**1.SOA:** определение, свойства, стандарты, спецификации, интерфейсы, специальные компоненты, способы клиент-серверного взаимодействия, платформы для разработки.

**SOA: Service-oriented architecture** – парадигма разработки программного обеспечения, основанная на применении распределенных слабосвязанных компонентов, обеспечивающих стандартные интерфейсы.

**SOA: основные свойства**

* независимость от аппаратной реализации узлов;
* независимость от ОС в узлах;
* независимость от языка программирования разработки сервиса;
* масштабируемость.

**SOA:** основные стандарты W3С: XML**,** SOAP, WSDL, UDDI**.**

**SOA:** спецификации второго уровня W3C (WS\*):

* WS-Policy,WS-PolicyAttachment,WS-PolicyAssertion (описание политик web-сервиса);
* WS-Addressing (механизм адресации web-cthdbcf);
* WS-Security (целостность и конфиденциальность web-сервисов);
* WS-Trust (механизм получения маркеров защиты);
* WS-SecureConversion (создание безопасной сессии обмена сообщениями);
* WS-SecurityPolicy (определяет набор утверждений политики безопасности);
* WS-Federation (объединение защищенных доменов);
* WS-Transfer (механизм обновления, создания и удаления ресурсов);
* WS-ResourceTransfer, WS-Fragment (обеспечивает частичный доступ к ресурсам);
* WS-MetadataExchange (механизм получения метаданных);
* WS-Enumeration (механизм получения данных больших размеров);
* WS-Eventing (механизм уведомления о событиях web-сервисов);
* WS-Management (SOAP-управление системами);
* WS-Discovery (механизмы публикации и поиска web-сервисов);
* WS-ReliableMessaging, WS-ReliableMessagingPolicy (механизмы надежной передачи сообщений между web-сервисами);
* WS-MakeConnection (установка соединения с сервисом не имеющего доступный адрес);
* WS-Coordination (механизмы взаимодействия web-сервисов);
* WS-AtomicTransaction (поддержка транзакций web-сервисов);
* WS-BusinessActivity (координация бизнес-взаимодействия web-сервисов).

**SOA: специальные компоненты** (как правило часть ESB)

* SOA Registry;
* Workflow Engine;
* Service Broker;
* SOA Supervisor.

**SOA Registry –** реестр сервисов, информация о сервисах и их интерфейсах.



**Workflow Engine –** программный компонент, предназначенный для построения модели бизнес процесса и выполнения бизнес процесса на основе группы сервисов. Другими словами, разработать новый сервис на основе последовательного выполнения нескольких сервисов.



**Service broker** –программный компонент, позволяющий трансформировать запрос пользователя в системе в запуск и координированную работу.



**SOA Supervisor** – служебный сервис, предназначенный для управления и мониторинга других служебных сервисов.

**SOA:** интерфейсы – REST, SOAP, JSON-RPC

**SOA:** способы клиент-серверного взаимодействия







**SOA:** Software AG webMethods OneData, Oracle SOA Suite 12C, Microsoft WCF.

1. **REST API:** определение, форматы передачи данных, HATEOAS, общепринятые правила REST API, платформы для разработки сервисов.

**REST**: Архитектурный стиль взаимодействия компонентов распределенного приложения.

**REST:** альтернатива RPC.

**REST**: форматы передачи данных – xml и json

**REST**: шесть обязательных ограничений:

* модель клиент-сервер;
* отсутствие состояния на стороне сервера, допускается сохранение состояния на клиенте или в другом сервисе (например, в БД);
* кэширование на стороне клиента, сервер явно управляет кэшированием;
* единообразие интерфейсов (идентификация ресурсов, манипуляция ресурсами через представления, самодостаточные сообщения, HATEOAS);
* код по требованию: допускается (необязательно) выгрузка на клиент апплетов или сценариев для расширения его функциональности.

**HATEOAS:** гипермедиа в качестве управления состоянием.

**Гипермедиа**: технология обработки, структурирования информации и произвольного доступа к ее элементам с помощью гиперсвязей.

**REST:** **общепринятые правила**

* Общий префикс для всех ресурсов сервиса
* Два типа ресурсов: коллекция, элемент коллекции
* Иерархическая связь
* Ограничить количество HTTP-статусов
* Версионность
* Постраничное получение данных
* Сортировка: параметр sort
* Все фильтры вынести за знак вопроса
* Обозначать в запросе формат сообщений (желательна поддержка нескольких форматов)

Платформы для разработки сервисов:

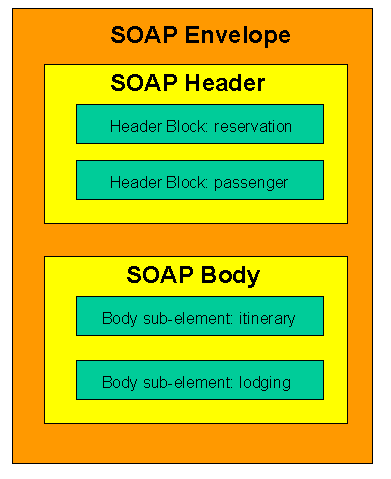
* на базе ASP.NET HTTP Handler
* на базе ASP.NET MVC WEB API

1. **SOAP:** определение, структура Envelop-сообщений, пространства имен, роли, принципы расширения.

**SOAP:** протокол обмена структурированными сообщениями в распределённой вычислительной среде.

* аббревиатура не расшифровывается;
* основывается на XML (предназначен для пересылки XML-сообщений);
* транспорт: HTTP, SMTP, FTP;
* не зависит о платформы.

**SOAP:** формат сообщений:



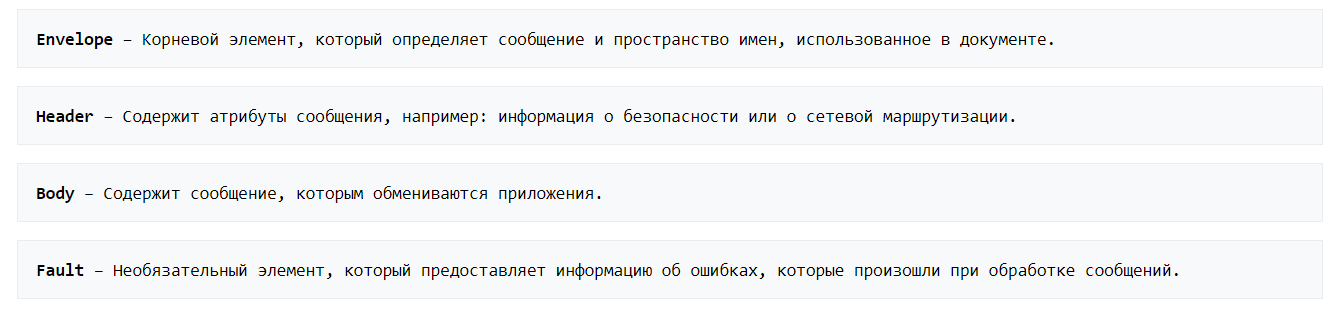
**SOAP:** стандартные пространства имен: env, enc, rpc, rep, xop, xmime

**SOAP:** узел – компонент SOA-архитектуры, который может отправлять или получать SOAP-сообщения.

**SOAP:** роль - набор правил, определяющих поведение узла, задается атрибутом env:role.

**SOAP:** тристандартные роли для узла (нет роли, промежуточный узел, конечный получатель), роль может быть пользовательской (задаваться собственным URI)

**SOAP:** структура

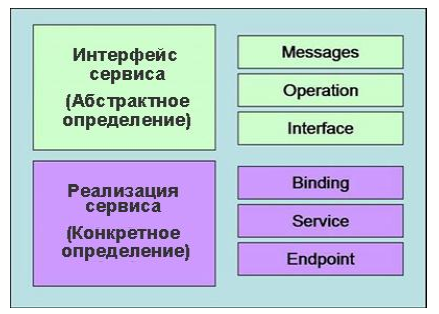


1. **WSDL:** определение, стандарты и версии, концептуальная модель, пространства имен, структура WSDL-документа.

