p>Content Enricher. Дополняет сообщения недостающими данными, полученными из внешнего источника данных.</p> <p>Content Filter. Удаляет конфиденциальные данные из сообщения и максимально сокращает объем сетевого трафика, удаляя ненужные данные из сообщения.</p> <p>Envelope Wrapper. Оболочка сообщений, включающая данные заголовка, используемые, например, для защиты, маршрутизации или аутентификации сообщения</p> <p>Normalizer. Преобразует или трансформирует данные в общий формат обмена, если организации используют разные форматы.</p>
REST	<p>Behavior. Применяется к ресурсам, выполняющим операции. Обычно такие ресурсы не содержат состояния и поддерживают только операцию POST.</p> <p>Container. Создает шаблон сущности, обеспечивая средства для динамического добавления и/или обновления вложенных ресурсов.</p> <p>Entity. Ресурсы, чтение которых может быть осуществлено операцией GET, но изменение возможно только с помощью операций PUT и DELETE.</p> <p>Store. Обеспечивает возможность создания и обновления сущностей с помощью операции PUT.</p> <p>Transaction. Ресурсы, поддерживающие транзакционные операции.</p>
Интерфейс сервиса	<p>Façade. Реализует унифицированный интерфейс для набора операций, чтобы обеспечить упрощенный интерфейс и уменьшить связанность систем.</p> <p>Remote Façade. Создает обобщенный унифицированный интерфейс для набора операций или процессов в удаленной подсистеме, обеспечивая обобщенный интерфейс для детализированных операций, чтобы упростить использование этой подсистемы и свести до минимума вызовы по сети</p> <p>Service Interface. Программный интерфейс, который может использоваться другими системами для взаимодействия с сервисом.</p>
SOAP	Data Contract. Схема, определяющая структуры данных, передаваемые с запросом к сервису.

	<p>Fault Contract. Схема, определяющая ошибки или сбои, которые могут быть возвращены из запроса к сервису.</p> <p>Service Contract. Схема, определяющая операции, которые может осуществлять сервис.</p>
--	---

Более подробно шаблоны Duplex и Request Response рассматриваются в статье «*Designing Service Contracts*» по адресу <http://msdn.microsoft.com/en-us/library/ms733070.aspx>.

Более подробно шаблон Request-Reply рассматривается в статье «*Request-Reply*» по адресу <http://www.eaipatterns.com/RequestReply.html>.

Более подробно шаблоны Atomic и Cross-service Transaction рассматриваются в статье «*WS-* Specifications*» по адресу <http://www.ws-standards.com/ws-atomictransaction.asp>.

Более подробно шаблоны Command, Document Message, Event Message, Durable Subscriber, Idempotent Receiver, Polling Consumer и Transactional Client рассматриваются в материале «*Messaging Patterns in Service-Oriented Architecture, Part I*» по адресу <http://msdn.microsoft.com/en-us/library/aa480027.aspx>.

Более подробно шаблоны Data Confidentiality и Data Origin Authentication рассматриваются в материале «*Chapter 2: Message Protection Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/aa480573.aspx>.

Более подробно шаблоны Replay Detection, Exception Shielding и Validation рассматриваются в материале «*Chapter 5: Service Boundary Protection Patterns*» по адресу <http://msdn.microsoft.com/en-us/library/aa480597.aspx>.

Более подробно шаблоны Claim Check, Content Enricher, Content Filter и Envelope Wrapper рассматриваются в материале «*Messaging Patterns in Service Oriented Architecture, Part 2*» (Шаблоны обмена сообщениями в сервисно-ориентированной архитектуре, Часть 2) по адресу <http://msdn.microsoft.com/en-us/library/aa480061.aspx>.

Более подробно шаблон Remote Façade рассматривается в статье «*P of EAA: Remote Façade*» по адресу <http://martinfowler.com/eaaCatalog/remoteFacade.html>.

Более подробно шаблоны REST, такие как Behavior, Container и Entity, рассматриваются в статье «*REST Patterns*» (Шаблоны REST) по адресу http://wiki.developer.mindtouch.com/REST/REST_Patterns.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- Более подробно распределенные системы рассматриваются в статье «*Enterprise Solution Patterns Using Microsoft .NET - Distributed Systems Patterns*» (Шаблоны корпоративных решений для Microsoft .NET – шаблоны распределенных систем) по адресу <http://msdn.microsoft.com/en-us/library/ms998483.aspx>.

- Сценариям Enterprise Service Bus посвящена статья «*Microsoft BizTalk ESB Toolkit*» по адресу <http://msdn.microsoft.com/en-us/library/dd897973.aspx>.
 - Более подробно шаблоны интеграции обсуждаются в статье «*Prescriptive Architecture Integration Patterns*» (Нормативные шаблоны интеграции архитектуры) по адресу <http://msdn.microsoft.com/en-us/library/ms978729.aspx>.
 - Более подробно шаблоны сервисов рассматриваются в статье «*Enterprise Solution Patterns Using Microsoft .NET - Services Patterns*» (Шаблоны корпоративных решений для Microsoft .NET – шаблоны сервисов) по адресу <http://msdn.microsoft.com/en-us/library/ms998508.aspx>.
 - Более подробно шаблоны безопасности для Веб-сервисов рассматриваются в статье «*Web Service Security*» по адресу <http://msdn.microsoft.com/en-us/library/aa480545.aspx>.
-

26

Проектирование размещаемых сервисов и сервисов в облаке

Обзор

Данная глава обращается к новым и активно развивающимся технологиям и подходам создания и использования сервисов и приложений, размещаемых удаленно. Такие сервисы и приложения доступны через Интернет и выполняются в среде, которую обычно называют *облаком (cloud)*, отсюда широко используемый термин *обработка данных в облаке (cloud computing)*. Поставщики решений для облака и поставщики услуг их размещения, как правило, обеспечивают готовые сервисные приложения с различными уровнями настраиваемости. В качестве альтернативы можно создать собственное приложение и разместить его либо в собственных системах, либо в облаке у провайдера услуг размещения.

Идея создания или использования сервисов и приложений, размещенных удаленно, все более привлекает и независимых поставщиков ПО (*independent software vendors, ISV*), и корпорации как способ сокращения затрат, повышения эффективности и расширения возможностей. Данная глава поможет понять природу и пути использования размещаемых в облаке сервисов и приложений. В ней описываются преимущества и типовые проблемы, возникающие при проектировании, а также ограничения и вопросы выбора технологий при построении и использовании этого типа приложений.

Обработка данных в облаке

Во многих отношениях обработка данных в облаке является результатом естественной эволюции инфраструктуры обработки данных и моделей приложений для создания и использования масштабируемых распределенных решений. С развитием методик построения приложений такого типа, развивались и возможности инфраструктуры, в которой они выполняются. Это взаимоусиливающее развитие обеспечило условия, когда инфраструктура может подготавливаться и обслуживаться во многом независимо от приложений, которые в ней размещаются. Это, в свою очередь, позволяет приложениям использовать преимущества поддерживаемых инфраструктурой сервисов и возможностей, уделяя при этом основное внимание бизнес-функциональности.

Многие организации смогли ощутить суммарные преимущества применения масштабируемых моделей приложений и соответствующих инфраструктур локально, в собственных данных центрах. Тем не менее, основной причиной восторга вокруг обработки данных в облаке является возможность использования внешней инфраструктуры для удаленного размещения приложений. Поставщик инфраструктуры занят вопросами оборудования, сетевого взаимодействия, энергоснабжения, охлаждения, а также операционной средой, которая обеспечивает управляемость, надежность и масштабируемость приложения, предоставляя организации возможность сосредоточиться на бизнес-функциональности приложения. Это дает множество преимуществ с точки зрения сокращения капиталовложений и эксплуатационных расходов, повышения пропускной способности, масштабируемости и доступности.

Чтобы использовать эти преимущества, архитектура размещаемых в облаке приложений, как правило, должна создаваться согласно определенной модели приложения. Это позволяет поставщику услуг размещения обобщать и оптимизировать свою операционную среду, обеспечивая управляемость, надежность или масштабируемость приложения.

Разные провайдеры размещения в облаке предъявляют разные требования к модели приложений. Некоторые используют виртуальные машины, когда приложение разрабатывается и упаковывается вместе с образом операционной системы и зависимыми инфраструктурами времени выполнения. Другие придерживаются модели приложения, которая обеспечивает более высокоуровневые абстракции для доступа к данным и хранения (как описывается позже в данной главе), а также для вычислений и взаимодействия. Тогда как третий придерживаются более высокоуровневых моделей приложений, основывающихся на приложениях с широкими возможностями настройки, которые направлены на определенную вертикальную функциональность приложения, такую как Планирование бизнес-ресурсов (Enterprise Resource Planning, ERP) или Управление отношениями с клиентами (Customer Relationship Management, CRM). Каждый из упомянутых подходов обладает определенными преимуществами и недостатками.

Более того, некоторые размещаемые удаленно приложения являются самодостаточными и предназначаются для пользователей, взаимодействующих с приложением через реализованный специально для этого UI. Некоторые из этих приложений поддерживают сервисы, таким образом, они одновременно обеспечивают UI и предоставляют собственную функциональность через API (часто предоставляемый через такие стандартные протоколы, как REST или SOAP), т.е. могут быть интегрированы в другие приложения, которые, в свою очередь, могут размещаться либо локально, либо удаленно. Некоторые размещаемые удаленно сервисы специально создаются для обеспечения функциональности, интегрируемой в другие приложения, и вообще не предоставляют UI.

Сервисы в облаке, как правило, можно разделить на такие категории, как сервисы хранения/обработки, бизнес-сервисы и сервисы розничной/оптовой торговли. Приведем общие примеры таких удаленных сервисов:

- **Бизнес-сервисы** – это такие сервисы, как биржевые сводки и котировки акций, системы счетов и платежей, средства обмена данными, сервисы продаж и порталы бизнес-информации.

- **Сервисы розничной/оптовой торговли** – это такие сервисы, как каталоги, системы запроса и заказа со склада, информация о погоде и трафике, карты и торговые порталы.
- **Сервисы хранения/обработки** – это такие сервисы, как системы хранения и обработки данных, системы резервного копирования данных, системы управления версиями, и также сервисы обработки для научных и технических задач.

Эти удаленные сервисы могут использоваться программами, выполняемыми локально, в data-центре организации или на компьютере пользователя, который может быть настольным компьютером либо любым другим устройством с поддержкой Интернета. Как правило, для реализации таких сценариев требуется привлечение и совместное использование различных технологий и методик; такой подход получил название программное обеспечение + сервисы (Software plus Services, S+S). Под S+S понимается такой подход к разработке приложений, который объединяет размещаемые сервисы и локально выполняемое ПО. Сочетание удаленных сервисов и выполняющегося локально ПО с насыщенными интегрированными интерфейсами может обеспечить более полное и эффективное решение, чем традиционные локальные приложения. S+S является результатом эволюции нескольких технологий, включая Сервисно-ориентированную архитектуру (Service Oriented Architecture, SOA), ПО как сервис (Software as a Service, SaaS), Платформа как сервис (Platform as a Service, PaaS) и сообщество-ориентированные архитектурные подходы Web 2.0.

Обработка данных в облаке – новая область. В данной главе представлены некоторые преимущества обработки данных в облаке и самые общие вопросы построения архитектуры, которые необходимо учитывать при создании или использовании размещаемых в облаке приложений и сервисов. В ближайшем будущем инфраструктуры, инструменты и среды размещения будут становиться все более доступными, что будет способствовать решению этих проблем.

Терминология размещаемых сервисов и сервисов в облаке

Рассмотрим некоторые термины, часто встречающиеся в данной главе, а также в сценариях с использованием сервисов в облаке и размещаемых сервисов:

- **Стандартный блок сервиса.** Сервис, созданный для использования или интеграции с другими приложениями или сервисами. Примером является сервис хранилища или размещаемый Сервис маркеров доступа (Security Token Service, STS), такой как Сервис управления доступом (Access Control Service) и Сервисная платформа Azure (Azure Services Platform).
- **Среда размещения в облаке.** Среда, обеспечивающая основную среду выполнения для размещения приложений, а также, возможно, стандартные блоки сервисов, бизнес-сервисы, сервисы социальных сетей и вспомогательные сервисы, такие как сервисы учета, оплаты услуг и управления.
- **Собственное приложение.** Приложение, создаваемое самостоятельно, как правило, для выполнения конкретной задачи, сценария или процесса; часто оно реализует

необходимую возможность, которая не может быть позаимствована у сторонних производителей.

- **Размещенное приложение.** Приложение (коробочное или собственное), размещенное как сервис. Оно может размещаться внутренне в вашей системе или внешне у партнера или поставщика услуг размещения.
- **Коробочное приложение.** Приложение, созданное сторонним производителем или поставщиком, которое может предоставлять только ограниченные возможности настройки через конфигурацию или подключаемые модули.
- **Платформа как сервис (Platform as a Service, PaaS).** Базовая операционная система и необязательные подключаемые стандартные блоки сервисов, которая позволяет выполнять собственные приложения или приложения сторонних производителей, полученные от поставщиков, в удаленной среде размещения в облаке.
- **ПО как сервис (Software as a Service, SaaS).** Приложения, осуществляющие комплексные бизнес-задачи или реализующие бизнес-сервисы, которые можно использовать как сервисы без каких-либо требований к внутренней реализации приложения, кроме композиции и UI.

Преимущества приложений в облаке

Размещаемые в облаке приложения и сервисы могут быть очень выгодными для ISV, а также для компаний, поставляющих сервисы или услуги размещения. Также такие приложения обеспечивают преимущества большим корпорациям, как правило, являющимся потребителями размещаемых решений и решений в облаке.

Преимущества для ISV и поставщиков услуг размещения

Основными преимуществами для ISV и компаний-поставщиков услуг размещения, создающих и предлагающих решения в облаке, являются:

- **Гибкость архитектуры.** Поставщики могут предлагать своим клиентам широкий диапазон вариантов развертывания, включая размещение необходимых им сервисов и предоставление пользователям возможности выбирать из ряда готовых функций или выбирать то, какие функции приложения они будут реализовывать самостоятельно. Это может сократить обязательства по реализации архитектуры для конечных пользователей, разрабатывающих сервисы.
- **Насыщенное взаимодействие с пользователем.** ISV и поставщики сервисов могут предлагать своим клиентам более насыщенное взаимодействие с пользователем за счет применения существующих специализированных сервисов (таких как Virtual Earth). Поставщики услуг размещения могут сочетать свои предложения с любыми другими сервисами в облаке, чтобы расширить свои услуги, а также упростить для конечных пользователей задачу по интеграции сервисов.
- **Повсеместный доступ.** Сервисы в облаке сохраняют данные и состояние пользователя и выполняют синхронизацию при повторном подключении пользователя из любого

местоположения. Такое поведение поддерживают как сценарии без подключения, так и сценарии без постоянного подключения, что особенно полезно для мобильных устройств, для которых невозможно обеспечить постоянное подключение или полосу пропускания.

ISV и поставщики услуг размещения сервисов могут также выходить на рынок с коммерческими предложениями для использования возможностей монетизации услуг. Приведем некоторые примеры:

- Поставщики могут воспользоваться незанятой нишней на рынке и предложить продукт, в настоящее время недоступный или доступный не повсеместно, или использовать облако для предложения более дешевых версий своих продуктов, что позволит защитить дорогостоящую лицензию.
 - Молодые компании могут использовать размещение в облаке для минимизации исходных капиталовложений и для использования свойств облака, таких как эластичность (способность расширяться по необходимости без больших исходных инвестиций).
 - Поставщики и пользователи могут создавать приложения, быстрее обеспечивающие прибыль за счет использования уже доступных вспомогательных служб. Например, они уже могут пользоваться системами платежей и бухгалтерского учета в облаке. Пользователи даже могут создавать виртуальные магазины, и для этого не понадобится больших инвестиций в ИТ-оборудование и организацию сети.
-

Преимущества корпоративных потребителей сервисов

Рассмотрим основные преимущества, предлагаемые предприятиям решениями в облаке:

- **Гибкость архитектуры.** Собственные разработчики могут создавать законченные решения, сочетающие сервисы в облаке с локальным кодом приложения и собственными сервисами. Отделы ИТ могут выбирать, какие функции приложения будут реализовывать самостоятельно, и докупать остальные необходимые сервисы.
- **Экономия денежных средств и времени.** Отделы ИТ могут выбирать наиболее подходящие сервисы в облаке для каждой задачи и комбинировать их для предоставления полнофункциональных приложений, сокращая при этом время и затраты на разработку. Кроме того, сокращение требований к собственной ИТ-инфраструктуре упрощает управление, безопасность и снижает затраты на обслуживание.
- **Эффективность, обусловленная ростом масштабов использования.** Компании могут сэкономить на использовании обычных отраслевых возможностей, дешевизна которых будет обусловлена ростом масштабов использования, и сосредоточиться на основной деятельности. Положительный эффект масштаба, обеспечиваемый размещаемыми приложениями, зависит от ряда факторов, включая меньшие затраты на собственную инфраструктуру и лучшее использование оборудования, что обеспечивает возможности снижения эксплуатационных расходов. Однако

необходимо найти сбалансированное соотношение между выгодами от роста масштабов и утратой контроля, что неизбежно при переходе от локальных к полностью удаленным приложениям.

- **Возможность автономной работы.** Облако может выступать в роли концентратора для изменяющих местопребывание пользователей. Данные и состояние пользователя могут храниться в облаке и синхронизироваться при повторном подключении пользователя. Пользователи могут свободно выполнять доступ с разных устройств (настольного компьютера, мобильных клиентов и пр.) с минимальными перенастройками сети.
-

Вопросы проектирования

При проектировании размещаемых сценариев и сценариев для облака и перед ISV, и перед корпоративными клиентами возникает ряд общих вопросов, которые можно разделить по определенным категориям. При разработке своей стратегии размещаемых сервисов и сервисов в облаке обратите внимание на следующие аспекты:

- [Изоляция и совместное использование данных](#)
 - [Безопасность данных](#)
 - [Хранение и расширяемость данных](#)
 - [Управление удостоверениями](#)
 - [Многотенантная архитектура](#)
 - [Локально или удаленно, создавать или покупать](#)
 - [Производительность](#)
 - [Композиция сервисов](#)
 - [Интеграция сервисов](#)
 - [Управление сервисами](#)
-

Изоляция и совместное использование данных

Поставщики услуг размещения могут реализовывать изолированные и совместно используемые базы данных и схемы баз данных. Существует три основные модели:

- **Отдельные базы данных.** Каждый пользователь имеет собственную базу данных с собственными схемами данных. Этот сценарий имеет преимущество простоты реализации, но если количество пользователей каждого сервера баз данных будет относительно невелико, это приведет к потере эффективности и быстрому повышению стоимости обслуживания инфраструктуры. Такой вариант является наиболее подходящим, если пользователи предъявляют особые требования по изоляции или безопасности данных, для обеспечения которых можно потребовать дополнительной платы.

- **Совместно используемые базы данных с разными схемами.** Все пользователи работают с одной базой данных, но имеют разные наборы предопределенных полей. Такой подход также прост в реализации, обеспечивает увеличение количества пользователей каждого сервера баз данных и повышение эффективности использования базы данных. Однако обычно он приводит к разряженному наполнению таблиц базы данных. Этот вариант наиболее полезен, если хранение данных для разных пользователей в одной таблице (смешивание) допустимо с точки зрения безопасности и изоляции, и когда заранее известно, какие поля потребуются.
- **Совместно используемые базы данных с совместно используемой схемой.** Все пользователи работают с одной и той же базой данных, и для хранения расширений данных используются специальные техники. Преимущество этого подхода в том, что вы можете предложить практически неограниченное количество собственных полей. Однако намного сложнее реализовать индексацию, поиск, запросы и обновление. Лучше всего такой вариант применять, если хранение данных для разных пользователей в одинаковых таблицах (смешивание) допустимо с точки зрения безопасности и изоляции, но сложно заранее предсказать, какие собственные поля потребуются.

В следующей таблице представлены преимущества и требования для трех описанных выше моделей изоляции и совместного использования. Преимущества и недостатки размещены в порядке убывания затрат и увеличения объема работ по разработке и эксплуатации.

	Преимущества	Недостатки
Отдельные базы данных	Простота реализации. Простота перевода приложения с локальной к размещаемой среде. Проще обеспечивать резервное копирование, восстановление и мониторинг, поскольку большинство существующих инструментов работают на уровне базы данных. Высокая изоляция данных.	Общие таблицы модели предметной области дублируются по базам данных пользователей. Более высокие расходы на оборудование.
Совместно используемая база данных с разными схемами	Более низкое потребление памяти. Большая плотность пользователей на каждом отдельно взятом сервере. Использование пользователями одних и тех же таблиц. Необходимость в компоненте доступа к данным, который перехватывает имена таблиц. Необходимость авторизации на уровне пользователя для доступа к данным.	Меньшая изоляция. Архивация и восстановление представляют собой довольно сложную задачу, требующую специального решения. Сложности с отслеживанием действий пользователей.
Совместно	Более низкое потребление памяти (меньше	Меньшая изоляция; требуются

<i>используемая база данных с совместно используемыми схемами</i>	<p>объектов базы данных).</p> <p>Большая плотность пользователей на каждом отдельно взятом сервере.</p>	<p>дополнительные усилия для обеспечения более высокой изоляции.</p> <p>Данные доступны всем пользователям.</p> <p>Архивация и восстановление представляют собой довольно сложную задачу, требующую специального решения.</p> <p>Сложности с отслеживанием действий пользователей.</p>
---	---	--

Приложения, оптимизированные для подхода с совместным использованием базы данных, могут быть более сложными и требовать больших капиталовложений и усилий на разработку. Однако, как правило, они поддерживают большее количество пользователей на каждый сервер и могут обеспечивать меньшие эксплуатационные расходы. Приложения, оптимизированные для подхода с общей схемой, проще и, как правило, обеспечивают меньшие затраты на эксплуатацию в долгосрочной перспективе, хотя эта экономия будет меньше, чем в случае применения подхода с совместно используемой базой данных.

Если требуется быстро создать приложение, используйте подход с разными базами данных, конфигурируя для каждого пользователя собственную базу данных. Для использования этого подхода не требуется никакого специального дизайна. Также применяйте его в случаях, когда отдельные пользователи предъявляют особенно высокие требования к безопасности данных, собираются хранить очень большие объемы данных или имеют очень большое число одновременно работающих конечных пользователей. Подход с разными базами данных также подойдет, если приложение должно без труда переходить от локального к размещаемому или от размещаемого к локальному и масштабироваться в случае необходимости.

Более высокая изоляция также пригодится, если предполагается предлагать услуги за дополнительную плату для отдельных пользователей. В этом случае необходимо использовать подход с отдельными базами данных или совместно используемой базой данных, но разными схемами. Однако если ожидается очень большое число пользователей, каждый из которых будет иметь относительно небольшое количество данных, используйте подход с меньшей изоляцией, такой как совместно используемая база данных с разными схемами или совместно используемая база данных с совместно используемыми схемами.

Безопасность данных

Безопасность размещаемых в облаке приложений должна быть весьма высока. Она реализуется с использованием множества уровней защиты, дополняющих друг друга для обеспечения защиты данных разными способами, в разных условиях, как против внутренних, так и внешних угроз. При планировании стратегии безопасности руководствуйтесь следующими рекомендациями:

- **Фильтрация.** Используйте промежуточный слой между пользователем и источником данных, который будет выступать в роли «сита» и обеспечивать пользователю

впечатление, что в базе данных хранятся только его данные. Это особенно важно в случае применения общего экземпляра базы данных для всех пользователей.

- **Разрешения.** Используйте списки управления доступом (ACL) для определения того, кто имеет право доступа к данным приложения, и что они могут делать с этими данными.
- **Шифрование.** Скройте все конфиденциальные данные пользователей так, чтобы их было невозможно прочитать, даже если они станут доступны неавторизованной стороне.

Шаблоны обеспечения безопасности данных

Выбирайте шаблон обеспечения безопасности, исходя из используемой многотенантной модели:

- **Доверенные подключения к базам данных** (применяется ко всем трем многотенантным моделям). Приложение всегда подключается к базе данных с использованием собственного удостоверения процесса приложения, которое не зависит от удостоверения тенанта, и сервер предоставляет приложению доступ к объектам базы данных, которые он может читать или обрабатывать. В рамках приложения должны быть реализованы дополнительные меры безопасности для предотвращения доступа к объектам базы данных отдельных конечных пользователей, которые не имеют права делать этого. Каждый тенант (организация), работающий с приложением, имеет множество наборов учетных данных, ассоциированных с учетной записью тенанта, и должен предоставлять доступ к приложению конечным пользователям, используя эти учетные данные. Эти конечные пользователи получают доступ к приложению посредством собственных учетных данных, ассоциированных с учетной записью тенанта, но приложение выполняет доступ к базе данных, используя единый набор учетных данных, ассоциированный с этим приложением. Это означает, что каждому приложению необходима всего одна учетная запись для доступа к базе данных (по одной для каждого тенанта). Как альтернативный вариант от STS можно получить шифрованные учетные данные входа для тенанта в целом и с помощью кода безопасности в приложении управлять тем, к каким данным имеет доступ каждый отдельный пользователь.
- **Безопасные таблицы базы данных** (применяется к модели с отдельными базами данных и совместно используемой базе данных с разными схемами). Предоставьте учетной записи тенанта доступ к таблице или другому объекту базы данных. В модели с отдельными базами данных предоставьте доступ к той или иной базе данных только тенанту, ассоциированному с этой базой данных. В модели с совместно используемой базой данных и разными схемами доступ к каждой таблице предоставьте только тенанту, ассоциированному с этой таблицей.
- **Шифрование данных тенанта** (применяется ко всем трем многотенантным моделям). Защитите данные с помощью симметричного шифрования, и секретный ключ тенанта – с помощью асимметричного шифрования (с применением пары открытый/секретный ключ). Используйте олицетворения для доступа к базе данных с применением

безопасного контекста тенанта и секретный ключ тенанта для дешифрования данных в базе данных, чтобы с ними можно было работать. Недостаток этого метода в невозможности индексации шифрованных столбцов, что означает необходимость компромисса между безопасностью данных и производительностью. Страйтесь избегать использования полей индекса, содержащих конфиденциальные данные.

- **Фильтрация данных тенанта** (применяется к модели с совместно используемой базой данных с совместно используемой схемой). Используйте представления SQL для выбора из таблиц подмножеств данных по ID тенанта или пользователя либо по идентификатору безопасности учетной записи тенанта. Представляйте тенантам доступ только к их представлениями, а не ко всей таблице. Это не допустит просмотр или доступ пользователей к любым строкам совместно используемых таблиц, принадлежащим другим тенантам или пользователям.

Хранение и расширяемость данных

Существует множество способов хранения размещаемых данных. Для реализации хранения данных в размещаемых приложениях разрабатывается два разных подхода: системы управления размещенными реляционными базами данных (*hosted relational database management systems, RDBMS*) и нереляционное хранилище в облаке. Реляционные базы данных обеспечивают хранение структурированных данных и больше подходят для транзакционных систем или приложений с интенсивным вводом/выводом. Также обычно такие базы данных обеспечивают меньшую задержку и расширенные возможности запросов. Под хранилищем в облаке, напротив, подразумевается хранилище для любых типов данных, которое размещается в облаке; сюда также относятся сервисы, обеспечивающие функциональность базы данных, сервисы для неструктурированных данных (например, хранилище файлов для цифровых мультимедиа), сервисы синхронизации данных и устройства хранения данных, подключаемые к сети (*network-attached storage, NAS*). Сервисы данных часто предоставляются по модели повременной оплаты или, в данном случае, оплате за гигабайт данных (включая и хранящиеся, и переданные данные).

Хранение в облаке предлагает ряд преимуществ, таких как возможность хранения и извлечения больших объемов данных из любой точки земного шара в любое время. Сервисы хранения данных быстры, недороги и практически неограниченно масштабируемые. Могут возникать проблемы с надежностью, но даже самые лучшие сервисы дают сбой время от времени. Также возможны неполадки с приложениями, чувствительными к большой задержке, поскольку каждое взаимодействие с сервисом хранения требует передачи данных по сети. Наконец, для систем хранения в облаке может быть проблематичной поддержка транзакций. Как правило, эти системы ориентированы на секционирование и доступность и не всегда могут гарантировать согласованность.

Хранилище Microsoft Azure (на момент написания данного документа находящееся на этапе ранней предварительной версии¹) включает ряд сервисов, обеспечивающих различные нужды хранилища и доступных через REST API:

¹ С 1 февраля 2010 года в промышленной эксплуатации (прим. научного редактора).

- **Сервисы хранилища таблиц** обеспечивают структурированное хранилище в форме таблиц, но эти таблицы не имеют определенной схемы, вместо этого они включают сущности, каждая из которых обладает рядом свойств. Такие популярные API, как LINQ, могут использоваться для работы с этими свойствами. Сервисы хранилища таблиц ориентированы на обеспечение высокого масштабируемости таблиц по очень малой цене. Однако это не реляционная база данных, поэтому в них нет многих возможностей, которые можно найти в RDBMS, таких как объединения и внешние ключи.
- **Сервисы хранилища больших бинарных объектов** предлагают хранилище для бинарных данных, размещаемых в определяемых пользователями контейнерах, которые организуют наборы больших бинарных объектов в рамках учетной записи хранилища.
- **Сервисы очередей** хранят сообщения, которые могут быть прочитаны с использованием семантики организации очереди любым клиентом, имеющим доступ к учетной записи хранилища.

Основную сложность при использовании RDBMS представляет расширяемость схемы. Это возможность расширять таблицу собственными полями без повторной компиляции или сборки приложения. Существует четыре подхода для расширения схем во время выполнения:

- **Фиксированные столбцы.** Этот шаблон моделирует поля расширения как набор фиксированных именованных столбцов для каждой расширяемой сущности (каждой таблицы). Количество фиксированных столбцов будет зависеть от природы сущности и шаблона ее использования. Потребуется слой доступа к данным, обеспечивающий инкапсуляцию и абстракцию фиксированных именованных столбцов и таблиц метаданных. При использовании подхода с фиксированными столбцами должны быть учтены следующие факторы:
 - Фильтрация на основании расширяемых столбцов представляет проблему в связи с предопределенностью типов данных. Например, использование типов данных переменной длины, таких как varchar, для всех расширяемых столбцов ограничивает возможность фильтрации по числовым значениям с применением операторов <, > и =. Возможным решением является выделение фиксированного количества полей для каждого общего типа данных или предоставление пользователю возможности помечать столбцы как доступные для поиска и использовать отдельную таблицу для хранения полей определенного типа.
 - Тогда как один из самых производительных и наиболее масштабируемых подходов к обеспечению расширяемости, как правило, он требует решения для индексированных столбцов не строкового типа.
 - Неправильная интерпретация базой данных значений null может привести к разреженному распределению данных и необоснованному чрезмерному расходованию пространства. Если один из тенантов расширяет только одно поле, тогда как остальные расширяют по 20 полей, база данных и страницы в памяти будут разрастаться. В Microsoft SQL Server 2008 предлагается

модификатор для столбцов SPARSE (Разреженный), который помогает несколько смягчить эту проблему.

- **Собственные схемы.** Этот шаблон используется в сочетании с многотенантной моделью с разными схемами. Каждая схема принадлежит определенному тенанту и содержит определенный, отличный от остальных, набор расширяемых строго типизированных столбцов. Для подхода с собственными схемами следует учесть следующие факторы:
 - Этот подход требует инкапсуляции и абстракции слоя доступа к данным и наличия обработчика запросов, хотя О/РМ-инфраструктуры, такие как Microsoft Entity Framework (EF) или инфраструктура с открытым исходным кодом NHibernate, могут упростить реализацию. Больше сведений по данным вопросам можно найти в материалах, приведенных в разделе [Дополнительные источники](#) в конце этой главы.
 - Каждый тенант имеет собственную таблицу, и схема таблицы меняется при каждом добавлении или удалении поля. Запросы работают с данными с плавающей точкой (не все столбцы строкового типа). Однако обновление схемы базы данных является непростой задачей из-за масштабного дублирования совместно используемых полей тенантов (в отличие от подхода с фиксированными столбцами, когда одна или более таблиц являются общими для всех тенантов).
 - Фильтрация по расширенным столбцам при таком подходе более производительна, чем при использовании шаблона с фиксированными столбцами, потому что в данном случае используются элементарные типы данных.
- **Таблица пар имя/значение.** Этот шаблон позволяет потребителям расширять модель данных произвольным образом (неограниченным количеством столбцов), сохраняя собственные данные в отдельной таблице и используя метаданные для определения меток и типов данных для собственных полей каждого тенанта. При использовании подхода учитывайте следующие факторы:
 - Он вводит дополнительный уровень сложности для функций базы данных, таких как индексация, запросы и обновление записей. Например, для извлечения данных необходимо больше операций объединения, также сложность представляют фильтрация и группировка.
 - Работать приходится лишь с одной базой данных. Однако если рост числа пользователей приводит к увеличению базы данных, она может масштабироваться вертикально с секционированием тенантов для использования отдельных баз данных.
 - Этот подход будет менее производительным, чем остальные, поскольку извлечение данных требует множества операций объединения.

- **XML-столбцы.** Этот шаблон позволяет потребителям расширять модель данных произвольным образом неограниченным числом расширений путем сохранения данных расширения в XML-столбце. При использовании подхода с XML-столбцами учтите следующие факторы:
 - Тогда как данный подход может казаться наиболее естественным выбором для обеспечения расширяемости, он обладает меньшей масштабируемостью (возможность добавлять записи) и производительностью (время ответа на запрос) по сравнению с остальными подходами.
 - Тогда как применение XML-столбцов в базе данных обуславливает относительно простые реализации, ISV и разработчикам потребуются дополнительные навыки для работы с XML в базе данных.
 - Для XML-столбцов можно определить индексы, но это вводит дополнительную сложность и требует больше памяти для хранения.

Управление удостоверениями

Все приложения и сервисы должны работать с удостоверением пользователя. Особенно важно это для размещаемых сценариев и сценариев в облаке, которые потенциально могут обслуживать очень большое количество пользователей, и каждый из этих пользователей может иметь собственную инфраструктуру удостоверений. Обычный подход для поставщиков услуг размещения – делегирование ответственности за управление собственными учетными данными самим пользователям. Идеальным является решение, использующее преимущество локального или интегрированного сервиса каталогов пользователя для поддержки единой регистрации (SSO) для локальных и всех внешних размещенных сервисов. Это сокращает объем работ по созданию индивидуальных или отдельных систем управления удостоверениями. Потребители могут настраивать доступ к приложению, используя привычные инструменты, и благодаря SSO пользователи могут выполнять доступ к приложению или сервису с помощью существующих учетных данных.

Чтобы реализовать такой сценарий, необходимо принять решение, основанное на отраслевых стандартах, действующих на всех платформах и во всех организациях. Как правило, следует придерживаться модели удостоверений на основании утверждений на базе интегрированного сервиса удостоверений, как показано на рис. 1. Это позволит отделить приложения и сервисы от механизма аутентификации.

