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

**Service-oriented architecture** – парадигма разработки программного обеспечения, основанная на применении распределенных слабосвязанных компонентов, обеспечивающих стандартные интерфейсы. Компоненты распределенной системы SOA – узлы - сервисы.

**Распределенное приложение** состоит из распределенных компонентов (узел - сервис).

**Сервис** - приложение, которое предоставляет сетевой интерфейс/API для других приложений)

**Основные свойства сервиса:**

* ориентирован на бизнес;
* автономен;
* повторное использование;
* четко описаны инструкции в терминах интерфейса (SLA);
* виден, доступен.

**Основные свойства SOA:**

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

**Плюсы SOA:**

* снижение затрат на разработку ПО;
* снижение затрат на владение;
* масштабирование.

**Основные стандарты W3С**: XML**,** SOAP, WSDL, UDDI.

**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-MakeCpnnection (установка соединения с сервисом не имеющего доступный адрес);

- WS-Coordination (механизмы взаимодействия web-сервисов);

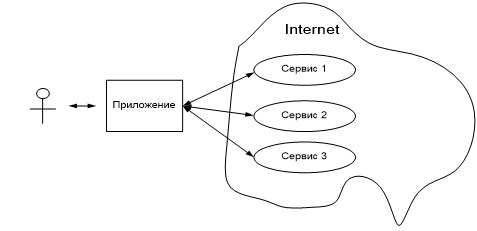
- WS-AtomicTransaction (поддержка транзакций web-сервисов);

- WS-BusinessActivity (координация бизнес-взаимодействия web-сервисов).

**Интерфейсы:** REST, SOAP, JSON-RPC.

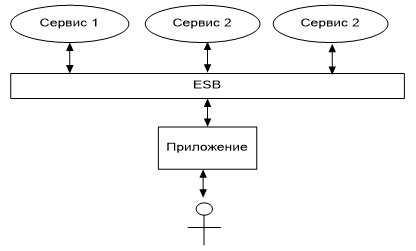
**SLA - System Level Agreement** - формальный документ, описывающий перечень услуг, обязательства, уровень качества предоставляемых услуг:

* сроки действия соглашения;
* дни и часы действия сервиса;
* правила безопасности;
* гарантированные характеристики (среднее время отклика и тд) и тд;



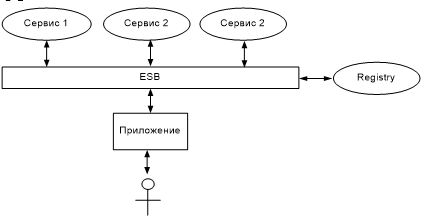
**ESB - Enterprise Service Bus** - программный компонент обеспечивающий обмен сообщениями между различными информационными системами, имеющих SOA архитектуру:

* синхронный и асинхронный вызов сервисов;
* гарантированная доставка сообщений;
* поддержка транзакций;
* маршрутизация сообщений;
* мониторинг, аудит, протоколирование.

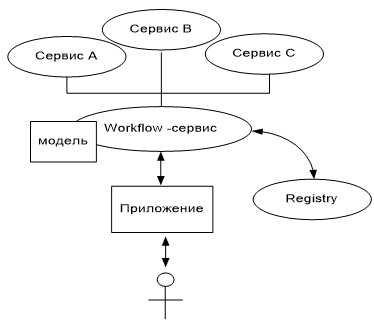


**Специальные компоненты** (как правило часть ESB):

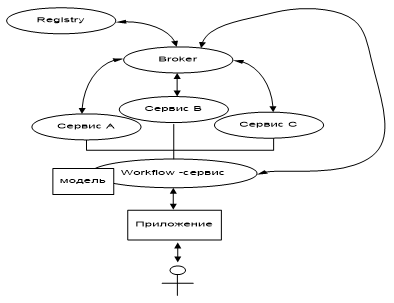
* **SOA Registry** - реестр сервисов, информация о сервисах и их интерфейсах.



* **Workflow Engine** - программный компонент, предназначенный разработки модели бизнес процесса и ее выполнения на основе последовательного выполнения нескольких сервисов.



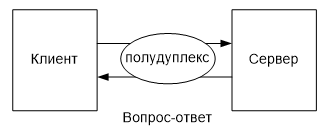
* **Service Broker** - программный компонент, позволяющий трансформировать запрос пользователя в запуск и скоординированную работу.

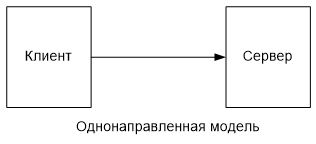


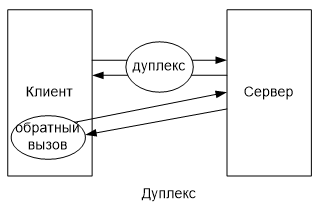
* **SOA Supervisor** - служебный сервис, предназначенный для управления и мониторинга других служебный сервисов.

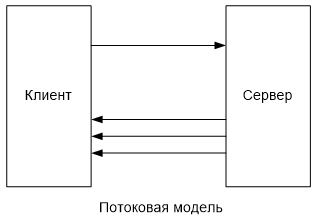
**Identity Service** - сервис аутентификации.

