УТВЕЖДАЮ

Заведующий кафедрой программной инженерии

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_В.В. Смелов

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Экзаменационные вопросы по**

**дисциплине «Программирование в Интернет» для студентов 4 курса специальности ПОИБМС**

1. SOA: определение, сервис, типовые компоненты, способы клиент-серверного взаимодействия.

**сервис-ориентированная архитектура -** стиль архитектуры информационной системы, который позволяет создавать приложения путем комбинации слабосвязанных распределенных компонент.

компоненты распределенной системы SOA – узлы - **сервисы**.

**сервис –** логическое представление реальной программы, базы данных, бизнес-процесса и т.п., определенное в терминах функциональности.

**сервис** – это видимый ресурс, выполняющий повторяющуюся задачу и описанный внешней инструкцией.

Свойства soa и сервиса

-   независимость от аппаратной реализации узлов;

-   независимость от операционной системы в узлах;

- независимость от языка программирования разработки сервиса;

-   масштабируемость.

-   сервис ориентирован на бизнес;

-   сервис автономен;

-   повторное использование;

-   четко описанная инструкция в терминах интерфейса (SLA);

-   сервис виден (доступен).

**интерфейсы – REST, SOAP, JSON-RPC**

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

**Спецификации:**

**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).

**ESB – Enterprise Service Bus:** программный компонент обеспечивающий обмен сообщениями между различными информационными системами, имеющих сервис-ориентированную структуру.

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

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

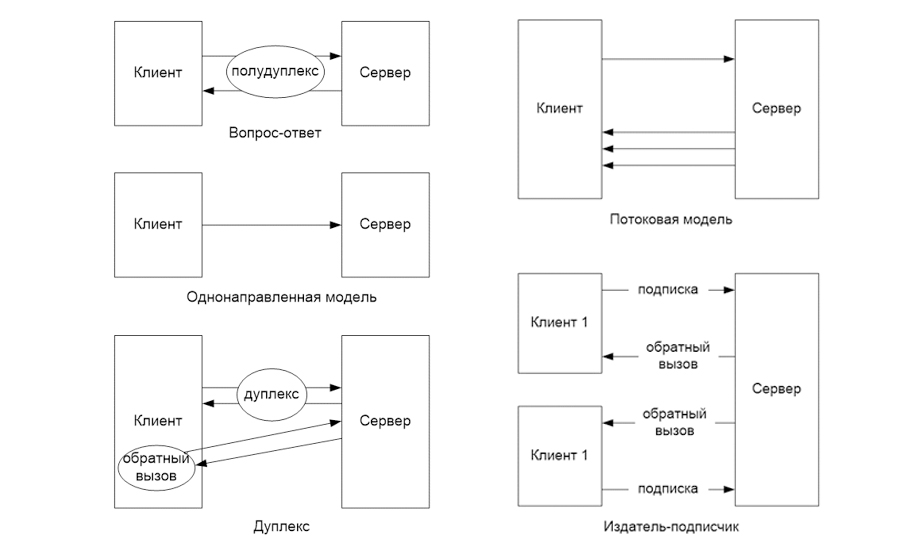
-   **Workflow Engine** - программный компонент, предназначенный для построить (построить модель бизнес процесса) и выполнить бизнес процесс на основе группы сервисов. Другими словами разработать новый сервис на основе последовательного выполнения нескольких сервисов;

-   **Service Broker** - программный компонент, позволяющий трансформировать запрос пользователя в системе в запуск и координированную работу;

- **Identity Service -**  сервис аутентификации, предназначен для реализации SSO (Single Sign On), протоколы токен-авторизации (JWT), OpenID Connect, OAuth2. IDaaS. (AWS IAM)

-   **SOA Supervisor** - служебный сервис, предназначенный для управления и мониторинга других служебных сервисов. **Health Check Service** – служебный сервис для проверки работоспособности других сервисов, часто в составе Supervisor; **Health Check API:** API для проверки работоспособности. Системы мониторинга серверов: Nagios, Zabbiks, Icinga (Nagios Core), dotcom.monitor.

1. **Способы клиент-серверного взаимодействия**

****

1. Web-сервис: определение, архитектурные стили (типы интерфейсов).

REST, SOAP,

Такое же веб-приложение, которое используется для работы других веб-приложений.