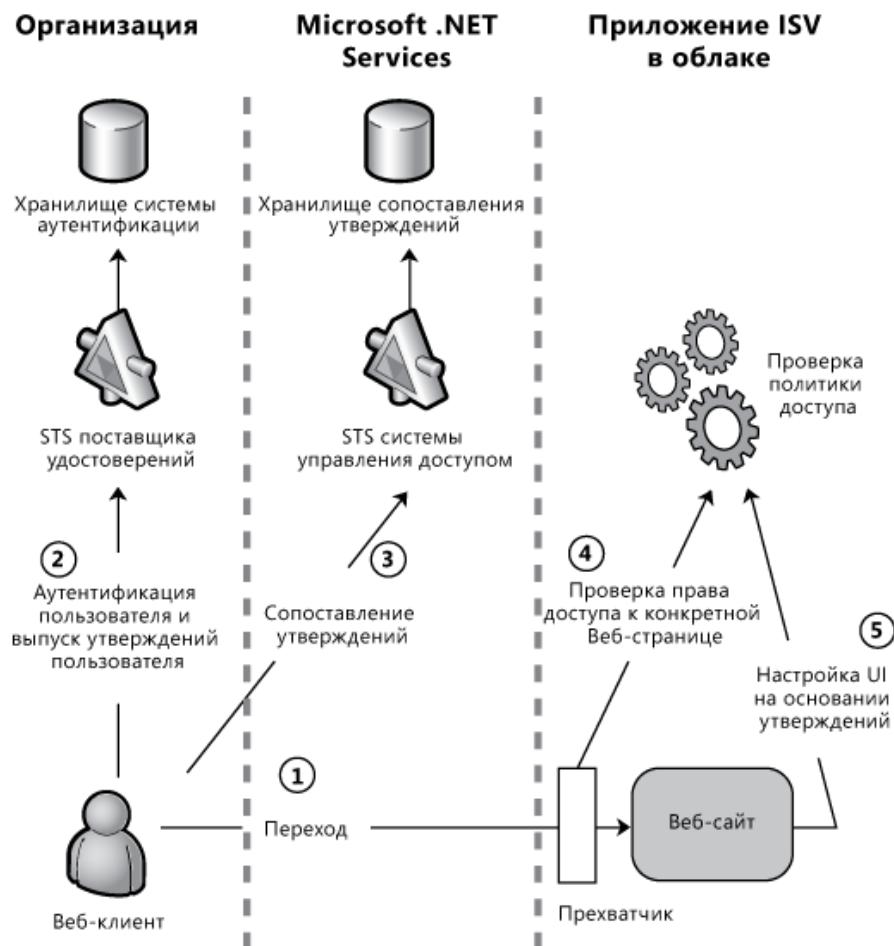


Рис. 43

Модель удостоверений на основании утверждений на базе интегрированного сервиса удостоверений

Существующая система удостоверений пользователя с каждым запросом к приложению передает криптографически подписанный маркер доступа, содержащий набор утверждений о каждом пользователе. Компании, предоставляющие услуги размещения, которые доверяют системе удостоверений пользователя, могут создавать приложения и сервисы, выполняющие авторизацию только необходимых утверждений. Система удостоверений пользователя должна реализовывать STS, который аутентифицирует пользователей, создает и подписывает маркеры доступа стандартного формата и предоставляет сервис для выпуска этих маркеров, основанный на отраслевых стандартах, таких как WS-Trust и WS-Federation. Microsoft Geneva Framework¹ и Geneva Server² во многом обеспечивают необходимую для реализации этих требований инфраструктуру. При реализации модели удостоверений на основании утверждений следует обратить внимание на следующие вопросы:

¹ Переименовано в Windows Identity Foundation или WIF (прим. научного редактора).

² Переименовано в Active Directory Federation Services v2 или ADFS v2 (прим. научного редактора).

- Если доступно подходящее хранилище удостоверений, используйте его для обеспечения единой регистрации для всех локальных приложений, размещаемых Веб-приложений и других размещаемых сервисов.
- Для небольших или ориентированных на потребителя приложений, если нет доступного хранилища удостоверений, используйте такой сервис, как .NET Services Access Control, интегрированный в Windows Live¹, или решение от стороннего производителя.
- Может возникнуть необходимость внесения изменений в утверждения, сформированные локальным STS, для выполнения требований поставщика услуг размещения. Или поставщикам услуг размещения может понадобиться реализовать системы преобразований для разных STS-механизмов пользователей. Уровень преобразований обеспечит .NET Access Control Service, или его можно реализовать самостоятельно с помощью Geneva Framework. Более подробная информация о Geneva Framework представлена по адресу <http://msdn.microsoft.com/en-us/security/aa570351.aspx>.
- Пути реализации основанных на стандартах продуктов разных поставщиков могут отличаться очень незначительно. Проблемы с совместимостью и возможностью взаимодействия возникают, только если эти продукты следуют стандартам не строго или обеспечивают немного разную их реализацию.
- Если решено создавать собственный STS, обеспечьте его защиту от атак. В этом сервисе содержатся все данные аутентификации, поэтому его уязвимость приведет к уязвимости приложений. Также обеспечьте устойчивую к ошибкам и надежную реализацию STS, которая способна обработать все предполагаемые объемы запросов. Сбой STS приведет к тому, что пользователи не смогут выполнять доступ ко всем приложениям, зависящим от него.

Многотенантная архитектура

Отдельные тенанты совместно используют оборудование и инфраструктуру поставщика услуг размещения, а также базы данных и системы баз данных (каждый тенант – это организация, в которой может быть более одного пользователя). Поставщики сервисов должны обеспечивать для размещаемых сервисов платформу с соответствующей емкостью и производительностью. Также они должны продумать, как будут контролировать систему затрат и обеспечивать настройку через конфигурацию. Существует четыре общих этапа перехода к эффективной многотенантной архитектуре с поддержкой пользовательской конфигурации. Эти этапы описываются в следующих разделах:

- **Собственная архитектура.** Каждый потребитель выполняет отдельную копию ПО, предназначенную только для него, и единственный способ поддерживать множество потребителей – предоставление им разных копий ПО. Более того, поскольку

¹ В финальной версии .NET Access Control Services интеграция с Windows Live не поддерживается (прим. научного редактора).

практически ничего не обеспечивает возможности настройки через конфигурацию, каждая копия включает настройки конкретного пользователя в форме собственного кода расширения, собственных процессов и/или собственных расширений данных. Хотя, технически, ПО поставляется как сервис (оно не выполняется локально у потребителя), повышения эффективности от роста масштабов достичь невозможно, поскольку все потребители выполняют разные экземпляры ПО. Такая модель может быть отправной точкой для проверки правильности бизнес-модели, но от нее необходимо отказаться, как только количество потребителей начинает увеличиваться. Она не годится для управления тысячами потребителей.

- **Настраиваемая архитектура.** ПО настраивается для каждого конкретного пользователя через конфигурацию и без применения собственного кода. Все тенанты выполняют один и тот же код. Однако архитектура все равно не является многотенантной, и каждый потребитель выполняет собственную копию кода, несмотря на то что копии идентичны. Разделение может быть либо виртуальным (виртуальные машины на одном сервере), либо физическим (выполнение на разных компьютерах). Эта модель является существенным улучшением по сравнению с описанной выше моделью собственной архитектуры, тем не менее, архитектура по-прежнему не допускает настройки через конфигурацию, и вычислительные мощности не разделяются между экземплярами. Поэтому поставщик не может добиться повышения экономической эффективности от роста масштабов использования.
- **Многотенантная архитектура.** UI может настраиваться для каждого тенанта в отдельности, как и бизнес-правила, и модель данных. Настройку каждый тенант выполняет исключительно через конфигурацию с использованием инструментов для самостоятельной настройки, что снимает ответственность за настройку с поставщика сервиса. Этот уровень является практически идеальным вариантом SaaS; единственный недостаток – никакой возможности горизонтального масштабирования, расширение может быть достигнуто только через вертикальное масштабирование.
- **Масштабируемая архитектура.** Такая архитектура поддерживает многотенантность и конфигурацию, плюс возможность горизонтального масштабирования приложения. Новые экземпляры ПО могут без труда добавляться в пул экземпляров для обеспечения динамической поддержки возрастающей нагрузки. Соответствующее секционирование данных, дизайн компонентов без сохранения состояния и совместный доступ к метаданным являются частью дизайна. На этом уровне появляется подсистема балансировки нагрузки тенантов (Tenant Load Balancer) (реализованная с использованием механизмов циклического обслуживания или на основании правил). Эта подсистема обеспечивает повышение степени использования ресурсов размещающей стороны, таких как ЦП и хранилище. Это означает, что общая нагрузка распределена по всей доступной инфраструктуре. Также периодически выполняется реорганизация данных для усреднения нагрузки по обработке данных на каждый экземпляр. Архитектура является масштабируемой, многотенантной и настраиваемой через конфигурацию.

Локально или удаленно, создавать или покупать

Размещаемые в облаке приложения позволяют ISV и поставщикам услуг размещения получить экономическую эффективность от роста масштаба; и в условиях конкурентного рынка эта прибыль будет пущена на уменьшение расходов потребителей. Однако переход к удаленным и размещаемым сценариям означает, что предприятия должны согласиться с частичной утратой контроля над приложениями, данными и уровнями обслуживания. Предприятия, кроме принятия решения о том, будут ли они создавать собственные приложения или покупать их у сторонних производителей, должны взвесить все плюсы и минусы перехода к таким сервисам. В следующей таблице представлены различия между сценариями создания и покупки размещаемых приложений.

	Локальное	Размещаемое	Сервис в облаке
Создание	Приложение, разрабатываемое самостоятельно и выполняемое в собственном дата-центре.	Приложение, разрабатываемое самостоятельно и выполняемое на ресурсах компании-поставщике услуг размещения.	Размещаемая у поставщика среда разработки и выполнения.
Покупка	Коробочное приложение, приобретаемое в готовом виде и выполняемое в собственном дата-центре.	Коробочное приложение, приобретаемое в готовом виде и выполняемое на ресурсах компании-поставщике услуг размещения.	Размещенное приложение, приобретаемое и размещаемое у поставщика.

Выгода, получаемая от повышения эффективности вследствие роста масштаба, должна быть сбалансирована с уменьшением контроля, связанным с переходом от локальных к полностью размещаемым приложениям. Принимая решение о переходе к решению в облаке, о подходе к разработке приложения и о том, где размещать свое приложение, руководствуйтесь следующими рекомендациями:

- Используйте **локальное** размещение, если должны иметь полный контроль над приложением и данными и предъявляемые требования по безопасности препятствуют использованию размещаемых сервисов, либо использование таких сервисов запрещено национальными политиками. При локальном размещении приложения в собственной инфраструктуре вы можете:
 - Создавайте собственное приложение, если не можете найти подходящее готовое или хотите полностью контролировать функциональные возможности приложения.
 - Используйте готовое коробочное приложение, если оно удовлетворяет всем требованиям и приемлемо по цене.
- Размещайте приложение **у поставщика услуг размещения**, если предполагаете оптимизировать свои операции, но желаете сохранить контроль над ПО. Например, можно развертывать настроенный пакет Enterprise Resource Planning (ERP) у

поставщика услуг размещения и передать им ответственность за управление энергопотреблением, оборудованием, сетью и операционной системой. Как правило, поставщик услуг размещения будет обеспечивать очень специфические требования вашей организации, такие как, например, настройка Virtual Private Network (VPN), добавление специализированного оборудования и многое другое. При размещении приложений во внешней компании-поставщике услуг размещения:

- Выбирайте одно из готовых коробочных приложений поставщика услуг размещения, если оно отвечает вашим требованиям. Доступность готовых коробочных приложений может определить выбор поставщика услуг размещения.
- Создавайте собственное приложение, если не можете найти поставщика, предлагающего подходящее готовое приложений. В этом случае необходимо учесть стоимость и время разработки собственного приложения.
- Используйте **сервисы в облаке** (размещенные у поставщика ПО), если покупаете SaaS-приложение у поставщика услуг размещения или ISV; если можете обеспечить спецификацию требуемого приложения; если не имеете ресурсов, времени или навыков для создания приложения; или если существующее стандартное или настроенное приложение необходимо немедленно. Другая причина использования внутренних свойств стандартных блоков сервисов в облаке (таких как эластичность) – создание самого приложения. Приобретая сервисы приложений у поставщика:
 - Выбирайте готовое коробочное приложение, созданное поставщиком ПО, если оно отвечает имеющимся краткосрочным и долгосрочным требованиям. Это SaaS-подход.
 - Выбирайте предоставляемую поставщиком платформу размещения для выполнения собственного приложения, если не можете найти подходящего готового приложения. При этом необходимо учесть стоимость и время разработки собственного приложения. Это PaaS-подход.

Производительность

Размещаемые в облаке приложения должны быть масштабируемыми для обеспечения поддержки увеличивающегося количества сервисов и увеличивающейся нагрузки для каждого сервиса и тенанта. При проектировании сервисов руководствуйтесь следующими рекомендациями для масштабирования приложений:

- По возможности создавайте сервисы и компоненты без сохранения состояния. Это сократит использование памяти для сервиса и улучшит возможности горизонтального масштабирования и балансировки нагрузки сервисов.
- Используйте асинхронные входящие и исходящие вызовы, что позволяет приложениям продолжать выполнение одновременно с выполнением операций ввода/вывода.

- Изучите возможности платформы размещения, которые могут способствовать улучшению производительности. Например, в Microsoft Azure используйте очереди для управления запросами и рабочие процессы для обработки в фоновом режиме.
- Используйте пулы ресурсов для потоков, сетевых подключений и подключений к базе данных.
- Обеспечьте оптимальные условия для одновременной работы, используя блокировки только в случае крайней необходимости.

При масштабировании хранилища данных и приложений руководствуйтесь следующими рекомендациями:

- При масштабировании секции данных разделяйте данные подписчиков на меньшие секции, чтобы обеспечить требования по производительности. Применяйте такие схемы, как Хэширование (для разукрупнения содержимого) и Временная (основанная на времени или диапазоне данных, в рамках которых данные считаются действительными).
- Реализуйте динамическое повторное секционирование для автоматического повторного секционирования данных, когда база данных достигает заданного максимального размера.
- Для масштабирования хранилища данных и приложений рассмотрите стандартные шаблоны и специальные техники и реализации, предоставляемые платформой размещения, такие как секционирование данных, балансировка нагрузки, обработка отказов и географическое распределение.

Композиция сервисов

Корпоративным пользователям необходимо иметь доступ к разным хранилищам документов, типам данных, источникам информации и приложениям, осуществляющим определенные функции. Традиционно пользователи напрямую взаимодействовали с каждым хранилищем или приложением, часто используя для этого специальные изолированные приложения. Однако со временем корпорации постарались консолидировать системы, часто используя для этого Веб-порталы или фасадные-приложения, подключающиеся к соответствующим нижестоящим приложениям.

С появлением сервисов и SOA-приложений отделы ИТ получили возможность предоставлять приложения и данные как сервисы, либо размещенные локально, или приобретенные как SaaS. Портфели сервисов могут по-прежнему предоставлять сочетание традиционных локальных приложений, размещенных локально сервисов и удаленных сервисов через порталы, которые скрывают пользователя от реализаций и позволяют отделам ИТ быстро и без труда переходить на различные сервисы. Однако дизайны и технологии S+S и SaaS обеспечивают отделам ИТ и корпоративным потребителям возможность полной интеграции сервисов. Интеграция сервисов поможет реализовать модель *многие-к-одному*, при которой все приложения и сервисы доступны пользователю через архитектуру композиции, предоставляющую их как одно приложение (рис. 2). Механизм интеграции сервисов

объединяет группы приложений в портфели и предоставляет их через насыщенный клиент, который может взаимодействовать с любым сервисом или приложением.



Рис. 2

Механизм интеграции сервисов может объединить множество сервисов в единый интерфейс

Корпоративным пользователям размещаемых в облаке сервисов обычно понадобится создавать комбинированные системы, использующие предоставляемые поставщиками услуг размещения сервисы, и предлагать их через локальные порталы и портфели. Эффективная пользовательская комбинированная архитектура может интегрировать данные для конечных пользователей из многих источников. Это сокращает избыточный ввод данных, улучшает взаимодействие персонала и повышает осведомленность о невыполненных задачах и их статусе. Бизнес-данные становятся более наглядными, что помогает пользователям принимать осознанные бизнес-решения. Композиция унифицированного решения, использующего размещаемые в облаке сервисы, обычно включает следующие три уровня:

- **Слой источников входных данных.** К источникам входных данных относятся размещаемые в облаке сервисы, внутренние приложения, внутренние базы данных, Веб-сервисы, неструктурированные файлы и пр. Внутренние ресурсы могут предоставляться через портфели ИТ-сервисов.
- **Слой композиции.** Слой, в котором происходит агрегация необработанных данных и их обработка для предоставления пользователю в новом унифицированном виде. Основная функция этого слоя – преобразование данных в бизнес-данные и аналитику процесса. Обычно слой включает следующее:
 - Компоненты, управляющие доступом, данными, рабочим процессом и правилами.
 - Агенты сервисов, которые согласовывают схемы и обмениваются сообщениями с приложениями, базами данных, Веб-сервисами и другими ресурсами.

- Компоненты управления удостоверениями, которые аутентифицируют и авторизуют пользователей и управляют учетными данными для взаимодействия с Веб-сервисами.
 - Компоненты агрегации данных, которые преобразуют данные соответственно модели сущностей приложений.
- **Ориентированный на работу с пользователем слой.** Этот слой представляет пользователю составные данные в централизованном интегрированном ориентированном на задачу пользовательском интерфейсе.

При использовании размещаемых в облаке сервисов как части составного пользовательского интерфейса для интеграции внешних и внутренних сервисов обычно требуется рабочий процесс или поэтапные процессы. Обычным решением является применение брокера интеграции, состоящего из модульного подключаемого конвейера и связанных сервисов метаданных для управления перемещением и маршрутизацией сообщений. К типовым операциям брокера интеграции относятся:

- **Безопасность.** Подсистема безопасности выполняет аутентификацию источника данных или цифровой подписи, проводит дешифрацию данных и проверяет их на наличие угроз безопасности, таких как вирусы. Для управления доступом операции по обеспечению безопасности могут быть скординированы с существующими политиками безопасности.
- **Валидация.** Подсистема валидации сравнивает данные с соответствующими схемами и отклоняет данные, не отвечающие им.
- **Преобразование.** Подсистема преобразования выполняет преобразование данных в соответствующий формат.
- **Рабочий процесс синхронизации.** Подсистема синхронизации использует рабочие процессы и правила для определения логических адресов и порядка передачи сообщений в соответствующие места назначения. Она также может управлять транзакциями бизнес-процесса для обеспечения согласованности данных.
- **Маршрутизация.** Подсистема маршрутизации, используя правила маршрутизации, которые определяют физические адреса, передает сообщения с данными в заданные места назначения. Возможно использование данных сообщения для определения адресов на основании содержимого.

Интеграция сервисов

Размещаемые в облаке решения могут способствовать решению некоторых проблем, присущих традиционному ПО, но при этом потребители этих сервисов сталкиваются с новыми совершенно другими сложностями. При переходе к размещаемым в облаке сервисам и приложениям обратите внимание на следующие вопросы:

- **Управление удостоверениями.** Операции по добавлению, обновлению и удалению пользователей должны быть перенесены на удаленные сервисы. Если внешний сервис

зависит от удостоверения пользователя, что очень вероятно для SaaS и S+S, удаленными должны быть также процессы регистрации и отзыва прав доступа пользователей. Кроме того, может потребоваться преобразование локального удостоверения пользователя в конкретные роли, возможно, через интегрированный сервис, для снижения миграции или дублирования удостоверений отдельных пользователей на хосте удаленных сервисов. Политики учетных записей корпоративных пользователей, такие как сложность пароля и блокировка учетной записи, также должны быть совместимы с политиками поставщика удаленных сервисов. Невозможность использования SSO может привести к повышению расходов, затрат на обслуживание и операционной неэффективности.

- **Данные.** Требования операций с данными, таких как Extract, Transform, Load (ETL) и интеграция данных, должны быть проанализированы на предмет совместимости с возможностями сервисов. Размещаемые сервисы могут не поддерживать шаблоны хранения составных данных, что влияет на дизайн сущностей данных и архитектуру приложения. Кроме того, возможно, понадобится принять дополнительные меры защиты данных, чтобы скомпенсировать недостаток физической безопасности, которая обеспечивалась при локальном размещении. Однако приложения могут хранить конфиденциальные или личные данные локально и использовать сервисы в облаке только для неконфиденциальных данных. Также необходимо спланировать процесс передачи данных провайдеру сервиса и то, как они будут передаваться другому провайдеру в случае возникновения такой необходимости.
- **Операции.** Локальные сервисы интеграции и клиентские приложения могут быть несовместимы с сервисами, предоставляемыми поставщиком сервисов, даже при использовании стандартных протоколов. Необходимо обеспечить поставщику сервисов возможность формировать соответствующие сведения отчетности и выработать стратегию интеграции этих данных с собственными системами управления и составления отчетов. Возможно, потребуется пересмотреть соглашения об условиях обслуживания (Service Level Agreements, SLA), чтобы обеспечить их выполнение в условиях все возрастающей зависимости от поставщика сервисов. Предприятия также должны быть готовы реализовать службу поддержки, которая будет выступать в роли первой инстанции для пользователей, и определить процедуры решения проблем с поставщиком сервисов.
- **Безопасность.** Политики конфиденциальности предприятия должны быть совместимы с политиками поставщика сервисов. Также должны быть обеспечены правила для действий, которые могут выполняться пользователями, такие как ограничения по размеру транзакции и другие бизнес-правила, даже если они не относятся к возможностям удаленного сервиса. Это может усложнить инфраструктуру интеграции сервисов. Также потребуются процедуры и политики для обеспечения безопасности и целостности данных в случае сбоя сервиса или подключения. Для аутентификации, шифрования и использования цифровых подписей необходимо будет приобрести сертификаты у сертифицированных поставщиков и, возможно, реализовать Инфраструктуру открытых ключей (Public Key Infrastructure, PKI). Кроме того, интеграция может потребовать внесения изменений в правила межсетевых экранов, а для

обеспечения фильтрации данных приложений и валидации XML-схемы понадобится обновление их оборудования и ПО.

- **Возможность подключения.** Некоторым типам приложений в облаке, таким как сервисы оперативной обработки транзакций и сервисы реального времени (Передача голоса по IP (voice over IP, VoIP) и Microsoft Office Communications Server), для эффективной работы необходимо наличие широкополосного Интернет-подключения хорошего качества. В некоторых регионах и странах такое подключение может быть недоступным. Кроме того, сервисы, требующие больших объемов передачи данных, такие как сервисы архивации и доставки файлов, как правило, через сетевое соединение будут выполняться медленнее, чем при локальной реализации, чтобы может представлять проблему. Однако обмен сообщениями и другие подобные сервисы не так зависят от полосы пропускания и не так сильно страдают от периодической потери подключения.
- **Соглашения об условиях обслуживания (SLA).** Для всестороннего сотрудничества с поставщиками и принятия решений относительно приобретения контрактов на обслуживание потребуются определенные навыки и опыт. Возможно, при использовании сервисов, размещенных у удаленного поставщика, придется пересматривать SLA, чтобы убедиться в возможности их выполнения.
- **Соблюдение правовых норм.** Соблюдение правовых и корпоративных норм может зависеть от производительности поставщика сервисов, либо существует вероятность возникновения конфликтов между корпоративными и правовыми нормами, если поставщик территориально находится в другом государстве или регионе. Также с получением отчетов о соответствии правовым нормам от поставщика сервисов могут быть связаны дополнительные расходы. Локальные законы и политики могут запрещать выполнение в сценариях с размещением некоторых типов приложений, таких как приложения для банковского обслуживания.

Управление сервисами

Поставщики сервисов в облаке сталкиваются с определенными сложностями при размещении и предложении сервисов, которые выполняются в облаке, особенно в вопросах доставки и поддержки сервисов. Некоторые ISV просто создают приложения и предоставляют их для размещения, тогда как другие производители создают и размещают свои приложения самостоятельно. Строя планы о создании размещаемых сервисов, необходимо учесть несколько аспектов. Некоторые из них касаются только ISV, размещающих собственные сервисы, тогда как другие относятся только к компаниями, предоставляющим услуги размещения:

- **Управление уровнем обслуживания.** Разные корпоративные пользователи могут требовать внесения некоторых изменений или дополнений в стандартные соглашения об условиях обслуживания с поставщиками услуг размещения. Это может усложнить задачу по выполнению всех требований всех потребителей. Решение с размещением в облаке может использоваться потребителями как способ повышения доступности и производительности, таким образом, ожидания от применения такого решения будут

выше, чем для локального приложения. Управление и удовлетворение этим ожиданиям может представлять собой сложную задачу, поскольку обычно для этого требуется управление зависимостями (такими как поставщики сетевых услуг и электропитания) и выполнение разных требований потребителей, находящихся в различных точках земного шара. При размещении сервисов для множества предприятий, особенно если они находятся в разных часовых поясах или переживают пиковые нагрузки в разное время в течение дня или недели, должны быть тщательно спланированы обслуживание и периоды простоя сервисов.

- **Управление пропускной способностью и непрерывностью.** Поставщики сервисов не будут обладать тем же пониманием возможных изменений требований по пропускной способности, как специалисты заказчика, что может привести к неожиданным пикам использования, для которых требуется возможность получения дополнительной пропускной способности. Перспективное планирование затруднено, поскольку рост и шаблоны использования каждого потребителя будут отличаться, и об этих отличиях поставщик не будет проинформирован заранее или будет знать очень немного. При реализации и настройке сервисов очень сложно обеспечить выполнение требований множества пользователей. Краткосрочные решения по пропускной способности, скорее всего, приведут к большим издержкам в длительной перспективе, чем план ее поэтапного увеличения, но долгосрочное планирование намного сложнее реализовать, не имея в своем распоряжении оценок потребителей.
- **Поддержка пользователей.** Персонал службы поддержки должен знать и принимать во внимание требования и сценарии использования, чтобы предоставлять оптимальную поддержку пользователям. В условиях использования каждого сервиса большим количеством потребителей служба поддержки будет получать огромное количество обращений, связанных со сбоями или проблемами, возникающими при работе с сервисами. Персонал службы поддержки должен уметь определять стоимость оказываемых услуг каждому пользователю, особенно для моделей, когда техническая поддержка предоставляется как платная дополнительная услуга. Идеальным вариантом является предоставление для размещаемого в облаке решения активной технической поддержки, когда поставщик узнает о проблемах в ходе мониторинга работоспособности решения и проводит профилактику этой проблемы. Также могут использоваться механизмы самообслуживания через предоставление пользователям системы отслеживания проблем.

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Доступность данных, Передача данных, Преобразование данных, Интеграция и композиция, Производительность и надежность и Взаимодействие с пользователем, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
-----------	---------

Доступность данных	<p>Polling (Опрос). Один источник, как правило, через равные промежутки времени опрашивает остальные о наличии изменений.</p> <p>Push (Выталкивание). Источник, данные в котором были изменены, передает изменения в приемник данных при каждом изменении данных или через равные промежутки времени.</p> <p>Publish/Subscribe. Смешанный подход, сочетающий в себе черты опроса и выталкивания. При внесении изменений в источник данных он публикует событие-уведомление об изменении, на которое может подписаться приемник данных.</p>
Передача данных	<p>Asynchronous Data Transfer (Асинхронная передача данных). Основанный на сообщениях метод, при котором отправитель и получатель обмениваются данными, не ожидая ответа.</p> <p>Synchronous Data Transfer (Синхронная передача данных). Интерфейсный метод, при котором отправитель и получатель обмениваются данными в режиме реального времени.</p>
Преобразование данных	<p>Shared Database (Совместно используемая база данных). Все интегрируемые приложения выполняют чтение данных напрямую из одной и той же базы данных.</p> <p>Maintain Data Copies (Хранение копий данных). Сохраняет копии базы данных приложения, чтобы другие приложения могли читать данные (и, возможно, обновлять их).</p> <p>File Transfer (Передача файлов). Доступность данных обеспечивается через передачу файла, являющегося фрагментом базы данных приложения, для того чтобы другие приложения могли загружать данные из файлов.</p>
Интеграция и композиция	<p>Broker (Посредник). Скрывает детали реализации вызова удаленного сервиса путем их инкапсуляции в другой слой, отдельный от бизнес-компонентта.</p> <p>Composition (Композиция). Объединяет множество сервисов, приложений или документов в интегрированный интерфейс, обеспечивая при этом безопасность, валидацию, преобразование и связанные задачи для каждого источника данных.</p> <p>Portal Integration (Интеграция в портал). Создает приложение портала, которое отображает в унифицированном UI данные, извлекаемые из множества приложений. Пользователь может осуществлять необходимые задачи на основании данных, отображаемых в портале.</p>
Производительность и надежность	<p>Server Clustering. Организовывает инфраструктуру приложения таким образом, что используемые серверы представляются пользователям и приложениям как виртуальные унифицированные вычислительные ресурсы, что позволяет улучшить доступность и масштабируемость либо и то, и другое.</p> <p>Load-Balanced Cluster. Устанавливает сервис или приложение на множество серверов, конфигурированных для разделения рабочей нагрузки. Хосты с балансировкой нагрузки параллельно отвечают на запросы разных клиентов, даже на множество запросов одного клиента.</p> <p>Failover Cluster. Устанавливает приложение или сервис на множество серверов, конфигурированных так, чтобы замещать друг друга в случае отказа. Каждому серверу в кластере поставлен в соответствие, по крайней мере, один сервер этого кластера, который является резервным для него.</p>

<p>Взаимодействие с пользователем</p>	<p>Universal Web (Универсальный Веб). Обеспечивается максимальная доступность и, в то же время, простота развертывания с возможностью работы в Веб-браузерах без каких-либо специальных расширений.</p> <p>Experience First (Опыт прежде всего). Повышает качество взаимодействия с пользователем путем использования оптимизированных возможностей компьютера и устройства.</p>
--	--

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

Больше информации о Microsoft Azure можно найти в материале «Azure Services Platform» по адресу <http://www.microsoft.com/azure/default.mspx>.

Более подробно система управления удостоверениями Microsoft «Geneva» рассматривается в статье «*Geneva Simplifies User Access to Applications and Services*» (Geneva упрощает доступ пользователей к приложениям и сервисам) по адресу <http://msdn.microsoft.com/en-us/security/aa570351.aspx>.

Архитектуре S + S посвящены следующие источники *Developer Center*¹ сайта MSDN:

- «*Multi-Tenant Data Architecture*» (Многотенантная архитектура данных) по адресу <http://msdn.microsoft.com/en-us/architecture/aa479086.aspx>.
- «*Software + Services (S+S)*» по адресу <http://msdn.microsoft.com/en-us/architecture/aa699384.aspx>.
- Веб-трансляция Жанпаоло Каррапо (Gianpaolo Carraro) «*Software + Services for Architects*» (S + S для архитекторов) по адресу <http://www.microsoft.com/feeds/msdn/en-us/architecture/media/SaaS/ssForArchitects.aspx>.

Архитектуре S + S посвящены следующие материалы *Architecture Journal*² сайта MSDN:

- «*A Planet Ruled by Software Architectures*» (Планетой правят архитектуры ПО) по адресу <http://msdn.microsoft.com/en-us/architecture/bb906059.aspx>.
- «*Head in the Cloud, Feet on the Ground*» (Витаем в облаках, но крепко стоим на ногах) по адресу <http://msdn.microsoft.com/en-us/library/dd129910.aspx>
- «*Enterprise Mash Ups*» (Корпоративные составные приложения) по адресу <http://msdn.microsoft.com/en-us/architecture/bb906060.aspx>.
- «*Implications of Software + Services Consumption for Enterprise IT*» (Сложности с использованием S + S для корпоративных ИТ) по адресу <http://msdn.microsoft.com/en-us/architecture/bb906061.aspx>.

¹ Центр разработчиков (прим. переводчика).

² Журнал по архитектуре (прим. переводчика).

- «*Microsoft Office as a Platform for Software + Services*» (Microsoft Office как платформа для S + S) по адресу <http://msdn.microsoft.com/en-us/architecture/bb906062.aspx>.
- «*The Internet Service Bus*» (Шина Интернет-сервисов) по адресу <http://msdn.microsoft.com/en-us/architecture/bb906065.aspx>.

Инфраструктура с открытым исходным кодом NHibernate обсуждается в статье «*NHibernate Forge*» по адресу <http://nhforge.org/Default.aspx>.

27

Проектирование офисных бизнес-приложений

Обзор

В данной главе рассмотрена концепция офисных бизнес-приложений (Office Business Applications, OBA), типовая архитектура ОВА и характерные для нее компоненты. Также описаны некоторые общие сценарии, в которых применение ОВА наиболее целесообразно, предложено руководство по проектированию и важные шаблоны ОВА. Кроме того, в этой главе приведены сведения по интеграции ОВА с Microsoft Office SharePoint Server (MOSS) и бизнес-приложениями (LOB).

Офисные бизнес-приложения — это класс составных корпоративных приложений. Они обеспечивают решения, которые объединяют основные возможности сетевых бизнес-систем (с широко развертываемыми и активно используемыми сервисами для повышения эффективности бизнеса) и приложений, составляющих систему Microsoft Office. Пользователь взаимодействует с бизнес-логикой, реализованной в ОВА, через пользовательские формы. Такой подход обеспечивает насыщенный пользовательский интерфейс, упрощающий анализ бизнес-данных и помогающий интегрировать существующие внутренние или внешние системы.

ОВА обычно интегрированы с новыми или существующими LOB-приложениями. Они используют насыщенный UI и возможности автоматизации клиентов Office для упрощения сложных процессов, требующих взаимодействия с пользователем, и помогают сократить количество ошибок и улучшить существующие в организации процессы. В сущности, в ОВА клиентские офисные приложения используются в качестве посредников между существующими LOB-системами и пользователями. На рис. 1 показаны ключевые компоненты и слои ОВА. Следует обратить внимание на слой Средств совместной работы между слоями Представления и Сервисов приложения. В этом слое располагаются компоненты для хранения и управления общими рабочими потоками в ориентированном на работу с документами стиле.

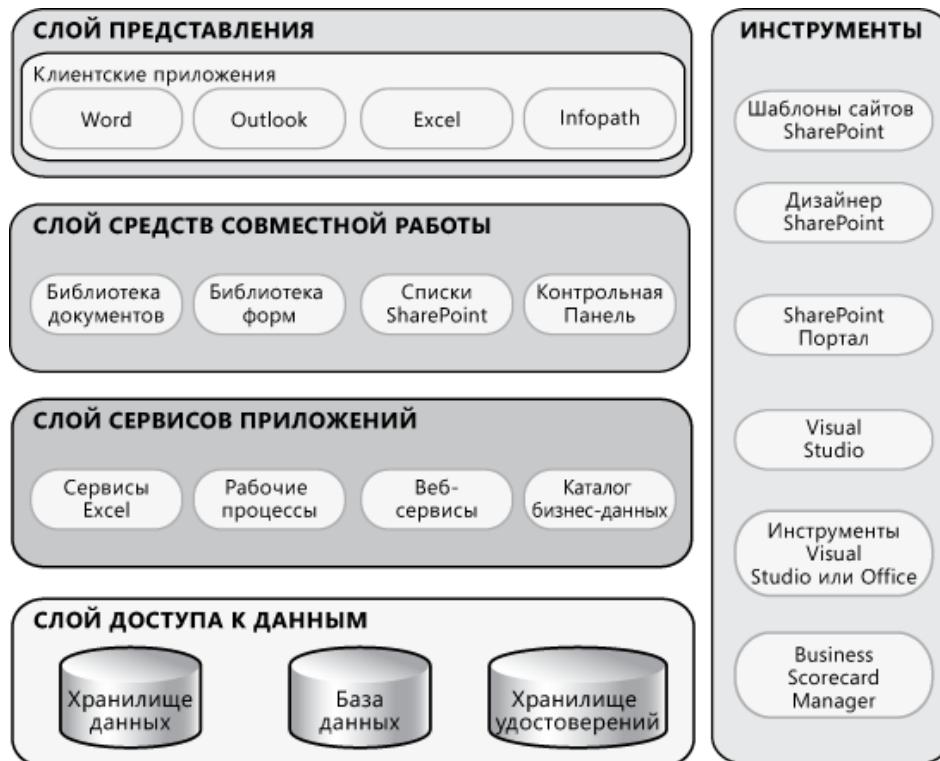


Рис. 44

Ключевые компоненты ОВА

Компоненты офисных бизнес-приложений

ОВА состоит из целого ряда взаимодействующих приложений и сервисов, обеспечивающих комплексное решение бизнес-проблемы. Офисные бизнес-приложения могут включать такие элементы:

- **Клиентские приложения Microsoft Office.** К клиентским приложениям относятся клиентское приложение для обмена сообщениями и координации совместной деятельности Outlook®, Word, Excel, приложение для сбора данных InfoPath® и средства создания презентаций PowerPoint®. Пользовательские формы Outlook могут применяться для размещения элементов управления UI с возможностью интеграции бизнес-логики и данных из различных источников. Word и Excel предлагают возможности программирования расширений в виде Панелей задач (Task Pane), Смарт-тегов (Smart Tags) и элементов Ленты (Ribbon), что позволяет объединить обычную работу с документом со структурированными бизнес-данными и процессами. Смарт-теги обеспечивают распознавание идентификаторов в тексте документа (например, телефонных номеров, идентификационных номеров или номеров счетов клиентов) через их сопоставление с регулярными выражениями. Наряду с данными в документе могут быть представлены соответствующие действия.
- **Windows SharePoint Services (WSS).** Построенная на основе Windows Server платформа WSS предлагает возможности управления содержимым и координации совместной деятельности, что помогает улучшить бизнес-процессы и эффективность группы. WSS применяется в офисных бизнес-приложениях для хранения и совместного

использования документов, форм и списков, для поддержки синхронизации в автономном режиме и управления задачами.