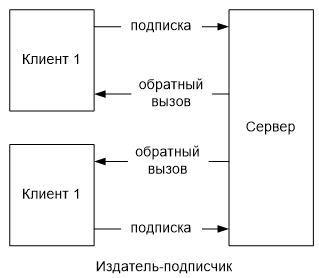
**Способы клиент-серверного взаимодействия**:











**Платформы для разработки:** AWS, Azure, Microsoft WCF, Oracle

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

**Representational State Transfer** - передача состояния представления.

**REST** - архитектурный стиль взаимодействия компонентов распределенного приложения.

Нет официального стандарта, но REST использует стандарты HTTP, URL/URI, XML, JSON.

Два основныхтипа ресурса – коллекции и элемент коллекции\

**Шесть обязательных ограничений:**

- модель клиент-сервер;

- отсутствие состояния на стороне сервера, сохранение состояния допускается на стороне клиента, допускается сохранение состояния 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);\

- слоистая система

- для клиента сервер должен предоставлять конечным;

- код по требованию: допускается (необязательно) выгрузка на клиент апплетов или сценариев для расширения его функциональности.

**Форматы передачи данных:** XML, JSON

**HATEOAS** – гипермедиа в качестве управления состоянием - технология обработки, структурирования информации и произвольного доступа к ее элементам с помощью гиперсвязей, WWW – реализация гипермедиа.

Платформы для разработки сервисов: Web API, Servlet, HTTP Handler,

**недостатки**: клиент и сервер должны быть включены, ограниченность методами HTTP, нэйминг для сложных

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

**SOAP - Simple Object Access Protocol** – простой протокол доступа к объекту. - сетевой протокол прикладного уровня

Транспорт: HTTP (весь Envelop перемещается в теле, Content-Type:application/soap+xml), TCP (сами должны придумать первое сообщение, в котором будет указано что теперь пойдет Soap сообщение)

Первоначально - реализация RPC, расширение протокола XML-RPC, позже - произвольный обмен XML-сообщениями.

Спецификация поддерживается W3C, последняя версия 1.2, в версии 1.2. аббревиатура SOAP не расшифровывается.

**СТРУКТУРА ENVELOPE-СООБЩЕНИЙ:**

Сообщение SOAP - это простой XML-документ, который состоит:

* ***Envelope*** - обязательный элемент, который определяет начало и конец сообщения. Благодаря этому элементу, клиент, принимающий сообщение, знает, когда сообщение полностью получено.
* ***Header*** - опциональный элемент, который может включать в себя любые опциональные атрибуты сообщения. Дает возможность определять дополнительные свойства приложения. Можно использовать несколько раз в файле.

Атрибуты:

* Actor - клиент может определить получателя элемента header.
* MustUnderstand - указывает является ли этот элемент обязательным или опциональным
* ***Body*** - обязательный элемент, который включает в себя данные XML, которые определяют сообщение, которое должны быть отправлено
* ***Fault*** - опциональный элемент, который предоставляет информацию об ошибках, которые возникают в процессе обработки сообщения
  + env: Сode – код сообщения;
  + env: Reason – пояснение к коду;
  + env: Node – узел назначения;
  + env: Role – роль назначения;
  + env: Detail – детальная информация.

Внутри каждого элемента Envelope должен быть один элемент Body.

Элемент Envelope определяется с помощью **пространства имён ENV** и элемента Enveloper.



**Главное свойство (принцип расширения)**: может расширятся за счет namespace (на базе soap можно строить свои собственные протоколы)

***Пространство имён в XML*** — это стандарт, описывающий именованную группу имен элементов и атрибутов, служащую для обеспечения их уникальности в XML-документе.

Пространства имен XML играют важную роль в сообщениях SOAP.

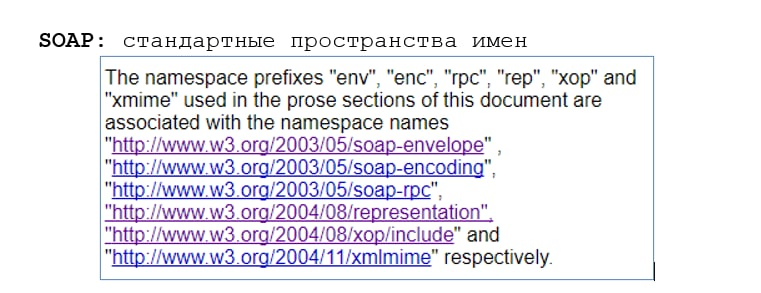
SOAP сообщение может включать в себя несколько различных XML - элементов в Header и Body, и, чтобы избежать столкновения имен, каждый из этих элементов должно быть идентифицировано уникальным пространством имен.

Использование пространств имен XML делает SOAP гибким и расширяемым протоколом.

Envelope, определяет пространство имен стандартных элементов SOAP - Envelope, Header и Body.

Каждый блок заголовка в элементе Header должен иметь собственное пространство имен. Это особенно важно, поскольку пространства имен помогают приложениям SOAP идентифицировать блоки заголовков и обрабатывать их отдельно.

Реальная сила пространств имен XML выходит за рамки простого предотвращения конфликтов имен, а также правильного управления версиями и обработкой. Использование полных имен для SOAP и данных приложения сообщает получателю SOAP, как обрабатывать сообщение, и какие схемы XML применять для проверки его содержимого. Например, различия в конкретной версии блока заголовка могут повлиять на то, как получатель обрабатывает сообщения, поэтому определение версии блока заголовка по его пространству имен позволяет получателю переключать модели обработки или отклонять сообщения, если он не поддерживает указанную версию.