**WSDL:** средство XML-описания интерфейса для доступа к web-службе; платформонезависимое описание.

**Стандарты и версии:** W3C *wsdl 1.1*, текущая версия 2.0

**WSDL: концептуальная модель WSDL 2.0**

****

**WSDL:** пространства имен: wsdl, wsdi, wsdx, wrpc, wsoap, whttp, xs, xsi

<!-- структура wsdl-документа -->

<definition>

<types>

<!-- оперделение типов, используемых web-сервисом -->

<!-- для пл.-независимого WSDL используется синтаксис XML Schema -->

</types>

<message>

<!-- сообщения, используемые web-сервисом -->

<!-- сообщений, может быть несколько -->

<!-- каждое сообщение может состоять из нескольких частей -->

</message>

<portType>

<!-- методы, предоставляемые web-сервисом -->

<!-- может быть несколько портов -->

<!-- определены операции web-сервиса и используемые сообщения -->

</portType>

<binding>

<!-- протоколы связи, используемые web-сервисом -->

<!-- форматы сообщений и детали протокола для каждого порта -->

</binding>

<service>

<!— набор портов связанных с сервисом -->

</service>

</definition>

1. **ASMX:** определение ASMX-сервиса, порядок разработки, принципы применения, утилита WSDL.EXE.

**ASMX: Active Server Method Extended;** технология Microsoft для разработки web-сервисов, основанная на XML, WSDL, SOAP;

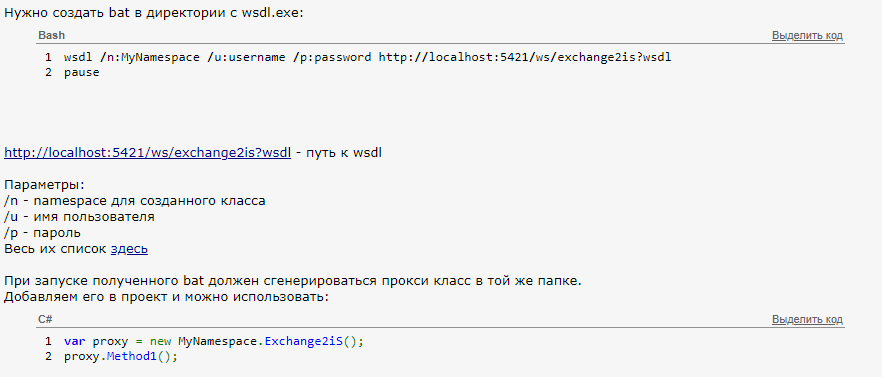
* В качестве хоста только IIS
* Модель взаимодействия только полудуплекс
* Имеет только 1 конечную точку
* Транспорт только HTTP

**Веб-сервис** — это веб-приложение, предоставляющее открытый (значит опубликованный) интерфейс, пригодный для использования другими приложениями и интерфейсами.

**Разработка ASMX сервиса:**

1. В Visual Studio добавить к проекту ASMX-службу.
2. В WebService(): Добавить пространство имен(Namespace) и описание(Description).
3. В WebMethod(): описание методов(Description), описание возвращаемого результата, добавить имя сообщения в WSDL(MessageName).
4. Добавить proxy-класс на клиент.

**Утилита WSDL.exe:**



1. **WCF:** определение WCF-сервиса, коммуникационная модель, WCF-контракты, WCF-хостинг, конечные точки, стандартные привязки, основные отличия от ASMX-сервисов, поведение и безопасность WCF-сервиса, порядок разработки WCF/RPC и WCF/REST-сервисов, разработка WCF-сервиса с несколькими конечными точками.

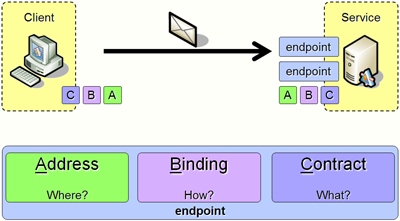
**WCF:** Windows Communication Foundation; технология основанная на .NET FRAMEWORK для разработки приложений SOA-архитектуры.

**Веб-сервис** — это веб-приложение, предоставляющее открытый (значит опубликованный) интерфейс, пригодный для использования другими приложениями и интерфейсами.

**WCF:** основные принципы

* разработка сервиса должна быть простой и иметь способность к расширению;
* один API для всех коммуникационных протоколов;
* сервис должен функционировать по отрытым телекоммуникационным стандартам;
* сервис должен поддерживать стандарты WS-\*;
* сервис должен поддерживать REST, RPC и др. архитектуры;

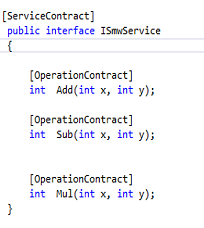
**WCF:** коммуникация модель



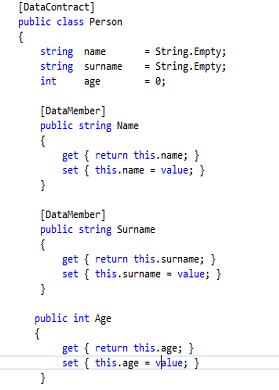
**WCF:** контракты

* контракт службы;
* контракт данных;
* контракт сообщений.

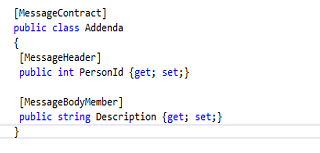
**WCF:** контракт службы

****

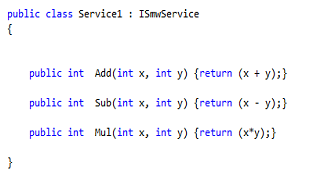
**WCF:** контракт данных – указывает каким образом данные будут сериализованы.



**WCF:** контракт сообщений

****

**WCF:** реализация сервиса

****

**WCF:** хост **–** контейнер для сервиса (любое С#-приложение). Основное назначение хоста: присоединить WCF-инфраструктуру, создать список конечных точек.