- **MOSS.** MOSS расширяет возможности WSS, предлагая дополнительные функции уровня предприятия для управления содержимым, рабочими процессами, поиском, порталами и персонализированными сайтами. Офисные бизнес-приложения используют MOSS для получения доступа к этим функциям, сервисы Excel — для формирования отчетов, каталог бизнес-данных (Business Data Catalog, BDC) — для доступа к бизнес-приложениям (LOB), а инфраструктура безопасности — для получения возможности единой регистрации (Single Sign On, SSO).
- **Технологии и сервисы.** Сервисы Excel обеспечивают пользователям возможность создавать документы в Excel, как обычно, и сохранять их на сервере SharePoint. Конечные пользователи получают возможность просматривать и работать с документами через Веб-браузер, а разработчики могут взаимодействовать с бизнес-логикой документов программно. Кроме того, офисные бизнес-приложения с помощью встроенной в MOSS функциональности Windows Workflow Foundation (WF) могут перехватывать процесс (например, подтверждение заказа товара), исправлять ошибки пользователей и, соответственно, сокращать связанные с ними задержки. Для создания Веб-сайтов, отвечающих требованиям компании, используются элементы визуального отображения Веб-страницы и Веб-части ASP.NET.
- **Функции координации совместной деятельности.** Координация совместной деятельности осуществляется посредством Microsoft Office Communications Server (OCS), программного обеспечения Microsoft Office Groove® и Microsoft Exchange Server.
- **Инструменты разработки.** Сюда входят SharePoint Central Administration, SharePoint Designer, Visual Studio и Visual Studio Tools для Office.

Основные сценарии использования офисных бизнес-приложений

Взаимодействие офисных бизнес-приложений (ОБА) осуществляется на базе открытых стандартов, стандартных форматов файлов и Веб-сервисов. Для описания метаданных объектов ОБА-решений используются XML-схемы. Все продукты Microsoft Office на всех уровнях поддерживают сервисы. Для создания бизнес-документов в качестве схем по умолчанию используются форматы файлов OpenXML с возможностью взаимодействия. Существует три основных сценария применения ОБА, соответственно которым эти приложения разделяются на три категории. К ним относятся:

- **Управление корпоративной информацией** обеспечивает пользователям возможность на основании их ролей находить данные и работать с ними.
- **Бизнес-аналитика** позволяет выполнять бизнес-анализ посредством таких возможностей, как серверные Excel-решения.
- **Унифицированный обмен сообщениями** открывает возможность обмена данными и координации совместной деятельности для упрощения управления группой.

Эти категории подробно рассматриваются в следующих разделах.

Управление корпоративной информацией

Использование клиентских офисных приложений в сценариях управления корпоративной информацией обеспечивает возможность поиска и работы с данными на основании бизнес-ролей пользователей или поставленных задач. Эти приложения могут взаимодействовать непосредственно с LOB-системой, которая предоставляет данные. Однако, как показано на рис. 2, более распространенным сценарием в бизнес-средах является использование MOSS или WSS в качестве инструмента управления содержимым клиентских офисных документов.

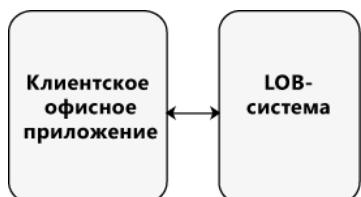


Рис. 45а

Прямое взаимодействие клиентского офисного приложения с LOB-системой

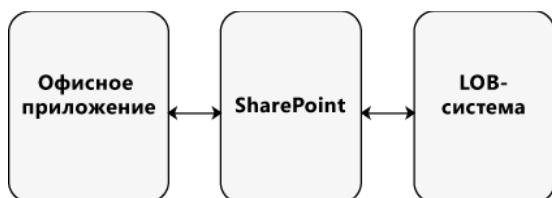


Рис. 26

Взаимодействие клиентского офисного приложения с LOB-системой через SharePoint

Использование SharePoint-решения позволяет реализовать управление версиями и рабочими процессами для файлов, связанных с клиентскими офисными приложениями. Кроме того, многие файлы могут быть изменены в среде SharePoint, при этом механизмы MOSS используют Excel для создания и отображения отчетов. В результате, многие ключевые сценарии основываются на использовании SharePoint с клиентскими офисными приложениями. Нижеперечисленные шаблоны офисных ОВА, подробно описанные далее в этой главе, применяются для реализации сценариев управления корпоративной информацией:

- Шаблон [Extended Reach Channel](#) (Расширенный канал доступа) позволяет увеличить аудиторию пользователей LOB-приложения благодаря использованию офисных приложений в качестве канала.
- Шаблон [Document Workflow](#) (Рабочий процесс обработки документов) обеспечивает поддержку управления ориентированными на работу с документами процессами, позволяет применять лучшие практики и улучшить базовые бизнес-процессы.
- Шаблон [Collaboration](#) (Координация совместной деятельности) согласует структурированные бизнес-процессы с неструктурированной деятельностью пользователей.

Бизнес-аналитика

Сценарии бизнес-аналитики позволяют проводить бизнес-анализ посредством таких возможностей, как серверные Excel-решения. Нижеперечисленные шаблоны офисных бизнес-приложений, подробно описываемые далее в данной главе, используются для реализации сценариев бизнес-аналитики:

- Шаблон [Document Integration](#) (Интеграция документов) делает возможным формирование офисных документов из LOB-приложений; позволяет информационным работникам встраивать LOB-данные в документы Office путем взаимодействия с ними в процессе создания документа; разрешает обработку на сервере документов, содержащих LOB-данные.
- Шаблон [Composite UI](#) (Составной UI) поддерживает объединение множества UI приложений в документ Office или Веб-страницу SharePoint.
- Шаблон [Data Consolidation](#) (Консолидация данных) предлагает более естественный способ работы с LOB-данными, предоставляя пользователям средства поиска данных во множестве LOB-приложений с последующей обработкой результатов поиска. Этот шаблон предполагает использование шаблона Discovery Navigation (Поисковая навигация).

Унифицированный обмен сообщениями

Сценарии унифицированного обмена сообщениями поддерживают взаимодействие и координацию совместной деятельности, что упрощает управление группой. Шаблон [Notification and Tasks](#) (Уведомления и задачи) позволяет реализовывать сценарии унифицированного обмена сообщениями, о чем пойдет речь далее в этой главе. Шаблон Notification and Tasks предполагает использование программы Outlook в качестве основного UI для получения и работы с задачами и предупреждениями, сформированными LOB-приложением.

Общие шаблоны ОВА

Офисные бизнес-приложения весьма разнообразны — начиная с очень простых, и заканчивая исключительно сложными решениями. Обычно в состав ОВА входит один или несколько общих шаблонов, которые рассматриваются в следующих разделах:

- [Расширенный канал доступа](#)
- [Интеграция документов](#)
- [Рабочий процесс обработки документов](#)
- [Составной UI](#)
- [Консолидация данных \(Поисковая навигация\)](#)
- [Координация совместной деятельности](#)
- [Уведомления и задачи](#)

Расширенный канал доступа

Приложения типа Extended Reach Channel позволяют увеличить аудиторию пользователей за счет использования офисного приложения в качестве канала. Шаблон Extended Reach Channel применяется для реализации следующих сценариев:

- Устранение избыточности, существующей в среде предприятия и связанной с повтором работ, например, функция Outlook, которая позволяет задавать время встречи для работы над текущими проектами.
- Расширение функциональности LOB для увеличения аудитории пользователей, например, приложение самообслуживания, которое позволяет сотрудникам обновлять личные данные.
- Улучшение существующей системы, не пользующейся популярностью у пользователей в силу ее избыточности или отсутствия соответствующих навыков работы с ней.
- Сбор данных от пользователей по электронной почте и автоматическое обновление системы.

Шаблон Extended Reach Channel поддерживает два разных шаблона интеграции: Direct Integration (Прямая интеграция) и Mediated Integration (Опосредованная интеграция). В следующих разделах эти шаблоны описаны более подробно.

Шаблон Direct Integration

Шаблон Direct Integration описывает порядок прямого предоставления LOB-функциональности с помощью клиентских офисных приложений для увеличения целевой аудитории пользователей. Доступ к LOB-интерфейсам в этом шаблоне реализуется непосредственно в клиенте Office или расширяется до существующего поведения, такого как ведение календаря. Клиентское приложение может осуществлять доступ к LOB-данным через Веб-сервис или просто отображать выходные данные (например, HTML), сформированные LOB-системой, как показано на рис. 3.

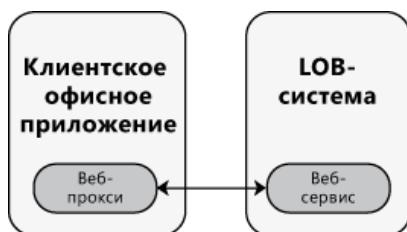


Рис. 46а

Шаблон Direct Integration с использованием Веб-сервисов

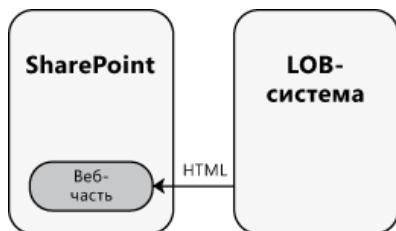


Рис. 36

Шаблон Direct Integration с использованием HTML

Шаблон Mediated Integration

Шаблон Mediated Integration описывает возможности использования хранилищ метаданных, таких как BDC, для создания дополнительного уровня абстракции. Абстракция обеспечивает общие подходы к управлению LOB-документами, включая систему безопасности с механизмом SSO, основанным на сопоставлении учетных данных. Этот шаблон предлагает больше возможностей для компоновки сервисов и данных в составной UI. Промежуточное звено, в роли которого может выступать BDC, осуществляет сбор данных из несопоставимых источников и предоставляет их в виде совместимых с Office форматов и сервисов, с которыми могут работать клиентские приложения. Рис. 4 иллюстрирует шаблон Mediated Integration.

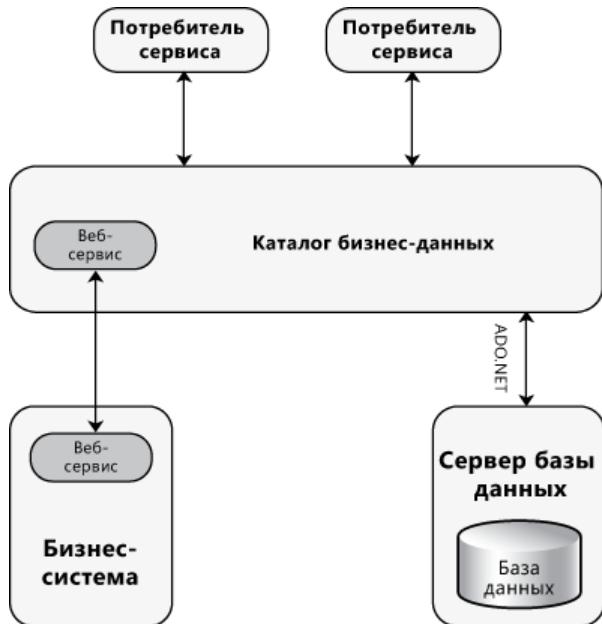


Рис. 47

Шаблон Mediated Integration

Интеграция документов

Приложения типа Document Integration делают возможным формирование документов Office из LOB-приложений; позволяют информационным работникам встраивать LOB-данные в документы Office путем взаимодействия с LOB-данными в процессе создания документа; обеспечивают возможность обработки документов, содержащих LOB-данные, на стороне сервера. Шаблон Document Integration применяется для реализации следующих сценариев:

- Сокращение дублирования LOB-данных, хранящихся в отдельных документах Office настольных систем пользователей.
- Обеспечение офисных приложений определенными подмножествами LOB-данных для реализации таких задач, как автоматическое составление стандартных писем или формирование отчетов.
- Создание документов Office, включающих элементы LOB-данных в соответствующем формате и автоматически обновляющихся при изменении данных. Избегайте создания типовых компоновок вручную — в офисных приложениях они должны по возможности создаваться с использованием шаблонов.

- Создание документов, требующих специальной обработки LOB-данных на стороне сервера. Встраивание этих данных должно осуществляться с применение открытых стандартов.
- Прием входящих документов, обработка встроенных данных и их применение в LOB-системе.

Шаблон Document Integration поддерживает четыре разных шаблона интеграции, в которых передача данных в и из LOB-систем осуществляется с помощью XML. Самый простой шаблон — Application Generated Documents (Создаваемые приложением документы). Кроме того, существует еще три шаблона интеллектуальной интеграции документов: Embedded LOB Information (Встроенные LOB-данные), Intelligent Documents/Embedded LOB Template (Интеллектуальные документы/Встроенный LOB-шаблон) и Intelligent Documents/LOB Information Recognizer (Интеллектуальные документы/Распознаватель LOB-данных). Более подробно эти шаблоны рассматриваются в следующих разделах.

Шаблон Application Generated Documents

Шаблон Application Generated Documents описывает методы объединения LOB-системой бизнес-данных и документа Office посредством пакетной обработки на стороне сервера, хотя выполнение данной операции на стороне клиента также возможно. К типовым примерам относятся экспорт данных в таблицы Excel или формирование отчетов и писем в Word. Это наиболее популярный шаблон интеграции данных и документов.



Рис. 48

Шаблон Application Generated Documents

Шаблон Intelligent Documents/Embedded LOB Information

Шаблон Intelligent Documents/Embedded LOB Information описывает механизмы встраивания LOB-данных непосредственно в тело документа Office или в качестве XML-части документа с предоставлением через элемент управления содержимым. В качестве альтернативы для отображения LOB-данных офисное приложение может использовать Office Custom Task Pane¹ (СТР). Информационный работник получает возможность просматривать эти данные, выполнять поиск и встраивать их в документ. На рис. 6 представлена схема шаблона Embedded LOB Information.

¹ Настраиваемая область задач (прим. переводчика).



Рис. 49

Шаблон *Intelligent Documents/Embedded LOB Information*

Шаблон Intelligent Documents/Embedded LOB Template

Шаблон Intelligent Documents/Embedded LOB Template описывает возможности объединения метаданных LOB-системы с разметкой документа (элементы управления содержимым, XML-схемы, закладки, именованные диапазоны и смарт-теги) по заданному шаблону.

Формирование документа в процессе выполнения осуществляется путем объединения шаблона с соответствующими экземплярами LOB-данных. Объединение может выполняться посредством подключаемого модуля клиентского офисного приложения или на сервере.

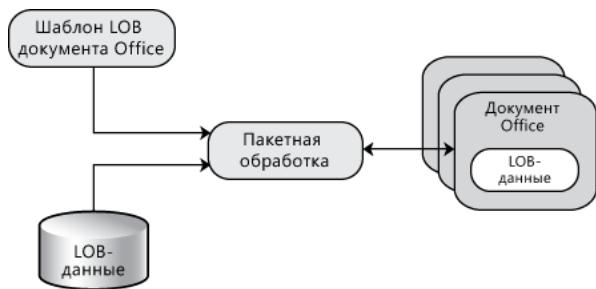


Рис. 50

Шаблон *Intelligent Documents/Embedded LOB Template*

Шаблон Intelligent Documents/LOB Information Recognizer

Шаблон Intelligent Documents/LOB Information Recognizer описывает принципы представления данных, распознаваемых LOB-системой, с помощью метаданных и разметки документа, например, с помощью элементов управления содержимым, XML-схем, закладок, именованных диапазонов и смарт-тегов. Приложение может использовать эти данные для обновления LOB-системы или для предоставления дополнительной функциональности пользователям. На стороне сервера, используя соответствующие данные, приложение может запускать рабочий процесс. На стороне клиента приложение может предоставлять контекстно-зависимые данные, скажем, сведения о клиенте, имя которого распознано в документе Word.

Рабочий процесс обработки документов

Приложения типа Document Workflow обеспечивают поддержку управления и мониторинга ориентированных на работу с документами процессов, позволяют применять лучшие практики и улучшать базовые бизнес-процессы. Шаблон Document Workflow используется для реализации следующих сценариев:

- Приложения, обменивающиеся данными, обычно по электронной почте, или выполняющие многошаговые задачи, например, прогнозирование, бюджетирование и контроль происшествий.

- Приложения, в которых необходимо выполнение определенных нормативных или корпоративных процедур с сохранением данных аудита.
- Приложения, выполняющие сложные задачи по обработке документов и условной маршрутизации или реализующие лучшие практики на базе правил.

При реализации данного шаблона должны быть учтены требования рабочего процесса. Однако, по возможности, необходимо избегать создания пользовательских компонентов рабочего процесса и отдавать предпочтение возможностям рабочих процессов SharePoint. Подход Document Workflow поддерживает два шаблона интеграции, запускающих рабочие процессы:

- **Шаблон LOB Initiated Document Workflow (Инициируемый LOB рабочий процесс обработки документов).** С помощью какого-либо действия, скажем, сохранения в библиотеке документов SharePoint или отправки формы InfoPath, документы автоматически передаются в рабочий процесс обработки документов SharePoint. Рабочий процесс может отправить документ следующему получателю в списке, сохранить копии или обработать его в зависимости от требований приложения.
- **Шаблон Cooperating Document Workflow (Взаимодействующий рабочий процесс обработки документов).** Определенные типы взаимодействия документов и LOB-систем должны подчиняться некоторым правилам или препятствовать выполнению определенных действий. В качестве примеров можно привести предотвращение редактирования отправленного документа на определенном этапе выполнения процесса, извлечение необходимых данных и их возвращение в LOB-систему. Данный шаблон обычно использует взаимодействующий рабочий процесс SharePoint, который обеспечивает логику процесса, в то время как интеллектуальный документ обеспечивает механизмы взаимодействия с LOB. В сложных сценариях LOB-система может также обновлять документ при его прохождении через рабочий процесс.

Составной UI

Приложения типа Composite UI поддерживают композицию множества пользовательских интерфейсов приложений в рамках документа Office или Веб-страницы SharePoint. Шаблон Composite UI применяется для реализации следующих сценариев:

- Приложения, собирающие и отображающие несколько различных типов данных на одной странице UI или в одном окне.
- Приложения, которые используют данные, предоставляемые множеством сетевых систем, и отображают их на одной странице UI или в одном окне.
- Приложения, которые должны предоставлять пользователям настраиваемый составной интерфейс с возможностью изменений соответственно любым требованиям.

Реализация шаблона Composite UI должна выполняться соответственно стандартам Office. Избегайте создания собственных компонентов, если доступны компоненты Веб-частей с

требуемой функциональностью. Данный подход поддерживает несколько шаблонов интеграции, которые обеспечивают объединение данных в составной UI:

- **Шаблон Context Driven Composite User Interface (Управляемый контекстом составной пользовательский интерфейс).** Контекст определяет структуру UI. Контекстные данные могут быть статическими (например, конфигурация приложения или вкладка, добавленная в представление Outlook) или динамическими (например, отображение или сокрытие данных вкладки в исходном документе). Каждая область составного UI представляет данные через клиентский компонент Office. Однако пользователи в ходе выполнения не могут динамически изменять связи между компонентами документа и исходными данными LOB-системы.
- **Шаблон Mesh Composite View (Объединенное композитное представление).** UI содержит такие компоненты, как Веб-части ASP.NET или MOSS, работающие совместно для получения данных одной или разных LOB-систем. Например, часть, отвечающая за представление клиента в системе управления отношениями с клиентами (Customer relationship management, CRM), в момент формирования представления может быть подключена к части, представляющей список открытых заказов системы планирования бизнес-ресурсов (Enterprise resource planning, ERP). При выборе клиента система CRM формирует событие и передает данные выбранного клиента в часть открытых заказов, которая отображает статус данного заказа.
- **Шаблон RSS and Web Services Composition (Композиция RSS и Веб-сервисов).** Данный шаблон является специализированной версией шаблона Mesh Composite View. Он объединяет данные, опубликованные посредством RSS-каналов или Веб-сервисов. Различные Веб-части представления данных SharePoint (или пользовательские части) форматируют и представляют опубликованные данные в UI. В качестве примера можно привести составное представление каталогов нескольких поставщиков, где каждый опубликованный элемент содержит ссылку на Веб-страницу поставщика с дополнительными сведениями.
- **Шаблон Analytics (Аналитика).** Данный шаблон является специализированной версией шаблона Mesh Composite View и предлагает конечным пользователям панель анализа данных. Для отображения данных и диаграмм в составном UI используются сервисы Excel Services и Веб-части Excel Services, предложенные в MOSS 2007. Для отображения пользовательских данных и данных LOB-системы и других источников могут применяться и другие Веб-части. Веб-часть Ключевой показатель производительности (Key Performance Indicator, KPI), предложенная MOSS, является очень удобным компонентом панели анализа данных. С ее помощью пользователи получили возможность определять KPI на основании данных любого списка SharePoint, включая список BDC.

Консолидация данных (Поисковая навигация)

Приложения типа Data Consolidation предлагают более естественный способ работы с LOB-данными, предоставляя пользователям возможность применения средств поиска по множеству LOB-приложений с последующей обработкой результатов поиска. Эти приложения

работают с достаточно большими объемами данных LOB-сущностей. Шаблон Data Consolidation использует шаблон Discovery Navigation и применяется для реализации следующих сценариев:

- Приложения, обеспечивающие возможности поиска внутри одной LOB-системы.
- Приложения, обеспечивающие возможности поиска в нескольких LOB-системах.
- Приложения, обеспечивающие возможности поиска в разнообразных LOB-системах и других источниках данных.

Шаблон Data Consolidation

Шаблон Data Consolidation обеспечивает единый механизм поиска, объединяя результаты множества операций поиска в одном или более источниках в одно результирующее множество. При этом он предоставляет не просто ссылки на результаты в виде Унифицированных идентификаторов ресурсов (Uniform Resource Identifiers, URI), но также действия, связанные с обнаруженными элементами. Рис. 8 иллюстрирует шаблон Data Consolidation для формирования индекса содержимого.

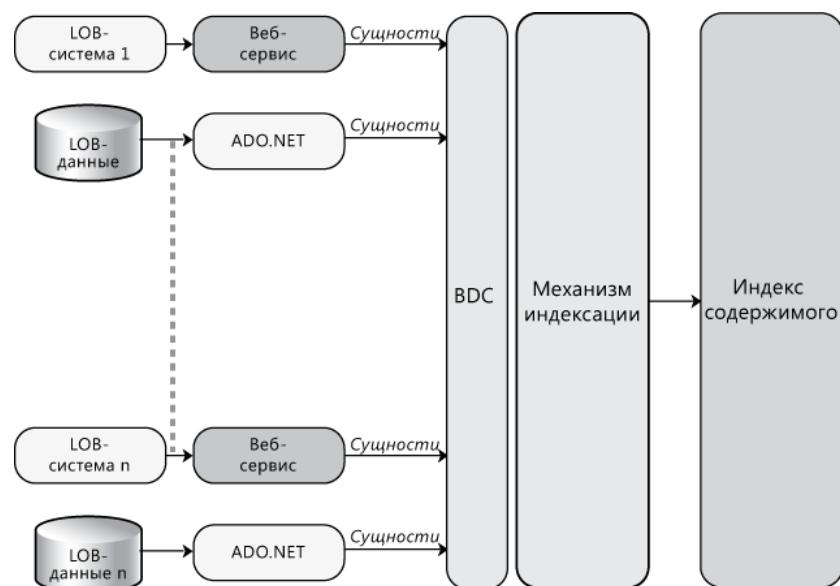


Рис. 51

Индекс содержимого включает данные, полученные из нескольких источников

Запуск LOB-процесса

Производный от Data Consolidation шаблон использует ссылки действий, которые запускают выполнение LOB-операций, таких как запуск рабочего процесса или обработка документа, как показано на рис. 9.

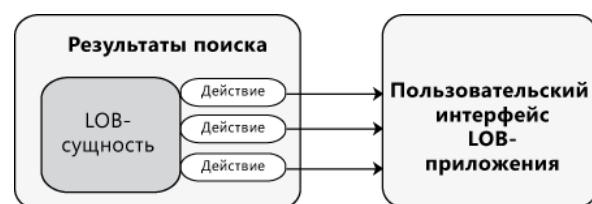


Рис. 52

Запуск LOB-процесса с помощью действия элемента, присутствующего в результатах поиска

Координация совместной деятельности

Приложения типа Collaboration согласуют структурированные бизнес-процессы с неструктуризованными действиями пользователей. Шаблон Collaboration применяется для реализации следующих сценариев:

- Приложения, включающие взаимодействие с пользователями, следствием которого является необходимость работы с LOB-системой, например, изучение конъюнктуры рынка перед оформлением заказа.
- LOB-приложения, получающие содержимое и пользовательские данные в неструктурированном виде, после чего должны представить их в структурированном формате.
- Приложения, предоставляющие данные в неструктурированной форме с возможностью ее редактирования пользователями, например, вики-сайт или форум.

Шаблон Collaboration использует MOSS-шаблоны Team Site (Портал проекта), благодаря чему пользователи получают возможность координировать совместную деятельность вокруг определенной бизнес-проблемы. Подобное взаимодействие осуществляется с помощью библиотек документов, списков обсуждений и задач, календарных планов группы и других элементарных функций управления проектами. Подготовка к работе и наполнение сайта содержимым может осуществляться с помощью LOB-данных; ссылки на LOB-процессы предоставляются на сайте в виде библиотек и списков. Доступ реализуется через документы Office или Веб-браузер.

Уведомления и задачи

Приложения, обеспечивающие поддержку уведомлений и задач, могут использовать приложение Outlook в качестве основного UI для получения и работы с задачами и предупреждениями, сформированными LOB-приложением. Кроме Outlook, SharePoint предоставляет сервисы уведомления и задач, которые могут взаимодействовать с большинством систем электронной почты через Simple Mail Transfer Protocol (SMTP). Шаблон Notifications and Tasks применяется для реализации следующих сценариев:

- Приложения, назначающие задачи и формирующие уведомления для конечных пользователей.
- Приложения, которые интегрируют множество LOB-операций и должны уведомлять пользователей о статусе или требованиях процесса.

Подход Notifications and Tasks, основанный на применении электронной почты, поддерживает несколько шаблонов интеграции, которые обеспечивают уведомление пользователей о задачах и статусе:

- **Шаблон Simple Task and Notification Delivery (Простая доставка уведомлений и задач).** LOB-система доставляет задачи и уведомления пользователям

однонаправленным потоком данных в виде задач Outlook и сообщений электронной почты. Детали задачи или уведомления встроены в тело задачи или сообщения электронной почты, но вносимые в них изменения не возвращаются в LOB-систему. Варианты доставки задач и уведомлений включают доставку на Microsoft Exchange Server (модель с активным источником данных), использование подключаемого модуля Outlook для извлечения задач и уведомлений (модель с пассивным источником данных) или публикацию RSS-канала, на который могут подписаться пользователи.

- **Шаблон Direct Task Synchronization (Прямая синхронизация задач).** LOB-система отправляет пользователям задачи через Exchange или Outlook в виде синхронизированного двунаправленного потока данных. Пользователи и LOB могут обновлять задачи в любое время, при этом изменения передаются в LOB-систему. Задача может являться составляющей частью рабочего процесса LOB.
- **Шаблон Mediated Task Synchronization (Опосредованная синхронизация задач).** Данный шаблон представляет собой разновидность шаблона Direct Task Synchronization, где MOSS выступает в роли посредника между LOB-системой и Outlook, выполняющим синхронизацию задач. LOB-система публикует задачи в список задач SharePoint, который синхронизирован с задачами Outlook посредством встроенного в Outlook механизма синхронизации. Обновления задачи в Outlook автоматически возвращаются в SharePoint. При этом формируется событие, сигнализирующее об изменении, благодаря чему код обработчика получает возможность выполнить обновление LOB-системы.
- **Шаблон Intelligent Tasks and Notifications (Интеллектуальные задачи и уведомления).** Благодаря ссылкам на действия, которые находятся в настраиваемой области задач Outlook, пользователи могут запускать выполнение определенного действия на основании отправленных LOB-системой задач или уведомлений. К типовым задачам относятся автоматический вход в LOB-систему, поиск необходимых данных и их обновление. В качестве примера можно привести процедуру утверждения отпусков сотрудников: руководитель получает из отдела кадров электронные письма с запросами об отпусках; Настраиваемая область задач (СТР) содержит ссылки на действия; щелчком соответствующей ссылки руководитель подтверждает или отклоняет запрос работника, обновляя LOB-систему.
- **Шаблон Form-based Tasks and Notifications (Задачи и уведомления на основании форм).** Данный шаблон представляет собой разновидность шаблона Intelligent Tasks and Notification, где сообщение электронной почты содержит вложенную форму InfoPath, предварительно заполненную LOB-системой. Пользователь может открыть сообщение, заполнить форму и отправить ее в LOB-систему. InfoPath обеспечивает проверку данных, специальные расчеты и вспомогательную логику для помощи пользователю при заполнении формы. Чтобы обеспечить пользователю удобство работы, настраиваемая область задач InfoPath может также предоставлять дополнительные сведения, извлекаемые из LOB-системы. Разновидность этого шаблона использует MOSS InfoPath Forms Services, благодаря чему пользователи могут заполнять формы в Веб-браузере без необходимости установки InfoPath.

Общие вопросы проектирования

Проектирование офисных бизнес-приложений требует поддержки определенных сценариев и подходящих для них клиентских офисных приложений. Кроме использования базовых шаблонов, рассматриваемых в предыдущем разделе, при проектировании ОВА руководствуйтесь следующими рекомендациями:

- **Отдавайте преимущество шаблону опосредованной интеграции перед шаблоном прямой интеграции.** Если ОВА проектируется как расширенный насыщенный канал, интерфейсы могут быть реализованы непосредственно в документах. Например, таблица Excel может содержать пользовательские формы ввода данных. Однако такой подход требует написания собственного кода и ограничивает возможности повторного использования функциональности. Шаблон опосредованной интеграции позволяет отделить интерфейсы от физических документов, воспользовавшись преимуществами SharePoint, в частности Business Data Catalog.
- **Используйте структуры на базе OpenXML для встраивания LOB-данных в документы.** OpenXML является международным стандартом European Computer Manufacturers Association (ECMA)¹, поддерживаемым приложениями Office 2007 и многими другими независимыми поставщиками и платформами. Применение OpenXML позволяет совместно использовать данные офисных приложений и приложений, разработанных для других платформ.
- **Создавайте шаблоны LOB-документов для общих компоновок, которые будут использоваться повторно.** LOB-шаблон содержит разметку и метаданные, ассоциированные с LOB, которые впоследствии могут быть связаны с определенными экземплярами LOB-данных. Иначе говоря, новые документы генерируются путем объединения LOB-данных с шаблонами документов. Конечные пользователи могут создавать собственные документы без привлечения разработчиков, а сложные документы могут формироваться посредством пакетной обработки данных на стороне сервера.
- **Используйте MOSS для управления просмотром и процессом визирования документов.** MOSS предлагает функции поддержки основного рабочего процесса для просмотра и визирования документов. WF удовлетворяет более сложным требованиям по обработке, расширяя возможности рабочих процессов SharePoint.
- **Используйте шаблон Collaboration для координации совместной деятельности пользователей.** Большинство LOB-приложений замечательно подходят для обработки структурированных бизнес-процессов. Однако они не слишком хороши для обработки неструктурированных бизнес-процессов, возникающих при взаимодействии с пользователями. Сайт, на котором реализована совместная деятельность пользователей, решает эти проблемы с помощью интерфейса, настроенного на

¹ Европейская ассоциация производителей вычислительной техники (прим. переводчика).

поддержку общей работы пользователей. Данный шаблон реализован в SharePoint-шаблоне Team Site.

- **Примите во внимание требования удаленной синхронизации данных.** Создаваемые, обновляемые и распространяемые документы должны быть синхронизированы с LOB-системой, после чего сохранены для дальнейшего использования. Несмотря на замечательные возможности по обработке действий транзакционного типа, LOB-системы не подходят для выполнения значительных объемов работ между отдельными действиями.

Вопросы безопасности

Офисные бизнес-приложения используют для предоставления данных и функциональности несколько типов клиентских приложений и имеют доступ к корпоративным LOB-данным, поэтому безопасность играет в них важную роль. Важно обеспечить безопасность доступа ко всем ресурсам и защитить данные, передаваемые по сети. При создании ОВА руководствуйтесь следующими рекомендациями:

- Реализуйте SSO, чтобы пользователи могли использовать для доступа к клиентским приложениям и сетевым функциям свои текущие учетные данные или учетные данные, проверенные интегрированным сервисом, таким как Active Directory или SharePoint.
- По возможности шифруйте сообщения, передаваемые во внешнюю по отношению к вашей защищенной сети среду. Для защиты сетевого соединения между серверами и клиентами могут использоваться механизмы шифрования на уровне канала, например, Internet Protocol Security (IPSec).
- Применение модели доверенная подсистема позволит выполнять доступ к данным, используя учетные данные ролей, и сократить требуемое количество подключений. Более подробно модель доверенная подсистема рассматривается в главе 19, [«Физические уровни и развертывание»](#).
- Где это необходимо, фильтруйте данные на сервере, чтобы предотвратить разглашение конфиденциальных данных клиентских приложений.

Вопросы развертывания

Развертывание ОВА-решений осуществляется с помощью пакета Windows Installer или технологии Click Once:

- **Click Once** требует минимального участия пользователя, небольших усилий для разработчика и обеспечивает автоматизированное обновление. Тем не менее, он может использоваться для развертывания только отдельного решения, не являющегося частью большего решения. Этот подход нельзя применять для развертывания дополнительных файлов или разделов реестра; он не обеспечивает

пользователю возможность конфигурации установки; и не позволяет создавать фирменный стиль установщика.

- **Windows Installer** дает возможность развертывания дополнительных компонентов и настроек реестра; взаимодействует с пользователем для конфигурации установки; поддерживает настраиваемую фирменную установку. Однако он требует расширенной настройки, больших усилий разработчиков и не поддерживает автоматического обновления.

Шаблоны проектирования

Основные шаблоны проектирования организованы по категориям, таким как Координация совместной деятельности, Составной UI, Консолидация данных, Интеграция документов, Рабочий процесс обработки документов, Расширенный канал доступа и Задачи и уведомления, и представлены в следующей таблице. Рассмотрите возможности применения этих шаблонов при принятии проектных решений для каждой из категорий.

Категория	Шаблоны
Координация совместной деятельности	Collaboration. Координирует неструктурированную деятельность пользователей со структуризованными бизнес-процессами.
Составной UI	Analytics. Специализированная версия шаблона Mesh Composite View, предлагающая конечным пользователям панель анализа данных. Context Driven Composite User Interface. Использует контекстную информацию для определения композиции UI. Mesh Composite View. Использует такие компоненты UI, как Веб-части ASP.NET или компоненты MOSS, которые функционируют совместно для предоставления данных одной или разных LOB-систем. RSS and Web Services Composition. Специализированная версия шаблона Mesh Composite View, объединяющая данные, опубликованные посредством RSS-каналов или Веб-сервисов.
Консолидация данных	Discovery Navigation. Позволяет пользователям осуществлять поиск данных во множестве LOB-приложений и с последующей обработкой результатов.
Интеграция документов	Application Generated Documents. LOB-система объединяет бизнес-данные и документ Office, используя пакетную обработку на стороне сервера. Embedded LOB Information. LOB-данные встраиваются непосредственно в тело документа Office или встраиваются как XML-часть документа с предоставлением через элемент управления содержимым. Embedded LOB Template. Шаблон объединяет метаданные LOB-системы с разметкой документа, например, элементами управления содержимым, XML-схемами, закладками, именованными диапазонами и смарт-тегами. LOB Information Recognizer. Данные, распознаваемые LOB-системой, представляются посредством метаданных и разметки документа,

	например, элементов управления содержимым, XML-схем, закладок, именованных диапазонов и смарт-тегов.
<i>Рабочий процесс обработки документов</i>	<p>Cooperating Document Workflow. Набор взаимодействий документов и LOB-систем, которые должны подчиняться определенным правилам или препятствовать выполнению определенных действий.</p> <p>LOB Initiated Document Workflow. Документы автоматически передаются в рабочий процесс обработки документов SharePoint с помощью какого-либо действия, такого как сохранение в библиотеке документов SharePoint или отправка формы InfoPath.</p>
<i>Расширенный канал доступа</i>	<p>Direct Integration. Доступ к LOB-интерфейсам реализуется непосредственно в клиенте Office или расширяется до существующего поведения, такого как ведение календаря.</p> <p>Mediated Integration. Промежуточное звено, в роли которого может выступать BDC, осуществляет сбор данных из несопоставимых источников и преобразует их в совместимые с Office форматы или сервисы, которые могут быть использованы клиентскими приложениями.</p>
<i>Задачи и уведомления</i>	<p>Direct Task Synchronization. LOB-система отправляет пользователям задачи через Exchange или Outlook в виде синхронизированного двунаправленного потока данных.</p> <p>Form-based Tasks and Notifications. Разновидность шаблона Intelligent Tasks and Notification, где сообщение электронной почты содержит вложенную форму InfoPath, предварительно заполненную LOB-системой.</p> <p>Intelligent Tasks and Notifications. Благодаря ссылкам на действия, которые находятся в Outlook CTP, пользователи могут инициировать выполнение определенного действия на основании переданных LOB-системой задач или уведомлений.</p> <p>Mediated Task Synchronization. Разновидность шаблона Direct Task Synchronization, где MOSS выступает в роли посредника между LOB-системой и Outlook, выполняющего синхронизацию задач.</p> <p>Simple Task and Notification Delivery. LOB-система односторонним потоком данных доставляет задачи и уведомления пользователям в виде задач Outlook и сообщений электронной почты.</p>

Шаблонам ОВА посвящена книга Роба Баркера, Иоанны Биксел, Адама Буэнца, Стива Фокса, Джона Холидея, Бушана Нене и Картика Равиндрана «*6 офисных бизнес-приложений для Office SharePoint® Server 2007*». Эком, 2009. Отрывок из этой книги, «*Getting Started with Office Business Applications*» (Начало работы с офисными бизнес-приложениями), можно найти на сайте <http://msdn.microsoft.com/en-us/library/bb614539.aspx>

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Automating Public Sector Forms Processing and Workflow with Office Business Application*» (Автоматизация обработки данных форм и рабочих процессов с помощью офисных

бизнес-приложений) по адресу

<http://blogs.msdn.com/singaporedp/archive/tags/OBA/default.aspx>.