**Роль** - это набор правил, определяющих поведение узла, задается атрибутом env:role.

Три стандартные роли для узла:

* нет роли
* промежуточный узел
* конечный получатель

Роль может быть пользовательской, задаваться собственным URI

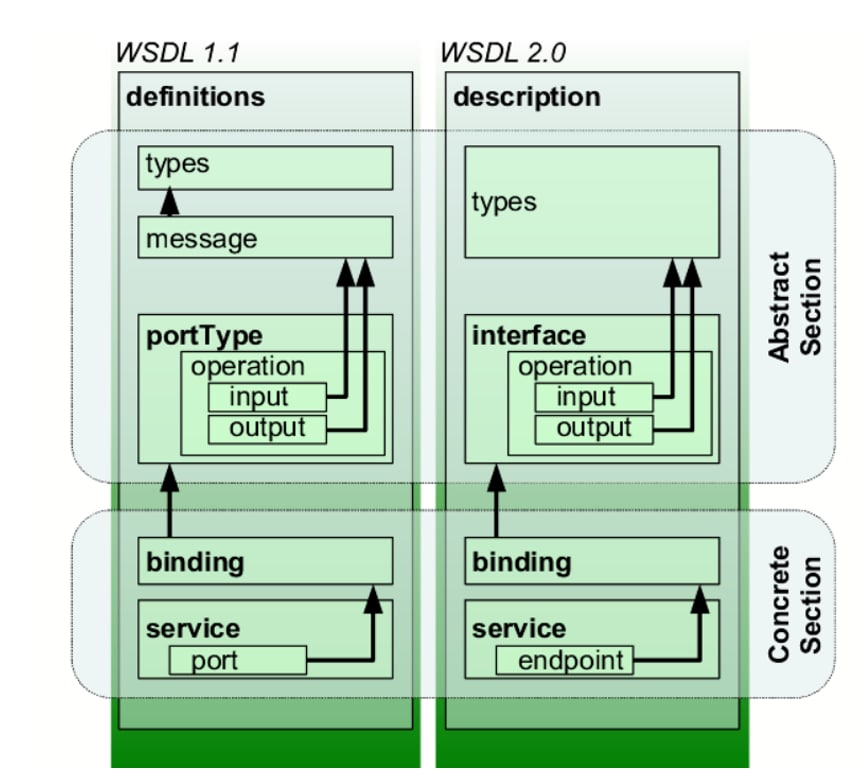
Может быть указана во всех блоках и указывает, кому предназначена информация блока.

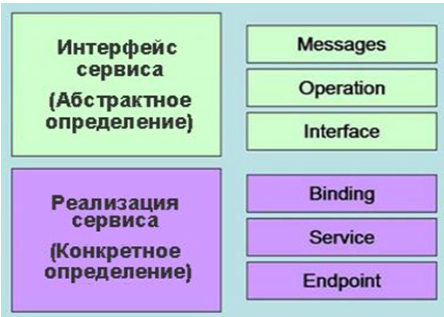
**Узел** - компонент SOA архитектуры, который может получать и отправлять сообщения.

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

**WSDL: Web Services Description Language** -средство XML-описания интерфейса для доступа к web-службе; платформонезависимое описание. Чтобы формальным способом описать сервис, основан на xml чтобы в автоматическом режиме сгенерировать клиента. На основе wsdl можно сгенерировать прокси и стан (класс, который обменивается сообщениями с прокси (прокси на стороне сервера))

Версия 1.1(2001), **версия 2.0**(2007, статус рекомендации).





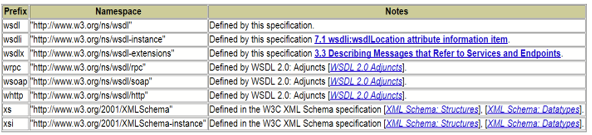
***Абстрактная*** - описывается в языке WSDL с помощью системы типов, XML-схемы, в терминах сообщений, которые этот сервис отправляет и получает.

1. **Шаблоны обмена** сообщениями определяют последовательность и количество сообщений.
2. Элемент **operation** связывает шаблоны обмена сообщениями с одним или несколькими сообщениями.
3. Элемент **interface** группирует ***operation*** независимо от транспорта и способа доставки.

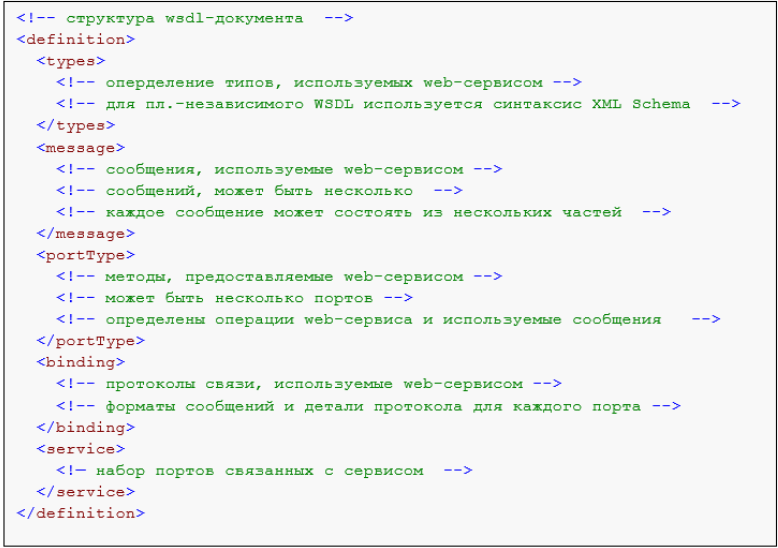
В ***конкретной части***

1. Элементы **binding** задают транспорт и формат доставки для ***interface***.
2. Элемент **endpoint** связывает сетевой адрес в соответствие с *элементом* ***binding***.
3. Элемент **service** группирует *элементы* ***endpoint***, которые реализуют общий ***interface***