1. REST: определение, ограничения, общепринятые правила, протоколы и форматы передачи данных, HATEOS.

**REST**: **Representational State Transfer** - передача состояния представления. Архитектурный стиль взаимодействия компонентов распределенного приложения.

**представление** – это URI (идентификатор ресурса), управление ресурсом с помощью глаголов (GET, POST, PUT, DELETE).

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

**два основных типа ресурса** – коллекции и элемент коллекции: /api/users, /api/users/288.

**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 интерфей**сов (идентификация ресурсов, манипуляция ресурсами через представления, самодостаточные сообщения, HATEOAS);

-   **для клиента сервер** должен представляться конечным;

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

**HATEOAS: Hypermedia As The Engine Of Application State** – гипермедиа в качестве управления состоянием.

**Гипермедиа**: технология обработки, структурирования информации и произвольного доступа к ее элементам с помощью гиперсвязей (Тед Нильсон, 1965), WWW – реализация гипермедиа.

**REST:** **общепринятые правила**

**16.1.** **Общий префикс для всех ресурсов сервиса** … /API/…,

[http://API.BSTU.BY/](http://api.bstu.by/)...

**16.2.** **Два типа ресурсов**: коллекция (users, students, …), элемент коллекции /api/users/238, /api/students/ef3d26.

**16.3.** **Иерархическая связь** …/api/users/238/cars/aah4899

16.4. **Ограничить количество HTTP-статусов, сопроводить сообщение дополнительным кодом** (например 20003, 404001,…),сделать отдельный ресурс (HATEOAS link) для пояснения ошибок<http://ccc/api/errors/20003>.

16.5.Подавление статуса ответа …/api/students/ef3d26?status\_code=200.

16.6. **Версионность**  /api/students/ef3d26?v=7.

16.7. **Постраничное получение данных**: параметры limit offset.

16.8. **Сортировка:** параметр sort.

16.9. **Все фильтры вынести за знак вопроса**: …/api/students?minbday=1998101&maxbday=20001231&gender=m.

16.10. **Пользователь получает только то, что хочет**: …/api/students?field=bday,surname,gender.

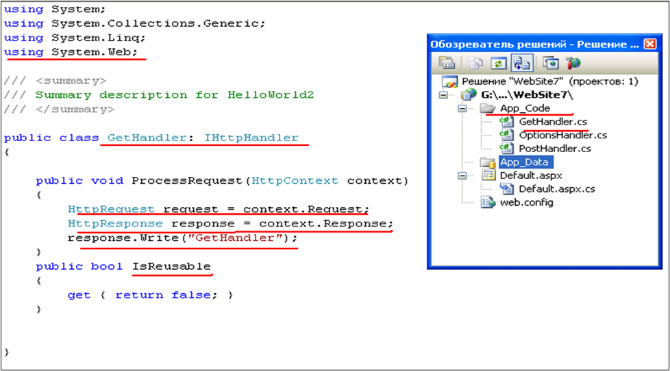
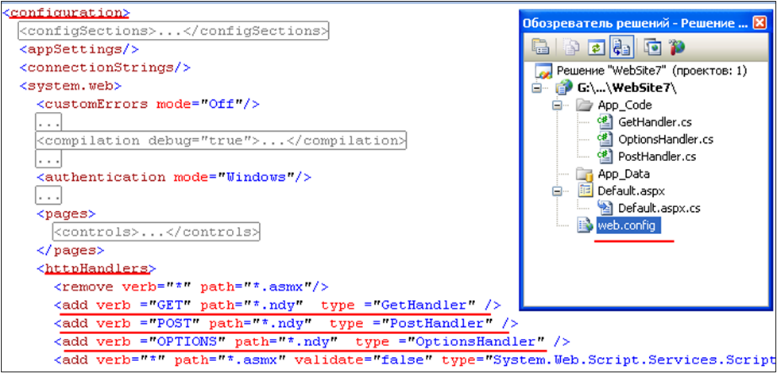
16.11. **Обозначать в запросе формат сообщений** (желательна поддержка нескольких форматов):