- «*Getting Started with Office Business Applications*» (Знакомство с офисными бизнес-приложениями) по адресу <http://msdn.microsoft.com/en-us/library/bb614538.aspx>.
- «*OBA (Reference Application Pack) RAP for E-Forms processing*» (Справочный пакет ОВА для обработки электронных форм) по адресу <http://msdn2.microsoft.com/en-us/architecture/bb643796.aspx>.
- Слайды PowerPoint и исходный код по адресу <http://msdn2.microsoft.com/en-us/architecture/bb643796.aspx>.
- «OBA Central» (Центр ОВА) по адресу <http://www.obacentral.com/>.
- «*Integrating LOB Systems with the Microsoft Office System*» (Интеграция LOB-систем с системой Microsoft Office) по адресу <http://msdn.microsoft.com/en-us/architecture/bb896607.aspx>.
- «*Understanding Office Development*» (Понимание принципов проектирования офисных приложений) по адресу <http://msdn.microsoft.com/en-us/office/aa905371.aspx>.

28

Проектирование LOB-приложений SharePoint

Обзор

Данная глава рассказывает об архитектуре типового бизнес-приложения (Line-Of-Business, LOB) SharePoint и его компонентах, знакомит с ключевыми сценариями и принципами проектирования. Также в ней рассматриваются процесс развертывания, основные шаблоны и вопросы проектирования таких приложений.

Microsoft Windows Server® является основной операционной системой для выполнения LOB-приложений SharePoint. SharePoint имеет тесную связь с платформой Microsoft, используя Internet Information Services (IIS) в качестве интерфейсного Веб-сервера для размещения Веб-сайтов и SQL Server в качестве сетевого хранилища для описаний сайтов, описаний типов содержимого, опубликованного содержимого и данных конфигурации. LOB-приложения SharePoint можно конфигурировать для публикации содержимого, предназначенного для Интернет, через Веб-сайты. Для обслуживания большего числа пользователей Веб-сайты могут развертываться на Веб-ферме и интегрироваться с ASP.NET для представления LOB-данных этим сайтам. При этом для UI могут использоваться Веб-части, стили, темы, шаблоны, серверные и пользовательские элементы управления ASP.NET. На рис. 1 представлены основные функции и слои LOB-приложения SharePoint.

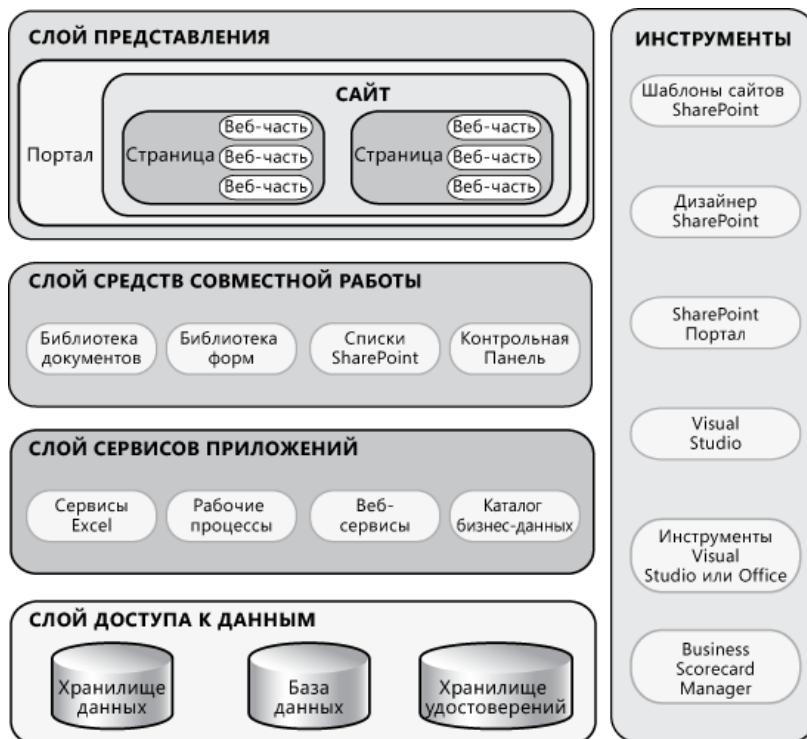


Рис. 53

Основные функции LOB-приложений SharePoint

Офисные бизнес-приложения (Office Business Applications, OBAs), о которых шла речь в главе 27, могут быть интегрированы с LOB-процессами для обеспечения насыщенных пользовательских интерфейсов для доступа к данным, анализа и работы с данными с помощью бизнес-порталов, созданных на базе Windows SharePoint Services (WSS) и Microsoft Office SharePoint Server (MOSS).

Логические слои LOB-приложения SharePoint

Рассмотрим, какие слои могут присутствовать в LOB-приложении SharePoint:

- **Слой представления.** Это UI приложения. Через Веб-браузер пользователи подключаются к порталу SharePoint Server, который состоит из Веб-страниц. Эти Веб-страницы можно компоновать с помощью Веб-частей, обеспечивающих насыщенную композицию на уровне представления. Веб-части также доступны для клиентских приложений Office. Кроме того, можно создавать собственные Веб-части для реализации специальных функций приложения.
- **Слой средств совместной работы.** Документы Office, например, таблицы Excel, хранятся в библиотеках документов. Формы, позволяющие автоматизировать задачи приложений Office, хранятся в библиотеках форм. Слой средств совместной работы также реализует функции создания и публикации отчетов в виде списков SharePoint или листов Excel. Кроме того, выходные данные могут формироваться в форме панели с данными из множества сервисов. В дополнение к этому, как говорилось в предыдущей главе, клиентские офисные приложения применяются для обработки данных и координации совместной деятельности.

- **Слой сервисов приложения.** Это доступный для повторного использования слой, который предоставляет сервисы, используемые слоем представления и слоем средств совместной работы. Данный слой включает сервисы Excel для формирования отчетов, рабочие процессы, использующие Windows Workflow Foundation (WF) для реализации бизнес-процессов или управления жизненным циклом документа, и другие бизнес-сервисы. Кроме того, доступ к данным для пользователей может быть реализован посредством Business Data Catalog (BDC)¹.
- **Слой данных.** Этот слой инкапсулирует механизмы хранения и доступа ко всем типам данных, используемых приложением. Сюда входят роли и удостоверения, операционные данные и содержащие LOB-данные хранилища данных.

Развертывание физического уровня

В предыдущем разделе описана логическая группировка компонентов или функциональности LOB-приложения SharePoint по слоям. Однако необходимо понимать также и физическое распределение компонентов по серверам своей инфраструктуры. Рассмотрим общие сценарии и рекомендации:

- Для обеспечения максимальной надежности и производительности развертывание базы данных для SharePoint осуществляется на отдельном сервере базы данных или в отдельном кластере базы данных.
- В нераспределенном сценарии выполняйте развертывание слоев представления, средств совместной работы и сервисов приложения на одном Веб-сервере или Веб-ферме.
- В нераспределенном сценарии выполняйте развертывание компонентов слоя представления (порталы, сайты, страницы и Веб-части) на одном Веб-сервере или Веб-ферме, а остальные слои и компоненты размещайте на другом сервере приложений или ферме приложений.
- Для достижения максимальной производительности при интенсивной нагрузке развертывание компонентов слоя сервисов приложения следует выполнять на отдельном сервере приложений или ферме приложений.

Основные сценарии и функции

LOB-приложения SharePoint предполагают использование открытых стандартов, стандартных форматов файлов и Веб-сервисов. Описания метаданных объектов LOB-решений SharePoint основаны на XML-схемах. Все продукты Microsoft Office на всех уровнях поддерживают сервисы и используют для создания бизнес-документов обеспечивающие возможность взаимодействия форматы файлов OpenXML.

MOSS способствует обеспечению функций управления содержимым и реализации бизнес-процессов в LOB-приложениях SharePoint. Сайты SharePoint поддерживают публикацию

¹ Каталог бизнес-данных (прим. переводчика).

специального содержимого, управление содержимым, управление записями и требования бизнес-аналитики. Кроме того, можно также проводить эффективный поиск людей, документов и данных; принимать участие в бизнес-процессах на основе форм; получать доступ и анализировать большие объемы бизнес-данных. Рассмотрим функции LOB-приложений SharePoint:

- **Рабочий процесс.** MOSS интегрирована с WF и позволяет разработчикам создавать простые рабочие процессы и подключать их к библиотекам документов в SharePoint. С помощью SharePoint Designer можно создавать также пользовательские рабочие процессы.
- **Бизнес-аналитика.** MOSS предоставляет интерактивные порталы бизнес-аналитики, которые поддерживают обработку и анализ данных. Пользователи получают возможность создавать панели для отображения данных из множества источников без необходимости написания кода. Ключевые показатели производительности (Key Performance Indicator, KPI) могут быть определены посредством сервисов Excel, списков SharePoint, кубов SQL Server Analysis Services и множества других источников. Размещение этих данных в SharePoint позволяет активно использовать их в других сервисах SharePoint, например, для поиска и рабочих процессов.
- **Управление содержимым.** Добавление функциональности Microsoft Content Management Server (MCMS)¹ в MOSS позволяет использовать широкие возможности управления Веб-содержимым, доступные непосредственно из платформы SharePoint.
- **Поиск.** Общий сервис Enterprise Search (Поиск в среде предприятия) в MOSS обеспечивает расширенный и расширяемый сбор, индексирование, формирование запросов, поиск по ключевым словам и полнотекстовый поиск.
- **Каталог бизнес-данных.** BDC обеспечивает возможность предоставления корпоративных данных Веб-частям, серверу форм InfoPath и функциям поиска. Разработчики могут с помощью BDC создавать приложения, благодаря которым пользователи взаимодействуют с LOB-данными через привычные интерфейсы.
- **Формат файлов OpenXML.** Применение формата файлов OpenXML в офисных приложениях обеспечивает весь спектр возможностей обработки документов на стороне сервера.

Общие принципы проектирования

SharePoint предоставляет множество базовых функций для работы с LOB-приложением, однако существует несколько ключевых аспектов проектирования, на которые следует обратить внимание. К ним относятся взаимодействие с пользователем и выбор клиентского интерфейса, а также вопросы эксплуатации и обслуживания. При проектировании LOB-приложения SharePoint руководствуйтесь следующими рекомендациями:

¹ Сервер управления содержимым (прим. переводчика).

- **Обеспечьте пользователя интерфейсом, соответствующим его роли.** Для разных пользовательских ролей должны предлагаться разные варианты UI. SharePoint обладает функциональностью, позволяющей автоматически изменять интерфейс в зависимости от ролей и групп пользователя. Применение групп доступа или выбор целевой аудитории позволит обеспечить пользователей только соответствующими опциями.
- **Интегрируйте LOB-системы с клиентскими приложениями Office.** Выбирайте шаблоны (например, шаблон Direct Integration или Mediated Integration) для интеграции LOB-системы с клиентскими офисными приложениями соответственно специфике решений и функциональным требованиям. Используйте ADO.NET или Веб-сервисы для шаблона Direct Integration. Используйте MOSS в качестве сервера приложений промежуточного уровня для шаблона Mediated Integration. Более подробно эти шаблоны рассматриваются в главе 27, «[Проектирование офисных бизнес-приложений](#)».
- **Избегайте тесного связывания слоев.** Используйте Веб-сервисы для разрешения зависимостей между слоями и избегайте слишком тесного их связывания.
- **Учитывайте требования удаленной синхронизации данных.** Все создаваемые, обновляемые или распространяемые документы следует синхронизировать с LOB-системой и сохранять для дальнейшего использования. Несмотря на замечательные возможности по обработке действий транзакционного типа, LOB-системы не подходят для выполнения значительных объемов работ между отдельными действиями
- **С помощью сервисов предоставьте доступ к серверным LOB-данным для использования в SharePoint и ОВА-приложениях.** Открывая доступ к своей сетевой системе данных через сервисы, вы обеспечиваете возможность SharePoint и расширениям ОВА запрашивать, обрабатывать и изменять формат данных для пользователя. Таким образом, SharePoint можно применять для расширения поведения сетевой системы без необходимости написания дополнительного кода.

Специальные вопросы проектирования

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям дизайна. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- [Каталог бизнес-данных](#)
- [Хранилище документов и содержимого](#)
- [Сервисы Excel](#)
- [Сервисы форм InfoPath](#)
- [Объектная модель SharePoint](#)

- [Веб-части](#)
 - [Рабочий процесс](#)
-

Каталог бизнес-данных

BDC позволяет предоставить доступ к корпоративным данным Веб-частям, серверу форм InfoPath и функциям поиска. Разработчики могут применять BDC для создания приложений, позволяющих пользователям взаимодействовать с LOB-данными через привычные интерфейсы. При разработке BDC-приложений руководствуйтесь следующими рекомендациями:

- Убедитесь, что структура источников данных позволяет BDC работать с ними напрямую, и определите, как эти данные будут использоваться, например, для поиска, профилей пользователей или простого отображения. Во избежание предоставления излишних данных необходимо задать соответствующую область поиска. Для обеспечения наилучшей производительности применяйте только самые новые драйверы доступа к источникам данных.
 - Выполните аутентификацию пользователей и процессов при подключении к источникам данных. Для аутентификации сетевых источников данных используйте возможности единой регистрации (Single Sign On, SSO), предлагаемые SharePoint.
 - Применение BDC Definition Editor (Редактор описаний), который входит в Office Server SDK, позволит сократить количество ошибок при создании файла определения приложения (Application Definition File, ADF). Если редактирование ADF выполняется вручную, загрузите схему BDCMetadata.xsd в Visual Studio, чтобы избежать ошибок.
 - В случае необходимости используйте BDC Security Trimmer (Триммер безопасности) для обеспечения настраиваемой фильтрации экземпляров сущностей по ролям безопасности.
 - Избегайте перегрузки промежуточной области.
-

Хранилище документов и содержимого

Офисные документы, например, таблицы Excel, хранятся в библиотеках документов. Настольные офисные приложения могут использоваться для консолидации разнообразного содержимого из множества источников данных. При хранении содержимого в SharePoint руководствуйтесь следующими рекомендациями:

- При хранении документов в библиотеках документов определяйте дополнительные централизованные метаданные для каждого типа документа, используя типы содержимого и их возможности наследования. Типы содержимого корневых сайтов автоматически распространяются на дочерние сайты и могут быть задействованы в них. Кроме того, новые типы содержимого могут быть унаследованы от существующих типов и расширены. Использование этого подхода позволит

упростить настройку типов содержимого, и в этом случае не придется управлять каждым типом содержимого в отдельности.

- Заранее определяйте и планируйте требуемые типы содержимого, задавайте уникальные имена полей метаданных и их связи, шаблоны документов и пользовательские формы с типами содержимого. Если необходимо обеспечить доступ к типу содержимого из любого дочернего сайта, создавайте его на уровне сайта. Если тип содержимого должен быть доступен только для определенного списка, создавайте его на уровне этого списка.
- Настраивайте Document Information Panel (Область сведений о документе) для сбора метаданных типа содержимого, чтобы отслеживать и редактировать метаданные документов. В Document Information Panel можно добавить бизнес-логику или проверку данных.
- Сохраняйте настраиваемые пользователем справочные данные или неизменяемые данные в списках. Однако не стоит отождествлять списки SharePoint с таблицами базы данных. В базе данных храните изменяемые или транзакционные данные. Если список будет запрашиваться из приложения многократно, кэшируйте его содержимое в **DataTable** или **DataSet**.
- Не заменяйте файловые системы библиотеками документов SharePoint, не пытайтесь использовать библиотеки документов SharePoint в качестве механизма управления версиями или как платформу координации совместной деятельности группы разработчиков. Библиотека документов предназначена исключительно для хранения документов, требующих совместного использования и управления.
- Учтите, что один контейнер списков в библиотеке или списке документов может включать максимум 2000 элементов. Создавайте собственный UI, если необходимо работать с контейнером списка, включающим более 2000 элементов.
- Организация документов с помощью папок, а не фильтрованных представлений, поможет ускорить поиск.

Сервисы Excel

Сервисы Excel состоят из трех основных компонентов. Компонент Excel Calculation Services (Сервисы вычислений Excel) обеспечивает загрузку рабочей книги, выполнение расчетов, обновление внешних данных и настройку сеансов. Excel Web Access (Веб-клиент Excel) представляет собой Веб-часть, которая отображает и поддерживает работу с рабочей книгой Excel в браузере. Excel Web Services (Веб-сервисы Excel) – это Веб-сервис, размещаемый в SharePoint и обеспечивающий разработчикам методы для создания собственных приложений на базе рабочей книги Excel. Планируя использование сервисов Excel, руководствуйтесь следующими рекомендациями:

- Выполняйте аутентификацию всех пользователей и защитите свои файлы подключения к данным Office (Open Data Connection). Настройте аутентификацию Kerberos или SSO для сервисов Excel, чтобы получить возможность аутентификации баз данных SQL Server, расположенных на других серверах.

- Перед публикацией рабочих книг настройте доверяемые расположения файлов и доверяемые библиотеки подключений к данным. Публикуйте только необходимые данные.
- Перед публикацией убедитесь в том, что рабочие книги Excel хранятся в доверяемых каталогах, а файлы подключения к данным Office загружены в надежные библиотеки.

Сервисы форм InfoPath

Благодаря сервисам форм InfoPath пользователи получают возможность работать с формами для браузера, которые создаются на базе шаблонов, хранятся в SharePoint и предоставляются через InfoPath. Если формы, созданные на базе совместимых с браузером шаблонов (.xsn), развернуты на сервере, выполняющем InfoPath Form Services, их можно открывать в Веб-браузере на компьютерах, на которых не установлен Office InfoPath 2007. Если Office InfoPath 2007 установлен, формы открываются в нем. Для обеспечения работоспособности форм InfoPath в сервисах форм (Form Services) руководствуйтесь следующими рекомендациями:

- Создавайте единообразные формы, которые выглядят и выполняются одинаково, независимо от того, где они отображаются, в клиентском приложении Office (Word, Excel или PowerPoint) или с помощью Веб-интерфейса SharePoint Server.
- Используйте панель задач Design Checker в InfoPath для проверки совместимости отображаемых в браузере форм. Создавая формы для отображения в браузере, выберите опцию «Enable browser-compatible features only» (Включить только совместимые с браузером функции), чтобы скрыть неподдерживаемые элементы управления.
- Используйте не одно представление со скрытым содержимым, а несколько представлений. Это позволит улучшить производительность форм и сократить время их отклика. Не полагайтесь на минимум безопасность, обеспечиваемую скрытием данных при использовании представлений.
- Активируйте специальную защиту, чтобы сохранить целостность шаблонов форм и предотвратить их изменение пользователями. Выкладывая формы на общедоступные сайты, убедитесь в том, что шаблоны форм недоступны сценариям или автоматизированным процессам, в противном случае, они будут уязвимыми для атак типа «отказ в обслуживании» (Denial of Service, DoS). Кроме того, общедоступные формы не должны включать конфиденциальные данные, такие как аутентификационные данные, имена серверов или баз данных.
- Не используйте InfoPath Form Services при проектировании решений для формирования отчетов, которые требуют больших объемов данных.
- Если необходимо сформировать отчет, отправляйте данные формы в базу данных. Все конфиденциальные данные, поступающие через формы, сохраняйте в базе данных.

- Использование файлов универсального подключения к данным (Universal Data Connection, UDC) обеспечит возможности гибкого управления подключениями к данным и повторного использования.
 - При настройке состояния сеанса для сервисов форм InfoPath используйте Form View (Представление формы).
-

Объектная модель SharePoint

SharePoint предлагает объектную модель, которая позволяет создавать код для автоматизации процессов. Например, можно реализовать специальную систему управления версиями документов или собственные политики регистрации изменений. При написании кода с использованием объектной модели SharePoint руководствуйтесь следующими рекомендациями:

- После использования удаляйте созданные объекты SharePoint для высвобождения неуправляемых ресурсов. Также убедитесь, что удаление объектов SharePoint правильно реализовано в обработчиках исключений.
 - Правильно выберите подход к кэшированию и избегайте кэширования конфиденциальных или изменяющихся данных. Если необходимо кэширование, загружайте данные объектов SharePoint в **DataSet** или **DataTable**. Однако при кэшировании объектов SharePoint необходимо обеспечить синхронизацию и безопасность потоков.
 - Повысящая уровень привилегий, обратите внимание, что только вновь созданным объектам SharePoint будут назначены привилегии с повышенным уровнем.
-

Веб-части

Веб-части обеспечивают насыщенную композицию на уровне представления. Можно создавать собственные Веб-части для реализации специальных функций и можно использовать стандартные Веб-части, предоставляемые SharePoint и другими средами, такими как ASP.NET. Веб-части обеспечивают взаимодействие с сетевыми LOB-приложениями или Веб-сервисами, с их помощью создаются составные настраиваемые интерфейсы, позволяющие реализовывать персонализацию в LOB-приложениях SharePoint. Если для Веб-частей необходимо обеспечить разрешения, превышающие стандартные права ASP.NET, создавайте собственную политику доступа к коду. При проектировании Веб-частей руководствуйтесь следующими рекомендациями:

- Определите функциональность, которую хотите реализовать в Веб-частях, и источники данных, с которыми будут работать Веб-части. Используйте команды Веб-частей, чтобы пользователи могли выполнять отдельные действия. Собственные свойства распределите по категориям, чтобы отличать их от стандартных свойств Веб-частей.
- Для проектирования Веб-частей используйте многослойный подход. Разделение логики представления, бизнес-логики и логики доступа к данным обеспечивает

удобство и простоту обслуживания в дальнейшем. Проектируйте Веб-части таким образом, чтобы каждая из них выполняла только одну функцию, что улучшает возможности повторного использования. По возможности создавайте настраиваемые или конфигурируемые Веб-части.

- Реализуйте для Веб-частей соответствующие меры безопасности. Выполняйте развертывание Веб-частей в глобальный кэш сборок только в том случае, если нет необходимости в обеспечении безопасности отдельных пользователей.
- Используйте зоны Веб-частей для размещения Веб-частей, управляемых пользователями в процессе выполнения. В случае применения главной страницы ASP.NET, интегрируйте Web Part Manager (Диспетчер Веб-частей) в главные страницы, которые будут использоваться страницами Веб-частей. Избегайте определения атрибутов стилей непосредственно в элементах управления Веб-частей.
- Корректно удаляйте объекты SharePoint и неуправляемые ресурсы, создаваемые в Веб-частях.

Рабочий процесс

SharePoint позволяет разработчикам создавать простые рабочие процессы и подключать их к библиотекам документов в SharePoint. Кроме того, есть возможность создавать пользовательские рабочие процессы с помощью SharePoint Designer или Visual Studio. При проектировании рабочих процессов руководствуйтесь следующими рекомендациями:

- Четко представляйте, какие бизнес-процессы или его части будут автоматизированы. Приступая к непосредственной реализации рабочих процессов убедитесь, что существующие бизнес-процессы верны и хорошо задокументированы. Проконсультируйтесь со специалистом предметной области или бизнес-аналитиком для анализа существующих бизнес-процессов.
- Правильно выберите технологию рабочих процессов, которая будет соответствовать бизнес-требованиям. Например, для реализации простых требований, таких как визирование документа, может быть достаточно стандартных рабочих процессов SharePoint. Если стандартные рабочие процессы не обеспечивают требуемой функциональности, создавайте собственные рабочие процессы с помощью SharePoint Designer.
- Создавайте собственные рабочие процессы в Visual Studio, если бизнес-требования требуют сложных рабочих процессов или интеграции с LOB-системами. Чтобы расширить возможности информационных работников, создавайте действия рабочего процесса, которые можно зарегистрировать с помощью SharePoint Designer.
- Создавая собственные рабочие процессы, правильно выбирайте их тип, соответственно реализуемому сценарию. Можно использовать последовательную модель и модель на основе состояний. Реализуйте расширенный инструментарий отладки в коде.

- При отладке собственных рабочих процессов задавайте детальный уровень протоколирования.
- При обновлении старых рабочих процессов применяйте контроль версий сборок и изменяйте глобально уникальный идентификатор (Globally Unique Identifier, GUID) решения. Разворачивая новые версии, не забывайте о том, как это может повлиять на выполняющиеся текущие экземпляры рабочих процессов.
- Формируйте отдельные списки журналов и списки задач для рабочих процессов, создаваемых конечными пользователями.
- Для удобства обслуживания ассоциируйте рабочие процессы с определенным типом содержимого. Это позволит использовать рабочий процесс в разных библиотеках содержимого, но обслуживание рабочего процесса будет выполняться только в одном месте (эта функциональность доступна для стандартных рабочих процессов и рабочих процессов Visual Studio, но не для рабочих процессов, созданных в SharePoint Designer). Имейте в виду, что для элемента списка может выполняться только один экземпляр рабочего процесса одного типа; экземпляры рабочих процессов запускаются только для элементов списка, но не для самого списка.

Вопросы выбора технологий

Следующие рекомендации помогут правильно выбрать технологию реализации рабочего процесса SharePoint. Предлагаем также руководство по созданию Веб-частей для пользовательских интерфейсов SharePoint:

- Используйте WF, если требуется создавать надежные рабочие процессы с автоматической поддержкой безопасности, обеспечивающие обмен данными с использованием транзакций, широкий выбор вариантов передачи данных и кодирования, встроенное хранение и отслеживание действий.
- Используйте BizTalk Server, если требуется создавать рабочие процессы, реализующие сложную оркестровку и обеспечивающие надежное хранение и пересылку сообщений.
- Используйте Microsoft BizTalk ESB Toolkit, если требуется обеспечивать взаимодействие с не-Microsoft системами, осуществлять электронный обмен данными (Electronic Data Interchange, EDI) или реализовывать шаблоны сервисной шины предприятия (Enterprise Service Bus, ESB).
- Используйте MOSS, если бизнес-уровень включает единственный сайт SharePoint и не требует реализации доступа к данным других сайтов. MOSS не подходит для сценариев с несколькими сайтами.
- Если нет необходимости в обеспечении обратной совместимости с SharePoint 2003, создавайте Веб-части ASP.NET путем наследования от класса **System.Web.UI.WebControls.WebParts.WebPart**. Когда требуется поддерживать

SharePoint 2003, реализуйте наследование от класса **Microsoft.SharePoint.WebPartPages.WebPart**.

Принципы развертывания

Большая часть функциональности LOB-приложений SharePoint обеспечивается стандартными функциями SharePoint. Что касается дополнительных артефактов, таких как компоненты, они должны развертываться таким образом, чтобы обеспечить возможность SharePoint доступа и работы с ними. При проектировании стратегии развертывания LOB-приложений SharePoint руководствуйтесь следующими рекомендациями:

- Определите область действия своих функций: ферма, Веб-приложение, семейство сайтов или один сайт.
 - Организовывайте собственные функции в решения.
 - Развертывайте свои сборки в папку BIN, а не в глобальный кэш сборок, чтобы использовать преимущества низкоуровневого механизма обеспечения безопасности доступа к коду.
 - После развертывания протестируйте решение, используя учетную запись без прав администратора.
-

Шаблоны проектирования

Основные шаблоны проектирования представлены в следующей таблице. Используйте эти шаблоны, принимая решения по проектированию собственных LOB-приложений SharePoint.

Категория	Шаблоны проектирования
Рабочие процессы	Data-driven workflow. Рабочий процесс, последовательность задач которого определяется значениями данных рабочего процесса или системы. Human workflow. Рабочий процесс, задачи которого выполняются вручную пользователями. Sequential workflow. Рабочий процесс, задачи которого выполняются последовательно, когда выполнение каждой последующей задачи начинается лишь после завершения предыдущей. State-driven workflow. Рабочий процесс, последовательность задач которого определяется состоянием системы.

Более подробно шаблоны Data-Driven Workflow, Human Workflow, Sequential Workflow и State-Driven Workflow рассматриваются в следующих материалах:

- «*Windows Workflow Foundation Overview*» по адресу <http://msdn.microsoft.com/en-us/library/ms734631.aspx>.
- «*Workflow Patterns*» по адресу <http://www.workflowpatterns.com/>.

Дополнительные источники

Электронная версия списка используемых источников по MOSS и WSS для создания LOB-приложений SharePoint доступна по адресу <http://www.microsoft.com/architectureguide>.

- «*Developing Workflow Solutions with SharePoint Server 2007 and Windows Workflow Foundation*» (Разработка решений рабочих процессов с помощью SharePoint Server 2007 и Windows Workflow Foundation) по адресу <http://msdn.microsoft.com/en-us/library/cc514224.aspx>.
 - «*Best Practices: Common Coding Issues When Using the SharePoint Object Model*» (Лучшие практики: общие вопросы программирования при работе с объектной моделью SharePoint) по адресу <http://msdn.microsoft.com/en-us/library/bb687949.aspx>.
 - «*Best Practices: Using Disposable Windows SharePoint Services Objects*» (Лучшие практики: использование высвобождаемых объектов Windows SharePoint Services) по адресу <http://msdn.microsoft.com/en-us/library/aa973248.aspx>.
 - «*InfoPath Forms Services Best Practices*» (Лучшие практики InfoPath Forms Services) по адресу <http://technet.microsoft.com/en-us/library/cc261832.aspx>.
 - «*White paper: Working with large lists in Office SharePoint Server 2007*» (Техническое описание: работа с большими списками в Office SharePoint Server 2007) по адресу <http://technet.microsoft.com/en-us/library/cc262813.aspx>.
-

Приложения

В этот раздел руководства включены приложения, предлагающие обзор платформы приложений Microsoft. Здесь представлены матрицы технологий, охватывающие такие темы, как представление, доступ к данным, интеграция и рабочий процесс, которые помогут оценить и выбрать подходящие технологии для конкретного сценария приложения. Кроме того, вы познакомитесь с возможностями Enterprise Library от patterns & practices, которые помогают ускорить проектирование и разработку приложений, предлагая готовые решения для сквозной функциональности. В приложениях также представлен каталог шаблонов patterns & practices; в нем перечислены основные шаблоны, которые пригодятся при проектировании архитектуры.

Руководство включает следующие приложения:

- Приложение А: [Платформа приложений Microsoft](#)
 - Приложение Б: [Матрица технологий слоя представления](#)
 - Приложение В: [Матрица технологий слоя доступа к данным](#)
 - Приложение Г: [Матрица интеграционных технологий](#)
 - Приложение Д: [Матрица технологий рабочего процесса](#)
 - Приложение Е: [Enterprise Library от patterns & practices](#)
 - Приложение Ж: [Каталог шаблонов patterns & practices](#)
-

Приложение

A

Платформа приложений Microsoft

Обзор

Данное приложение начинается с обзора функциональных возможностей платформы Microsoft. Это обеспечит представление о существующих технологиях тем, кто еще не знаком с разработкой приложений на базе .NET Framework и серверных технологий Microsoft.

Следующий раздел, «Поиск данных и ресурсов», расскажет, как находить необходимую информацию на множестве Интернет-сайтов компании Microsoft.

Далее приведен обзор .NET Framework и Общезыковой среды выполнения (Common Language Runtime, CLR), за которым следуют несколько разделов, посвященных технологиям платформы приложений Microsoft, предлагаемым для создания разных типов приложений, координации совместной деятельности, интеграции, доступа к данным и рабочих процессов. Затем рассматриваются такие продукты, как SQL Server и IIS (Веб-сервер), инструменты разработки, как Visual Studio, и внешние библиотеки.

В состав платформы приложений Microsoft входят продукты, компоненты инфраструктуры, сервисы среды выполнения и .NET Framework. Все они перечислены в следующей таблице.

Категория	Технологии
Инфраструктура приложения	Common Language Runtime (CLR) .NET Framework
Координации совместной деятельности / Интеграция / Рабочий процесс	Windows Workflow Foundation (WF) Microsoft Office SharePoint Server (MOSS) Microsoft BizTalk Server
Доступ к данным	ADO.NET Core ADO.NET Data Services Framework

	ADO.NET Entity Framework ADO.NET Sync Services Language Integrated Query (LINQ)
<i>Сервер баз данных</i>	Microsoft SQL Server
<i>Инструменты разработки</i>	Microsoft Visual Studio Microsoft Expression® Studio design software
<i>Мобильное приложение</i>	.NET Compact Framework ASP.NET Mobile Silverlight Mobile
<i>Насыщенное клиентское приложение</i>	Windows Forms Windows Presentation Foundation (WPF)
<i>Насыщенное клиентское Интернет-приложение (RIA)</i>	Microsoft Silverlight
<i>Сервисы</i>	ASP.NET Web Services (ASMX) Windows Communication Foundation (WCF)
<i>Веб-технология</i>	ASP.NET
<i>Веб-сервер</i>	Internet Information Services (IIS)

Поиск данных и ресурсов

В данном разделе предлагается план проектирования архитектуры и описаны лучшие практики разработки приложений для .NET Framework и платформы приложений Microsoft. Несмотря на то, что данное руководство содержит много полезной информации, включая пошаговую реализацию типовых сценариев, оно не может охватить всех деталей и аспектов. Поэтому компания Microsoft поддерживает библиотеку, где собраны подробные руководства по всем предлагаемым ею технологиям, продуктам и сервисам, о чем пойдет речь в следующем разделе.

Как Microsoft организует техническую информацию в глобальной сети Интернет

Тем, кто только начинает свое знакомство с огромной коллекцией технической документации Microsoft, полезно будет узнать несколько вещей, которые помогут быстро находить необходимую информацию.

Прежде всего, компанией Microsoft поддерживается несколько больших Веб-сайтов разной направленности. На корпоративном Веб-сайте Microsoft (<http://www.microsoft.com>) представлена вся маркетинговая информация о продуктах компании, включая технические данные. Допустим, необходимо узнать официальную позицию компании по сервисно-ориентированной архитектуре (Service Oriented Architecture, SOA). Для получения доступа к этим сведениям достаточно перейти на сайт Microsoft.com и ввести в строке поиска «SOA». Сайт Microsoft.com используется для поддержания имиджа компании и описания преимуществ продуктов Microsoft. Все технические данные вынесены на специально предназначенные для этого Веб-сайты.

На главной странице Microsoft.com можно найти ссылки на техническую информацию, которая находится на других ресурсах. На главной странице можно найти такие категории, как «*Highlights*» (Главное), «*Latest Releases*» (Новые продукты), «*Using Your Computer*» (Пользователям ПК), «*For Business*» (Для бизнеса), и две категории для профессионалов, «*For IT Professionals*» (ИТ-специалистам) и «*For Developers*» (Разработчикам). Microsoft выделяет разработчиков программного обеспечения среди других ИТ-специалистов. Это сделано, главным образом, потому что для них предоставляется огромное количество сведений, и поскольку им необходима качественно иная информация по сравнению, скажем, с сетевыми администраторами или пользователями. Тогда как многие ссылки внутри этих категорий меняются, основные Веб-сайты технической информации остаются неизменными. Все материалы для разработчиков представлены на сайте Microsoft Developer Network (MSDN) по адресу <http://msdn.microsoft.com>. Основным сайтом для других ИТ-специалистов является Microsoft TechNet, который можно найти по адресу <http://technet.microsoft.com>.