**ПРОСТРАНСТВО ИМЕН**



**СТРУКТУРА WSDL-ДОКУМЕНТА**

****



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

**ASMX: Active Server Method Extended -** технология Microsoft для разработки web-сервисов(2002, 2007), основанная на XML, WSDL, SOAP; официальное название в MSDN «XML Web Services»; легкая технология: для работы с ней не обязательно знать XML, SOAP и WSDL.

ASMX сервис - RPC сервис, web сервис, полудуплекс, частый случай WSF,формат данных soap, транспорт http, встраивается в asp .net приложение, требуется IIS



Веб-сервис представлен одним обычным классом с одной лишь обязательной особенностью – некоторые его методы помечены специальным атрибутом [WebMethod]. Такие методы класса становятся веб-методами веб-сервиса с соответствующей сигнатурой вызова. Этот класс должен обладать конструктором по умолчанию. При каждом новом запросе IIS его инстанциирует дефолтным конструктором и вызывает соответствующий метод.

Вторая обязательная часть минимальной конструкции – это файл с расширением asmx, внутри которого необходимо указать этот класс.

**Утилита wsdl.exe** для генирации прокси и стан на стороне сервера, на входе wsdl файл (что сгенерировал сервер и сгенерирует клиент)

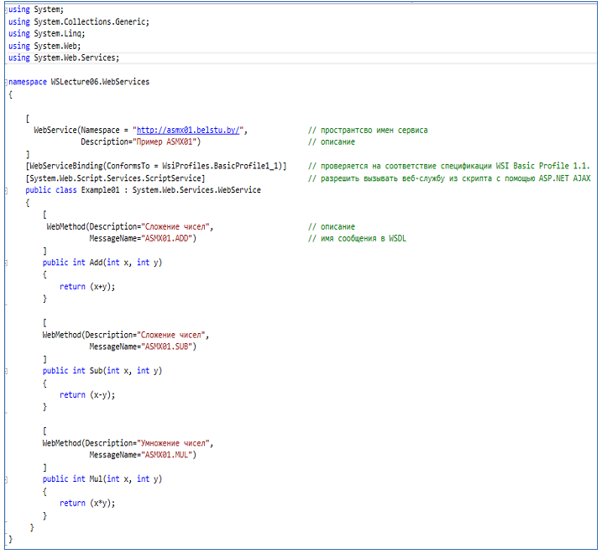
является соответствующей для asmx техникой потребления SOAP веб-сервисов. По wsdl файлу или ссылке она генерирует прокси-класс – специальной класс, максимально упрощающий обращение к данному веб-сервису.

Утилита расположена в папке C:\Program Files (x86)\Microsoft SDKs\Windows, более того, она там представлена в разных версиях, в зависимости от версии .net, разрядности, версии windows и visual studio.

Как известно, wsdl описание веб-сервиса в технологии ASMX генерируется автоматически. Однако иногда возникает обратная задача: по данному wsdl файлу разработать соответствующий ему веб-сервис. Решается она с помощью той же утилиты wsdl.exe. Она может создать необходимый скелет из классов и вам останется только реализовать программную логику веб-методов.

Принципы разработки:

1.добавить ASM-службу



# **6.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;** платформа для разработки приложений SOA-архитектуры, в основе SOAP, .NET. Поддерживает не только Web сервисы

Главная особенность: один сервис может иметь много точек, сервис хостит. сам себя запускает (как обычное .net приложение), можем встроить в IIS, в любое ASP .Net приложение

первая версия 2006 (Indigo).

**Основные принципы**

- разработка сервиса должна быть простой и иметь способность к расширению его функциональных возможностей;

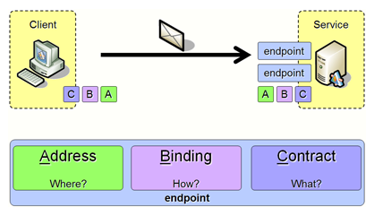
- один API для всех коммуникационных протоколов;

- сервис должен функционировать по открытым телекоммуникационным стандартам;

- сервис должен поддерживать стандарты WS-\*;

- сервис должен поддерживать REST, RPC и др. архитектуры;

**Коммуникация модель**



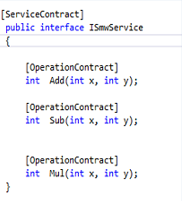
**Адрес конечной точки** указывает, где находится конечная точка в виде сетевого адреса, в основном это - URI, идентификатор и набор необязательных заголовков.

**Привязка конечной точки** указывает, как конечная точка взаимодействует с миром, включая: транспортный протокол, кодирование и требования безопасности. Привязка имеет имя, пространство имен и набор составных элементов привязки. Имя и пространство имен привязки однозначно идентифицируют ее в метаданных службы.