**WCF:** конечная точка – адрес, привязка, контракт.

**WCF:** привязка – инкапсулирует все технологии, позволяющие переправить сообщение от одной конечной точке к другой.



**WCF:** привязка: безопасность:

* WS-ReliableMessaging
* WS-Security

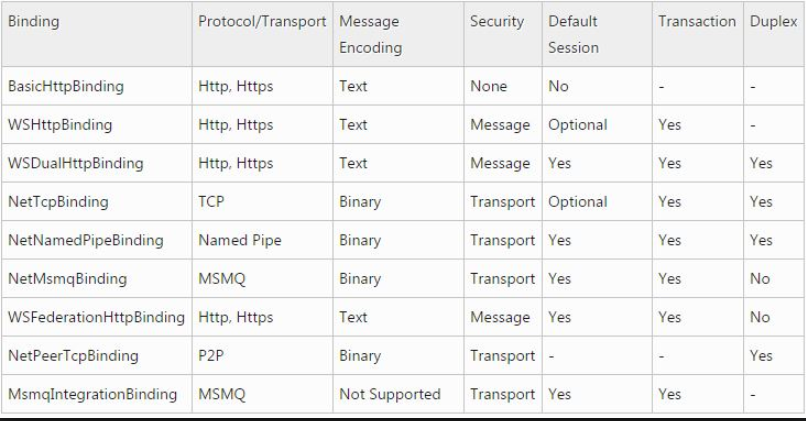
**WCF:** привязка: кодирование:

* текст (ASCII, UTF-8, UTF-16);
* двоичная (проприетарные алгоритмы);
* МТОМ.

**WCF:** привязка: транспорт:

* HTTP;
* TCP;
* IPC Named Pipes;
* MSMQ;
* Custom.

**WCF:** привязка: стандартные:

****

**Отличия asmx-сервиса и wcf-сервиса:**

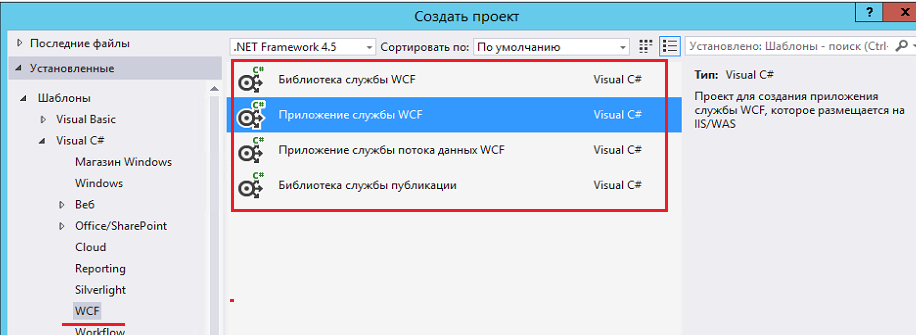
1. asmx-сервис работает только по протоколу http в качестве транспорта, wcf может работать как по http, так и по tcp, named pipe, msmq.

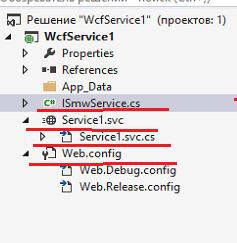
2. asmx-сервис можно размещать только на iis-сервере, у wcf в качестве хоста может выступать любое консольное или графическое .NET-приложение, Windows-служба или IIS

3. asmx-сервис может использовать только одну конечную точку, wcf может использовать по одной конечной точке на каждый используемый протокол передачи данных.

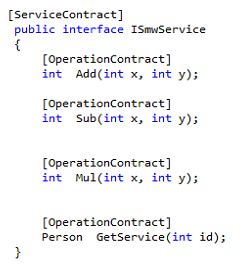
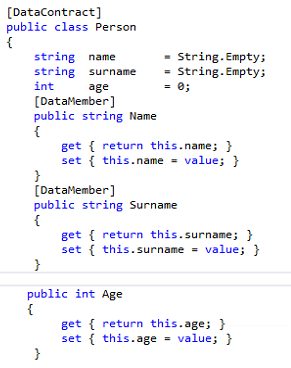
4. asmx-сервис работает только по полудуплексной модели взаимодействия, wcf – по полудуплексной, дуплексной, однонаправленной, потоковой и издатель-подписчик.