Microsoft Developer Network

На сайте MSDN представлены такие разделы: Developer Centers (Центры разработки, обычно ориентированы под каждый отдельный инструмент разработки, язык, технологию или техническую предметную область), Library (Библиотека — огромное хранилище доступного для поиска содержимого), Downloads (Загрузки), Support Forums (Форумы поддержки) и Communities (Сообщества, где можно ознакомиться с мнением других участников и при желании внести свой вклад).

Существуют и другие не менее важные специализированные сайты. На сайте Channel9 (<http://channel9.msdn.com>) представлены неофициальные видео-ролики, которые обычно посвящены технологиям и инструментальным средствам Microsoft. В этих роликах специалисты по разработке программного обеспечения или архитекторы продуктов Microsoft рассказывают о технологиях, над которыми они работают, или обсуждают перспективы развития инструментов и технологий. CodePlex (<http://codeplex.com>) — это сайт компании Microsoft для размещения проектов с открытым исходным кодом. Здесь можно просмотреть проекты, над которыми работают открытые сообщества, или даже начать собственный проект. Группа patterns & practices Microsoft разрабатывает все свои предложения в открытых сообществах CodePlex, что позволяет реализовать обратную связь внутри сообщества на всех этапах жизненного цикла проекта.

Кроме того, на сайте MSDN есть чаты по технической тематике. Также можно подписаться на Веб-трансляции, что весьма полезно для изучения развивающихся или уже существующих технологий.

Microsoft TechNet

Microsoft TechNet предлагает подобную представленной на сайте MSDN информацию и возможности. Не удивительно, что на сайте TechNet вместо раздела Developer Centers представлен раздел TechCenters (Технические центры). Кроме этого, на сайте TechNet рассматриваются технические аспекты таких вопросов, как проектирование, развертывание, функционирование сетевой инфраструктуры, и предлагается руководство по установке и управлению продуктами Microsoft.

Технология .NET Framework

Если брать укрупненно, .NET Framework состоит из виртуальной среды выполнения, библиотеки классов и сервисов среды выполнения, используемых при разработке и выполнении .NET-приложений. Изначально .NET Framework была выпущена как среда выполнения с базовым набором классов для создания приложений.

В библиотеку Base Class Library (BCL) включен базовый набор классов, покрывающий широкий спектр потребностей при разработке в различных областях, включая UI, доступ к данным, подключение к базе данных, криптографию, вычислительные алгоритмы и сетевое взаимодействие.

На следующем уровне, поверх BCL, представлены основные технологии разработки .NET приложений. Сюда входят библиотеки классов, сервисы среды выполнения, сгруппированные по функциональному назначению, например, для реализации насыщенного клиентского приложения или доступа к данным. По мере развития платформы Microsoft .NET к основным технологиям добавляются новые, такие как WCF, WPF и WF.

Общеязыковая среда выполнения

.NET Framework включает виртуальную среду, которая управляет требованиями среды выполнения приложения. Эта среда называется общеязыковой средой выполнения (Common Language Runtime, CLR). CLR обеспечивает работу виртуальной машины таким образом, что программистам не нужно беспокоиться о частоте ЦП или других характеристиках оборудования, на котором будет выполняться приложение. Приложения, выполняемые в CLR, называются управляемыми. Приложения Microsoft .NET Framework разрабатываются с использованием управляемого кода (кода, который будет выполняться внутри CLR), хотя некоторые функции (такие как драйверы устройств, которые используют API ядра) часто реализованы с помощью неуправляемого кода. CLR также обеспечивает поддержку сервисов безопасности, управления памятью и обработки исключений.

Доступ к данным

Платформа Microsoft поддерживает следующие технологии доступа к данным:

- **ADO.NET Core.** ADO.NET Core предоставляет возможности извлечения, обновления и управления данными. Сюда входят поставщики для доступа к SQL Server, OLE DB, Open Database Connectivity (ODBC), SQL Server Compact Edition и баз данных Oracle.
- **ADO.NET Data Services Framework.** Эта инфраструктура предоставляет доступ к данным из любого источника с поддержкой Linq (обычно Entity Data Model) посредством Веб-сервисов REST, доступных по HTTP. Взаимодействие с данными может выполняться непосредственно через универсальные идентификаторы ресурсов (Uniform Resource Identifier, URI). Веб-сервис можно настроить таким образом, чтобы он возвращал данные в формате Atom или JavaScript Object Notation (JSON).
- **ADO.NET Entity Framework.** Эта инфраструктура обеспечивает строго типизированный доступ к реляционным базам данных. Она переводит модель данных от физической структуры реляционных таблиц к концептуальной модели, которая точно отображает

бизнес-объекты. Entity Framework предлагает обычную модель Entity Data Model в среде ADO.NET, что позволяет разработчикам определить гибкое сопоставление с реляционными данными. Такое сопоставление обеспечивает возможность изолировать приложение от изменений в базовой схеме хранения данных. Entity Framework также поддерживает технологию LINQ to Entities, которая поддерживает LINQ для бизнес-объектов, доступных через Entity Framework. Используя Entity Framework в качестве объектно-реляционного сопоставления (Object/Relational Mapping, O/RM), разработчики применяют к бизнес-объектам технологию LINQ to Entities. Эти бизнес-объекты будут преобразованы Entity Framework в Entity SQL, который сопоставлен с моделью Entity Data Model под управлением Entity Framework. Кроме того, разработчики могут работать непосредственно с Entity Data Model и задействовать Entity SQL в своих приложениях.

- **ADO.NET Sync Services¹.** Механизм ADO.NET Sync Services входит в состав Microsoft Sync Framework и применяется для реализации синхронизации для баз данных с поддержкой ADO.NET. Поддерживается синхронизация данных с приложениями без постоянного подключения. Через определенные промежутки времени осуществляется сбор данных от клиентской базы с последующей синхронизацией с серверной базой данных.
- **Language Integrated Query² (LINQ).** LINQ предлагает библиотеки классов, которые расширяют C# и Visual Basic встроенным синтаксисом языка запросов. Прежде всего, это технология запросов, поддерживаемая разными библиотеками .NET Framework. Например, технология LINQ to Entities включена в библиотеки ADO.NET Entity Framework, LINQ to XML входит в состав **System.Xml**, а LINQ to Objects – в состав системных библиотек .NET Framework. Запросы могут выполняться к разным форматам данных, включая **DataSet** (LINQ to **DataSet**), XML (LINQ to XML), объекты в памяти (LINQ to Objects), ADO.NET Data Services (LINQ to Data Services) и реляционные данные (LINQ to Entities).
- **LINQ to SQL.** Облегченное решение со строгой типизацией данных для построения запросов к SQL Server. Технология LINQ to SQL предназначена для простого и быстрого сохранения объектов, когда классы промежуточного уровня очень близко сопоставляются со структурами таблиц базы данных. Сценарии использования LINQ to SQL будут интегрированы и поддерживаться в ADO.NET Entity Framework, начиная с версии .NET Framework 4.0, но LINQ to SQL все равно будет поддерживаться. Более подробно эти вопросы рассматриваются в блоге группы ADO.NET по адресу <http://blogs.msdn.com/adonet/archive/2008/10/31/clarifying-the-message-on-l2s-futures.aspx>.

¹ Сервисы синхронизации ADO.NET (прим. переводчика).

² Встроенный язык запросов (прим. переводчика).

Мобильные приложения

Платформа .NET предлагает следующие технологии для мобильных приложений:

- **Microsoft .NET Compact Framework.** Эта версия Microsoft .NET Framework создана специально для мобильных устройств. Используйте данную технологию для мобильных приложений, которые предполагается выполнять на устройствах как автономные приложения или приложения без постоянного подключения.
- **ASP.NET for Mobile.** Данная версия ASP.NET ориентирована исключительно для мобильных устройств. Приложения ASP.NET for Mobile можно размещать на обычном Веб-сервере IIS. Используйте эту технологию для мобильных Веб-приложений, когда необходимо обеспечить поддержку большой номенклатуры мобильных устройств и браузеров, которые могут рассчитывать на гарантированное сетевое соединение.
- **Silverlight for Mobile.** Данная версия Silverlight-клиента требует установки подключаемого модуля Silverlight на мобильное устройство. Используйте эту технологию для переноса существующих приложений Silverlight на мобильные устройства. Для создания более насыщенного UI можно воспользоваться другими технологиями.

Насыщенное клиентское приложение

Windows-приложения выполняются в .NET Framework. Данная технология предлагает целый ряд вариантов насыщенных клиентских приложений:

- **Windows Forms.** Это стандартная технология создания UI для .NET Framework. Даже при наличии возможностей WPF, Windows Forms по-прежнему остается отличным решением для создания UI, тем более, если группа разработчиков уже имеет опыт работы с Windows Forms, а UI приложения не предполагает интенсивного использования графики или воспроизведения потокового мультимедиа.
- **Приложение Windows Presentation Foundation (WPF).** WPF-приложения обеспечивают более широкие графические возможности: двухмерную и трехмерную графику, независимость от разрешения, расширенную поддержку документов и полиграфического оформления, анимацию с временной шкалой, потоковое аудио и видео, векторную графику. Для реализации UI, привязки данных и определения событий WPF использует Extensible Application Markup Language (XAML). WPF также включает расширенные возможности привязки данных и шаблонов. В результате развертывания с использованием приложения браузера XAML (XAML browser application, XBAP) могут быть получены как настольные WPF-приложения, так и WPF-приложения браузера. WPF-приложения обеспечивают совместную работу разработчиков и дизайнеров, когда разработчики занимаются реализацией бизнес-логики, а дизайнеры — оформлением и поведением.
- **Windows Forms с пользовательскими элементами управления WPF.** Данный подход позволяет использовать преимущества более мощных возможностей UI, обеспечиваемых элементами управления WPF. WPF можно добавлять в

существующее приложение Windows Forms. Следует иметь в виду, что элементы управления WPF лучше работают на мощных клиентских компьютерах.

- **WPF с пользовательскими элементами управления Windows Forms.** Эта технология позволяет дополнить WPF элементами управления, которых нет в WPF. С помощью элемента управления **WindowsFormsHost** из сборки **WindowsFormsIntegration** можно добавить элементы управления Windows Forms. Однако существуют некоторые ограничения и несогласованности, связанные с перекрытием элементов управления, фокусом интерфейса и методами формирования визуального представления, которые отличаются для разных технологий.
- **XAML Browser Application (XBAP) с использованием WPF.** Эта технология выполняет WPF-приложение в защищенной программной среде внутри Microsoft Internet Explorer или Mozilla Firefox в Windows. В отличие от Silverlight, может использоваться большая часть инфраструктуры WPF, но существуют некоторые ограничения, связанные с доступом к системным ресурсам из частично доверяемой изолированной программной среды. XBAP требует установки на клиентском компьютере Windows Vista или обоих подключаемых модулей: .NET Framework 3.5 и браузера XBAP. XBAP — отличный выбор, если требуемые функции недоступны в Silverlight и имеется возможность определять клиентскую платформу и требования к доверию.

Насыщенное Интернет-приложение

В состав платформы приложений Microsoft входит технология Silverlight, позволяющая создавать насыщенные Интернет-приложения (Rich Internet Application, RIA). RIA размещаются на Веб-сервере, например, Windows Server Internet Information Services (IIS). Для создания RIA предлагаются следующие варианты:

- **Silverlight.** Это оптимизированная для браузера версия WPF с поддержкой разных платформ и браузеров. По сравнению с XBAP, Silverlight меньше и быстрее устанавливается, но не поддерживает трехмерную графику и документы с внутренним форматированием. Благодаря небольшому размеру и поддержке разных платформ, Silverlight является отличным решением для WPF-приложений, в которых не требуется обеспечивать поддержку расширенных графических возможностей, предоставляемых WPF.
- **Silverlight с AJAX.** Silverlight поддерживает Asynchronous JavaScript and XML (AJAX) и предоставляет объектную модель AJAX для сценариев JavaScript, размещаемых в Веб-странице. Эту функциональность можно использовать для обеспечения фонового взаимодействия между компонентами страницы и сервером, что позволит уменьшить время отклика пользовательского интерфейса.

Сервисы

Платформа .NET предлагает следующие технологии разработки сервисных приложений:

- **Windows Communication Foundation (WCF).** WCF предлагает управляемый подход для реализации распределенных вычислений, обеспечивает широкие возможности взаимодействия и включает прямую поддержку сервисов. WCF поддерживает ряд протоколов, включая HTTP, TCP, Microsoft Message Queuing и протокол именованных каналов.
 - **ASP.NET Web services (ASMX).** ASMX предлагает более простой подход к реализации распределенных вычислений и возможностей взаимодействия, но поддерживает только протокол HTTP.
-

Рабочий процесс

Платформа .NET предлагает следующие варианты реализации рабочих процессов:

- **Windows Workflow Foundation (WF).** WF является основополагающей технологией реализации рабочего процесса. Являясь инструментарием для профессиональных разработчиков и независимых поставщиков программных продуктов (Independent Software Vendor, ISV), которые создают последовательные рабочие процессы или рабочие процессы на основе конечных автоматов, WF поддерживает следующие типы рабочих процессов: Sequential (Последовательный), State-Machine (На основе конечных автоматов), Data Driven (Управляемый данными) и Custom (Пользовательский). Рабочие процессы могут создаваться с помощью Windows Workflow Designer (Дизайнер рабочих процессов) в Visual Studio.
- **Workflow Services.** Workflow Services интегрирует WCF с WF для обеспечения сервисов на базе WCF для рабочего процесса. Начиная с Microsoft .NET Framework 3.5, WCF расширен поддержкой рабочих процессов, представленных как сервисы, а также возможностью вызывать сервисы из рабочих процессов. Расширение WCF также позволило вызывать сервисы из рабочих процессов. Кроме того, в состав Visual Studio 2008 включены новые шаблоны и инструменты, поддерживающие сервисы рабочих процессов.
- **Microsoft Office SharePoint Services (MOSS).** MOSS — это платформа управления содержимым и координации совместной деятельности с поддержкой рабочих процессов на базе WF. MOSS предлагает решение для реализации бизнес-процессов и координации совместной деятельности в контексте сервера SharePoint. Рабочие процессы для визирования документов можно создавать непосредственно в интерфейсе MOSS. Кроме того, можно создавать рабочие процессы с помощью SharePoint Designer или Windows Workflow Designer в Visual Studio. Для настройки рабочих процессов используется объектная модель WF в Visual Studio.
- **Microsoft BizTalk Server.** BizTalk обладает собственным механизмом реализации рабочих процессов, ориентированным на оркестровку, например, корпоративную интеграцию с рабочими процессами уровня системы. Следующая версия BizTalk, возможно, будет использовать WF, а также XLANG (расширение языка описания Веб-сервисов (Web Service Definition Language, WSDL) для моделирования оркестровки и координации совместной деятельности сервисов), которая применяется в качестве

текущей технологии оркестровки в BizTalk. Использование BizTalk Orchestration Services позволяет определять общий дизайн и поток длительных слабосвязанных бизнес-процессов внутри приложений и между ними.

MOSS и сервер BizTalk не являются составляющими .NET Framework или Visual Studio — это отдельные продукты, но все они входят в состав платформы Microsoft.

Веб-приложения

Платформа .NET включает технологию ASP.NET для создания Веб-приложений и простых Веб-сервисов. Приложения ASP.NET должны размещаться в рамках Веб-сервера, например, IIS. Для создания Веб-приложений с использованием ASP.NET предлагаются следующие технологии:

- **ASP.NET Web Forms.** Это стандартная технология проектирования и реализации UI для Веб-приложений .NET. Приложение ASP.NET Web Forms достаточно установить только на Веб-сервере, никакие компоненты на компьютер клиента устанавливать не требуется.
- **ASP.NET Web Forms с AJAX.** Используйте AJAX с ASP.NET Web Forms для асинхронной обработки запросов между клиентом и сервером. Это позволит уменьшить время отклика, обеспечить насыщенный клиентский интерфейс и уменьшить число обращений к серверу. AJAX входит в состав ASP.NET в .NET Framework 3.5 и более поздних версиях.
- **ASP.NET Web Forms с элементами управления Silverlight.** Элементы управления Silverlight позволяют расширять готовое приложение ASP.NET, улучшая его пользовательский интерфейс, без необходимости создания абсолютно нового Silverlight-приложения. Таким образом, появляется возможность добавлять «островки» содержимого Silverlight в существующее приложение.
- **ASP.NET MVC.** Эта технология позволяет использовать ASP.NET для создания приложений на базе шаблона Model-View-Controller (MVC). ASP.NET MVC поддерживает разработку через тестирование и четкое разделение функций обработки UI и формирования визуального представления UI. Данный подход помогает избежать смешивания данных представления с кодом логики.
- **ASP.NET Dynamic Data.** Эта технология позволяет создавать управляемые данными приложения ASP.NET, которые используют функциональность LINQ to Entities. Технология предусматривает модель быстрой разработки управляемых данными LOB-приложений с поддержкой возможностей формирования шаблонов и полнофункциональной настройки.

Веб-сервер – Internet Information Services

В состав платформы Microsoft входят сервисы IIS, обеспечивающие всестороннюю поддержку публикации в Интернете, включая транспортные сервисы, клиентские приложения,

инструменты администрирования, возможность подключения к базе данных и приложению, связь с шифрованием. IIS поддерживает следующие сервисы:

- **Сервис World Wide Web.** Этот сервис обеспечивает все необходимые функции для публикации гипертекстовых документов, а также для доставки других типов содержимого, использующего HTTP. Он обеспечивает высокую производительность, хорошее сжатие, широкий выбор вариантов реализации безопасности и аутентификации.
- **Сервис File Transfer Protocol (FTP).** Этот сервис позволяет получать и отправлять файлы с помощью FTP, хотя аутентификация ограничена обычным методом.
- **Сервис Gopher.** Этот сервис поддерживает сетевой протокол распределенного поиска и извлечения документов. Сегодня он применяется крайне редко.
- **Internet Database Connector.** Это интегрированный шлюз и шаблонный механизм, использующий сценарии, для обеспечения доступа сервиса World Wide Web к базам данных через открытый интерфейс взаимодействия с базами данных (Open Database Connectivity, ODBC). Как правило, заменяется более новыми технологиями доступа к данным и написания сценариев, такими как ASP.NET и ASP.NET Data Services.
- **Протокол Secure Sockets Layer (SSL) для взаимодействия клиент/сервер.** Данный механизм предлагает поддержку связи с шифрованием по протоколу HTTP. Таким образом, обеспечивается более безопасный обмен данными между клиентами и серверами, чем отправка содержимого в виде открытого текста.
- **Internet Service Manager Server.** Эта консоль администрирования и связанные с ней инструменты предлагают локальные и удаленные функции администрирования для IIS.
- **Интеграция с ASP.NET.** IIS 7.0 и более поздние версии были разработаны специально для тесной интеграции с ASP.NET с целью улучшения производительности и сокращения нагрузки на сервер при использовании ASP.NET для создания и доставки содержимого.

Сервер базы данных – SQL Server

Реляционная база данных является наиболее общим подходом для хранения и доступа к данным в корпоративном приложении. Платформа приложений Microsoft предлагает SQL Server в качестве системы управления базами данных для ваших приложений. На рынке программного обеспечения доступно несколько версий SQL Server: начиная с локальной базы данных (SQL Server Express), выполняющейся в единственном экземпляре, и заканчивая такими приложениями уровня предприятия, как SQL Server Enterprise Edition.

Технологии доступа к данным, входящие в состав .NET Framework, позволяют получить доступ к данным в любой версии SQL Server, поэтому вам не придется изменять свое приложение, если требуется перейти на более масштабную версию.

Среда разработки Visual Studio

Платформа .NET предлагает комплексную среду разработки Visual Studio Team System. Microsoft Visual Studio является основной средой разработки приложений .NET и доступна в нескольких версиях. Каждая из версий предназначена для определенных групп разработчиков, принимающих участие в цикле разработки приложения.

Visual Studio Team System позволяет использовать любой язык программирования для написания приложений для .NET Framework. Являясь интегрированной средой разработки (Integrated Development Environment, IDE), Visual Studio обеспечивает все необходимые инструменты для проектирования, разработки, отладки и развертывания насыщенных клиентских приложений, RIA, Веб- и мобильных приложений, сервисов и офисных решений. Для получения необходимого сочетания функций можно установить несколько версий одновременно.

Другие инструменты и библиотеки

Кроме Visual Studio, существуют и другие инструменты и инфраструктуры, которые ускоряют разработку или улучшают операционное управление:

- System Center предлагает набор инструментов и сред для мониторинга приложений на уровне предприятия, их развертывания, конфигурирования и управления. Более подробную информацию по «Microsoft System Center» можно найти по адресу <http://www.microsoft.com/systemcenter/en/us/default.aspx>.
- Expression Studio предлагает инструменты графического дизайна для создания насыщенных интерфейсов и анимации. Более подробную информацию по «Microsoft Expression» можно найти по адресу <http://www.microsoft.com/expression/products/Overview.aspx?key=studio>.

Решения patterns & practices

Более подробную информацию об основных решениях, предлагаемых группой Microsoft patterns & practices, можно найти в следующих источниках:

- **Composite Client Application Guidance for WPF** (**Руководство по проектированию составных клиентских приложений для WPF**) упрощает создание модульных приложений, как для настольных сценариев, так и для Silverlight. Больше информации можно найти в разделе «Composite Client Application Guidance» по адресу <http://msdn.microsoft.com/en-us/library/cc707819.aspx>.
- **Enterprise Library** содержит набор блоков приложений, реализующих сквозную функциональность. Больше информации можно найти в разделе «Enterprise Library» по адресу <http://msdn.microsoft.com/en-us/library/cc467894.aspx>.
- **Software Factories** (**Фабрики ПО**) позволяют ускорить разработку определенных типов приложений, таких как Smart Client, WPF-приложения и Веб-сервисы. Больше информации по этому вопросу можно найти в материале «patterns & practices»:

Catalog by Application Type» (patterns & practices: каталог по типам приложений) по адресу <http://msdn.microsoft.com/en-gb/practices/bb969054.aspx>.

- **Unity Application Block (Блок Unity)** обеспечивает функции для реализации внедрения зависимостей, обнаружения сервисов и инверсии управления, как для корпоративных сценариев, так и для Silverlight. Больше информации можно найти в разделе «*Unity Application Block*» по адресу <http://msdn.microsoft.com/en-us/library/dd203101.aspx>.
- **Подробное руководство** по корпоративной архитектуре, проектированию, разработке и сценариями развертывания. Сюда входят такие сценарии, как основы разработки решений, разработка клиентских и серверных решений, создание сервисов. Более подробная информация представлена на главной странице сайта patterns & practices по адресу <http://msdn.microsoft.com/en-us/library/ms998572.aspx>.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу <http://www.microsoft.com/architectureguide>.

Более подробная информация о .NET Framework представлена на следующих ресурсах:

- «*.NET Framework 3.5 Overview*» (Обзор .NET Framework 3.5) по адресу <http://msdn.microsoft.com/en-us/library/a4t23ktk.aspx>.
- «*Overview of the .NET Framework*» (Обзор .NET Framework) по адресу [http://msdn.microsoft.com/en-us/library/zw4w595w\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/zw4w595w(VS.71).aspx).
- «*Overview of the .NET Compact Framework*» (Обзор .NET Compact Framework) по адресу [http://msdn.microsoft.com/en-us/library/w6ah6cw1\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/w6ah6cw1(VS.80).aspx).

Более подробная информация о Веб-сервисах представлена на следующих ресурсах:

- «*Windows Communication Foundation*» по адресу <http://msdn.microsoft.com/en-us/library/ms735119.aspx>.
- «*XML Web Services Using ASP.NET*» (Веб-сервисы XML с использованием ASP.NET) по адресу <http://msdn.microsoft.com/en-us/library/ba0z6a33.aspx>.

Более подробная информация о сервисах рабочих процессов представлена на следующих ресурсах:

- «*Microsoft BizTalk ESB Toolkit*» по адресу <http://msdn.microsoft.com/en-us/library/dd897973.aspx>.
- «*Workflows in Office SharePoint Server 2007*» (Рабочие процессы в Office SharePoint Server 2007) по адресу <http://msdn.microsoft.com/en-us/library/ms549489.aspx>.
- «*Windows Workflow Foundation (WF)*» по адресу <http://msdn.microsoft.com/en-us/netframework/aa663328.aspx>.

Более подробная информация о других возможностях платформы Microsoft представлена на следующих ресурсах:

- Возможности доступа к данным более подробно рассматриваются в материале «*Data Platform Development*» (Разработка платформы данных) <http://msdn.microsoft.com/en-gb/data/default.aspx>.
 - Веб-сервер IIS обсуждается в статье «*A High-Level Look at Microsoft Internet Information Server*» (Общее представление о Microsoft Internet Information Server) <http://msdn.microsoft.com/en-us/library/ms993571.aspx>.
 - СУБД SQL Server более подробно рассматривается в разделе «*SQL Server*» по адресу <http://msdn.microsoft.com/en-gb/sqlserver/default.aspx>.
 - Visual Studio Team System посвящен материал «*Visual Studio 2008 Overview*» (Обзор Visual Studio 2008) по адресу <http://msdn.microsoft.com/en-us/vstudio/products/bb931331.aspx>.
-

Приложение

Б

Матрица технологий слоя представления

Обзор

Данное приложение поможет понять, на какие компромиссы придется пойти при выборе технологии представления и как выбор конкретной технологии повлияет на дизайн.

Приведенные здесь рекомендации будут полезны при выборе технологии представления для конкретного сценария и типа приложения.

Выбор технологии представления зависит как от типа разрабатываемого приложения, так и от типа пользовательского интерфейса, который планируется реализовать для этого приложения. Раздел Обзор технологий представления рассматривает доступные варианты технологий для каждого типа приложения. Раздел Преимущества и недостатки поможет сделать осознанный выбор технологии на основании преимуществ и недостатков каждой из них. Раздел Общие сценарии и решения позволит сопоставить сценарий создаваемого приложения с общепринятыми решениями технологий представления.

Обзор технологий представления

В следующих разделах описаны технологии Microsoft, доступные для каждого из четырех базовых архетипов приложений: мобильные приложения, насыщенные клиентские приложения, насыщенные Интернет-приложения (Rich Internet Application, RIA) и Веб-приложения.

Мобильные приложения

Для мобильных приложений используются следующие технологии представления:

- **Microsoft .NET Compact Framework.** Эта версия Microsoft .NET Framework создана специально для мобильных устройств. Используйте данную технологию для

мобильных приложений, которые предполагается выполнять на устройствах как автономные приложения или приложения без постоянного подключения.

- **ASP.NET for Mobile.** Данная версия ASP.NET ориентирована исключительно для мобильных устройств. Приложения ASP.NET for Mobile можно размещать на обычном Веб-сервере IIS. Используйте эту технологию для мобильных Веб-приложений, когда необходимо обеспечить поддержку большой номенклатуры мобильных устройств и браузеров, которые могут рассчитывать на гарантированное сетевое соединение.
- **Microsoft Silverlight for Mobile.** Данная версия Silverlight-клиента требует установки подключаемого модуля Silverlight на мобильное устройство. Используйте эту технологию для переноса существующих приложений Silverlight на мобильные устройства. Для создания более насыщенного UI можно воспользоваться другими технологиями.

Насыщенные клиентские приложения

Для насыщенных клиентских приложений используются следующие технологии представления:

- **Windows Forms.** Это стандартная технология создания UI для .NET Framework. Даже при наличии возможностей WPF, Windows Forms по-прежнему остается отличным решением для создания UI, тем более, если группа разработчиков уже имеет опыт работы с Windows Forms, а UI приложения не предполагает интенсивного использования графики или воспроизведения потокового мультимедиа.
- **Приложение Windows Presentation Foundation (WPF).** WPF-приложения обеспечивают более широкие графические возможности: двухмерную и трехмерную графику, независимость от разрешения, расширенную поддержку документов и полиграфического оформления, анимацию с временной шкалой, потоковое аудио и видео, векторную графику. Для реализации UI, привязки данных и определения событий WPF использует Extensible Application Markup Language (XAML). WPF также включает расширенные возможности привязки данных и шаблонов. В результате развертывания с использованием приложения браузера XAML (XAML browser application, XBAP) могут быть получены как настольные WPF-приложения, так и WPF-приложения браузера. WPF-приложения обеспечивают совместную работу разработчиков и дизайнеров, когда разработчики занимаются реализацией бизнес-логики, а дизайнеры — оформлением и поведением.
- **Windows Forms с пользовательскими элементами управления WPF.** Данный подход позволяет использовать преимущества более мощных возможностей UI, обеспечиваемыми элементами управления WPF. WPF можно добавлять в существующее приложение Windows Forms. Следует иметь в виду, что элементы управления WPF лучше работают на мощных клиентских компьютерах.
- **WPF с пользовательскими элементами управления Windows Forms.** Эта технология позволяет дополнить WPF элементами управления, которых нет в WPF. С помощью

элемента управления **WindowsFormsHost** из сборки **WindowsFormsIntegration** можно добавить элементы управления Windows Forms. Однако существуют некоторые ограничения и несогласованности, связанные с перекрытием элементов управления, фокусом интерфейса и методами формирования визуального представления, которые отличаются для разных технологий.

- **XAML Browser Application (XBAP) с использованием WPF.** Эта технология выполняет WPF-приложение в защищенной программной среде внутри Microsoft Internet Explorer или Mozilla Firefox в Windows. В отличие от Silverlight, может использоваться большая часть инфраструктуры WPF, но существуют некоторые ограничения, связанные с доступом к системным ресурсам из частично доверяемой изолированной программной среды. XBAP требует установки на клиентском компьютере Windows Vista или обоих подключаемых модулей: .NET Framework 3.5 и браузера XBAP. XBAP — отличный выбор, если требуемые функции недоступны в Silverlight и имеется возможность определять клиентскую платформу и требования к доверию.

Насыщенные Интернет-приложения

Для RIA-приложений используются следующие технологии представления:

- **Silverlight.** Это оптимизированная для браузера версия WPF с поддержкой разных платформ и браузеров. По сравнению с XBAP, Silverlight меньше и быстрее устанавливается, но не поддерживает трехмерную графику и документы с внутренним форматированием. Благодаря небольшому размеру и поддержке разных платформ, Silverlight является отличным решением для WPF-приложений, в которых не требуется обеспечивать поддержку расширенных графических возможностей, предоставляемых WPF.
- **Silverlight с AJAX.** Silverlight поддерживает Asynchronous JavaScript and XML (AJAX) и предоставляет объектную модель AJAX для сценариев JavaScript, размещаемых в Веб-странице. Эту функциональность можно использовать для обеспечения фонового взаимодействия между компонентами страницы и сервером, что позволит уменьшить время отклика пользовательского интерфейса.

Веб-приложения

Для Веб-приложений используются следующие технологии представления:

- **ASP.NET Web Forms.** Это стандартная технология проектирования и реализации UI для Веб-приложений .NET. Приложение ASP.NET Web Forms достаточно установить только на Веб-сервере, никакие компоненты на компьютер клиента устанавливать не требуется.
- **ASP.NET Web Forms с AJAX.** Используйте AJAX с ASP.NET Web Forms для асинхронной обработки запросов между клиентом и сервером. Это позволит уменьшить время отклика, обеспечить насыщенный клиентский интерфейс и

уменьшить число обращений к серверу. AJAX входит в состав ASP.NET в .NET Framework 3.5 и более поздних версиях.

- **ASP.NET Web Forms с элементами управления Silverlight.** Элементы управления Silverlight позволяют расширять готовое приложение ASP.NET, улучшая его пользовательский интерфейс, без необходимости создания абсолютно нового Silverlight-приложения. Таким образом, появляется возможность добавлять «островки» содержимого Silverlight в существующее приложение.
- **ASP.NET MVC.** Эта технология позволяет использовать ASP.NET для создания приложений на базе шаблона Model-View-Controller (MVC). ASP.NET MVC поддерживает разработку через тестирование и четкое разделение функций обработки UI и формирования визуального представления UI. Данный подход помогает избежать смешивания данных представления с кодом логики.
- **ASP.NET Dynamic Data.** Эта технология позволяет создавать управляемые данными приложения ASP.NET, которые используют функциональность LINQ to Entities. Технология предусматривает модель быстрой разработки управляемых данными LOB-приложений с поддержкой возможностей формирования шаблонов и полнофункциональной настройки.

Преимущества и недостатки

В следующих таблицах приведен список преимуществ и недостатков каждой из технологий представления, описанных в предыдущих разделах.

Мобильные приложения

Технология	Преимущества	Недостатки
<i>.NET Compact Framework</i>	<p>Выполняется на клиентском компьютере для обеспечения лучшей производительности и уменьшения времени отклика.</p> <p>Не требует постоянного сетевого подключения.</p> <p>Имеет привычную для Windows Forms модель программирования.</p> <p>Visual Studio предоставляет поддержку дизайнера.</p> <p>Обычно устанавливается в постоянную память (ROM) устройства.</p>	<p>Имеет ограниченный набор API по сравнению с настольным приложением Windows Forms.</p> <p>Требует больше ресурсов на стороне клиента, чем приложение ASP.NET for Mobile.</p> <p>Предполагает более сложное развертывание, чем приложение ASP.NET for Mobile.</p>
<i>ASP.NET for Mobile</i>	<p>Поддерживает большую номенклатуру устройств, включая все устройства с установленным Веб-браузером.</p> <p>Приложение не занимает памяти</p>	<p>В Visual Studio 2008 удалена поддержка функциональности для создания дизайна, но визуальное представление элементов управления по-</p>

	<p>устройства, поскольку не требует установки.</p> <p>Имеет привычную для ASP.NET Web Forms модель программирования.</p> <p>Шаблоны для поддержки дизайнера в Visual Studio можно загрузить из Веб.</p>	<p>прежнему будет формироваться на устройствах.</p> <p>Для выполнения требуется постоянное сетевое подключение.</p> <p>Производительность и время отклика зависят от пропускной способности и задержки.</p> <p>Теперь многие устройства имеют полную поддержку HTML, поэтому могут выполнять стандартные приложения ASP.NET.</p>
<i>Silverlight for Mobile</i>	<p>Предлагает насыщенный UI и визуализацию, включая двухмерную и векторную графику, анимацию.</p> <p>Silverlight-код настольных приложений может успешно выполняться в Silverlight for Mobile.</p> <p>Имеется изолированное хранилище для хранения объектов вне кэша браузера.</p>	<p>Использует больше ресурсов устройств, чем Веб-приложение.</p> <p>Настольные приложения Silverlight, выполняемые на мобильных устройствах, могут потребовать оптимизации, чтобы учесть ограничения памяти и менее производительное оборудование.</p> <p>Требует установки подключаемого модуля Silverlight.</p> <p>Из-за необходимости установки подключаемого модуля номенклатура устройств, на которых может выполняться данное приложение, может быть ограничена по сравнению с Веб-приложениями.</p>

Насыщенные клиентские приложения

Технология	Преимущества	Недостатки
<i>Windows Forms</i>	<p>Имеет привычную модель программирования.</p> <p>Microsoft Visual Studio обеспечивает поддержку дизайнера.</p> <p>Обеспечивает хорошую производительность на широкой номенклатуре клиентских устройств.</p>	<p>Не поддерживает трехмерной графики, потокового мультимедиа, документов с внутренним форматированием или иных расширенных функций UI, которые доступны в WPF, например, стили и шаблоны UI.</p> <p>Требует установки на клиентский компьютер.</p>
<i>Windows Forms с пользовательскими элементами управления WPF</i>	<p>Позволяет добавлять насыщенный UI в существующие приложения Windows Forms.</p> <p>Обеспечивает стратегию</p>	<p>В зависимости от сложности UI может понадобиться более мощное аппаратное обеспечение для поддержки графики.</p>

	перехода к полнофункциональным WPF-приложениям.	Нельзя перекрывать элементы управления Windows Forms и WPF.
<i>Приложение WPF</i>	<p>Обеспечивает насыщенный UI и визуализацию. В том числе поддерживает двухмерную, трехмерную и векторную графику, независимость от разрешения, документы с внутренним форматированием и анимацию.</p> <p>Поддерживает потоковое мультимедиа с переменной полосой пропускания (Адаптивная потоковая передача мультимедиа).</p> <p>XAML упрощает описание UI, привязки данных и событий.</p> <p>Поддерживает интеграцию кода и дизайна, создаваемых раздельно разработчиком и дизайнером.</p>	<p>В зависимости от сложности UI может понадобиться более мощное аппаратное обеспечение для поддержки графики.</p> <p>Expression Blend может быть менее привычной средой разработки, чем Visual Studio.</p> <p>В WPF меньше встроенных элементов управления, чем в Windows Forms.</p>
<i>WPF с элементами управления Windows Forms</i>	Позволяет дополнить WPF отсутствующими элементами управления. Так, например, добавить элемент управления таблица данных, который изначально не предусмотрен в WPF ⁷² .	<p>Требует WindowsFormsHost.</p> <p>Могут возникнуть сложности с получением фокуса и входных данных для перехода через границы.</p> <p>Нельзя перекрывать элементы управления Windows Forms и WPF.</p> <p>Элементы управления WPF и Windows Forms используют разные техники формирования визуального представления, что может привести к несоответствиям при их отображении на разных платформах.</p>
<i>XBAP с использованием WPF</i>	<p>Позволяет выполнять развертывание WPF-приложения в Веб.</p> <p>Обеспечивает все преимущества WPF касательно насыщенной визуализации и UI.</p> <p>Проще в развертывании и обновлении, чем приложение WPF или Windows Forms.</p>	<p>Может выполняться только на клиентском компьютере с установленной .NET Framework 3.5 и более поздними версиями и подключаемым модулем браузера XBAP.</p> <p>Может выполняться только в браузерах Internet Explorer и Mozilla Firefox. Кроме того, могут присутствовать некоторые</p>

⁷² Доступно в WPF Toolkit (прим. научного редактора).