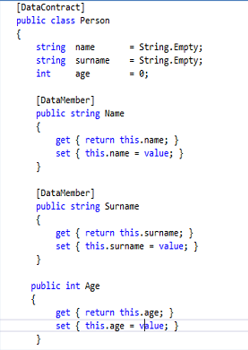
**Контракт конечной точки** определяет, что передает конечная точка, и, по сути, представляет собой набор сообщений, организованных в операции, которые имеют базовые шаблоны обмена сообщениями (MEP), такие как односторонний, дуплексный и запрос/ответ. Подобно привязкам службы, каждый контракт службы имеет имя и пространство имен, которые однозначно идентифицируют его в метаданных службы.

**WCF-КОНТРАКТЫ**

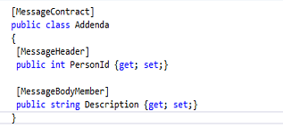
* **контракт службы** - предоставляет клиенту, а также внешнему миру информацию о предложениях конечной точки и протоколах, которые будут использоваться в процессе связи



* **контракт данных** - указывает каким образом данные будут сериализованы. Данные, которыми обменивается сервис, определяются контрактом данных.



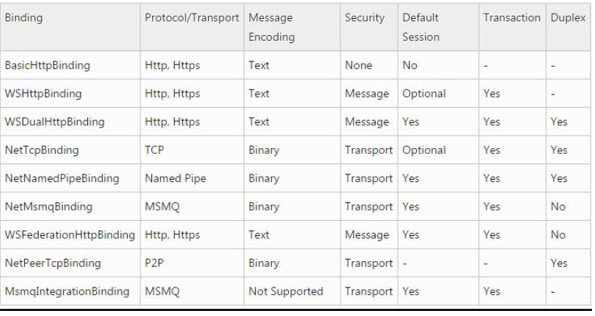
* **контракт сообщений** - описывает формат сообщений – контракт данных контролируется контрактом сообщения. Он в первую очередь выполняет настройку форматирования типов параметров сообщений SOAP. WCF использует формат SOAP для связи.



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

**Конечная точка** – адрес, привязка, контракт.

**Стандартные привязки:**

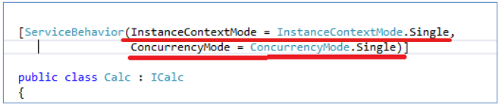


**Разница между WCF и ASMX веб сервисами**

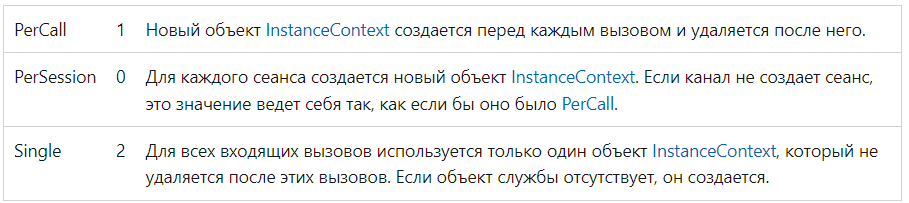
ASMX простой, но во многих отношениях ограничен по сравнению с WCF.

1. ASMX веб сервисы могут быть размещены только в IIS, в то время как WCF сервис имеет больше вариантов хостинг (IIS, WAS (Windows Process Activation Services), Console Application, Windows NT Services, WCF provided Host)
2. ASMX веб сервисы ограничены в поддержке одного лишь HTTP, в то время как WCF поддерживает HTTP, TCP, MSMQ, Named Pipes. (ключевое!)
3. Безопасность ASMX ограничена. WCF поддерживает больше протоколов. Обычно аутентификация и авторизация ASMX производится используя конфигурацию безопасности IIS и ASP.NET и безопасность транспортного слоя. Для слоя сообщений может быть использовано WSE. WCF предоставляет согласованную программную модель безопасности для любого протокола и поддерживает много таких же возможностей как IIS и WS-\* протоколы безопасности. Дополнительно, WCF предоставляет поддержку claims-based авторизации которая позволяет контролировать ресурсы более четко чем безопасность на основе ролей (role-based security). WCF безопасность остается согласованной не зависимо от того какой хост был использован для реализации WCF сервиса.

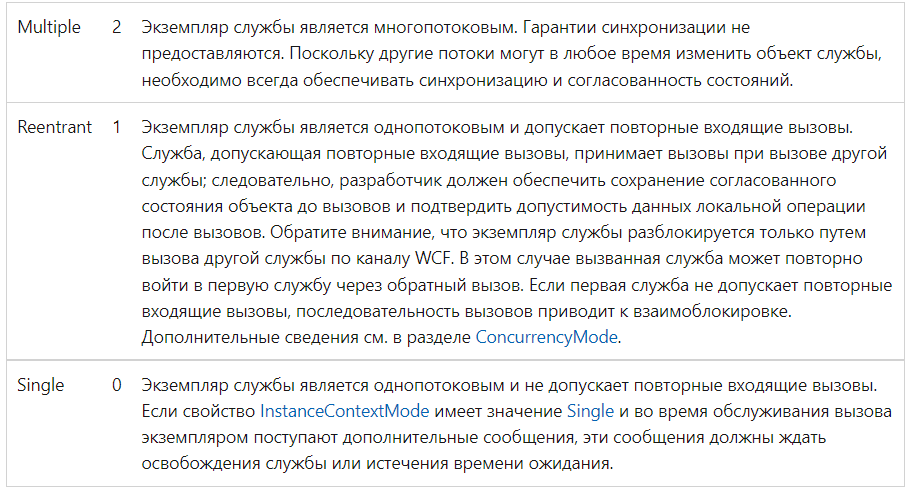
**Service Behavior**

****

**InstanceContextMode** - Задает число экземпляров службы, доступных для обработки вызовов, содержащихся во входящих сообщениях.