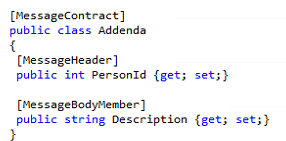
**Разработка WCF RPC:**

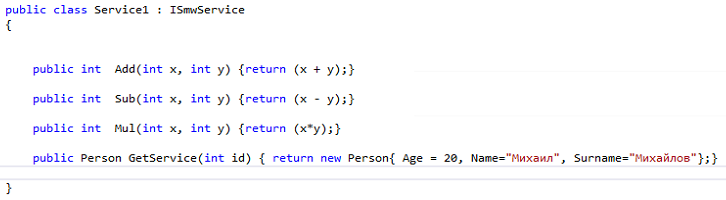
****

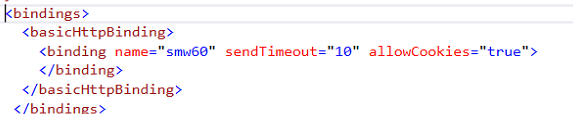
****

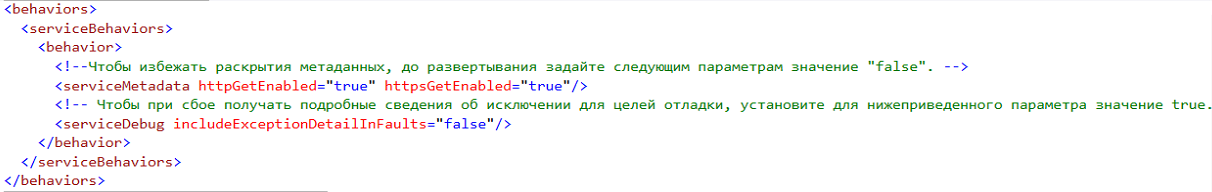


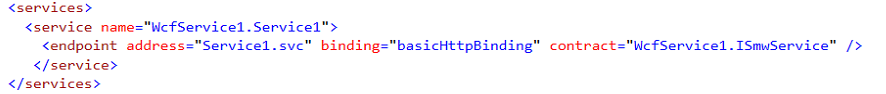
** **

****

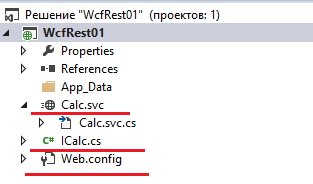
****

****

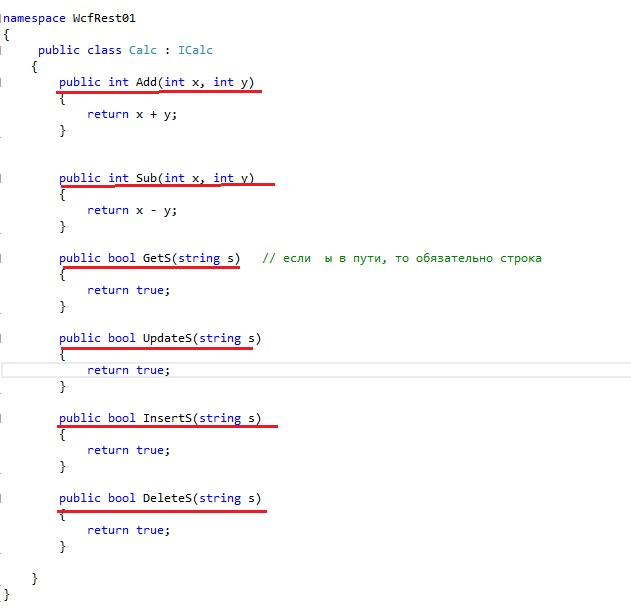
****

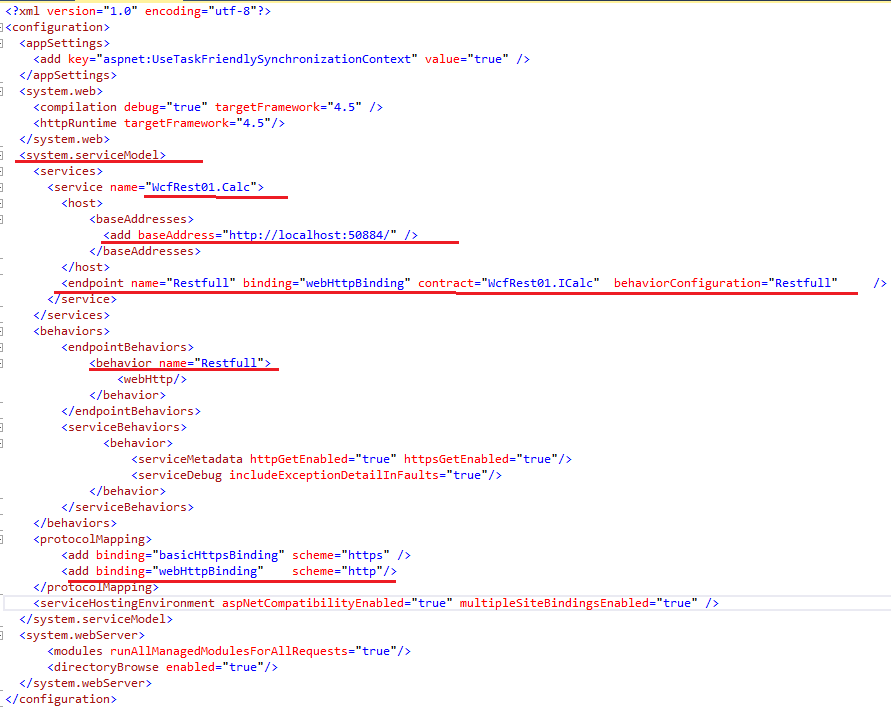
****

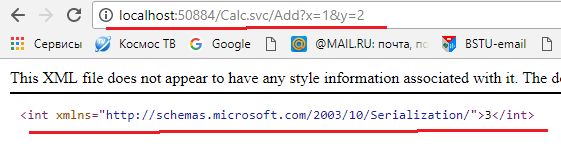
**WCF/REST:** разработка WCF-службы с REST-интерфейсом





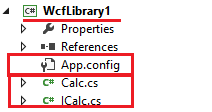




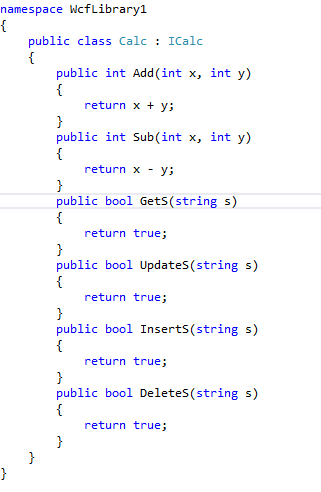


**WCF-сервис с несколькими конечными точками**

**WCF:** создать приложение ***библиотека службы WCF***

****

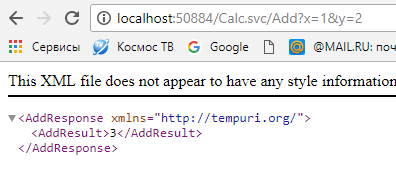
**WCF:** контракт и реализация сервиса

****

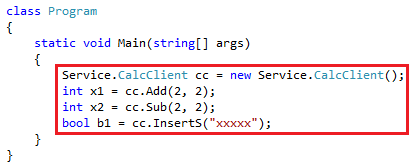
**WCF:** конфигурационный файл сервера

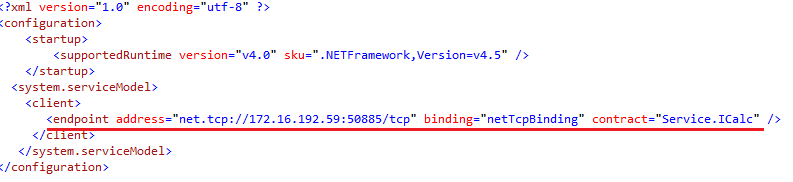


**WCF:** REST-клиент

****

**WCF:** WCF/TCP-клиент

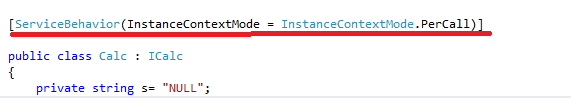
****

****

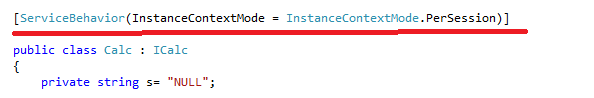
**Поведение WCF-сервиса**

**WCF: Service Behavior/Instance Context Mode –** определяет когда объект сервиса будет создан

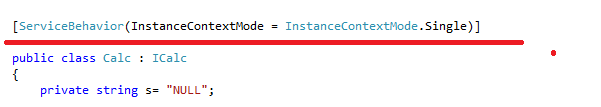
**WCF: Per Call**

****

**WCF: Per Session**

****

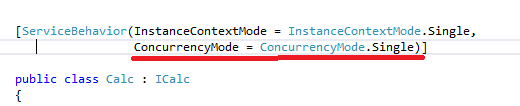
**WCF: Single**

****

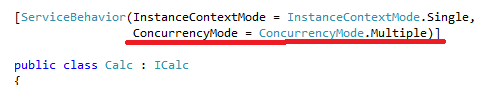
**WCF:** поведение сервиса Per Session необходимо сопоставлять с binding

**WCF: Service Behavior/Concurrency Mode -** получает или задает, поддерживает ли служба один поток, несколько потоков или повторные входящие вызовы.

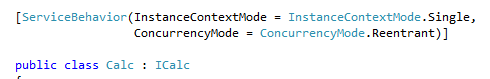
**WCF: Single**

****

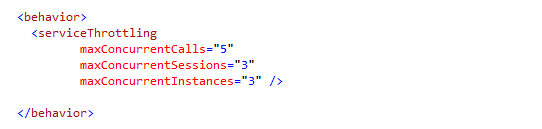
**WCF: Multiple**

****

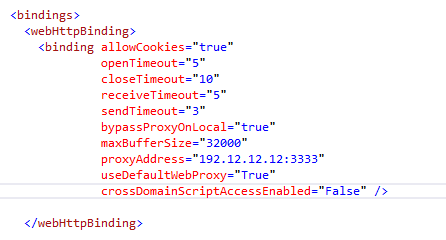
**WCF: Reentrant**

****

**WCF: Service Behavior/serviceThrottling -** Определяет механизм регулирования службы Windows Communication Foundation (WCF)