		ограничения по доступу к ресурсам на стороне клиента.
--	--	---

Насыщенные Интернет-приложения

Технология	Преимущества	Недостатки
<i>Silverlight</i>	<p>Облегченная установка для клиентских компьютеров.</p> <p>Обеспечивает большинство преимуществ WPF по UI и визуализации, таких как потоковое мультимедиа, двухмерная и векторная графика, анимация, независимость от разрешения.</p> <p>Изолированное хранилище обеспечивает отделение кэша приложения от кэша браузера.</p> <p>Поддерживает видео высокой четкости.</p> <p>Обработка на стороне клиента обеспечивает улучшенное взаимодействие с пользователем и меньшее время отклика по сравнению с Веб-приложениями.</p> <p>Поддерживает широкий спектр языков программирования, таких как C#, Visual Basic .NET, Ruby и Python.</p> <p>Поддерживает фоновую обработку в безоконном режиме в качестве замены JavaScript.</p> <p>Обеспечивает поддержку разных платформ, включая Mac и Linux.</p> <p>Обеспечивает поддержку разных браузеров, включая Firefox и Safari.</p>	<p>Требует установки подключаемого модуля Silverlight на стороне клиента.</p> <p>Expression Blend может быть менее привычной средой разработки, чем Visual Studio.</p> <p>Отсутствует присущая WPF расширенная поддержка трехмерной графики и документов с внутренним форматированием.</p> <p>Из-за различий между XAML и элементами управления возникают сложности с переходом от WPF или XBAP.</p>
<i>Silverlight c AJAX</i>	<p>Позволяет использовать существующие библиотеки AJAX и процедуры приложения Silverlight.</p> <p>Посредством взаимодействия с сервером обеспечивает возможность динамического создания и удаления объектов Silverlight при работе пользователя с приложением. Это открывает дополнительные возможности для интерактивных интерфейсов.</p>	<p>Модель программирования будет непривычной, если вы до сих пор работали только с обычным ASP.NET или Silverlight.</p>

Веб-приложения

Технология	Преимущества	Недостатки
ASP.NET Web Forms	<p>Интерфейс разработки напоминает Windows Forms для Веб.</p> <p>Не зависит от клиента.</p> <p>Не требует установки на клиентский компьютер.</p> <p>Обеспечивает поддержку разных платформ и браузеров.</p> <p>Обеспечивает поддержку создания дизайна в Visual Studio.</p> <p>Доступен широкий спектр элементов управления.</p>	<p>UI поддерживает только HTML и Dynamic HTML (DHTML).</p> <p>На стороне клиента хранятся только cookies и состояние представления.</p> <p>Для обновления содержимого страницы требуется возвращение всех данных на сервер и обновление страницы.</p> <p>Время отклика UI довольно велико, поскольку вся обработка выполняется на сервере.</p>
ASP.NET Web Forms с AJAX	<p>Меньшее время отклика UI и более насыщенное взаимодействие с пользователем.</p> <p>Поддерживает отложенную загрузку.</p> <p>Обеспечивает возможность частичного обновления страниц.</p> <p>Входит в состав ASP.NET 3.5.</p>	<p>Модель программирования будет непривычной, если вы до сих пор работали только с обычным ASP.NET.</p> <p>Не работает, если на клиентском компьютере отключен JavaScript.</p>
ASP.NET Web Forms с элементами управления Silverlight	<p>Позволяет вводить насыщенные возможности визуализации Silverlight и UI в существующие приложения ASP.NET.</p> <p>Обеспечивает стратегию перехода на приложения, полностью созданные на Silverlight.</p>	<p>Требует установки подключаемого модуля Silverlight на клиентском компьютере.</p> <p>Expression Blend может быть менее привычной средой разработки, чем Visual Studio.</p>
ASP.NET MVC	<p>Поддерживает разработку через тестирование.</p> <p>Обеспечивает разделение обработки UI и формирования визуального представления UI.</p> <p>Позволяет создавать удобные для пользователей и средств поиска URL.</p> <p>Обеспечивает полный контроль над разметкой.</p> <p>Обеспечивает полный контроль над формированием визуального представления содержимого.</p> <p>Управление навигацией осуществляется путем настройки,</p>	<p>Не поддерживает состояние представления.</p> <p>Не поддерживает события элемента управления.</p>

	что значительно сокращает требуемый объем кода.	
ASP.NET Dynamic Data	<p>Позволяет создавать сайты, полностью управляемые данными, с автоматическим формированием визуального представления.</p> <p>Имеется встроенная поддержка LINQ-запросов.</p> <p>Имеется встроенная поддержка ADO.NET Entity Framework.</p> <p>LINQ позволяет моделировать базы данных путем сопоставления объектов и данных.</p>	На данный момент данную технологию поддерживают всего несколько элементов управления.

Общие сценарии и решения

В следующих разделах предлагается руководство по выбору подходящего типа технологии представления для четырех базовых архетипов приложений: мобильных приложений, насыщенных клиентских приложений, RIA и Веб-приложений.

Мобильные приложения

При выборе технологии представления для мобильных приложений руководствуйтесь следующими рекомендациями:

Используйте **.NET Compact Framework**, если:

- Создаете мобильное приложение, которое должно поддерживать сценарии без постоянного подключения или сценарии работы в автономном режиме.
- Создаете мобильное приложение, которое будет выполняться на клиентском компьютере для улучшения производительности и уменьшения времени отклика.

Используйте **ASP.NET for Mobile**, если:

- Группа разработчиков имеет опыт работы с ASP.NET, и нужно обеспечить поддержку максимально широкой номенклатуры устройств.
- Создаете приложение, которое не должно требовать установки на стороне клиента и зависеть от подключаемых модулей.
- Создаете приложение с постоянным сетевым подключением.
- Необходимо обеспечить, чтобы приложение занимало минимальный объем памяти в устройстве, или минимальное использование ресурсов устройства.

Используйте **Silverlight for Mobile**, если:

- Создаете мобильное Веб-приложение с использованием возможностей Silverlight для реализации насыщенной визуализации и UI.

- На целевых устройствах легко получить доступ к Silverlight или этот подключаемый модуль уже установлен.
-

Насыщенные клиентские приложения

При выборе технологии представления для насыщенных клиентских приложений руководствуйтесь следующими рекомендациями:

Используйте **Windows Forms**, если:

- Ваша группа имеет опыт разработки приложений Windows Forms, и вы по экономическим соображениям не можете позволить себе переход на другую технологию.
- Выполняется расширение или изменение существующего приложения Windows Forms.
- Не требуется поддержка насыщенного мультимедиа и анимации.

Используйте **WPF**, если:

- Создаете насыщенное клиентское приложение с применением возможностей WPF по реализации насыщенной визуализации и UI.
- Создаете насыщенное клиентское приложение для развертывания в Веб с помощью XBAP.

Используйте **Windows Forms с пользовательскими элементами управления WPF**, если:

- Уже имеете готовое приложение Windows Forms и хотите воспользоваться такими преимуществами WPF, как расширенные графические возможности, документы с внутренним форматированием, потоковое мультимедиа и анимация.

Используйте **WPF с элементами управления Windows Forms**, если:

- Создаете насыщенное клиентское приложение с применением WPF и элементов управления, не предоставляемыми в WPF.

Используйте **XBAP**, если:

- Необходимо выполнить развертывание уже существующего приложения WPF в Веб.
 - Необходимо воспользоваться возможностями WPF по реализации насыщенной визуализации и UI, которые недоступны в Silverlight.
-

Насыщенные Интернет-приложения

При выборе технологии представления для RIA руководствуйтесь следующими рекомендациями:

Используйте **Silverlight**, если:

- Необходимо использовать возможности Silverlight по реализации насыщенной визуализации, потокового мультимедиа и UI.

- Создаете приложение, для которого необходимо обеспечить простое развертывание и возможность отложенной загрузки отдельных модулей.
- Создаете приложение, которое планируется использовать в разных браузерах и на разных платформах.

Используйте **Silverlight с AJAX**, если:

- Необходимо обеспечить возможность динамического управления экземплярами объектов в рамках объектной модели Silverlight с вашей Веб-страницы.
- Необходимо манипулировать элементами управления Silverlight на основании взаимодействия пользователя с Веб-страницей.

Веб-приложения

При выборе технологии представления для Веб-приложений руководствуйтесь следующими рекомендациями:

Используйте **ASP.NET Web Forms**, если:

- Группа разработчиков уже имеет опыт создания приложений ASP.NET Web Forms.
- Необходимо расширить или изменить существующее приложение ASP.NET Web Forms.
- Приложение планируется выполнять на совершенно разных клиентских компьютерах.
- Создаете приложение, для выполнения которого на клиентских компьютерах ничего не требуется устанавливать.
- Необходимо реализовать простые функции, такие как Create, Read, Update и Delete (CRUD), без насыщенного UI или анимации.

Используйте **ASP. NET Web Forms с AJAX**, если:

- Требуется создать приложение ASP.NET Web Forms с более насыщенным пользовательским интерфейсом и меньшим временем отклика.
- Требуется обеспечить поддержку отложенной загрузки и частичного обновления страницы.

Используйте **ASP.NET Web Forms с элементами управления Silverlight**, если:

- Необходимо расширить существующее приложение ASP.NET Web Forms возможностями Silverlight по реализации насыщенной визуализации и UI.
- Планируется перенести существующее Веб-приложение на Silverlight.

Используйте **ASP.NET MVC**, если:

- Необходимо реализовать шаблон Model-View-Controller (MVC).
- Необходимо обеспечить полный контроль разметки.

- Необходимо обеспечить четкое разделение обработки UI и формирования визуального представления UI.
- Необходимо реализовать разработку через тестирование.

Используйте **ASP.NET Dynamic Data**, если:

- Необходимо быстро создать управляемое данными приложение.
- Необходимо использовать язык запросов LINQ или модель данных Entity Framework.
- Необходимо использовать встроенные средства моделирования LINQ для упрощения сопоставления объектов с данным.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- Все о Silverlight можно найти на официальном сайте Silverlight по адресу <http://silverlight.net/default.aspx>.
- Больше сведений о «Islands of Richness» (Острова насыщенности) можно найти по адресу <http://blogs.msdn.com/brada/archive/2008/02/18/islands-of-richness-with-silverlight-on-an-asp-net-page.aspx>.

Приложение

В

Матрица технологий слоя доступа к данным

Обзор

Данное приложение поможет понять, на какие компромиссы придется пойти при выборе технологии доступа к данным и как выбор конкретной технологии повлияет на дизайн.

Приведенные здесь рекомендации будут полезны при выборе технологии доступа к данным для конкретного сценария и типа приложения.

Выбор технологии доступа к данным зависит как от типа разрабатываемого приложения, так и от типа используемых бизнес-сущностей. В разделе Обзор технологий доступа к данным рассматриваются существующие технологии доступа к данным. Раздел Преимущества и недостатки расскажет обо всех представленных на рынке технологиях доступа к данным. Раздел Общие сценарии и решения позволит сопоставить сценарий создаваемого приложения с общепринятыми решениями технологий доступа к данным.

Обзор технологий доступа к данным

Платформа Microsoft поддерживает следующие технологии доступа к данным:

- **ADO.NET Core.** ADO.NET Core предоставляет возможности извлечения, обновления и управления данными. Сюда входят поставщики для доступа к SQL Server, OLE DB, Open Database Connectivity (ODBC), SQL Server Compact Edition и баз данных Oracle.
- **ADO.NET Data Services Framework.** Эта инфраструктура предоставляет доступ к данным из любого источника с поддержкой Linq (обычно Entity Data Model) посредством Веб-сервисов REST, доступных по HTTP. Взаимодействие с данными может выполняться непосредственно через универсальные идентификаторы ресурсов (Uniform Resource

Identifier, URI). Веб-сервис можно настроить таким образом, чтобы он возвращал данные в формате Atom или JavaScript Object Notation (JSON).

- **ADO.NET Entity Framework.** Эта инфраструктура обеспечивает строго типизированный доступ к реляционным базам данных. Она переводит модель данных от физической структуры реляционных таблиц к концептуальной модели, которая точно отображает бизнес-объекты. Entity Framework предлагает обычную модель Entity Data Model в среде ADO.NET, что позволяет разработчикам определить гибкое сопоставление с реляционными данными. Такое сопоставление обеспечивает возможность изолировать приложение от изменений в базовой схеме хранения данных. Entity Framework также поддерживает технологию LINQ to Entities, которая поддерживает LINQ для бизнес-объектов, доступных через Entity Framework. Используя Entity Framework в качестве объектно-реляционного сопоставления (Object/Relational Mapping, O/RM), разработчики применяют к бизнес-объектам технологию LINQ to Entities. Эти бизнес-объекты будут преобразованы Entity Framework в Entity SQL, который сопоставлен с моделью Entity Data Model под управлением Entity Framework. Кроме того, разработчики могут работать непосредственно с Entity Data Model и задействовать Entity SQL в своих приложениях.
- **ADO.NET Sync Services.** Механизм ADO.NET Sync Services входит в состав Microsoft Sync Framework и применяется для реализации синхронизации для баз данных с поддержкой ADO.NET. Поддерживается синхронизация данных с приложениями без постоянного подключения. Через определенные промежутки времени осуществляется сбор данных от клиентской базы с последующей синхронизацией с серверной базой данных.
- **Language Integrated Query (LINQ).** LINQ предлагает библиотеки классов, которые расширяют C# и Visual Basic встроенным синтаксисом языка запросов. Прежде всего, это технология запросов, поддерживаемая разными библиотеками .NET Framework. Например, технология LINQ to Entities включена в сборки ADO.NET Entity Framework, LINQ to XML входит в состав **System.Xml**, а LINQ to Objects – в состав системных библиотек .NET Framework. Запросы могут выполняться к разным форматам данных, включая **DataSet** (LINQ to **DataSet**), XML (LINQ to XML), объекты в памяти (LINQ to Objects), ADO.NET Data Services (LINQ to Data Services) и реляционные данные (LINQ to Entities).
- **LINQ to SQL.** Облегченное решение со строгой типизацией данных для построения запросов к SQL Server. Технология LINQ to SQL предназначена для простого и быстрого сохранения объектов, когда классы промежуточного уровня очень близко сопоставляются со структурами таблиц базы данных. Сценарии использования LINQ to SQL будут интегрированы и поддерживаться в ADO.NET Entity Framework, начиная с версии .NET Framework 4.0, но LINQ to SQL все равно будет поддерживаться. Более подробно эти вопросы рассматриваются в блоге группы ADO.NET по адресу <http://blogs.msdn.com/adonet/archive/2008/10/31/clarifying-the-message-on-l2s-futures.aspx>.

Преимущества и недостатки

В следующих таблицах приведен список преимуществ и недостатков каждой из технологий доступа к данным, описанных в предыдущих разделах. В каждой таблице рассматривается ряд сценариев использования: объектно-реляционный доступ к данным, доступ к данным без подключения и в автономном режиме, SOA- и сервисные сценарии, а также n-уровневые и общие сценарии. После таблиц приведены некоторые общие рекомендации по применению технологий доступа к данным.

Объектно-реляционный доступ к данным

Технология	Преимущества	Недостатки
<i>ADO.NET Entity Framework (EF)</i>	<p>Отделяет используемую структуру базы данных от логической модели данных.</p> <p>Entity SQL обеспечивает согласованный язык запросов для всех источников данных и типов баз данных.</p> <p>Четко разделяет метаданные на архитектурные слои.</p> <p>Позволяет разработчикам бизнес-логики реализовывать доступ к данным без знания особенностей базы данных.</p> <p>Обеспечивает широкие возможности создания дизайна в Visual Studio для визуализации структуры сущностей данных.</p> <p>Использование модели поставщика обеспечивает возможность применения к разным базам данных.</p>	<p>Требует изменения дизайна сущностей и запросов, если ранее использовался более традиционный метод доступа к данным.</p> <p>Использует отдельные объектные модели.</p> <p>Имеет больше слоев абстракции, чем LINQ to DataSet.</p> <p>Может использоваться вместе или без LINQ to Entities.</p> <p>При изменении структуры базы данных необходимо еще раз генерировать модель сущность-данные и повторно выполнить развертывание библиотеки EF.</p>
<i>LINQ to Entities</i>	<p>Поддержка LINQ для реляционных данных в ADO.NET Entity Framework.</p> <p>Обеспечивает строго типизированный доступ к реляционным данным посредством LINQ.</p> <p>Поддерживает LINQ-запросы к объектам, созданным по модели сущность-данные EF.</p> <p>Обработка выполняется на стороне сервера.</p>	Требует ADO.NET Entity Framework.
<i>LINQ to SQL</i>	<p>Простой способ чтения/записи объектов, когда объектная модель данных соответствует физической модели базы данных.</p> <p>Обеспечивает строго типизированный доступ к SQL-данным посредством LINQ.</p> <p>Обработка выполняется на стороне сервера.</p>	<p>Функциональность интегрирована в Entity Framework, начиная с версии .NET Framework 4.0.</p> <p>Передает LINQ-запросы напрямую в базу данных, а не через поставщика, и поэтому работает только с Microsoft SQL Server.</p>

Доступ к данным без подключения и в автономном режиме

Технология	Преимущества	Недостатки
<i>LINQ to DataSet</i>	Позволяет выполнять полнофункциональные запросы к DataSet .	Вся обработка выполняется на стороне клиента.
<i>ADO.NET Sync Services</i>	Поддерживает синхронизацию между базами данных, координацию совместной деятельности и сценарии работы в автономном режиме. Синхронизация может выполняться в фоновом режиме. Для синхронизации между базами данных используется веерная (<i>hub-and-spoke</i>) архитектура.	Необходимо реализовать собственный механизм отслеживания изменений ¹ . Обмен большими объемами данных в процессе синхронизации может снизить производительность.

Сценарии SOA/Сервис

Технология	Преимущества	Недостатки
<i>ADO.NET Data Services Framework</i>	Данные могут быть отправлены непосредственно через URI с помощью REST-подобных механизмов. Данные могут быть возвращены в формате Atom или JSON. Для упрощения выпуска новых интерфейсов сервисов поддерживается облегченный механизм управления версиями. Клиентские библиотеки .NET Framework, Silverlight и AJAX позволяют разработчикам взаимодействовать непосредственно с объектами и обеспечивают строго типизированный доступ посредством LINQ к ADO.NET Data Services. Клиентские библиотеки .NET Framework, Silverlight и AJAX предлагают привычный API для	Применима исключительно к сервисно-ориентированных сценариев.

¹ Это не совсем точно, может потребоваться реализация собственного механизма отслеживания изменений и решения конфликтов (прим. научного редактора).

	работы с Windows Azure Tables, SQL Data Services ¹ и другими сервисами Microsoft.	
<i>LINQ to Data Services</i>	<p>Позволяет формировать LINQ-запросы к клиентским данным, возвращаемым ADO.NET Data Services.</p> <p>Поддерживает LINQ-запросы к REST-данным.</p>	Используется только с клиентской инфраструктурой ADO.NET Data Services.

N-уровневые и общие сценарии

Технология	Преимущества	Недостатки
<i>ADO.NET Core</i>	<p>Включает .NET-поставщиков для обеспечения доступа с подключением к широкому набору хранилищ данных.</p> <p>Обеспечивает возможности хранения и работы с данными без подключения.</p>	<p>Написание программного кода осуществляется напрямую для конкретных поставщиков, что снижает возможности повторного использования.</p> <p>Структура реляционной базы данных может не соответствовать объектной модели; в этом случае требуется создать слой сопоставления данных.</p>
<i>ADO.NET Data Services Framework</i>	<p>Данные могут быть отправлены непосредственно через URI с помощью схемы типа REST.</p> <p>Данные могут быть возвращены в формате Atom или JSON.</p> <p>Для упрощения выпуска новых интерфейсов сервисов включена облегченная схема управления версиями.</p> <p>Модель поставщика позволяет использовать любые источники данных IQueryable.</p> <p>Клиентские библиотеки .NET Framework, Silverlight и AJAX предлагают привычный API для работы с Windows Azure Tables, SQL Data Services и другими сервисами Microsoft.</p>	<p>Применяется исключительно для сервисно-ориентированных сценариев.</p> <p>Обеспечивает ресурсно-ориентированный сервис, хорошо сопоставимый с сервисами с интенсивной обработкой данных. Однако если преимущественное большинство сервисов ориентировано на операции, могут потребоваться большие трудозатраты.</p>
<i>ADO.NET Entity Framework</i>	Строго разделение метаданные на архитектурные слои.	Требует изменения дизайна сущностей и запросов, если ранее использовался более

¹ SQL Data Services переименован в SQL Azure Database, и не предоставляет REST доступ к данным (прим. научного редактора).

	<p>Поддерживает LINQ to Entities для формирования запросов к сложным объектным моделям. Использование модели поставщиков данных обеспечивает возможность применения к разным базам данных.</p> <p>Позволяет создавать сервисы с четко определенными границами, а также контракты данных/сервисов для отправки и получения строго определенных сущностей через границы сервиса.</p> <p>Экземпляры сущностей вашей Entity Data Model сериализуются и используются Веб-сервисами напрямую.</p> <p>Полная гибкость структуризации полезной нагрузки — на сервер передаются отдельные сущности, коллекции сущностей или диаграмма сущностей.</p> <p>В будущем будет реализована возможность передачи через границы сервисов абсолютно не зависящих от метода хранения объектов.</p>	<p>традиционный метод доступа к данным.</p> <p>Объекты сущностей могут отправляться по сети или может использоваться шаблон Data Mapper для преобразования сущностей в объекты, которые представляют собой более общие типы DataContract.</p> <p>Планируемая поддержка РОСО устранит необходимость преобразования объектов при их передаче по сети.</p> <p>Подход с созданием конечных точек сервиса для получения обобщенных диаграмм сущностей является в меньшей степени сервисно-ориентированным, чем использование конечных точек, которые обусловливают применение более строгих контрактов для типов полезной нагрузки, которые могут быть приняты.</p>
<i>LINQ to Objects</i>	<p>Позволяет создавать LINQ-запросы к объектам в памяти. Предлагает новый подход к извлечению данных из коллекций.</p> <p>Может использоваться напрямую с коллекциями, поддерживающими IEnumerable или IEnumerable<T>.</p> <p>Может использоваться для формирования запросов к строкам, метаданным на основе отражения и папкам с файлами.</p>	Rаботает исключительно с объектами, которые реализуют интерфейс IEnumerable .
<i>LINQ to XML</i>	<p>Позволяет создавать LINQ-запросы к XML-данным. Сравнима с объектной моделью документа (Document Object Model, DOM), которая помещает XML-документ в память, однако значительно проще в использовании.</p>	<p>Сильно зависит от универсальных классов.</p> <p>Технология не оптимизирована для работы с XML-документами, полученными из не пользующихся доверием источников, которые требуют использования других технологий</p>

	Результаты запросов можно применять в качестве параметров конструкторов объектов XElement и XAttribute .	снижения рисков.
<i>LINQ to SQL</i>	Предлагает простой способ извлечения и обновления данных как объектов, когда объектная модель и модель базы данных одинаковы.	В .NET Framework 4.0 Entity Framework будет рекомендованным решением доступа к данным для сценариев LINQ-запросов к реляционной базе данных. LINQ to SQL будет поддерживаться и развиваться на базе отзывов, полученных от сообщества разработчиков.

Общие рекомендации

При выборе технологии доступа к данным руководствуйтесь следующими рекомендациями:

- **Гибкость и производительность.** ADO.NET Core обеспечит максимальную производительность и гибкость. ADO.NET Core предоставляет самые широкие возможности и является наиболее серверно-ориентированным решением. При использовании ADO.NET Core не забывайте о компромиссе между дополнительной гибкостью и необходимостью написания собственного кода. Имейте в виду, что преобразование к пользовательским объектам уменьшает производительность. Если вам необходима тонкая инфраструктура, использующая поставщики ADO.NET и поддерживающая изменения в базе данных через конфигурацию, воспользуйтесь Enterprise Library Data Access Application Block.
- **Объектно-реляционное сопоставление (O/RM).** Если вам требуется решение на базе O/RM и/или поддержка множества баз данных, воспользуйтесь Entity Framework. Это идеальный вариант реализации сценариев Domain Model.
- **Сценарий работы в автономном режиме.** Если требуется поддерживать сценарий без подключения, воспользуйтесь объектами **DataSet** или Sync Framework.
- **N-уровневый сценарий.** Существуют такие варианты передачи данных через слои или уровни: передача объектов сущностей, объекты передачи данных (Data Transfer Objects, DTO), которые сопоставляются с сущностями, объекты **DataSet** и пользовательские объекты. При создании ресурсно-ориентированных сервисов (REST) используйте ADO.NET Data Services. При создании сервисов, ориентированных на операции (SOAP), используйте сервисы Windows Communication Foundation (WCF) с явным определением контрактов сервисов и данных.

- **SOA и сценарии сервисов.** Если база данных предоставляется как сервис, используйте ADO.NET Data Services. Если требуется сохранять данные в облаке, используйте SQL Data Services¹.
- **Microsoft Windows Mobile.** Многие технологии доступа к данным требуют привлечения слишком большого количества ресурсов, поэтому не могут использоваться на большинстве устройств Windows Mobile, имеющих ограниченные возможности памяти. Воспользуйтесь базой данных SQL Server Compact Edition и ADO.NET Sync Services для сохранения данных на мобильном устройстве и их синхронизации с серверной системой базы данных. В сценариях Windows Mobile могут также использоваться такие функции, как репликация слиянием.

Общие сценарии и решения

В следующих разделах предлагается руководство по выбору подходящего типа технологии доступа к данным.

Используйте **ADO.NET Core**, если:

- Необходимо применить низкоуровневые API для управления доступом к данным в создаваемом приложении.
- Необходимо использовать уже разработанные поставщики ADO.NET.
- Используется традиционная логика доступа к данным базы данных.
- Нет необходимости в дополнительной функциональности, предлагаемой другими технологиями доступа к данным.
- Создаете приложение, которое должно поддерживать доступ к данным без подключения.

Используйте **ADO.NET Data Services Framework**, если:

- Хотите получить доступ к данным, предоставляемым как сервис через REST.

Используйте **ADO.NET Entity Framework**, если:

- Концептуальная модель должна использоваться совместно приложениями и сервисами.
- Необходимо сопоставить один класс множеству таблиц через наследование.
- Необходимо формировать запросы к реляционным хранилищам, отличным от семейства продуктов Microsoft SQL Server.
- Имеется объектная модель, которую необходимо сопоставить с реляционной моделью с помощью гибкой схемы.
- Необходима гибкость отделения схемы сопоставления от объектной модели.

¹ Переименовано в SQL Azure Database (прим. научного редактора).

Используйте **ADO.NET Sync Services**, если:

- Необходимо создать приложение с поддержкой сценариев без постоянного подключения.
- Используете Windows Mobile и хотите выполнять синхронизацию с центральным сервером базы данных.

Используйте **LINQ to Data Services**, если:

- Используются данные, возвращаемые клиентской частью ADO.NET Data Services.
- Необходимо выполнять запросы к клиентским данным, используя синтаксис LINQ.
- Необходимо выполнять запросы к REST-данным, используя синтаксис LINQ.

Используйте **LINQ to DataSets**, если:

- Необходимо выполнять запросы к **Dataset**, в том числе запросы на объединение таблиц.
- Хотите работать с общим языком запросов, а не заниматься написанием итеративного кода.

Используйте **LINQ to Entities**, если:

- Используете ADO.NET Entity Framework.
- Необходимо выполнять запросы к строго типизированным сущностям.
- Необходимо выполнять запросы к реляционным данным, используя синтаксис LINQ.

Используйте **LINQ to Objects**, если:

- Необходимо выполнять запросы к коллекции.
- Необходимо выполнять запросы к папкам с файлами.
- Необходимо выполнять запросы к объектам в памяти, используя синтаксис LINQ.

Используйте **LINQ to XML**, если:

- В приложении используются XML-данные.
- Необходимо выполнять запросы к XML-данным, используя синтаксис LINQ.

Рекомендации по применению LINQ to SQL

LINQ to Entities рекомендуется применять для LINQ-запросов к реляционной базе данных. LINQ to SQL будет поддерживаться, но не будет основным объектом инноваций или доработок. Если вы уже применяете LINQ to SQL, можете продолжать использовать эту технологию. Но вновь создаваемые решений лучше реализовывать с LINQ to Entities. Более подробно эти вопросы рассматриваются в блоге группы разработки ADO.NET по адресу <http://blogs.msdn.com/adonet/>.

Рекомендации по применению мобильных технологий

Целых ряд рассматриваемых выше в данном приложении технологий недоступен в операционной системе Windows Mobile. На момент публикации в Windows Mobile недоступны следующие технологии:

- ADO.NET Entity Framework
 - ADO.NET Data Services Framework
 - LINQ to Entities
 - LINQ to SQL
 - LINQ to Data Services
 - ADO.NET Core; Windows Mobile поддерживает только SQL Server и SQL Server Compact Edition
-

То, какие продукты доступны в более поздних версиях Windows Mobile, можно найти в документации к ней.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- «ADO.NET» по адресу [http://msdn.microsoft.com/en-us/library/e80y5vhx\(vs.80\).aspx](http://msdn.microsoft.com/en-us/library/e80y5vhx(vs.80).aspx).
 - «ADO.NET Data Services» по адресу <http://msdn.microsoft.com/en-us/data/bb931106.aspx>.
 - «ADO.NET Entity Framework» по адресу <http://msdn.microsoft.com/en-us/data/aa937723.aspx>.
 - «Language-Integrated Query (LINQ)» по адресу <http://msdn.microsoft.com/en-us/library/bb397926.aspx>.
 - «SQL Server Data Services (SSDS) Primer» (SSDS для начинающих) по адресу <http://msdn.microsoft.com/en-us/library/cc512417.aspx>.
 - «Introduction to the Microsoft Sync Framework Runtime» (Введение в Microsoft Sync Framework Runtime) по адресу <http://msdn.microsoft.com/en-us/sync/bb821992.aspx>.
-

Приложение

Г

Матрица интеграционных технологий

Обзор

Данное приложение поможет понять, на какие компромиссы придется пойти при выборе интеграционной технологии и как выбор конкретной технологии повлияет на дизайн.

Приведенные здесь рекомендации будут полезны при выборе интеграционной технологии для конкретного сценария и типа приложения.

Выбор интеграционной технологии зависит от типа разрабатываемого приложения. В разделе Обзор интеграционных технологий перечислены и описаны существующие интеграционные технологии. Раздел Преимущества и недостатки расскажет обо всех представленных на рынке интеграционных технологиях. Раздел Общие сценарии и решения позволит сопоставить сценарий создаваемого приложения с общепринятыми решениями интеграционных технологий.

Обзор интеграционных технологий

Далее перечислены технологии, предлагаемые Microsoft для интеграции приложений:

- **Microsoft BizTalk® Server.** BizTalk Server предлагает полный набор адаптеров, оркестровку, обмен сообщениями и протоколы для создания систем с поддержкой интеграции корпоративных приложений (Enterprise Application Integration, EAI).
- **Microsoft Host Integration Server.** Host Integration Server обеспечивает платформу для подключения к приложениям IBM zSeries и iSeries. Кроме того, Host Integration Server поддерживает подключение к данным между Microsoft Message Queuing и IBM WebSphere MQ.

- **Microsoft Message Queuing.** Message Queuing позволяет интегрировать приложения с помощью очереди сообщений. Данная технология гарантирует доставку сообщений и безопасность, а также обеспечивает обмен сообщениями с учетом приоритетов. Она может поддерживать интеграцию с системами без постоянного подключения или системами, временно пребывающими в автономном режиме. Кроме того, Message Queuing поддерживает синхронные и асинхронные сценарии обмена сообщениями.
 - **Microsoft BizTalk ESB Toolkit.** Данный набор инструментов предлагает слабосвязанную архитектуру обмена сообщениями, построенную на базе сервисов BizTalk Server. Использование функций BizTalk Server позволяет обеспечить гибкую и расширяемую архитектуру, которая включает такие возможности, как преобразование, контроль доставки, безопасность сообщений, реестр сервисов, интеллектуальная маршрутизация и унифицированная обработка исключений.
-

Преимущества и недостатки

В следующей таблице представлен список преимуществ и недостатков каждой интеграционной технологии.

Технология	Преимущества	Недостатки
<i>BizTalk Server</i>	<p>Обеспечивает электронный документооборот между компаниями в форматах Electronic Data Interchange (EDI) и/или Extensible Markup Language (XML).</p> <p>Интегрируется с не-Microsoft системами.</p> <p>Легко расширяется для обеспечения возможностей ESB.</p> <p>WCF LOB-адаптеры (Windows Communication Foundation line-of-business adapters) позволяют создавать пользовательские адаптеры для применения в BizTalk или за его пределами.</p> <p>Включает адAPTERЫ для интеграции с такими системами, как SAP, Oracle и SQL.</p> <p>Предоставляет SOAP-адаптер для работы с Веб-сервисами.</p>	<p>Может привести к формированию тесно связанной инфраструктуры.</p> <p>Требует настройки для получения возможностей ESB.</p>
<i>Host Integration Server</i>	<p>Поддерживает сетевую интеграцию между Windows Server и мэйнфреймом IBM или компьютерами AS/400.</p> <p>Обеспечивает безопасный доступ к хосту и управление удостоверениями с поддержкой протокола безопасных соединений (Secure Sockets Layer, SSL)/протокола безопасности транспортного уровня (Transport Layer Security, TLS), технологии единой регистрации (Single Sign On, SSO) и синхронизации паролей.</p>	<p>Должен быть установлен в среде Windows Server.</p> <p>Требует Microsoft Visual Studio 2005 или более поздние версии.</p> <p>Требует Message Queuing с поддержкой маршрутизации.</p>

	<p>Обеспечивает интеграцию данных с поддержкой Message Queuing и Веб-сервисов XML.</p> <p>Включает инструмент доступа к данным для создания и управления подключениями к базам данных IBM DB2.</p> <p>Поддерживает масштабируемость на уровне предприятия и производительности с одновременными сессиями хостов, балансировкой нагрузки и горячей обработкой отказов.</p> <p>Адаптеры BizTalk для Host Systems поддерживают интеграцию с DB2, IBM WebSphere MQ, Host Applications и Host Files.</p>	
<i>Microsoft Message Queuing</i>	<p>С помощью методов на базе сообщений обеспечивает связь между приложениями в гетерогенных сетях.</p> <p>Поддерживает надежный обмен сообщениями между приложениями внутри предприятия и за его пределами.</p> <p>Поддерживает транзакционные возможности, например, гарантирует доставку сообщений только один раз, гарантирует очередность доставки, подтверждает факт извлечения сообщений из очереди получателя.</p> <p>Выполняет маршрутизацию сообщений на основании топологии сети, возможности подключения на транспортном уровне и необходимости концентрации сессий.</p> <p>Позволяет осуществлять доставку сообщений по HTTP с поддержкой SOAP Reliable Messaging Protocol¹ (SRMP).</p> <p>Поддерживает рассылку одного сообщения множеству получателей.</p> <p>Входит в состав Windows Server 2003 и более поздних версий.</p> <p>Поддерживает два режима развертывания: режим домена с доступом к Active Directory и режим рабочей группы.</p> <p>Включает конечные точки WCF для Message Queuing.</p>	<p>Необходимо продумать режим развертывания перед установкой и настройкой Message Queuing.</p> <p>Режим развертывания рабочей группы не позволяет кодировать сообщения, использовать внутренние сертификаты и не поддерживает обмен сообщениями между платформами.</p> <p>Должны использоваться независимые клиенты.</p> <p>Технология Message Queuing оптимизирована для удаленной отправки и локального получения сообщений. Поэтому следует избегать удаленного чтения очереди.</p> <p>Следует избегать функций, которые формируют запросы к Active Directory.</p> <p>Асинхронные уведомления, для которых используются события, могут быть утеряны.</p> <p>Конечные точки WCF требуют установки Microsoft .NET Framework 3.0 или более поздних версий.</p>
<i>Microsoft BizTalk ESB Toolkit</i>	<p>Обеспечивает динамическое разрешение конечных точек сервиса во время выполнения, что сокращает привязку к описанию конечной точки.</p> <p>Отделяет преобразование сообщений от</p>	<p>Требует установки BizTalk Server 2006 R2 или более поздних версий.</p> <p>Может требовать настройки для конкретных бизнес-сценариев.</p> <p>По умолчанию данные ESB Itinerary</p>

¹ Протокол надежного обмена сообщениями (прим. переводчика).