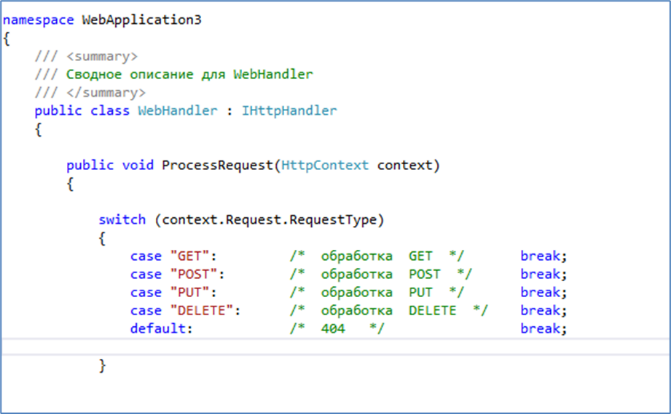
…/api/students.json?field=bday,surname,gender; один из форматов должен быть по умолчанию; могут применяться заголовки Accept и Content-Type со значениями application/xml и application/json для запроса или обозначения в ответе формата.

16.12. **Глобальный поис**к: ../api/search?q=19600107+Иванов.

16.13. Документация.

**создание REST-службы на базе ASP.NET HTTP Handler**

** **

****

**создание REST-службы на базе ASP.NET MVC WEB API просто контроллер наследовать от ApiController,**

1. RPC: определение, протоколы и форматы передачи данных, proxy/ stub-объекты.

RPC (Remote Procedure Call) — это технология, позволяющая выполнять вызовы функций или процедур, которые находятся на удалённом сервере, так, как если бы они выполнялись локально. RPC абстрагирует сложность сетевого взаимодействия, предоставляя разработчикам удобный интерфейс для вызова удалённых процедур.

**Протоколы для RPC**

1. **HTTP/HTTPS**:
   * Часто используется в REST и SOAP сервисах.
   * Обеспечивает простоту взаимодействия и широкую совместимость.
2. **gRPC**:
   * Основан на HTTP/2 и Protocol Buffers (protobuf).
   * Поддерживает двунаправленную передачу данных, потоковую передачу и сжатие данных.
   * Высокая производительность и поддержка различных языков программирования.
3. **JSON-RPC**:
   * Лёгкий протокол, использующий JSON для передачи данных.
   * Простой в реализации и использовании, особенно для веб-приложений.
4. **XML-RPC**:
   * Использует XML для кодирования вызовов и HTTP как транспортный протокол.
   * Менее эффективен по сравнению с JSON-RPC из-за объёма XML-документов.

**Форматы передачи данных**

1. **JSON**:
   * Читаемый человеком формат, широко используемый в веб-разработке.
   * Легко парсится в большинстве языков программирования.
2. **XML**:
   * Используется для SOAP и XML-RPC.
   * Более громоздкий, но поддерживает сложные структуры данных.
3. **Protocol Buffers (protobuf)**:
   * Бинарный формат от Google, используется в gRPC.
   * Более компактный и быстрый для сериализации/десериализации по сравнению с JSON и XML.
4. **MessagePack**:
   * Бинарный формат, похожий на JSON, но более компактный.
   * Быстрее в передаче и обработке.

**Proxy и Stub-объекты**

1. **Proxy (Прокси)**:
   * Локальный объект, представляющий удалённую процедуру.
   * Перехватывает вызовы и перенаправляет их на сервер.
   * Служит интерфейсом между клиентом и сервером.
2. **Stub (Заглушка)**:
   * На стороне сервера Stub получает запрос от Proxy.
   * Stub разбирает запрос и вызывает соответствующую локальную функцию или процедуру.
   * Stub затем собирает ответ и отправляет его обратно Proxy.

**Рабочий процесс**:

1. Клиент вызывает удалённую процедуру через Proxy.
2. Proxy сериализует запрос и отправляет его на сервер.
3. Stub на сервере принимает запрос, десериализует его и вызывает нужную процедуру.
4. Результат выполнения процедуры отправляется обратно через Stub и Proxy к клиенту.
5. SOAP: определение, структура сообщений, принцип расширения.