****

**WCF: Service Behavior/ binding**

****

**Безопасность WCF-сервиса**

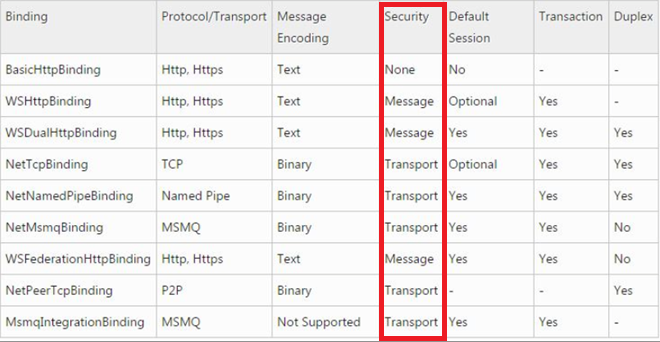
**WCF:** Аутентификация, авторизация, целостность сообщений, конфиденциальность сообщений

**WCF:** Безопасность транспорта (TLS/SSL)

**WCF:** Безопасность сообщений, спецификация WS-Security (цифровая подпись и шифрование фрагментов конверта)

**WCF:** Гибридная модель безопасности.

**WCF: binding/Security**

****

****

**WCF: message-аутентификация:** None**,** Windows (NTLM, Kerberos), Username(имя/пароль), Certificate (X509), IssueToken (служба токенов третьей стороны).

**WCF: transport-аутентификация:** None, Windows (NTLM, Kerberos), Certificate (X509),Basic (протокол Http Basic Authentication RFC 7615), Digest (Http Digest Access Authentication RFC 7616),NTLM (NT LAN Manager, Microsoft Windows NT)

1. **WCF Syndication Services:** стандарты ATOM, RSS, порядок разработки WCF Syndication Service.

**RSS:** семейство XML-форматов, предназначенного для описания новостных лент, анонсов и статей (199).

**RSS:** MIME: application/rss+xml.

**RSS:** пример

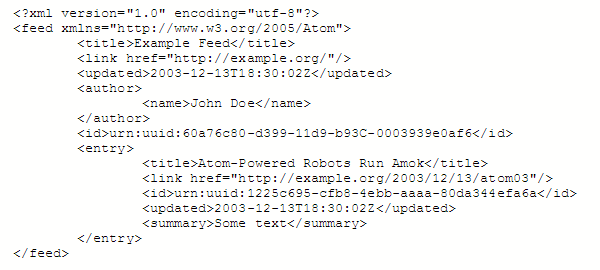
****

**ATOM**: формат описания web-ресурсов и протокола для их публикации (2005г.).

**ATOM**: последняя версия 1.0

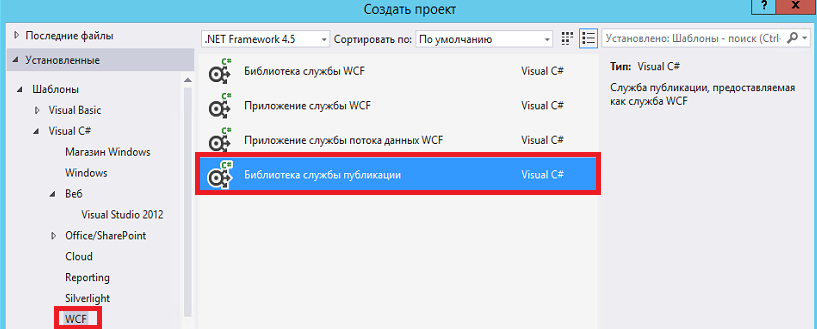
**ATOM**: MIME: application/atom+xml.

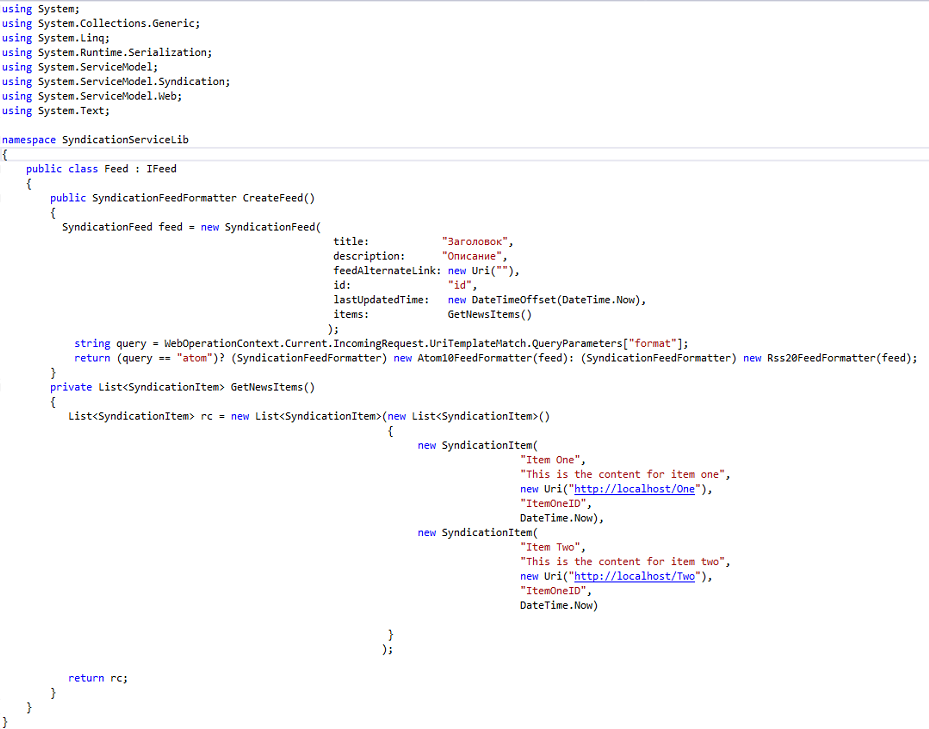
**ATOM**: примеры

****

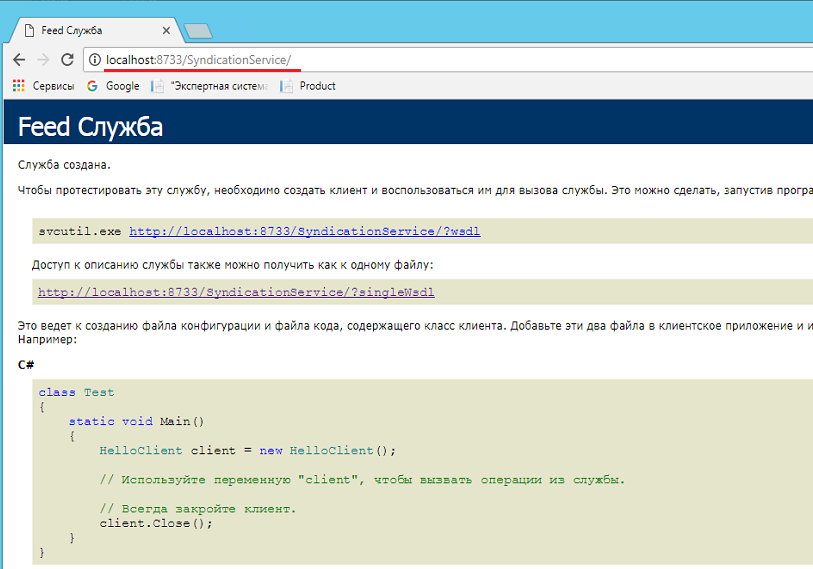
**ATOM/RSS:** ATOM более универсальный и чаще применяется; следует использовать его при разработке новых приложений.