	<p>приложения.</p> <p>Тесная интеграция с WCF обеспечивает безопасность и надежность обмена сообщениями.</p> <p>Выявление ошибок и формирование отчетов выполняется посредством унифицированной обработки исключений, как для системных, так и для бизнес-исключений.</p> <p>Включает системы преобразования адресов для связи с реестрами сервисов, такие как Universal Description, Discovery and Integration¹ (UDDI).</p> <p>Для маршрутизации и преобразования поддерживает подход на основании маршрутов.</p> <p>Поддерживает клиентские и серверные маршруты.</p> <p>Поддерживает возможность расширения системы преобразования адресов для создания пользовательских систем преобразования адресов.</p> <p>Поддерживает дизайнер BizTalk для создания маршрутов.</p> <p>Обеспечивает портал управления исключениями.</p> <p>Предоставляет все основные возможности, такие как обработка исключений, маршрутизация, преобразование адресов и другие, в виде Веб-сервисов.</p> <p>Отслеживание маршрутов выполняется посредством Business Activity Monitoring (BAM).</p>	Tracking не отслеживаются.
--	--	----------------------------

Общие сценарии и решения

В следующих разделах представлено руководство по выбору подходящей интеграционной технологии для приложения:

Используйте **BizTalk Server**, если:

- Необходимо обеспечить интеграцию с множеством Веб-сервисов через оркестровку, как часть сервисно-ориентированной архитектуры (SOA).
- Необходимо обеспечить поддержку процессов типа «бизнес-бизнес», включая такие отраслевые стандарты как EDIFACT, ANSCI X12, HL7, HIPAA или SWIFT.
- Требуется обеспечить параллельное выполнение сервисов.

¹ Универсальное описание, поиск и взаимодействие (прим. переводчика).

- Требуется обеспечить высоконадежное решение с выделенной масштабируемой инфраструктурой сервера без изменения кода.
- Необходима возможность измерения Ключевых показателей производительности (Key Performance Indicator, KPI) путем настройки BAM-решения для отслеживания данных приложения в реальном масштабе времени.
- Необходимо сократить бизнес-логику приложения до политик декларативных правил, которые легко привести в соответствие динамически изменяющимся бизнес-требованиям.

Используйте **Host Integration Server**, если:

- Необходимо обеспечить поддержку взаимодействия с приложениями IBM zSeries или iSeries.
- Необходимо интегрировать BizTalk с DB2, WebSphere MQ, Host-приложениями или Host-файлами.
- Необходимо интегрировать Message Queuing с WebSphere MQ.

Используйте **Microsoft Message Queuing**, если:

- Необходимо обеспечить поддержку взаимодействия между приложениями с использованием сообщений.
- Необходимо обеспечить интеграцию с не-Microsoft платформами.
- Необходимо обеспечить поддержку SRMP.

Используйте **Microsoft BizTalk ESB Toolkit**, если:

- Необходимо обеспечить поддержку маршрутного подхода.
- Необходимо обеспечить поддержку динамического разрешения и маршрутизации.
- Желаете использовать динамические преобразования.
- В вашей EAI-системе необходимо обеспечить поддержку надежного и унифицированного управления исключениями.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу
<http://www.microsoft.com/architectureguide>.

- Больше сведений о BizTalk можно найти в материале «*BizTalk Server*» по адресу <http://msdn.microsoft.com/en-us/biztalk/default.aspx>.
- Host Integration Server посвящена статья «*Host Integration Server*» по адресу <http://www.microsoft.com/hiserver/default.mspx>.
- MSMQ посвящена статья «*Microsoft Message Queuing*» по адресу <http://www.microsoft.com/windowsserver2003/technologies/msmq/default.mspx>.

- Лучшие практики MSMQ представлены в материале «*Programming Best Practices with Microsoft Message Queuing Services (MSMQ)*» (Лучшие практики программирования с MSMQ) по адресу <http://msdn.microsoft.com/en-us/library/ms811053.aspx>.
 - Много полезной информации можно найти в материале «*Microsoft BizTalk ESB Toolkit*» по адресу <http://msdn.microsoft.com/en-us/library/dd897973.aspx>.
-

Приложение

д

Матрица технологий рабочего процесса

Обзор

Данное приложение поможет понять, на какие компромиссы придется пойти при выборе технологии рабочего процесса и как выбор конкретной технологии повлияет на дизайн. Приведенные здесь рекомендации будут полезны при выборе технологии рабочего процесса для конкретного сценария и типа приложения.

Выбор технологии зависит от типа разрабатываемого рабочего процесса. В разделе Обзор технологий рабочего процесса рассматриваются существующие технологии рабочего процесса. Раздел Преимущества и недостатки расскажет обо всех представленных на рынке технологиях рабочего процесса. Раздел Общие сценарии и решения позволит сопоставить сценарий создаваемого приложения с общепринятыми решениями для технологий рабочего процесса.

Обзор технологий рабочего процесса

Платформа Microsoft поддерживает следующие технологии рабочего процесса:

- **Windows Workflow Foundation (WF).** WF является основополагающей технологией реализации рабочего процесса. Являясь инструментарием для профессиональных разработчиков и независимых поставщиков программных продуктов (Independent Software Vendor, ISV), которые создают последовательные рабочие процессы или рабочие процессы на основе конечных автоматов, WF поддерживает следующие типы рабочих процессов: Sequential (Последовательный), State-Machine (На основе конечных автоматов), Data Driven (Управляемый данными) и Custom (Пользовательский). Рабочие процессы могут создаваться с помощью Windows Workflow Designer (Дизайнер рабочих процессов) в Visual Studio.

- **Workflow Services.** Workflow Services интегрирует WCF с WF для обеспечения сервисов на базе WCF для рабочего процесса. Начиная с Microsoft .NET Framework 3.5, WCF расширен поддержкой рабочих процессов, представленных как сервисы, а также возможностью вызывать сервисы из рабочих процессов. Расширение WCF также позволило вызывать сервисы из рабочих процессов. Кроме того, в состав Visual Studio 2008 включены новые шаблоны и инструменты, поддерживающие сервисы рабочих процессов.
- **Microsoft Office SharePoint Services (MOSS).** MOSS — это платформа управления содержимым и координации совместной деятельности с поддержкой рабочих процессов на базе WF. MOSS предлагает решение для реализации бизнес-процессов и координации совместной деятельности в контексте сервера SharePoint. Рабочие процессы для визирования документов можно создавать непосредственно в интерфейсе MOSS. Кроме того, можно создавать рабочие процессы с помощью SharePoint Designer или Windows Workflow Designer в Visual Studio. Для реализации сложных рабочих процессов может использоваться объектная модель WF в Visual Studio.
- **Microsoft BizTalk Server.** BizTalk обладает собственным механизмом реализации рабочих процессов, ориентированным на оркестровку, например, корпоративную интеграцию с рабочими процессами уровня системы. Следующая версия BizTalk, возможно, будет использовать WF, а также XLANG (расширение языка описания Веб-сервисов (Web Service Definition Language, WSDL) для моделирования оркестровки и координации совместной деятельности сервисов), которая применяется в качестве текущей технологии оркестровки в BizTalk. Использование BizTalk Orchestration Services позволяет определять общий дизайн и поток длительных слабосвязанных бизнес-процессов внутри приложений и между ними.

Пользовательский и системный рабочий процесс

Термин *рабочий процесс* применяется для обозначения двух основных типов процессов:

- **Пользовательский рабочий процесс.** Этот тип рабочего процесса предполагает участие пользователей и подразделяется на несколько состояний и событий. Переход от одного состояния к другому осуществляется на основании условной оценки событий, которые возникают. Рабочий процесс преимущественно состоит из действий, выполняемых пользователями.
- **Системный рабочий процесс.** Этот тип рабочего процесса иногда называют *оркестровкой*. Обычно он используется как посредник между бизнес-сервисами и бизнес-процессами. Оркестровка не предполагает участия пользователей.

Преимущества и недостатки

В следующей таблице перечислены основные преимущества и недостатки каждой технологии рабочего процесса.

Технология	Преимущества	Недостатки
<i>Windows Workflow Foundation (WF)</i>	<p>Ориентированное на разработчика решение по созданию рабочих процессов.</p> <p>Поддерживает последовательный рабочий процесс, рабочий процесс на основе конечных автоматов и управляемый данными рабочий процесс.</p> <p>В Visual Studio предоставляется поддержка дизайнера.</p> <p>Включает поддержку безопасного и надежного обмена данными с использованием транзакций.</p> <p>Поддерживает длительные рабочие процессы, которые могут продолжать выполняться после перезагрузки системы.</p>	<p>Для размещения дизайнера в приложении необходим специальный код.</p> <p>Не поддерживает параллельного выполнения.</p>
<i>Workflow Services</i>	<p>Обеспечивает интеграцию WCF и WF.</p> <p>Позволяет предоставлять рабочие процессы клиентским приложениям в виде сервисов.</p> <p>Поддерживает координацию множества сервисов для завершения бизнес-процесса.</p> <p>При вызове Workflow Services среда выполнения WF автоматически подключает новые или существующие экземпляры рабочих процессов.</p> <p>В Visual Studio 2008 обеспечивается поддержка новых шаблонов и инструментов для Workflow Services.</p>	<p>Требует установки .NET Framework 3.5 или более поздней версии.</p> <p>При использовании учетных данных, отличных от применяемых по умолчанию, требуется написание дополнительного кода.</p>
<i>MOSS 2007 Workflow</i>	<p>Подсистема управления рабочими процессами основана на WF.</p> <p>Рабочий процесс для визирования документов может быть определен с помощью Веб-интерфейса.</p>	<p>Рабочие процессы привязаны к одному сайту и не имеют доступа к данным других сайтов.</p> <p>Не очень хорошо подходит для сложных интегрированных в бизнес-</p>

	<p>Условные или управляемые данными рабочие процессы могут определяться в SharePoint Designer.</p> <p>В Visual Studio могут создаваться пользовательские рабочие процессы с использованием компонентов и сервисов WF.</p> <p>Технология интегрирована с приложениями комплекта Microsoft Office.</p>	<p>приложения (Line-Of-Business, LOB) рабочих процессов.</p>
<i>BizTalk</i>	<p>Обеспечивает единое решение по управлению бизнес-процессами.</p> <p>Обеспечивает электронный документооборот между компаниями в форматах Electronic Data Interchange (EDI) и/или Extensible Markup Language (XML).</p> <p>Включает функции оркестровки для проектирования и выполнения длительных слабосвязанных бизнес-транзакций.</p> <p>Интегрируется с не-Microsoft системами.</p> <p>Легко расширяется для обеспечения возможностей сервисной шины предприятия (Enterprise Service Bus, ESB).</p> <p>WCF LOB-адаптеры позволяют создавать пользовательские адаптеры для применения в BizTalk или за его пределами.</p>	<p>Сохраняет состояние оркестровки на SQL Server, что может привести к задержкам при выполнении оркестровки.</p> <p>В текущей версии не используется WF. Возможно, поддержка WF будет реализована в следующей версии.</p>

Общие сценарии и решения

В следующих разделах представлено руководство по выбору подходящей технологии рабочего процесса для приложения.

Используйте **WF**, если:

- Необходимо создать собственное решение для рабочего процесса.
- Требуется поддержка дизайнера рабочих процессов в Visual Studio.
- Необходимо разместить дизайнер WF в своем приложении.

Используйте **Workflow Services**, если:

- Необходимо предоставлять рабочие процессы как сервисы.
- Необходимо вызывать сервисы из рабочего процесса.
- Для завершения бизнес-процесса необходимо координировать вызовы множества сервисов.

Если уже работаете с SharePoint, используйте рабочий процесс **MOSS 2007** в следующих случаях:

- Если необходимо обеспечить рабочий процесс для совместной работы пользователей.
- Если необходимо обеспечить рабочий процесс для списка или библиотеки SharePoint; например, чтобы обеспечить процесс визирования.
- Если необходимо расширить рабочий процесс SharePoint, добавив в него пользовательские задачи.
- Если желаете использовать в Visual Studio дизайнер рабочих процессов.

Используйте **BizTalk Server**, если:

- Необходим рабочий процесс, который будет работать с разными приложениями и системами.
- Необходим размещаемый на сервере системный рабочий процесс, обеспечивающий интеграцию систем предприятия.
- Разрабатывается приложение, которое должно осуществлять сбор данных от множества Веб-сервисов, как часть сервисно-ориентированной архитектуры (Service Oriented Architecture, SOA).
- Разрабатывается приложение с длительными бизнес-процессами, выполнение которых может продолжаться несколько дней.
- Требуется поддерживать процессы типа «бизнес-бизнес», основанные на отраслевых стандартах.
- Необходимо обеспечить параллельное выполнение сервисов.
- Необходимо представить бизнес-логику приложения в виде декларативных правил, которые легко привести в соответствие с изменениями бизнес-требований.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу
<http://www.microsoft.com/architectureguide>.

- Более подробно рабочие процессы MOSS 2007 рассматриваются в статье «*Workflows in Office SharePoint Server 2007*» по адресу <http://msdn.microsoft.com/en-us/library/ms549489.aspx>.
- WF посвящена статья «*Windows Workflow Foundation*» по адресу <http://msdn.microsoft.com/en-us/netframework/aa663328.aspx>.

- Более подробно Workflow Services обсуждаются в материале «*Workflow Services*» (Сервисы рабочего процесса) по адресу <http://msdn.microsoft.com/en-us/library/cc825354.aspx>.
 - BizTalk посвящен ресурс по адресу <http://msdn.microsoft.com/en-us/biztalk/default.aspx>.
 - Более подробно корпоративные рабочие процессы рассматриваются в статье «*Architecting Enterprise Loan Workflows and Orchestration*s» (Проектирование архитектуры корпоративных рабочих процессов и оркестровок обработки займов) по адресу <http://msdn.microsoft.com/en-us/library/bb330937.aspx>.
-

Приложение

E

Enterprise Library от patterns & practices

Обзор

В данном приложении рассматривается Enterprise Library от patterns & practices. Вы узнаете, как с помощью этой библиотеки можно просто и быстро реализовать сквозную функциональность, такую как протоколирование, обработку исключений и доступ к данным, в приложениях.

Назначение Enterprise Library

Библиотека Enterprise Library предназначена для реализации следующих целей:

- **Единообразие.** Все блоки приложений Enterprise Library характеризуются единообразием шаблонов проектирования и подходов реализации.
- **Расширяемость.** Все блоки приложений включают определенные точки расширения, которые позволяют разработчикам настраивать поведение блоков приложений через добавление собственного кода.
- **Простота использования.** Enterprise Library предлагает множество полезных функций, к которым относятся графический инструмент настройки, простая процедура установки, прозрачная и полная документация с примерами.
- **Интеграция.** Блоки приложений Enterprise Library предназначены для совместного использования, а проведенные тесты гарантируют качество их совместного функционирования. Кроме того, можно использовать блоки приложений по отдельности.

Составляющие Enterprise Library

В состав Enterprise Library входят следующие компоненты:

- Блоки приложений, которые включают пригодный для повторного использования код. Этот код может применяться при реализации решений для сквозной функциональности, например, протоколирования, обработки исключений, валидации и доступа к данным.
- Инструменты настройки, которые упрощают введение блоков Enterprise Library в приложение и задание данных конфигурации. Инструменты настройки включают самодостаточный редактор конфигурации и инструмент настройки, интегрированный с Visual Studio.
- Общие служебные функции для таких задач, как сериализация. Эти функции применяются в различных компонентах библиотеки и блоках приложений и могут использоваться разработчиками в их коде.
- Возможности инstrumentирования позволяют разработчикам и администраторам отслеживать поведение и производительность блоков приложений во время выполнения.
- Сценарии формируют исходный код Enterprise Library и копируют сборки в соответствующие каталоги.
- Служебные программы для установки счетчиков событий и производительности, предоставляемых Enterprise Library.
- Служебные программы для создания образцов баз данных, используемых в примерах Enterprise Library.
- Полный набор примеров приложений (по одному для каждого блока приложений), которые демонстрируют возможности использования блоков приложений. Они реализуют общие сценарии для каждого блока приложений и содержат ссылки на соответствующие разделы документации.
- Полный исходный код Enterprise Library, включая проекты Visual Studio и модульные тесты, которые могут использоваться разработчиками для расширения и изменения библиотеки и блоков приложений. Разработчики выполняют модульные тесты и пишут новые тесты. Это позволяет им убедиться в том, что приложения по-прежнему соответствуют проектным требованиям.

Блоки приложений

В следующей таблице перечислены и описаны блоки приложений, которые помогут в решении типовых задач разработки систем уровня предприятия.

Блок приложений	Описание
Caching Application Block (Блок кэширования)	Помогает включать локальный кэш в создаваемые приложения. Поддерживается кэш в памяти и (необязательно) резервное хранилище, которое может быть представлено либо

	базой данных, либо изолированным хранилищем. Данный блок предлагает всю необходимую функциональность по извлечению, добавлению и удалению кэшированных данных, поддерживает настраиваемые политики истечения срока действия и сборки мусора.
Cryptography Application Block (Блок шифрования)	Упрощает задачу по введению функциональности шифрования в приложения. Этот блок может использоваться для реализации целого ряда задач в приложениях, таких как шифрование данных, создание хэша данных, сравнение значений хэша для проверки наличия изменений в данных.
Data Access Application Block (Блок доступа к данным)	Упрощает задачи по реализации общей функциональности доступа к данным, например, чтения данных для отображения, передачи данных между слоями приложения, возвращения измененных данных в базы данных. Этот блок включает поддержку хранимых процедур и языка SQL, обеспечивает доступ к наиболее часто используемым функциям ADO.NET в виде простых в использовании классов.
Exception Handling Application Block (Блок обработки исключений)	Помогает разработчикам и лицам, вырабатывающим политики, в создании согласованной стратегии обработки исключений, формируемых во всех архитектурных слоях корпоративного приложения. Может выполняться протоколирование данных исключений, скрытие конфиденциальных данных путем замены исходного исключения другим, сохранение контекстных данных исключения путем помещения исходного исключения внутрь другого исключения.
Logging Application Block (Блок протоколирования)	Упрощает реализацию общих функций протоколирования. Блок позволяет записывать данные в Windows Event Log, сообщения электронной почты, базу данных, Windows Message Queuing, текстовый файл, WMI-событие или заданную папку.
Policy Injection Application Block (Блок внедрения политик)	Помогает разработчикам осуществлять управление сквозной функциональностью, максимизировать разделение функциональных областей, инкапсулировать поведение путем автоматического применения политик к экземплярам объектов. Разработчики определяют набор политик для целевых классов и их членов через конфигурацию или применение атрибутов к отдельным членам целевого класса.
Security Application Block (Блок безопасности)	Помогает разработчикам реализовать в приложениях общие функции, связанные с авторизацией, и кэшировать данные авторизации и аутентификации пользователя. Имея в распоряжении также функции Microsoft .NET Framework 2.0, разработчики могут легко реализовать общую функциональность, связанную с безопасностью.
Unity Application Block (Блок Unity)	Обеспечивает легковесный и расширяемый контейнер внедрения зависимостей (Dependency Injection, DI) с поддержкой внедрения конструкторов, свойств и вызовов методов. Разработчики могут использовать данный блок в сочетании с Enterprise Library для формирования объектов Enterprise Library и собственных бизнес-объектов, либо блок может использоваться как самостоятельный механизм

	внедрения зависимостей.
Validation Application Block (Блок валидации)	Предлагает разработчикам полезные функции по реализации структурированных и простых в обслуживании сценариев валидации. В этот блок входит библиотека средств проверки для валидации типов данных .NET Framework, такие как средства проверки пустой строки и диапазона чисел. Также он включает сложные средства проверки и поддерживает наборы правил.

БЛОК КЭШИРОВАНИЯ

Блок Caching Application Block позволяет включать локальный кэш в приложения, которые используют кэш в памяти и (необязательно) базу данных или изолированное резервное хранилище. Блок обеспечивает всю необходимую функциональность для извлечения, добавления и удаления кэшированных данных, поддерживает настраиваемые политики истечения срока действия и сборки мусора. Кроме того, есть возможность расширять функциональность путем создания собственных подключаемых поставщиков или использования поставщиков сторонних производителей, скажем, для поддержки распределенного кэширования или других функций. Кэширование может обеспечить значительное улучшение производительности и повышение эффективности во многих сценариях приложения.

Ключевые сценарии

Caching Application Block целесообразно использовать в следующих ситуациях:

- Многократно повторяющийся доступ к статическим данным или редко изменяемым данным.
- Доступ к данным, создание, доступ или передача которых требуют привлечения большого количества ресурсов.
- Работа с данными, которые всегда должны быть доступными, даже если источник данных (например, сервер) недоступен.

Применение

Caching Application Block оптимизирован для обеспечения высокой производительности и масштабирования. Более того, он безопасен с точки зрения исключений и потоков. Его можно расширять собственными политиками истечения срока действия и добавлять собственное резервное хранилище. Он применим для большинства общих сценариев кэширования данных, когда приложение и кэш находятся в одной системе. Это означает, что кэш является локальным и может использоваться только этим приложением. С учетом этих рекомендаций блок приложений идеально подходит для реализации следующих требований:

- Обеспечение согласованного и простого интерфейса с реализацией функциональности кэша для разных сред приложений, не зависящего от используемого хранилища кэша. Например, разработчики могут писать похожий

код для реализации кэширования в компонентах приложения, размещаемых в Internet Information Services (IIS), Enterprise Services и насыщенных клиентских приложениях. Кроме того, существуют одинаковые параметры настройки кэша для всех сред.

- Обеспечение настраиваемого и постоянного резервного хранилища. Блок поддерживает и изолированные хранилища, и резервные хранилища базы данных. Разработчики могут создавать дополнительных поставщиков резервного хранилища и добавлять их в блок, используя его параметры конфигурации. Также данный блок приложений может осуществлять симметричное шифрование данных элементов кэша перед сохранением в резервном хранилище.
- Обеспечение того, чтобы изменение параметров конфигурации кэша не приводило к изменению исходного кода приложения. Разработчики пишут код, в котором используется один или несколько именованных кэшей. Впоследствии операторы или разработчики системы могут конфигурировать каждый из этих именованных кэшей в отдельности с помощью инструментов настройки Enterprise Library.
- Элементы кэша требуют любого из следующих вариантов настройки срока действия: абсолютное время, скользящий временной интервал, расширенный формат времени (например, каждый вечер в полночь), зависимость от файлов или неограниченный срок действия.
- Изменение исходного кода блока для расширения или настройки.
- Использование множества типов хранилищ кэша (через разные диспетчеры кэша) в одном приложении.

Caching Application Block может использоваться с любыми из нижеперечисленных типов приложений:

- Windows Forms.
- Консольное приложение.
- Сервис Windows.
- Сервер COM+.
- Веб-приложение ASP.NET или Веб-сервис, если необходимы функции, не вошедшие в кэш ASP.NET.

Замечания

При работе с Caching Application Block обратите внимание на следующее:

- Блок должен развертываться в одном домене приложения. Каждый домен приложения может иметь одно или несколько хранилищ кэша (с или без резервных хранилищ).

- Хранилища кэша не могут использоваться совместно разными доменами приложений.
 - Несмотря на то, что есть данные, кэшированные в резервных хранилищах, могут быть зашифрованы, блок не поддерживает шифрование данных, кэшированных в памяти.
 - Блок не поддерживает защиту от несанкционированного доступа (подпись и верификацию элементов в кэше).
-

Блок шифрования

Блок Cryptography Application Block упрощает добавление таких функций, как шифрование данных, создание хэша данных, сравнение значений хэша для проверки наличия изменений в данных.

Ключевые сценарии

Cryptography Application Block целесообразно использовать для:

- Быстрого и простого шифрования и дешифрования данных.
 - Быстрого и простого создания хэша из данных.
 - Сравнения значений хэша для проверки наличия изменений в данных.
-

Применение

Cryptography Application Block идеально подходит для реализации следующих требований:

- Устранение необходимости в написании стандартного кода для выполнения задач хэширования, стандартного шифрования и дешифрования данных.
 - Сохранение согласованности практик шифрования как внутри приложения, так и в рамках всего предприятия.
 - Упрощение процесса обучения разработчиков через использование единой архитектурной модели безопасности во всех областях функциональности.
 - Добавление или расширение реализаций поставщиков шифрования.
 - Создание настраиваемой модели защиты ключа (Key Protection Model).
-

Замечания

При работе с Cryptography Application Block обратите внимание на следующее:

- Блок поддерживает только симметричные алгоритмы, использующие одинаковые ключи для шифрования и дешифрования.

- Блок не поддерживает автоматического управления ключами шифрования и хранилищем ключей.
-

Блок доступа к данным

Блок Data Access Application Block упрощает многие общие задачи доступа к данным, такие как чтение данных для отображения, передача данных между слоями приложения, возврат измененных данных в систему управления базами данных. Этот блок включает поддержку хранимых процедур и языка SQL, обеспечивает доступ к наиболее часто используемым функциям ADO.NET в виде простых в использовании классов.

Ключевые сценарии

Блок доступа к данным целесообразно использовать в следующих ситуациях:

- Использование **DataReader** или **DataSet** для извлечения множества строк данных.
 - Выполнение команды и извлечение выходных параметров или элемента с одним значением.
 - Выполнение множества операций внутри транзакции.
 - Извлечение XML-данных из SQL Server.
 - Обновление базы данных данными из объекта **DataSet**.
 - Добавление или расширение реализаций поставщиков баз данных.
-

Применение

Блок доступа к данным идеально подходит для реализации следующих требований:

- Обеспечение простоты и удобства использования функций, предоставляемых ADO.NET через лучшие практики.
 - Устранение необходимости в написании стандартного кода для выполнения стандартных задач доступа к данным.
 - Обеспечение согласованных практик доступа к данным как внутри приложения, так и в пределах всего предприятия.
 - Упрощение операции по изменению типа целевой базы данных через конфигурацию и сокращение объема кода, необходимого для перевода приложений на другие типы баз данных.
 - Устранение необходимости изучения разработчиками различных моделей программирования для разных типов баз данных.
-

Замечания

При работе с Data Access Application Block обратите внимание на следующее:

- Data Access Application Block является дополнением к ADO.NET, но не заменой. Если приложение должно извлекать данные особым способом или использовать функции, характерных для какой-то конкретной базы данных, используйте непосредственно ADO.NET.
-

Блок обработки исключений

Блок Exception Handling Application Block позволяет легко и быстро проектировать и реализовывать согласованную стратегию обработки исключений, формирующихся во всех архитектурных слоях приложения. Этот блок обеспечивает протоколирование сведений об исключениях, скрытие конфиденциальных данных путем замены исходного исключения другим, сохранение контекстных данных исключения путем заключения исходного исключения внутрь другого исключения.

Ключевые сценарии

Exception Handling Application Block позволяет разработчикам инкапсулировать логику операторов catch в компоненты приложения в виде пригодных для повторного использования обработчиков исключений. Данный блок целесообразно использовать в следующих ситуациях:

- Обертывание исключения. Используйте обработчик Wrap для размещения исключения внутри нового исключения.
 - Замена исключения. Используйте обработчик Replace для замены одного исключения другим.
 - Протоколирование исключения. Используйте обработчик Logging для форматирования данных исключения, таких как сообщение и записи трассировки стека, и их передачи в блок Logging Application Block для публикации.
 - Экранирование исключения в границах WCF-сервиса. Используйте обработчик Fault Contract Exception (Контракт уведомления о сбоях для исключения), который предназначен для работы в границах сервиса Windows Communication Foundation (WCF), для формирования нового Fault Contract (Контракт уведомления о сбоях) из исключения.
 - Распространение исключения, отображение понятных пользователю сообщений, уведомление пользователя и техническая поддержка. Используйте сочетание обработчиков блока для обработки особых типов исключений и при необходимости для повторного их формирования.
 - Локализация сообщений об исключении. С помощью обработчиков и их настройки задайте локализованные тексты сообщений для исключений.
-

Применение

Exception Handling Application Block идеально подходит для реализации следующих требований:

- Обеспечение поддержки обработки исключений во всех архитектурных слоях приложения, а не только на границах интерфейса сервиса.
 - Политики обработки исключений должны быть определены и обслуживаться на административном уровне через конфигурацию. Также должна быть обеспечена возможность сохранения и изменения правил, управляющих обработкой исключений, без изменения кода блока приложения.
 - Обеспечение общих функций обработки исключений, таких как протоколирование данных исключений, сокрытие конфиденциальных данных путем замены исходного исключения другим исключением, сохранение контекстных данных исключения путем помещения исходного исключения внутрь другого исключения.
 - Объединение обработчиков исключений с целью формирования необходимого ответа на исключение, например, протоколирования данных исключения с последующей заменой исходного исключения другим исключением.
 - Обеспечение единообразного вызова обработчиков исключений, чтобы их можно было использовать в разных местах внутри приложения и в разных приложениях.
 - Добавление или расширение реализаций обработчиков исключений.
 - Обеспечить обработку исключений через политики, а не просто их протоколирование.
-

Блок протоколирования

Блок Logging Application Block упрощает реализацию общих функций протоколирования, таких как запись данных в Windows Event Log, сообщения электронной почты, базу данных, Windows Message Queuing, текстовый файл, WMI-событие или заданный каталог.

Ключевые сценарии

Logging Application Block целесообразно использовать в следующих ситуациях:

- Формирование и протоколирование данных события в журнал Windows Event, сообщение электронной почты, базу данных, очередь сообщений, текстовый файл, событие инструментария управления Windows (Windows Management Instrumentation, WMI) или заданный каталог.
 - Изменение и форматирование контекстных данных внутри события с помощью шаблонов.
 - Отслеживание действий приложения и обеспечение идентификаторов, которые могут использоваться для компоновки данных события.
 - Предотвращение несанкционированного доступа к конфиденциальным данным с помощью списков управления доступом (Access Control List, ACL) для ограничения доступа к неструктурированным файлам или создание собственного модуля форматирования для шифрования данных журнала.
-

Применение

Logging Application Block идеально подходит для реализации следующих требований:

- Обеспечение единообразных практик протоколирования как внутри приложения, так и в рамках всего предприятия.
- Упрощение процесса обучения разработчиков за счет применения единообразной архитектурной модели.
- Обеспечение решения общих задач протоколирования в приложениях, которые помогут избежать необходимости многократного повторного написания одинакового кода.
- Добавление или расширение протоколирования и целевых объектов.

Замечания

При работе с Logging Application Block обратите внимание на следующее:

- Модули форматирования протоколов блока не обеспечивают шифрования протоколируемых данных.
- Обработчики трассировки получают данные протоколирования в виде открытого текста.
- Некоторые слушатели Logging Application Block дают сбой при выполнении в частично доверяемой изолированной программной среде.

Блок внедрения политик

Блок Policy Injection Application Block обеспечивает механизм автоматического применения политик к экземплярам объектов. Благодаря этому разработчики получают лучшие возможности управления сквозной функциональностью, разделения функциональных областей и инкапсуляции поведения. Разработчики определяют набор политик для целевых классов и их членов через конфигурацию Policy Injection Application Block или применение атрибутов к отдельным членам целевого класса.

Ключевые сценарии

Policy Injection Application Block целесообразно использовать в следующих ситуациях:

- Создание приложений из объектов, требующих инкапсуляции и разделения для обеспечения максимальной выгоды от независимости в работе и лучших возможностей повторного использования.
- Обеспечение возможности разработчикам, операторам и администраторам создавать, изменять, удалять и выполнять точную настройку политик перехвата через конфигурирование, обычно без необходимости изменения кода или

повторной компиляции приложения. Это снижает вероятность появления ошибок в коде, упрощает управление версиями и уменьшает время простоя.

- Повторное использование существующих экземпляров объектов. Это сокращает требования к коду по формированию новых экземпляров объектов и их подготовке через задание свойств или вызов методов. При этом по-прежнему разрешается использование конвейеров обработчиков для отдельных или всех членов класса.
- Сокращение необходимого объема работ и кода для выполнения общих задач в приложении, таких как протоколирование, валидация, авторизация и инструментирование.

Применение

Policy Injection Application Block идеально подходит для реализации следующих требований:

- Обеспечение готового решения, которое просто реализовать в новых и существующих приложениях, особенно в приложениях, уже использующих функции Enterprise Library.
- Управление сквозной функциональностью. Отсутствие управления может повлиять на независимость объектов, которым необходим доступ к общим функциям (таким как протоколирование или валидация).
- Предоставление разработчику и администратору возможности настройки поведения объектов в приложении через конфигурацию путем добавления или удаления обработчиков, которые выполняют общие задачи или добавляют специальные функции.
- Упрощение применения функций Enterprise Library Core и отдельных блоков приложений, которые реализуют часто используемые в корпоративных приложениях задачи.
- Сокращение продолжительности и стоимость разработки, сокращение количества ошибок в сложных приложениях, в которых применяются общие и совместно используемые задачи и сервисы.

Замечания

При работе с Policy Injection Application Block обратите внимание на следующее:

- Для активации только обработчиков предварительной и последующей обработки используется перехват, а не вставка кода напрямую.
- Блок не обеспечивает перехват для конструкторов класса.
- Как все технологии перехвата, добавляет в приложения необходимость в дополнительной обработке, несмотря на то что дизайн блока обеспечивает максимальное сокращение этой необходимости.

- Обработчики вызовов имеют доступ только к данным внутри сообщения вызова и не могут сохранять внутреннее состояние.
 - Внедрение политик может иметь место только для открытых членов целевого класса.
-

Блок безопасности

Блок Security Application Block позволяет без труда реализовывать общие функции, связанные с авторизацией, например, кэширование данных авторизации и аутентификации пользователя и интеграцию с функциями безопасности Microsoft .NET Framework.

Ключевые сценарии

Security Application Block целесообразно использовать в следующих ситуациях:

- Кэширование учетных данных, связанных с безопасностью, которые используются для авторизации.
 - Получение временного маркера для аутентифицированного пользователя и аутентификация пользователя с помощью маркера.
 - Завершение сеанса пользователя (истечение срока действия маркера).
 - Определение того, авторизован ли пользователь для выполнения задачи.
-

Применение

Блок Security Application Block идеально подходит для реализации следующих требований:

- Устранение необходимости написания одного и того же кода для выполнения стандартных задач, связанных с безопасностью, например, кэширования учетных данных и проверки аутентификации.
 - Обеспечение согласованности практик безопасности как внутри приложения, так и в рамках всего предприятия.
 - Упрощение процесса обучения разработчиков за счет применения единообразной архитектурной модели во всех областях предложенной функциональности.
 - Необходимо использовать собственные реализации поставщиков безопасности.
-

Замечания

При работе с Security Application Block обратите внимание на следующее:

- Caching Application Block является хранилищем по умолчанию для кэшированных данных, связанных с безопасностью. Несмотря на то, что Caching Application Block можно настроить на шифрование кэшированных данных в резервных хранилищах, этот блок не поддерживает шифрования кэшированных данных в памяти. Если

вопрос шифрования в памяти является для приложения ключевым, можно использовать собственный поставщик хранилища кэша, который поддерживает этот тип шифрования.