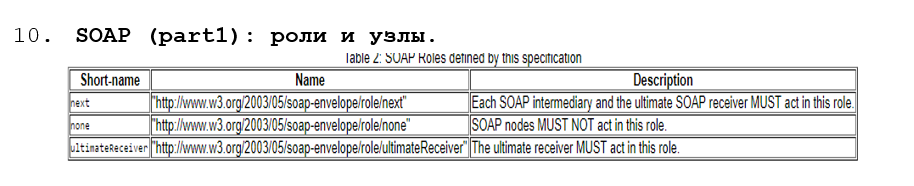
**Simple Object Access Protocol** – простой протокол доступа  к объекту. первоначально - реализация RPC, расширение протокола XML-RPC,   позже - произвольный обмен XML-сообщениями. спецификация поддерживается W3C, последняя версия 1.2, в версии 1.2. аббревиатура SOAP не расшифровывается.

**Сообщение SOAP** – это простой XML-документ, который состоит из следующих элементов:

* **Envelope**  
  Обязательный элемент, который определяет начало и конец сообщения. **Envelope («конверт»).**Это корневой элемент. Определяет [XML-документ](https://blog.skillfactory.ru/glossary/xml/) как сообщение SOAP с помощью *xmlns\_soap=»http://www.w3.org/2003/05/soap-envelope/».* Если в определении будет указан другой адрес, сервер вернет ошибку.
* **Header**  
  Опциональный элемент который может включать в себя любые опциональные (не обязательные) атрибуты сообщения.В заголовке могут использоваться **три атрибута**, которые указывают, как принимающая сторона должна обрабатывать сообщение, ***mustUnderstand*, *actor* и *encodingStyle*.** Значение ***mustUnderstand*** — 1 или 0 — говорит принимающему приложению о том, следует ли распознавать заголовок в обязательном или опциональном порядке. Атрибут ***actor*** задает конкретную конечную точку для сообщения. Атрибут ***encodingStyle*** устанавливает специфическую кодировку для элемента.
* **Body**  
  Обязательный элемент, который включает в себя данные XML, которые определяют сообщение, которые должно быть отправлено. Может содержать запрос к серверу или ответ от него.
* **Fault**  
  Опциональный элемент, который предоставляет информацию об ошибках, которые возникают в процессе обработки сообщения.

**Пространства имен, представленные в изображении:**

1. **"env"**:
   * **Описание**: Это пространство имен используется для элементов, относящихся к структуре SOAP-сообщения, например, Envelope, Header, Body, а также к элементам, связанным с обработкой ошибок (Fault).
2. **"enc"**:
   * **Описание**: Используется для обозначения правил кодирования данных в SOAP-сообщении. Пространство имен soap-encoding определяет, как данные должны быть сериализованы в формате XML.
3. **"rpc"**:
   * **Описание**: Это пространство имен используется для поддержки удалённого вызова процедур (Remote Procedure Call, RPC) с использованием SOAP. Оно определяет, как аргументы и возвращаемые значения процедур передаются в SOAP-сообщениях.
4. **"rep"**:
   * **Описание**: Используется для обозначения представлений данных в сообщениях. Обычно это касается описания того, как представляются данные, передаваемые через веб-сервисы.
5. **"xop"**:
   * **Описание**: XOP (XML-binary Optimized Packaging) используется для включения бинарных данных в SOAP-сообщение, что позволяет более эффективно передавать большие объекты, такие как файлы изображений или документы.
6. **"xmime"**:
   * **Описание**: Это пространство имен связано с использованием XML-MIME для указания типов содержимого и данных, которые могут быть представлены в MIME-формате в сообщениях SOAP.

****

**роль** -  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)

**Роль "none"** используется для указания, что определенный узел не должен обрабатывать сообщение SOAP. В контексте этой роли, узел игнорирует тело сообщения SOAP и не выполняет никаких действий, связанных с его содержимым.

**Роль "next"** указывает, что сообщение SOAP должно быть передано следующему узлу в цепочке обработки. Это обычно используется для маршрутизации сообщения к следующему получателю или обработчику, который должен продолжить обработку, основанную на контексте или маршруте.

**Роль "ultimateReceiver"** указывает на конечного получателя SOAP-сообщения — узел, который должен выполнить основную обработку сообщения. Этот узел, как правило, является тем, для кого предназначено сообщение, и он должен выполнить все необходимые действия с содержимым сообщения, такие как его интерпретация и выполнение логики бизнес-процесса.

**Использование заголовка (Header) для расширений**: Заголовок в SOAP-сообщении — это секция, предназначенная для передачи метаданных и управляющей информации. Например, в заголовке можно указать данные для аутентификации, информацию о маршрутизации или указания по обработке сообщения.