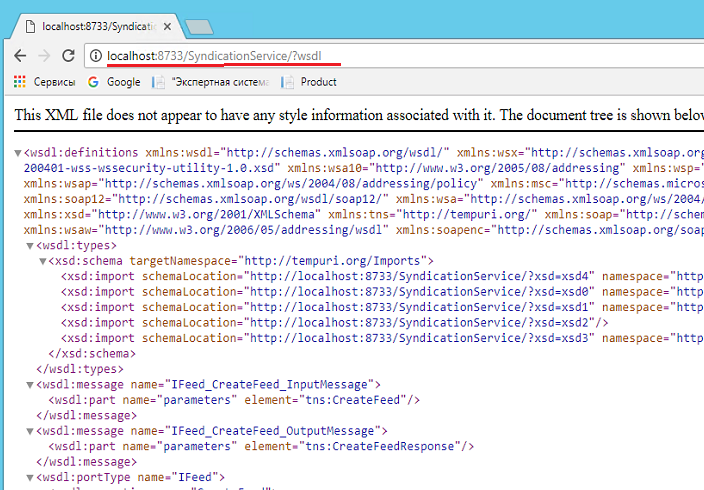
**ATOM/RSS:** реализация на платформе WCF .NET

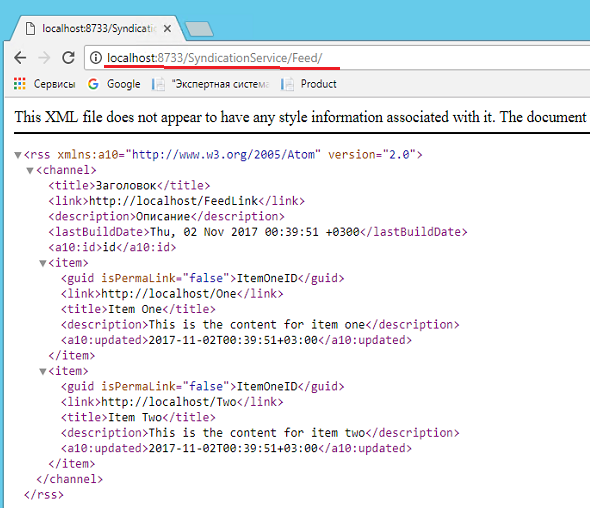
****

****

****

****

****

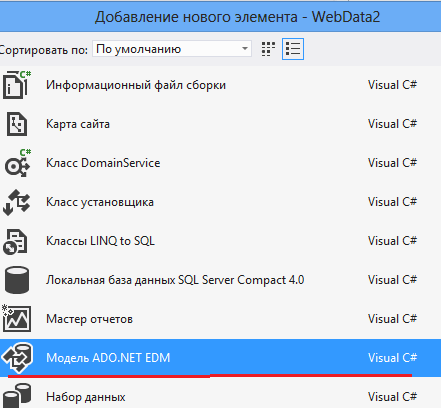
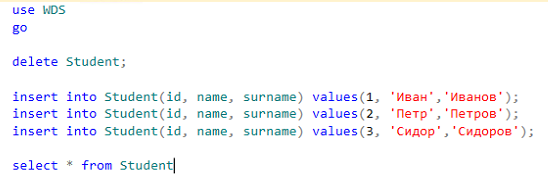
****

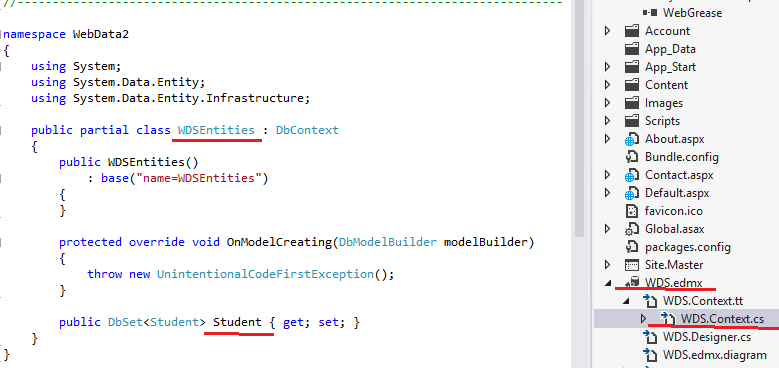
1. **WCF Data Services:** протокол Open Data Protocol, возможности предоставляемые OData-интерфейсом, порядок разработки Data Services, применение Data Services.

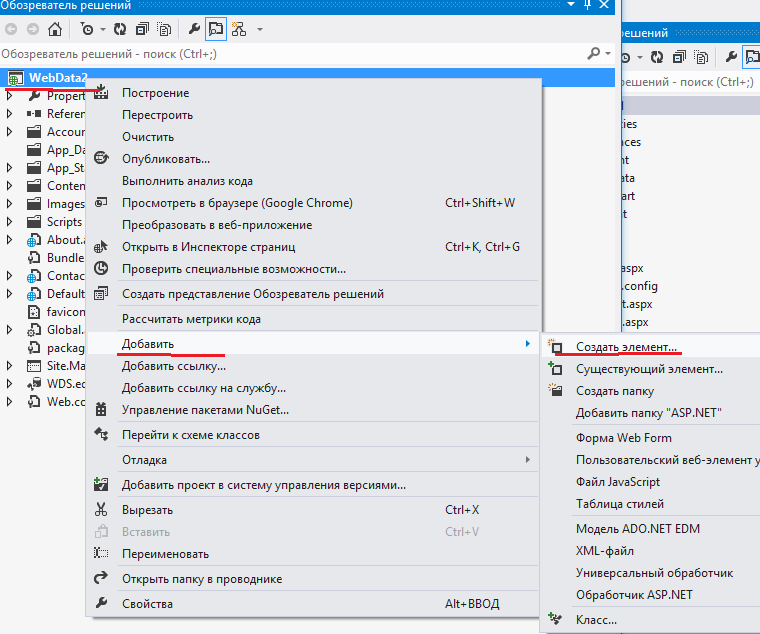
**OData: Open Data Protocol –** открытый web-протокол; позволяет выполнять операции с ресурсами и получать ответы в форматах XML, JSON.

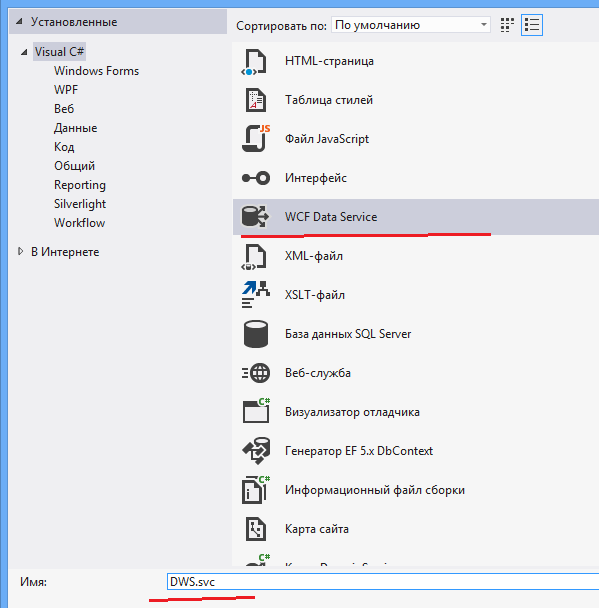
**OData:** поддерживается OASIS.