**ConcurrencyMode** - Указывает, какой режим выполнения операции поддерживается классом службы — однопотоковый или многопотоковый.



**Режимы безопасности передачи:**

**Нет**.

**Транспорт**. - наиболее простой способ с использованием протоколов связи, таких как TCP, IPC, Https и MSMQ.

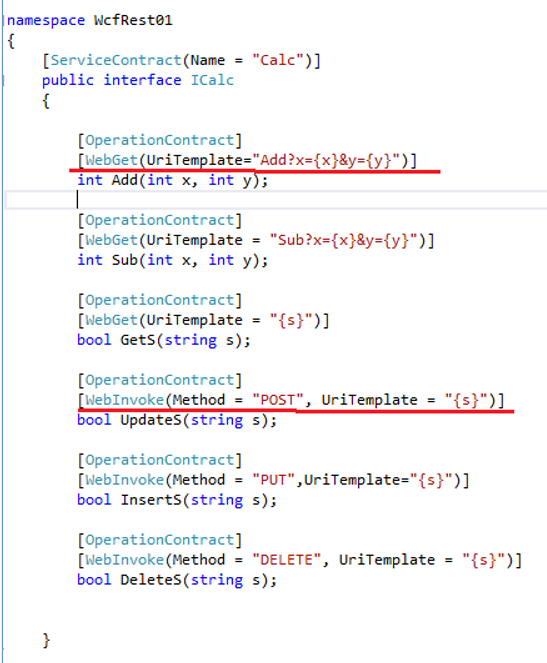
**Сообщение** — обеспечивает взаимную аутентификацию

**Смешанный** — этот режим безопасности используется не часто, и аутентификация клиента предлагается только на уровне клиента.

**И то, и другое.** Этот режим безопасности включает в себя как транспортную безопасность, так и безопасность сообщений

**Разработка WCF-службы с REST-интерфейсом**

webHttpBinding

****

**WCF-сервис с несколькими конечными точками**

****

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

**WCF Syndication Services:** - реализация WCF для новостных каналов

**ATOM** - формат описания web-ресурсов и протокола для их публикации (application/atom+xml)

ATOM более универсальный и чаще применяется; следует использовать его при разработке новых приложений.

<feed>

<id> — Идентификатор фида, постоянный URI.

<title> — Название канала. Не может быть пустым.

<updated> — Дата последнего обновления

<author>

<link> — Адрес связанного сайта. Тип связи определяется в атрибуте rel.

<category> — Задает категории, к которым относится канал. Элемент feed может включать несколько элементов category.

<generator> — Название программы, с помощью которой собран канал.

<icon> — Маленькое изображение, пиктограмма канала.

<logo> — Большое изображение, логотип канала.

<rights> — Информация об авторском и смежных правах.

<subtitle> — Человеко-читаемое описание или подзаголовок канала.

<entry>

<id>uuid:2e6c2527-997a-4310-8028-5626bef6fdbd;id=4</id>

<title type="text">Java</title>

<updated>2022-12-27T19:03:00Z</updated>

<content type="text">8</content>

</entry>

</feed>

**RSS** - семейство XML-форматов, предназначенного для описания новостных лент, анонсов и статей

<rss xmlns:a10="http://www.w3.org/2005/Atom" version="2.0">

<channel>

<title>lab 7</title>

<description>description 7 lab</description>

<language>ru-ru</language>

<managingEditor>tukaalena2@gmail.com</managingEditor>

<lastBuildDate>Tue, 27 Dec 2022 22:02:46 +0300</lastBuildDate>

<category>students</category>

<generator>C# RSS</generator>

<item>

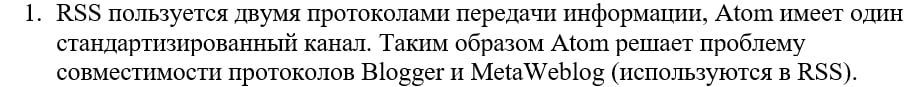
<title>Java</title>

<description>8</description>

</item>

</channel>

</rss>



Порядок разработки:

1.Создать проект Библиотека службы публикации

2. на клиента добавить reference на службу

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

**WCF Data Services**- реализация WCF, которая позволяет использовать протокол OData (доступ к реляционный базе данных через REST интерфейс

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

для получения JSON, необходимо обеспечить в заголовке заголовок ACCEPT:application/json или $format=json

Возможности предоставляемые OData-интерфейсом:

* Простое чтение данных (запросы без параметров) : одного элемента, коллекции
* Параметризованные запросы (поиск, сортировка, и т.д.)
* Добавление данных
* Модификация данных
* Удаление данных
* Добавление, модификация и удаление связей между объектами

Порядок разработки:

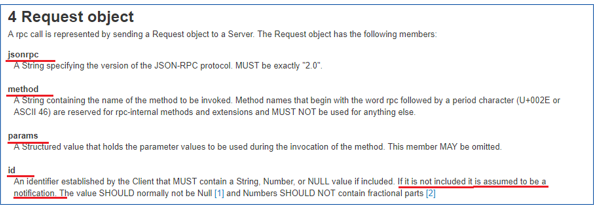
1. Создать модель ADO.NET EDM (подключиться к бд, выьрать таблицы и тд)
2. Создать WCF Data Service
3. Add connected service (1)
4. На клиенте добавить ссылку на службу (2)