**Пространства имен для уникальности**: Если вы хотите добавить новую информацию в заголовок SOAP-сообщения, например, для аутентификации или шифрования, вы можете использовать пространство имен, чтобы указать, что эти элементы принадлежат вашему расширению.

1. ASMX: определение, назначение, протоколы, proxy/stub-объекты.

**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. WSE(Web Services Enhancements – улучшение)- дополнение к ASMX основанное на спецификациях WS-\*.

Visual Studio, к любому проекту добавить ASM-службу

Утилита **WSDL.EXE** – это инструмент командной строки из состава .NET SDK, который используется для:

* **Генерации клиентского кода**: Создает прокси-класс на основе WSDL-документа веб-службы.
* **Генерации WSDL-документа**: Создаёт описание службы на основе существующего класса веб-службы.

Пример использования:

**Создание прокси-класса:**

wsdl.exe http://example.com/MyService.asmx?wsdl

1. Этот командный вызов создаёт файл MyService.cs, который можно использовать для вызова методов веб-службы.

**Создание WSDL-документа**:

wsdl.exe /serverInterface MyService.asmx

**Создание server** :

wsdl.exe /server /n:WebServices.Example01

ASMX-сервис — это веб-сервис, который использует расширения ASP.NET для обработки запросов SOAP.

**Порядок разработки ASMX-сервиса**

1. Создайте новый проект ASP.NET в Visual Studio.

2. Добавьте файл .asmx, например MyService.asmx.

3. Определите класс веб-службы и методы с использованием атрибута [WebMethod].

4. Разверните веб-сервис на сервере IIS или используйте встроенный сервер разработки Visual Studio для тестирования.

**Принципы применения**

* Используется для взаимодействия между разными приложениями и платформами с помощью SOAP.
* Применяется в корпоративных приложениях, где требуется высокая совместимость и стандартизация.

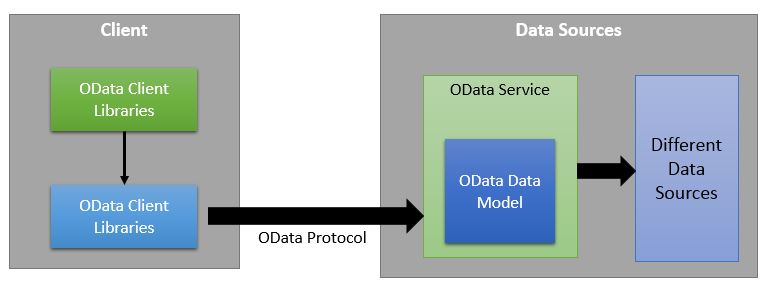
1. OData: определение, назначение, сравнение с SQL.

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

**Архитектура:**

На диаграмме ниже поясняется высокоуровневая архитектура OData. Существует четыре основные части OData.

* Модель OData
* Служба OData
* Протокол OData
* Клиентские библиотеки OData



**Модель OData:**

Модель OData используется для представления данных из разных источников данных в едином формате с использованием модели Entity Data (EDM).

**Служба OData:**

Это уровень обслуживания поверх модели OData, который предоставляет конечную точку, которая позволяет клиенту получать доступ к данным с использованием протокола OData и клиентской библиотеки OData.

Он преобразует различные форматы источников данных, такие как таблицы отношений SQL, список SharePoint, таблицы Windows Azure и т. д., в общий формат для использования клиентом.

**Протокол OData:**

Протокол OData позволяет клиенту отправлять запросы и получать ответы от службы OData.

Протокол OData — это набор взаимодействий RESTful.

Протокол OData **аналогичен HTTP** и используется для выполнения операций CRUD с использованием OData. **Он передает данные в форме XML или JSON.**

**Клиентские библиотеки OData:**

Клиентские библиотеки OData используются клиентским приложением для доступа к данным с использованием протокола OData.

Предоставление готовых библиотек для выполнения запросов OData и получения результатов упрощает жизнь разработчикам, создающим эти приложения.

Для доступа к OData доступно несколько клиентских библиотек OData. Некоторые из них:

· Microsoft .NET Framework 3.51: **платформа WCF Data Services** доступна для отдельной загрузки для .NET 3.x.