- При выполнении в частично доверяемой изолированной программной среде отсутствует поддержка диспетчера авторизации.
-

Блок Unity

Unity — это легковесный расширяемый контейнер внедрения зависимостей с поддержкой перехвата объектов, внедрения конструкторов, свойств и вызовов методов. Данный блок может использоваться с Enterprise Library для формирования объектов Enterprise Library и пользовательских бизнес-объектов.

Ключевые сценарии

Блок Unity Application Block целесообразно использовать в следующих ситуациях:

- Внедрение зависимостей посредством контейнера, который поддерживает внедрение конструкторов, свойств и вызовов методов и может управлять жизненным циклом экземпляров объектов.
 - Внедрение зависимостей для классов, которые зависят от других объектов или классов, и эти зависимости являются сложными или требуют абстракции.
 - Конфигурирование и изменение зависимостей во время выполнения.
 - Кэширование или сохранение контейнера при обратной передаче в Веб-приложение.
-

Применение

Unity Application Block идеально подходит для реализации следующих требований:

- Упрощение процедуры создания объекта, в особенности для иерархических структур и зависимостей объектов, что упрощает код приложения.
 - Абстрагирование требований за счет определения зависимостей во время выполнения или при конфигурации для упрощения управления сквозной функциональностью.
 - Повышение гибкости путем переноса конфигурации компонентов в контейнер.
 - Обеспечение возможности размещения сервиса, где клиенты смогут хранить или кэшировать контейнер. Это особенно полезно в Веб-приложениях ASP.NET, в которых разработчики могут сохранять контейнер в сеансе или приложении ASP.NET.
-

Unity Application Block не рекомендуется использовать в следующих случаях:

-
- Если объекты или классы не имеют зависимостей от других объектов или классов либо эти зависимости просты и не требуют абстракции.

Замечания

При работе с Unity Application Block обратите внимание на следующее:

- Внедрение зависимостей может привести к незначительному падению производительности.
 - Если имеются только простые зависимости, внедрение зависимостей может повысить сложность.
-

Блок валидации

Блок Validation Application Block предлагает ряд функций для реализации структурированных и простых в обслуживании механизмов валидации с помощью атрибутов и наборов правил, а также для интеграции с большинством типов технологий интерфейсов приложений.

Ключевые сценарии

Блок Validation Application Block целесообразно использовать в следующих ситуациях:

- Реализация структурированного и простого в обслуживании кода валидации полей, свойств и вложенных объектов и предотвращения внедрения злонамеренных данных в приложение.
 - Реализация бизнес-правил и обеспечение откликов на пользовательский ввод.
 - Многократная валидация данных в рамках приложения с применением одних и тех же одинаковых правил.
 - Объединение различных готовых средств проверки для обеспечения поддержки сложных сценариев и широкого спектра возможностей.
-

Применение

Validation Application Block идеально подходит для реализации следующих требований:

- Сохранение согласованности практик валидации практически для всех стандартных типов данных .NET в приложениях ASP.NET, Windows Forms и WCF.
 - Создание правил валидации с использованием конфигурации, атрибутов и кода.
 - Соотнесение множества наборов правил с одним классом и всеми его членами.
 - Применение одного или более наборов правил при валидации объекта и повторное использование бизнес-логики валидации.
-

Замечания

При работе с Validation Application Block обратите внимание на следующее:

- Некоторые технологии, такие как ASP.NET и Windows Forms, предлагают встроенные функции валидации. Поэтому в случаях, когда логика валидации должна применяться только в рамках этих технологий, работайте с Validation Application Block, только если логику валидации необходимо использовать многократно.
- WCF и другие приложения, работающие с XML-данными, могут использовать XML-схемы для проверки сообщений на уровне XML. Если логику валидации требуется применять только в рамках этих технологий, работайте с Validation Application Block, только если логику валидации необходимо использовать многократно.
- В самых простых случаях, когда требуется проверка лишь нескольких объектов, не стоит тратить время на добавление блока приложений.

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- «Enterprise Library» по адресу <http://msdn.microsoft.com/en-us/library/cc467894.aspx>.
- «The Caching Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511588.aspx>.
- «The Cryptography Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511721.aspx>.
- «The Data Access Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511547.aspx>.
- «The Exception Handling Application Block» по адресу <http://msdn2.microsoft.com/en-us/library/aa480461.aspx>.
- «The Logging Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511708.aspx>.
- «The Policy Injection Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511729.aspx>.
- «The Security Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511928.aspx>.
- «The Unity Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511654.aspx>.
- «The Validation Application Block» по адресу <http://msdn.microsoft.com/en-us/library/cc511802.aspx>.

- «*Enterprise Library Frequently Asked Questions*» (Часто задаваемые вопросы по Enterprise Library) по адресу
<http://www.codeplex.com/entlib/Wiki/View.aspx?title=EntLib%20FAQ>.

Приложение

Ж

Каталог шаблонов patterns & practices

Руководство по проектированию составных приложений для WPF и Silverlight

Категория	Шаблоны
Модульность	Service Locator (Локатор сервиса). Создает локатор сервиса, включающий ссылки на сервисы и инкапсулирующий логику для их размещения. В своих классах используйте локатор сервиса для получения экземпляров сервиса. Более подробно этот шаблон рассматривается в материале « <i>Service Locator</i> » по адресу http://msdn.microsoft.com/en-us/library/dd458903.aspx .
Тестируемость	Dependency Injection (Внедрение зависимостей). Экземпляры зависимостей не должны создаваться в классе явно, а должны быть представлены декларативно в его описании. Используйте объект Builder для получения действительных экземпляров зависимостей объекта и их передачи в объект при его создании и/или инициализации. Более подробно этот шаблон рассматривается в материале « <i>Dependency Injection</i> » по адресу http://msdn.microsoft.com/en-us/library/dd458879.aspx . Inversion of Control (Инверсия управления). Делегируйте функции выбора конкретного типа реализации зависимостей классов внешнему компоненту или источнику. Более подробно этот шаблон рассматривается в материале « <i>Inversion of Control</i> » по адресу http://msdn.microsoft.com/en-us/library/dd458907.aspx . Separated Presentation (Отделение представления). Отделите логику представления от бизнес-логики. Шаблон Separated Presentation может быть реализован по-разному, например, как Supervising Presenter или Presentation Model и т.п. Более подробно этот шаблон рассматривается в материале « <i>Separated Presentation</i> » по адресу http://msdn.microsoft.com/en-us/library/dd458859.aspx . Presentation Model (Модель презентации). Распределяет обязанности по визуальному отображению и реализации состояния и поведения пользовательского

	<p>интерфейса (UI) между разными классами, которые названы представление и модель презентации, соответственно. Класс представления управляет элементами управления UI. Класс модели презентации инкапсулирует поведение доступа к модели и обеспечивает открытый интерфейс, с которым легко работать представлению (например, посредством привязки данных), выступая, таким образом, в роли фасада для модели с характерным для UI состоянием и поведением. Более подробно этот шаблон рассматривается в материале «<i>Presentation Model</i>» по адресу http://msdn.microsoft.com/en-us/library/dd458863.aspx.</p> <p>Supervising Presenter (Наблюдающий презентатор) или Supervising Controller (Наблюдающий контроллер). Распределяет обязанности по визуальному отображению и поведению обработки событий между разными классами, которые названы представление и презентатор, соответственно. Класс представления управляет элементами управления UI, передавая события пользователей классу презентатора. Класс презентатора включает логику, которая позволяет отвечать на события, обновлять модель (бизнес-логику и данные приложения) и управлять состоянием класса представления. Более подробно этот шаблон рассматривается в материале «<i>Supervising Presenter</i>» по адресу http://msdn.microsoft.com/en-us/library/dd490821.aspx.</p>
--	---

Шаблоны передачи данных

Категория	Шаблоны
<i>Шаблоны перемещения данных</i>	<p>Data Replication (Репликация данных). Создает набор репликации и связь репликации для передачи данных между двумя местоположениями. Этот шаблон описывает общий процесс, который детализирован в шаблонах передачи данных, описываемых в этой таблице. Более подробно этот шаблон рассматривается в материале «<i>Data Replication</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978671.aspx.</p> <p>Master-Master Replication (Репликация главный-главный). Копирует данные из источника в заданное место, выявляет и разрешает любые конфликты обновления, возникшие с момента последней репликации (из-за изменений одинаковых данных источника и цели). Решение включает две перекрестные связи репликации между источником и целевым местоположением. Обе связи передают одинаковый набор репликации в обоих направлениях. Такие пары связей репликации называют <i>связанными</i>. Более подробно этот шаблон рассматривается в материале «<i>Master-Master Replication</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978735.aspx.</p> <p>Master-Subordinate Replication (Репликация главный-подчиненный). Копирует данные из источника в заданное место без учета обновлений, которые могли иметь место для набора репликации в целевом местоположении с момента последней репликации. Более подробно этот шаблон рассматривается в материале «<i>Master-Subordinate Replication</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978740.aspx.</p> <p>Master-Master Row-Level Synchronization (Синхронизация главный-главный на уровне строк). Использует пару связанных связей репликации между источником и целевым местоположением и контроллер синхронизации для управления синхронизацией в обоих направлениях. Для синхронизации более двух копий набора репликации создайте соответствующую пару связей репликации для каждой дополнительной копии. Более подробно этот шаблон рассматривается в материале «<i>Master-Master Row-Level Synchronization</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998434.aspx</p> <p>Master-Subordinate Snapshot Replication (Репликация снимка главный-</p>

	<p>подчиненный). Создает копию набора репликации источника в определенный момент времени (называется снимком), переносит эту копию в целевое местоположение и перезаписывает целевые данные. В этом случае любые изменения, которые могли произойти в целевом наборе репликации, заменяются новым набором репликации источника. Более подробно этот шаблон рассматривается в материале «<i>Master-Subordinate Snapshot Replication</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998430.aspx.</p> <p>Capture Transaction Details (Захват деталей транзакции). Создает дополнительные объекты базы данных, такие как триггеры и (теневые) таблицы, и записывает изменения всех таблиц, входящих в набор репликации. Более подробно этот шаблон рассматривается в материале «<i>Capture Transaction Details</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978709.aspx.</p> <p>Master-Subordinate Transactional Incremental Replication (Транзакционная инкрементная репликация главный-подчиненный). Запрашивает у источника сведения о завершенных транзакциях и повторяет транзакции в правильной последовательности при записи в целевое местоположение. Более подробно этот шаблон рассматривается в материале «<i>Master-Subordinate Transactional Incremental Replication</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998441.aspx.</p> <p>Master-Subordinate Cascading Replication (Каскадная репликация главный-подчиненный). Увеличивает число связей репликации между источником и целью, добавляя одну или более промежуточных целей между источником и конечными целевыми базами данных. Эти промежуточные цели являются хранилищами данных, принимающими набор репликации из источника и, таким образом, выступают в роли целевого местоположения для предыдущей связи репликации. При передаче данных по следующей связи репликации предыдущая промежуточная цель уже выступает в качестве источника, и так далее, пока не будет достигнута конечная цель каскада. Более подробно этот шаблон рассматривается в материале «<i>Master-Subordinate Cascading Replication</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978712.aspx.</p>
Прототипы шаблонов	<p>Maintain Data Copies (Хранение копий данных). Выполняет синхронную запись в копии данных из исходного приложения или синхронную публикацию данных в локальный кэш для дальнейшей их передачи асинхронным сервисом. Более подробно этот шаблон рассматривается в материале «<i>Patterns and Pattlelets</i>» (Шаблоны и прототипы) по адресу http://msdn.microsoft.com/en-us/library/ms998465.aspx.</p> <p>Application-Managed Data Copies (Управляемые приложением копии данных). Когда какое-либо приложение изменяет свою копию данных, оно также должно внести изменения и в другие копии. Приложение должно обеспечить обновление копий данных и/или производных данных той же транзакцией, которая выполняет изменение исходных данных. Более подробно этот шаблон рассматривается в материале «<i>Patterns and Pattlelets</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998465.aspx.</p> <p>Extract-Transform-Load (Извлечение-Преобразование-Загрузка). Тип передачи данных, который может обеспечивать выполнение сложных запросов для получения данных из гетерогенных источников, применение сложных операций с данными, которые включают агрегацию и очистку, но всегда выполняющий простую запись с замещением всех изменений в цели. Более подробно этот шаблон рассматривается в материале «<i>Patterns and Pattlelets</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998465.aspx.</p> <p>Topologies for Data Copies (Топологии копий данных). Архитектурные подходы по развертыванию копий данных на нескольких платформах. Более подробно этот</p>

	шаблон рассматривается в материале « <i>Patterns and Pattlets</i> » по адресу http://msdn.microsoft.com/en-us/library/ms998465.aspx .
--	---

Шаблоны корпоративных решений

Категория	Шаблоны
<i>Шаблоны развертывания</i>	<p>Deployment Plan (План развертывания). Создает план развертывания, описывающий, какие компоненты приложения на какие уровни будут развертываться. Если в процессе распределения компонентов по слоям выясняется, что какой-то слой не подходит для компонента, определите рентабельность изменения компонента для улучшения его работы с инфраструктурой или изменения инфраструктуры для большего соответствия компоненту. Более подробно этот шаблон рассматривается в материале «<i>Deployment Plan</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978676.aspx.</p> <p>Layered Application (Многослойное приложение). Разделяет компоненты решения на слои. Компоненты каждого слоя должны быть связанными и находится примерно на одном уровне абстракции. Каждый слой должен быть слабо связан с нижележащими слоями. Более подробно этот шаблон рассматривается в материале «<i>Layered Application</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978678.aspx.</p> <p>Three-Layered Services Application (Трехслойное сервисное приложение). Разделяет многослойную архитектуру на три слоя – слой представления, бизнес-слой и слой данных – для уменьшения связанности и увеличения связности. Более подробно этот шаблон рассматривается в материале «<i>Three-Layered Services Application</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978689.aspx.</p> <p>Tiered Distribution (Многоуровневое распределение). Структурирует серверы и клиентские компьютеры как набор физических уровней и распределяет компоненты приложений по соответствующим уровням. Более подробно этот шаблон рассматривается в материале «<i>Tiered Distribution</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978701.aspx.</p> <p>Three-Tiered Distribution (Трехуровневое распределение). Структурирует приложение в три физических уровня: клиентский уровень, уровень приложения и уровень базы данных. Более подробно этот шаблон рассматривается в материале «<i>Three-Tiered Distribution</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978694.aspx.</p>
<i>Распределенные системы</i>	<p>Broker (Брокер). Используйте шаблон Broker для скрытия деталей реализации вызова удаленного сервиса путем их инкапсуляции в отдельный слой. Более подробно этот шаблон рассматривается в материале «<i>Broker</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978706.aspx.</p> <p>Data Transfer Object (Объект передачи данных). Создает объект передачи данных (Data Transfer Object, DTO), содержащий все данные, необходимые для удаленного вызова. Изменяет сигнатуру удаленного метода, чтобы принимать DTO как один параметр и возвращать его клиенту. После того как DTO получено и сохранено как локальный объект, приложение может вызывать процедуры DTO без необходимости выполнения удаленных вызовов. Более подробно этот шаблон рассматривается в материале «<i>Data Transfer Object</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978717.aspx.</p>

	Singleton (Синглтон). Синглтон обеспечивает создание единственного экземпляра класса. Другие объекты могут выполнять доступ к данному экземпляру через глобально доступный метод класса, который возвращает ссылку на экземпляр. Кроме того, конструктор класса должен быть объявлен <code>private</code> , чтобы ни один другой объект не смог создать новый экземпляр. Более подробно этот шаблон рассматривается в материале « <i>Singleton</i> » по адресу http://msdn.microsoft.com/en-us/library/ms998426.aspx .
<i>Производительность и надежность</i>	<p>Server Clustering (Кластеризация сервера). Кластер серверов – это объединение двух или более серверов, представляющееся как один сервер. Таким образом, формируется виртуальный ресурс, который расширяет доступность, масштабируемость или и то, и другое. Более подробно этот шаблон рассматривается в материале «<i>Server Clustering</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998414.aspx.</p> <p>Load-Balanced Cluster (Кластер с балансировкой нагрузки). Обеспечивает установку сервиса или приложения на несколько серверов, которые сконфигурированы для разделения рабочей нагрузки. Такой тип конфигурации называют кластером с балансировкой нагрузки. Балансировка нагрузки масштабирует производительность таких серверных программ, как Веб-сервер, путем распределения клиентских запросов на множество серверов. Технологии балансировки нагрузки, которые обычно называют подсистемами балансировки нагрузки, принимают входящие запросы и в случае необходимости перенаправляют их на определенный узел. Узлы балансировки нагрузки одновременно отвечают на запросы разных клиентов (даже на множество запросов одного клиента). Более подробно этот шаблон рассматривается в материале «<i>Load-Balanced Cluster</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978730.aspx.</p> <p>Failover Cluster (Отказоустойчивый кластер). Отказоустойчивый кластер – это набор серверов, сконфигурированных таким образом, что если один сервер становится недоступным, другие серверы автоматически берут на себя работу вышедшего из строя сервера, продолжая обработку. Каждому серверу кластера назначен по крайней мере один резервный сервер из числа серверов этого же кластера. Более подробно этот шаблон рассматривается в материале «<i>Failover Cluster</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978720.aspx.</p>
<i>Шаблоны сервисов</i>	<p>Service Interface (Интерфейс сервиса). Создает компонент, обеспечивающий точку входа для взаимодействия потребителей с сервисом, и предоставляет слабо детализированный интерфейс, отделяя реализацию от бизнес-логики. Более подробно этот шаблон рассматривается в материале «<i>Service Interface</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998421.aspx.</p> <p>Service Gateway (Шлюз сервиса). Инкапсулирует код, реализующий клиентскую часть контракта, в отдельный компонент Service Gateway, который выступает в роли посредника для других сервисов, инкапсулируя детали подключения к источнику и выполняя все необходимые преобразования. Более подробно этот шаблон рассматривается в материале «<i>Service Gateway</i>» по адресу http://msdn.microsoft.com/en-us/library/ms998420.aspx.</p>
<i>Шаблоны Веб-представления</i>	Model-View-Controller (Модель-представление-контроллер). Этот шаблон распределяет основанные на пользовательском вводе данные предметной области, представления и действий в три отдельных класса. Модель управляет поведением и данными домена приложения, отвечает на запросы данных о состоянии (обычно поступающие от Представления) и на команды изменить состояние (обычно поступающие от Контроллера). Представление

	<p>управляет отображением информации. Контроллер интерпретирует входные данные пользователя, получаемые посредством мыши или клавиатуры, и информирует модель и/или представление о необходимости соответствующего изменения. Более подробно этот шаблон рассматривается в материале «<i>Model-View-Controller</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978748.aspx.</p> <p>Page Controller (Контроллер страниц). Этот шаблон позволяет принимать входные данные запроса страницы, вызывать запрошенные действия модели и определять соответствующее представление результирующей страницы. Отделяет логику диспетчеризации от кода представления. Где это необходимо, создавайте общий базовый класс для всех контроллеров страниц, чтобы избежать дублирования кода и повысить согласованность и тестируемость. Более подробно этот шаблон рассматривается в материале «<i>Page Controller</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978764.aspx.</p> <p>Front Controller (Контроллер запросов). Этот шаблон решает проблему децентрализации, присущую шаблону Page Controller, путем направления всех запросов через один контроллер. Сам контроллер обычно реализован в виде двух составляющих: обработчика и иерархии команд. Обработчик получает от Веб-сервера HTTP-запросы Post или Get и извлекает из них соответствующие параметры. Он использует параметры запроса для выбора необходимой команды и затем для передачи управления команде для обработки. Сами команды также входят в состав контроллера. Команды представляют определенные действия, описанные в шаблоне Command. Более подробно этот шаблон рассматривается в материале «<i>Front Controller</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978723.aspx.</p> <p>Intercepting Filter (Перехватывающий фильтр). Используйте этот шаблон для создания цепочки компонуемых фильтров при реализации общих задач предварительной и последующей обработки в процессе запроса Веб-страницы. Более подробно этот шаблон рассматривается в материале «<i>Intercepting Filter</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978727.aspx.</p> <p>Page Cache (Кэш страниц). Кэширует выходные данные, сформированные сервером для страниц, которые достаточно редко меняются, но часто запрашиваются, чтобы сократить загруженность сервера. Более подробно этот шаблон рассматривается в материале «<i>Page Cache</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978759.aspx.</p> <p>Observer (Наблюдатель). Используйте шаблон Observer для сохранения списка необходимых зависимостей (наблюдателей) в отдельном объекте (субъект). Пусть все отдельные наблюдатели реализуют общий интерфейс Observer, чтобы устранить прямые зависимости между субъектом и зависимыми объектами. Более подробно этот шаблон рассматривается в материале «<i>Observer</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978753.aspx.</p>
--	---

Шаблоны интеграции

Категория	Шаблоны
Слой интеграции	<p>Entity Aggregation (Агрегация сущностей). Вводит слой Entity Aggregation, обеспечивающий логическое представление сущностей на уровне предприятия с физическими связями, которые поддерживают доступ и обновление соответствующих им экземпляров в серверных хранилищах. Более подробно этот</p>

	<p>шаблон рассматривается в материале «<i>Entity Aggregation</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978573.aspx.</p> <p>Process Integration (Интеграция бизнес-процессов). Определяет модель бизнес-процессов, которая описывает отдельные этапы, составляющие сложную бизнес-функцию. Создает отдельный компонент диспетчера процессов, который может одновременно интерпретировать множество экземпляров этой модели и взаимодействовать с существующими приложениями для выполнения отдельных этапов процесса. Более подробно этот шаблон рассматривается в материале «<i>Process Integration</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978592.aspx.</p> <p>Portal Integration (Интеграция в портал). Создает приложение портала, отображающее данные, полученные из нескольких приложений, в унифицированном UI. Пользователи могут осуществлять необходимые задачи на основании данных, отображаемых на этом портале. Более подробно этот шаблон рассматривается в материале «<i>Portal Integration</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978585.aspx.</p>
<i>Топологии интеграции</i>	<p>Message Broker (Брокер сообщений). Расширяет решение интеграции. Брокер сообщений – это физический компонент, обеспечивающий связь между приложениями. Приложения взаимодействуют не друг с другом, а только с посредником - брокером сообщений. Приложение отправляет сообщение посреднику вместе с логическим именем получателей. Посредник находит приложения, зарегистрированные под данным логическим именем, и передает им сообщение. Более подробно этот шаблон рассматривается в материале «<i>Message Broker</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978579.aspx.</p> <p>Message Bus (Шина сообщений). Подключает все приложения через логический компонент, который называют шиной сообщений. Шина сообщений обеспечивает передачу сообщений между приложениями. Шина сообщений включает три основных элемента: набор согласованных с сообщениями схем, набор общих сообщений с командой и совместно используемую инфраструктуру для отправки сообщений шины получателям. Более подробно этот шаблон рассматривается в материале «<i>Message Bus</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978583.aspx.</p> <p>Publish/Subscribe (Публикация/Подписка). Обеспечивает классы для публикации событий, на которые могут подписаться другие приложения. Механизм Publish\Subscribe обеспечивает отправку событий или сообщений всем подписчикам. Более подробно этот шаблон рассматривается в материале «<i>Publish/Subscribe</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978603.aspx.</p>
<i>Системные подключения</i>	<p>Data Integration (Интеграция данных). Интегрирует приложения в логическом слое доступа к данным, открывая данные одного приложения (источник) для доступа из других приложений (цель). Более подробно этот шаблон рассматривается в материале «<i>Data Integration</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978572.aspx.</p> <p>Functional Integration (Функциональная интеграция). Интегрирует приложения в логическом бизнес-слое, открыв бизнес-функцию одного приложения (источник) для доступа из других приложений (цель). Более подробно этот шаблон рассматривается в материале «<i>Functional Integration</i>» по адресу http://msdn.microsoft.com/en-us/library/ms978578.aspx.</p> <p>Service-Oriented Integration (Сервисно-ориентированная интеграция). Для логической интеграции приложений в бизнес-слое, позволяет системам использовать и предоставлять Веб-сервисы XML. Используйте контракты Web Services Description Language (WSDL) для описания интерфейсов этих систем.</p>

	<p>Обеспечьте возможность взаимодействия, согласовав вашу реализацию с семейством спецификаций Web Services (WS-*). Более подробно этот шаблон рассматривается в материале «Service-Oriented Integration» по адресу http://msdn.microsoft.com/en-us/library/ms978594.aspx.</p> <p>Presentation Integration (Интеграция представления). Обеспечивает доступ к функциям приложения через UI путем моделирования ввода данных пользователем и чтения данных с экрана монитора. Более подробно этот шаблон рассматривается в материале «Presentation Integration» по адресу http://msdn.microsoft.com/en-us/library/ms978588.aspx.</p>
<i>Дополнительные шаблоны интеграции</i>	<p>Pipes and Filters (Каналы и фильтры). Реализует преобразования путем использования последовательности компонентов фильтров, где каждый фильтр получает входящее сообщение, применяет простое преобразование и передает преобразованное сообщение следующему компоненту. Проводят сообщения через каналы, соединяющие входы и выходы фильтров и буферизующие связь между фильтрами. Более подробно этот шаблон рассматривается в материале «Pipes and Filters» по адресу http://msdn.microsoft.com/en-us/library/ms978599.aspx.</p> <p>Gateway (Шлюз). Выносит всю функциональность доступа к внешней системе в один интерфейс. Данный шаблон устраняет необходимость понимания принципов подключения к внешней системе, упрощая, таким образом, процессы разработки и обслуживания, связанные с доступом ко внешним системам. Более подробно этот шаблон рассматривается в материале «Additional Integration Patterns» (Дополнительные шаблоны интеграции) по адресу http://msdn.microsoft.com/en-us/library/ms978722.aspx.</p>

Шаблоны безопасности Веб-сервисов

Категория	Шаблоны
Аутентификация	<p>Brokered Authentication (Аутентификация через посредника). Веб-сервис проверяет учетные данные, предоставленные клиентом, без необходимости прямого взаимодействия между двумя сторонами. Посредник аутентификации, которому доверяют обе стороны, передает маркер доступа клиенту. После этого клиент может предоставить Веб-сервису учетные данные, включая и маркер доступа. Более подробно этот шаблон рассматривается в материале «Brokered Authentication» по адресу http://msdn2.microsoft.com/en-us/library/aa480560.aspx. Следующие три шаблона описывают специальные реализации шаблона Brokered Authentication.</p> <p>Brokered Authentication: Kerberos (Аутентификация через посредника: Kerberos). Выполняет аутентификацию через посредника между клиентами и Веб-серверами с помощью протокола Kerberos. Более подробно этот шаблон рассматривается в материале «Brokered Authentication: Kerberos» по адресу http://msdn2.microsoft.com/en-us/library/aa480562.aspx.</p> <p>Brokered Authentication: X509 PKI (Аутентификация через посредника: X509 PKI). Для проверки учетных данных, предоставляемых запрашивающим приложением, использует аутентификацию через посредника по сертификатам X.509, выпущенными центром сертификации (Certificate Authority, CA) для инфраструктуры открытых ключей (Public Key Infrastructure, PKI). Более подробно этот шаблон рассматривается в материале «Brokered Authentication: X509 PKI» по адресу http://msdn2.microsoft.com/en-us/library/aa480565.aspx.</p> <p>Brokered Authentication: STS (Аутентификация через посредника: STS). Использует аутентификацию через посредника с маркером доступа, выпущенным</p>

	<p>сервисом маркеров доступа (Security Token Service, STS). И клиент, и Веб-сервис доверяют STS обеспечение маркеров доступа. Более подробно этот шаблон рассматривается в материале «<i>Brokered Authentication: STS</i>» по адресу http://msdn2.microsoft.com/en-us/library/aa480563.aspx.</p> <p>Direct Authentication (Прямая аутентификация). Веб-сервис выступает в роли сервиса аутентификации, проверяя учетные данные, полученные от клиента. Учетные данные, включающие доказательство владения на основании общих секретов, сверяются с хранилищем удостоверений. Более подробно этот шаблон рассматривается в материале «<i>Direct Authentication</i>» по адресу http://msdn.microsoft.com/en-us/library/aa480566.aspx.</p>
Авторизация	<p>Trusted Subsystem (Доверенная подсистема). Веб-сервис для доступа к дополнительным ресурсам действует как доверенная подсистема. Он использует собственные учетные данные (вместо пользовательских) для доступа к ресурсам. Более подробно этот шаблон рассматривается в материале «<i>Trusted Subsystem</i>» по адресу http://msdn2.microsoft.com/en-us/library/aa480587.aspx.</p>
Управление исключениями	<p>Exception Shielding (Экранирование исключений). Очищает небезопасные исключения, заменяя их исключениями с безопасной структурой. Возвращает клиенту только очищенные исключения или исключения с безопасной структурой. Исключения с безопасной структурой не включают конфиденциальные данные в сообщение об исключении и содержат подробных записей трассировки стека, поскольку это может привести к разглашению конфиденциальных данных о внутренних операциях Веб-сервиса. Более подробно этот шаблон рассматривается в материале «<i>Exception Shielding</i>» по адресу http://msdn2.microsoft.com/en-us/library/aa480591.aspx.</p>
Шифрование сообщений	<p>Data Confidentiality (Конфиденциальность данных). Использует шифрование для защиты конфиденциальных данных сообщения. Незашифрованные данные, которые называют <i>открытым текстом</i> (plaintext), преобразуются в <i>зашифрованные данные</i> (ciphertext). Данные шифруются с помощью алгоритма и ключа шифрования. По достижении места назначения зашифрованный текст преобразуется опять в обычный текст. Более подробно этот шаблон рассматривается в материале «<i>Data Confidentiality</i>» по адресу http://msdn.microsoft.com/en-us/library/aa480570.aspx.</p>
Выявление повторов сообщений	<p>Message Replay Detection (Выявление повторов сообщений). Кэширование идентификаторов входящих сообщений и использование выявления повторов сообщений позволяет распознавать и отклонять сообщения, для которых в кэше определения повторов уже существует соответствующая запись. Более подробно этот шаблон рассматривается в материале «<i>Message Replay Detection</i>» по адресу http://msdn2.microsoft.com/en-us/library/aa480598.aspx.</p>
Подписывание сообщений	<p>Data Origin Authentication (Аутентификация источника данных). Использование аутентификации источника данных позволяет получателю удостовериться, что сообщения не были повреждены при передаче (целостность данных) и что они исходят от предполагаемого отправителя (аутентичность). Более подробно этот шаблон рассматривается в материале «<i>Data Origin Authentication</i>» по адресу http://msdn2.microsoft.com/en-us/library/aa480571.aspx.</p>
Проверка сообщений	<p>Message Validator (Средство проверки сообщений). Логика проверки сообщений реализует строго определенную политику, которая устанавливает обязательные части сообщения запроса для его успешной обработки сервисом. Проверка полезной нагрузки XML-сообщения на соответствие XML-схеме (XSD) позволяет убедиться в правильности их структуры и соответствия ожиданиям обрабатывающего сервиса. Логика проверки также проверят сообщения на</p>

	соответствие определенным критериям: проверяется размер сообщения, его содержимое и используемые наборы символов. Сообщения, не соответствующие заданным критериям, отклоняются. Более подробно этот шаблон рассматривается в материале « <i>Message Validator</i> » по адресу http://msdn2.microsoft.com/en-us/library/aa480600.aspx .
<i>Развертывание</i>	Perimeter Service Router (Пограничный сервисный маршрутизатор). Спроектируйте промежуточный Веб-сервис, который будет выступать в роли пограничного сервисного маршрутизатора. Пограничный сервисный маршрутизатор обеспечивает внешний интерфейс пограничной сети для внутренних Веб-сервисов. Он принимает сообщения от внешних приложений и направляет их на соответствующий Веб-сервис частной сети. Более подробно этот шаблон рассматривается в материале « <i>Perimeter Service Router</i> » по адресу http://msdn2.microsoft.com/en-us/library/aa480606.aspx .

Дополнительные источники

Электронная версия списка используемых источников доступна по адресу

<http://www.microsoft.com/architectureguide>.

- Какие шаблоны входят в состав библиотеки Composite Application Library (Библиотека составного приложения), можно узнать из материала [Composite Application Guidance for WPF and Silverlight](#) (Руководство по проектированию составных клиентских приложений для WPF и Silverlight).
 - Шаблоны данных рассматриваются в статье [Data Patterns](#).
 - Шаблоны корпоративных решений рассматриваются в статье [Enterprise Solution Patterns Using Microsoft .NET](#) (Шаблоны корпоративных решений для Microsoft .NET).
 - Шаблоны интеграции рассматриваются в статье [Integration Patterns](#).
 - Безопасности Веб-сервисов посвящен материал [Web Service Security Guidance: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements \(WSE\) 3.0](#) (Руководство по безопасности Веб-сервисов: сценарии, шаблоны и руководство по реализации для Web Services Enhancements (WSE) 3.0).
-

РУКОВОДСТВО MICROSOFT® ПО ПРОЕКТИРОВАНИЮ АРХИТЕКТУРЫ ПРИЛОЖЕНИЙ 2е издание

Целевой аудиторией данного руководства являются разработчики ПО и архитекторы решений. Приведенные в нем рекомендации помогут проектировать и создавать эффективные высококачественные приложения на платформе Microsoft и .NET Framework в более сжатые сроки и с меньшими рисками. Здесь даются рекомендации по применению проверенных и подтвержденных принципов построения архитектуры, принципов и шаблонов проектирования. Руководство организовано по разделам соответственно основным аспектам архитектуры и дизайна и может использоваться и как справочник, и как учебный материал.

Данное руководство поможет Вам:

- Понимать базовые принципы построения архитектуры и дизайна, а также шаблоны для разработки успешных решений на платформе Microsoft и .NET Framework.
- Правильно выбирать стратегию и шаблоны проектирования, что облегчит проектирование слоев, компонентов и сервисов решения.
- Находить ключевые технические решения и правильно их реализовывать.
- Выявлять и реализовывать ключевые показатели качества и сквозную функциональность решения.
- Создавать возможные варианты базовой архитектуры решения.
- Правильно выбирать технологии для решения.
- Правильно выбирать решения и руководства группы *patterns & practices*, которые помогут в реализации собственных решений.

Несмотря на то, что данное руководство обеспечивает всестороннее рассмотрение вопросов, связанных с архитектурой и дизайном приложений, его нельзя считать полным и исчерпывающим исследованием в данной области. Это руководство является практическим обзором и удобным справочником по общим принципам архитектуры и дизайна приложения на платформе Microsoft и .NET Framework.

В частности, Вы не найдете в данном руководстве определенного или образцового архитектурного решения ни для одного конкретного сценария. Напротив, здесь предлагается краткий обзор принципов и шаблонов, следование которым обеспечит создание хорошей архитектуры и дизайна. Также здесь обозначены некоторые наиболее важные проблемы, на которые следует обратить внимание, и приведены рекомендации по их решению.

Поскольку основная масса приведенных здесь материалов основывается на общих принципах и не ориентирована ни на одну конкретную технологию, это руководство может применяться ко всем платформам приложений или технологиям. Однако чтобы помочь Вам выбрать лучшую технологию из всего существующего в настоящее время многообразия, в данное издание включены специальные рекомендации по технологиям Microsoft и .NET Framework и тому, как извлечь максимальную пользу от их применения в конкретных ситуациях.

patterns & practices

Проверенные практики –
гарантированные результаты

Сократите сроки и риски для проектов разработки ПО, используя прикладное техническое руководство группы *patterns & practices* Microsoft, которое включает и готовый к применению исходный код, и документацию.

Данное руководство поможет группам разработки:

В принятии ключевых решений по выбору дизайна и технологий. В нем предлагаются соответствующие архитектуры решений, технологии и продукты Microsoft для общих сценариев.

Понять ключевые принципы, которые обеспечивают достижение успеха. Здесь рассматриваются соответствующие шаблоны и предлагаются необходимые практики.

Начать работу с проверенного готового к использованию базового решения. Руководство предлагает протестированное ПО и исходный код, в котором реализованы рекомендации Microsoft.

Группа *patterns & practices* объединяет в себе опытных архитекторов, разработчиков, авторов и тестировщиков. В каждом нашем проекте мы активно сотрудничаем с сообществом разработчиков и специалистами отрасли, привлекая к работе над руководствами наилучшие умы современности.

Нам также нравится выступать в роли посредников и находить решения для реальных нужд наших потребителей в огромной номенклатуре продуктов и технологий Microsoft.

Нас можно найти по адресу:
msdn.microsoft.com/practices

Microsoft®