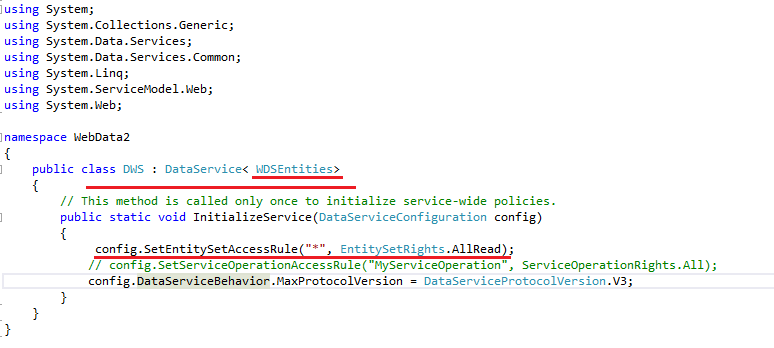
**WCF DS: ASP.NET**

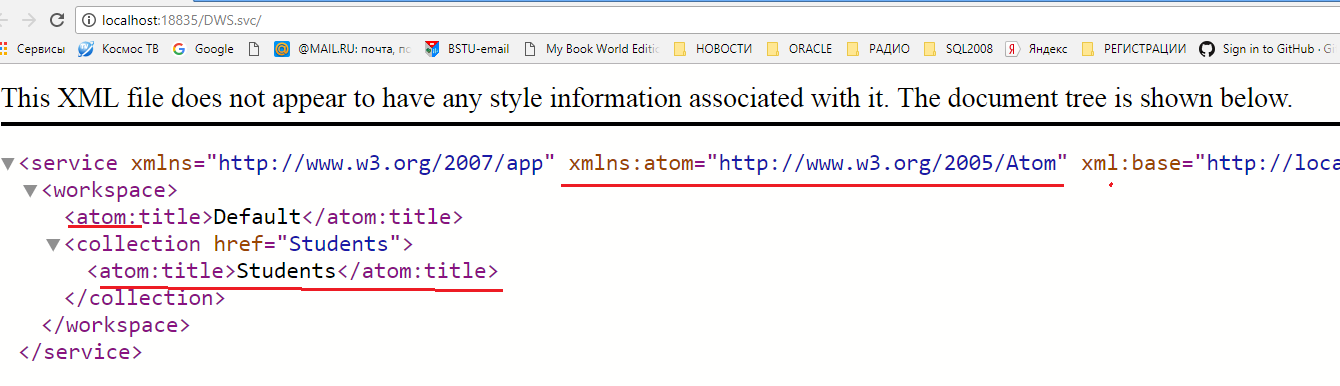
****

****

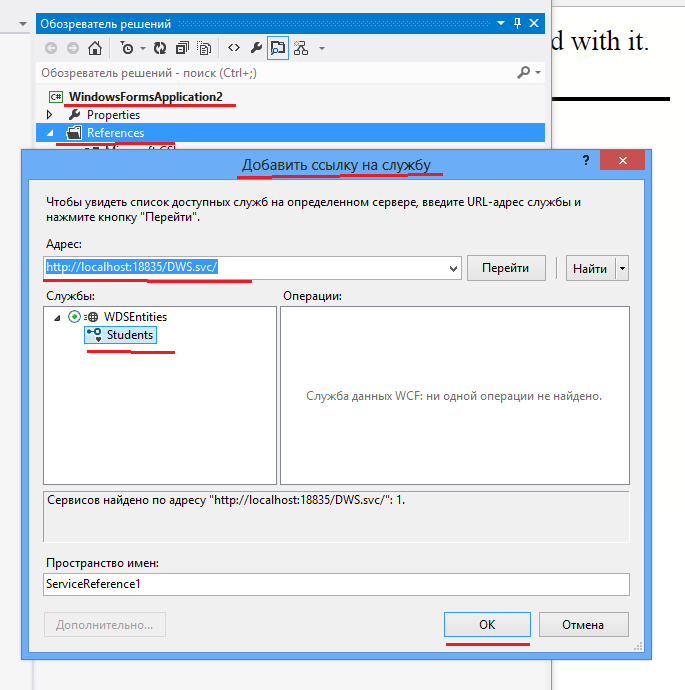
****

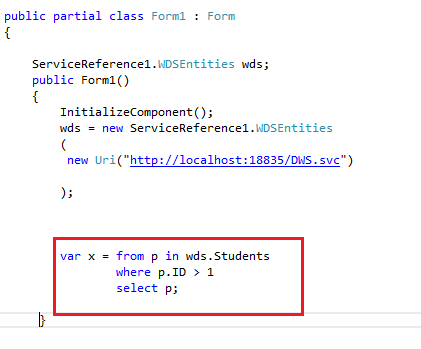
****

****

****

**WCF DS: клиент**

****

****

1. **JSON-RPC:**  определение JSON-RPC-сервиса, форматы запросов и ответов, обработка ошибок, пакеты запросов, реализация JSON-RPC на платформе Web API.

**JSON-RPC:** протокол удаленного вызова процедур, использующий формат JSON для передачи сообщений.

**JSON-RPC:** последняя версия 2.0.

**JSON-RPC:** объект запроса

{“jsonrpc”, “method”, “params”, “id”}

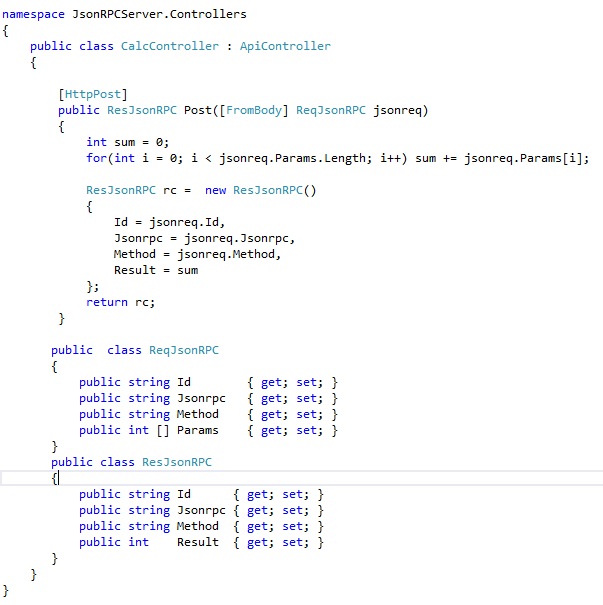
**JSON-RPC:** объект ответа

{“jsonrpc”, “result”, “method”, “id”}

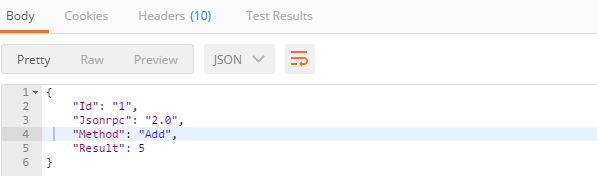
**JSON-RPC:** объект ответа (ошибка)

{“jsonrpc”,“error”{“message”,“data”,“code”},'id'}

**JSON-RPC:** начало примера

****

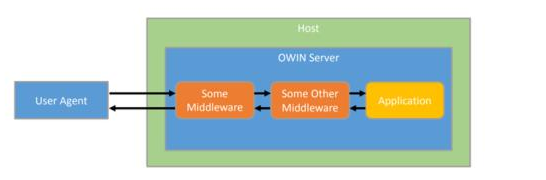
****

****

1. **ASP.NET CORE Nancy:** интерфейс OWIN,архитектура приложения, принцип разработки сервиса.