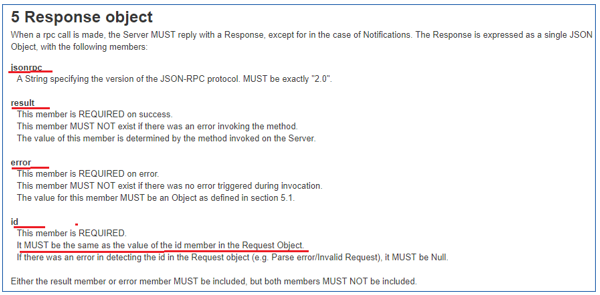
# **9.JSON-RPC:** определение JSON-RPC-сервиса, форматы запросов и ответов, обработка ошибок, пакеты запросов, реализация JSON-RPC на платформе Web API.

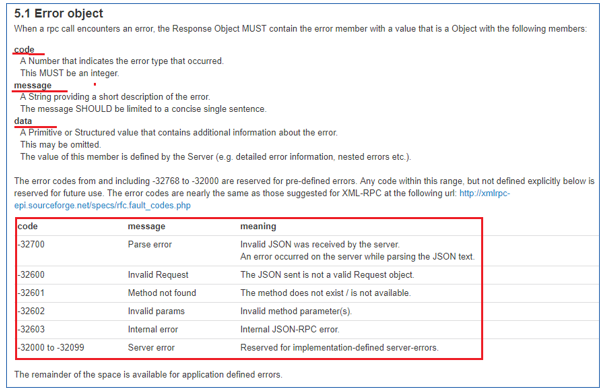
**JSON-RPC -** протокол удаленного вызова процедур, использующий формат JSON для передачи сообщений.

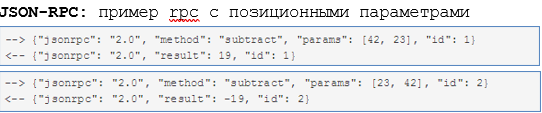
последняя версия 2.0.

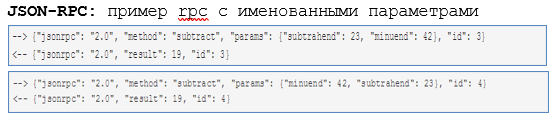
транспортHTTP/HTTPS или TCP Socket.

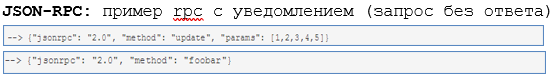




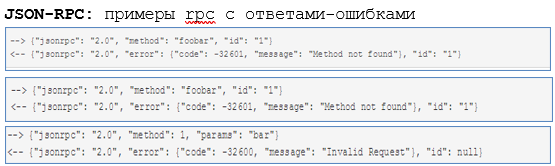


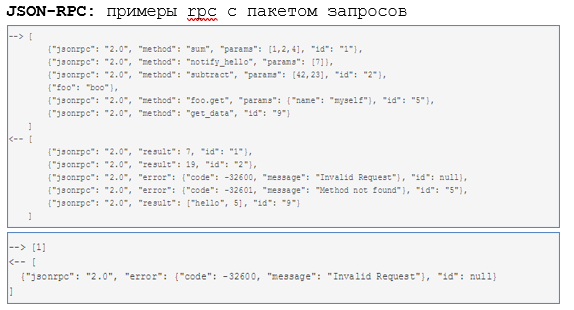


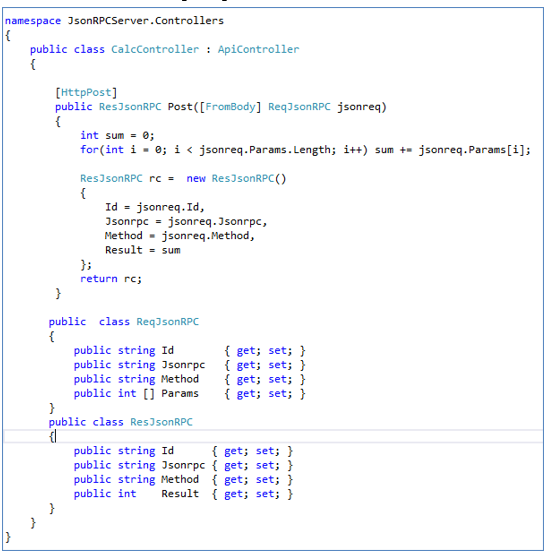




(примеры)

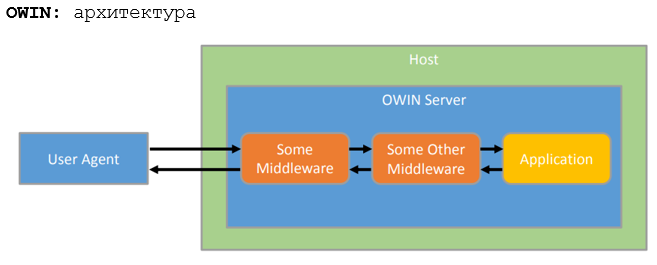






# 10.**ASP.NET CORE Nancy:** интерфейс OWIN,архитектура приложения, принцип разработки сервиса.

**OWIN: Open Web Interface for NET. 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.

****

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

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

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

4. OWIN: **middleware** – подключенные компоненты (модули), предназначенные для обработки запросов.

**Nancy** — это легкий, мощный, с открытым исходным кодом, очень простой веб-фреймворк, предназначенный для реализации сервисов на основе HTTP.Позволяет создавать независимые друг от друга модули

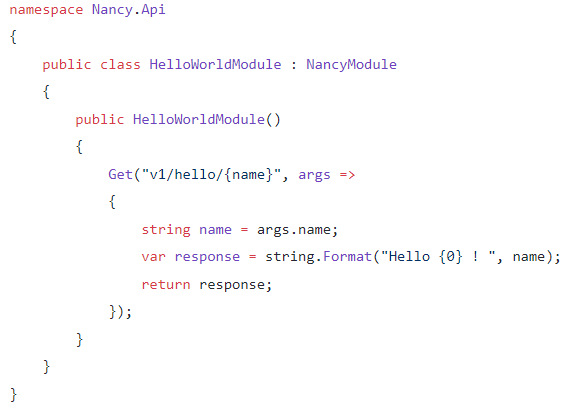
* **простая конфигурация**
* **простой путь**

**Принципы разработки:**

1. create an API .Net Core projects
2. скачать пакеты из Nuget:[**Nancy**](https://www.nuget.org/packages/Nancy/)**, Nancy.Owin**
3. в методе **Configure**() Startup.cs конфигурация Nancy:

app.UseOwin(x => x.UseNancy());

1. унаследовать **NancyModule** и в конструкторе



# **11.Event Storing:** назначение, принципы применения, примеры реализации

Event Sourcing- подход к хранению данных, при котором вместо конечного результата хранится череда событий произошедших с некоторой сущностью.

1. каждое событие имеет имя.

2. ES: произошедшие события неизменны.

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

**EventStoreDB** хранит ваши данные в виде серии неизменяемых событий с течением времени, предоставляя один из самых надежных доступных вариантов журнала аудита (характеристики аналогичны блокчейну).

Принципы разработки:

1.Скачать

2.запустить .exe ..db ./db ..log ./log

3.скачать пакет EventStore.ClientAPI.newCore





# **12.Микросервисы:** микросервисная архитектура, определение микросервиса, основные принципы разработки микросервиса, паттерны разработки, DevOps для микросервисов.

**Микросервисы:** один из подходов к разработке SOA-приложений, основной принцип – сервисы должны быть легкими (сервисы маленькие и протоколы легковесные). Легкий сервис – сервис, который может быть переписан за 2 недели (Real Estate).

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

**Цель**:

1) устойчивость к сбоям;

2) облегчить понимание и поддержку кода;

3) усилить работу команды программистов.

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

Популярные паттерны ***Tolerant Reader***, ***Consumer Driver***.

Требуется ***DevOps***- набор технологий нацеленных на интеграцию процессов разработки и информационно-техническому обслуживанию.

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.

# 13.**Docker**: назначение, архитектура, основные команды

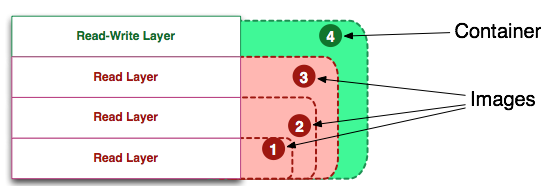
**Docker:** платформа для разработки, доставки и эксплуатации приложений. Основное назначение – упростить развертывание приложения

**Docker** = Docker Desktop(Engine) + Docker Hub

**Docker Hub** – облачный сервис для распространения **контейнеров**.

**Docker Engine** – механизм (сервис и приложение) для создания и функционирования **контейнеров**.

**image** – read only template – шаблон для чтения.



1. Определите среду своего приложения с помощью , **Dockerfile** чтобы его можно было воспроизвести где угодно.

docker build - создать image

docker run - создать и запустить контейнер

docker buid,

docker push

docker pull

docker run

docker stop

docker start

doker ps

docker images ls - список images

docker rm

docker image rm

docker search

# 14.**Docker-Compose**: назначение, основные команды.

**Compose** — это инструмент для определения и запуска многоконтейнерных приложений Docker. С Compose вы используете файл YAML для настройки служб вашего приложения. Затем с помощью одной команды вы создаете и запускаете все службы из вашей конфигурации.

1. Определите среду своего приложения с помощью , Dockerfileчтобы его можно было воспроизвести где угодно.
2. Определите службы, из которых состоит ваше приложение, docker-compose.yml чтобы их можно было запускать вместе в изолированной среде.
3. Build and run your app docker compose up

-d - фоновый режим



Наличие нескольких изолированных сред на одном хосте **Compose** использует имя проекта для изоляции сред друг от друга. Вы можете использовать это имя проекта в нескольких различных контекстах:

* на хосте разработки для создания нескольких копий одной среды, например, когда вы хотите запустить стабильную копию для каждой функциональной ветви проекта.
* на сервере CI, чтобы сборки не мешали друг другу, вы можете установить имя проекта на уникальный номер сборки
* на общем хосте или хосте разработки, чтобы разные проекты, которые могут использовать одни и те же имена служб, не мешали друг другу

**docker compose run** позволяет запускать одноразовые команды для ваших служб. Например, чтобы увидеть, какие переменные среды доступны webслужбе: $ docker compose run web env