**Nancy:** Framework для разработки self-hosting приложений в .NET и Core.

**OWIN** – интерфейс между net-web-сервером и серверным приложением (обработчиками запросов). Основная цель OWIN отделить web-сервер от серверного приложения. Можно разрабатывать отдельно сервер и приложение.



**Host** – приложение-процесс операционной системы, управляющий жизненным циклом OWIN Server.

**OWIN Server** - http-сервер, реализующий интерфейс OWIN.

**OWIN:** обеспечивает интерфейсы: между приложением и http-сервером, между http-сервером и Host, между http-сервером и middleware.

**Middleware** – подключенные компоненты (модули), предназначенные для обработки запросов

1. **Event Storing:** назначение, принципы применения, примеры реализации.

*(Сказать о новом подходе который существует сейчас связанном с микросервисами каждый сервис должен быть изолированный и иметь свою. Сервисы которые заполняют бд и которые читают)*

DDD: **Domain-Driven Design** - набор принципов направленных на создание систем объектов со сложной предметной областью.

цель – ускорить процесс проектирование программного обеспечения.

DDD: система имеет 4 уровня:

1. interface (пользовательское взаимодействие, взаимодействие с другими системами)
2. application (реализация, технические аспекты приложения, хранение данных, создание данных)
3. business (реализация доменного языка)
4. infrastructure (платформа и дополнительное программное обеспечение).

**Command-Query responsibility segregation(CQRS)**

CQRS: разделение ответственности на команды и запросы;

обычно 2 базы данных: для записи и для чтения. Базы данных синхронизируются.

**Event Sourcing (ES)**

ES: подход к хранению данных, при котором вместо конечного результата хранится череда событий произошедших с некоторой сущностью.

ES: каждое событие имеет имя.

ES: произошедшие события неизменны.

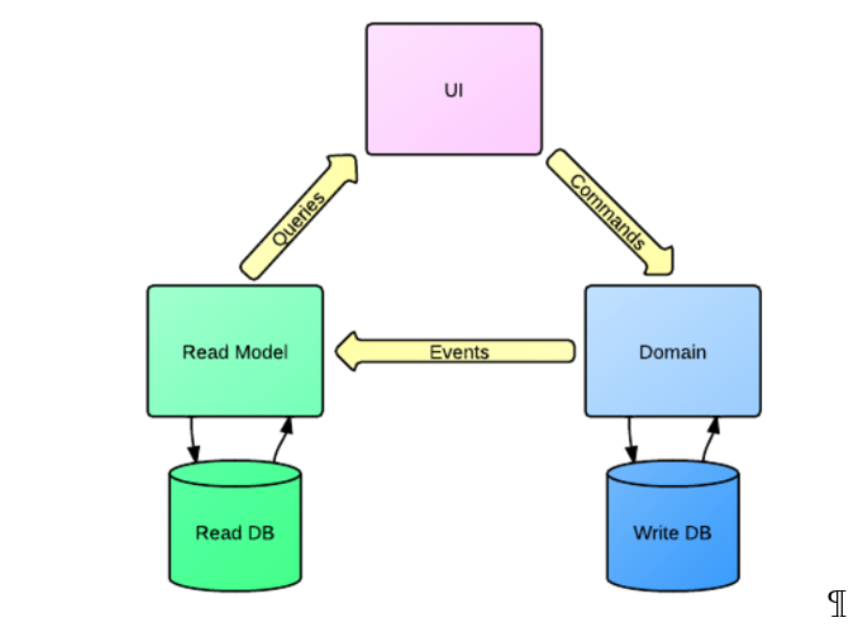
ES: проекция - вычисленные результаты для UI (аналог view).

ES: ориентация на процесс (workflow driven), а не на данные (data-input based).

ES: обычно применяется в архитектуре CQRS для реализации WriteDB.

EventStore: функциональная база данных с комплексной обработкой событий в JavaScript.

НАПРИМЕР ЕСТЬ РЕДАКТОР ТЕКСТА И CTRL+Z и CTRL+Y



1. Микросервисы:микросервисная архитектура, определение микросервиса, основные принципы разработки микросервиса, паттерны разработки, DevOps для микросервисов, оркестровка и хореография микросервисов.

**Микросервисы:** один из подходов к разработке SOA-приложений.

**Микросервисы:** цель:

1) устойчивость к сбоям;

2) облегчить понимание и поддержку кода;

3) усилить работу команды программистов.

**Микросервисы:** *микросервис* – сервис, выполняющий одну элементарную функцию; основной принцип разбиения – изменение сервиса не затрагивает другие сервисы.

**Микросервисы:** *микросервисная архитектура* -  набор принципов, которым должны соответствовать сервисы. Задается или в форме правил или основывается на применении готовых фреймворков (например, Karyon, Dropwiard,…).

*Основные принципы разработки микросервиса:*

* Легкость сервисов (сервисы маленькие и протоколы легковесные). Легкий сервис – сервис, который может быть переписан за 2 недели (Real Estate).
* Принцип разбиения – изменение сервиса не затрагивает другие сервисы.
* Правильное распределение функциональности
* ESB – не является приемлемым решением. Обычно применяется HTTP/REST, HTTP+HATEOAS, XML/JSON.
* Обеспечение устойчивости к сбоям, большего понимания кода, облегчения работы программиста.
* Распределенные процессы - каждый модуль в отдельном удаленном процессе.
* Применения микросервисной архитектуры.
* Не применяются синхронные вызовы.
* Распределенное управление данными.

*Популярные паттерны разработки:*

* Tolerant Reader - приложение будет получать информацию и обрабатывать только то, что будет использоваться, независимо от неожиданной дополнительной информации.
* Consumer Driver - позволяет предупредить коллизии и улучшить понимание при взаимодействии двух связанных компонентов(сервисов)

*DevOps для микросервисов:*

Требуется DevOps (Development & Operation)- набор технологий, нацеленных на интеграцию процессов разработки и информационно-техническое обслуживание.

*Цели DevOps*:

* сокращение выхода продукта на рынок
* снижение частоты отказов релизов
* сокращение времени на изменения
* сокращение времени на восстановление. Задача DevOps сделать согласованным процесс разработки и эксплуатации приложений.

*Основные задачи DevOps:*

* контроль версий
* непрерывная сборка
* непрерывное тестирование
* поддержка репозиториев артефактов
* конфигурация инфраструктуры
* мониторинг работоспособности и производительности.

 DevOps – следствие увеличения релизов (Agile-технологии) и усложнения инфраструктуры.

Автоматизация, продукты DevOps: Continuous Delivery, Continuous Integration.

*Оркестровка и хореография микросервисов:*

клиенты работающие с микросервисом могут быть реализованы с помощью оркестрового (прямые вызовы сервисов) или хореографического (сервисы подписываются на события клиента) принципов.