· Java: **odata4j** (включая Java на телефоне Android) поддерживает протокол OData.

· JavaScript: библиотека **OpenUI5,** поддерживаемая **SAP.**

· И многое другое.

**Возможности интерфейса:**

***1. Фильтрация данных****:*

· http://localhost:5000/odata/Students?$**filter**=Grade eq 5 - получение студентов с оценкой 5.

***2. Сортировка данных****:*

· http://localhost:5000/odata/Students?$**orderby**=Name desc - получение студентов, отсортированных по имени в обратном порядке.

***3. Проекция данных*** *(выбор конкретных полей):*

· http://localhost:5000/odata/Students?$**select**=Name,Grade - получение только имен и оценок студентов.

***4. Расширенные запросы****:*

· http://localhost:5000/odata/Students?$**top**=2 - получение первых двух студентов.

· http://localhost:5000/odata/Students?$**skip**=2 - пропуск первых двух студентов и получение остальных.

· http://localhost:5000/odata/Students?$**count**=true - получение общего количества студентов.

***5. Комбинированные запросы****:*

· http://localhost:5000/odata/Students?$filter=Grade gt 3 and Grade lt 6&$orderby=Grade desc - получение студентов с оценкой между 3 и 6, отсортированных по убыванию оценок.

***6. Поиск****:*

· http://localhost:5000/odata/Students?$**search**=Иванов - поиск студента с именем "Иванов".

***7. Группировка данных****:*

· http://localhost:5000/odata/Students?$apply=groupby((Grade), aggregate(Name with max as MaxName)) - группировка студентов по оценке с агрегацией максимального имени в каждой группе.

***8. Inline Count*** *(получение количества записей внутри ответа):*

· http://localhost:5000/odata/Students?$top=2&$count=true - получение первых двух студентов и общего количества студентов.

**Порядок разработки Data Services:**

1. Создание БД
2. Создание приложения ASP.NET
3. Создание EDM модели
4. Создание WCF Data Service

**Применение Data Services:**

* простое чтение данных
* параметризированные запросы
* добавление данных
* модификация данных
* удаление данных
* добавление/изменение/удаление связей между объектами

1. WCF: определение, назначение, протоколы, архитектура, конечные точки, контракты.

**WCF:** Windows Communication Foundation; технология основанная на .NET FRAMEWORK для разработки приложений SOA-архитектуры, первая версия 2006 (Indigo).

**WCF: основные принципы**

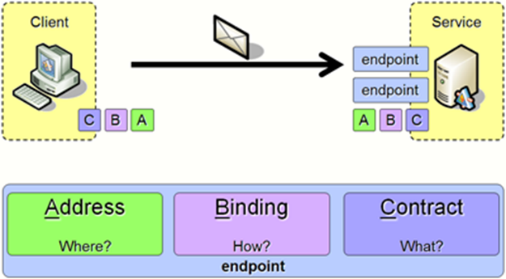
-разработка сервиса должна быть простой и иметь способность к расширению его функциональных возможностей;

-   один API для всех коммуникационных протоколов;

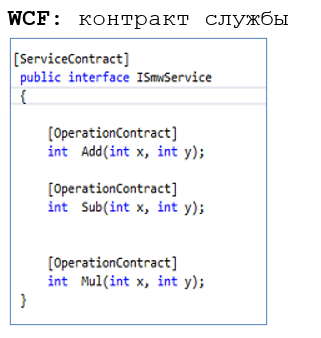
-   сервис должен быть интероперабельным (функционировать по отрытым телекоммуникационным стандартам);

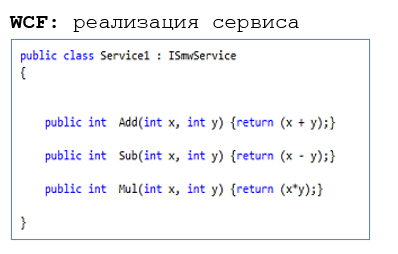
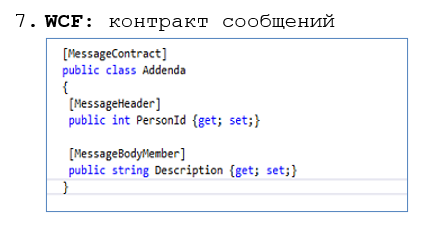
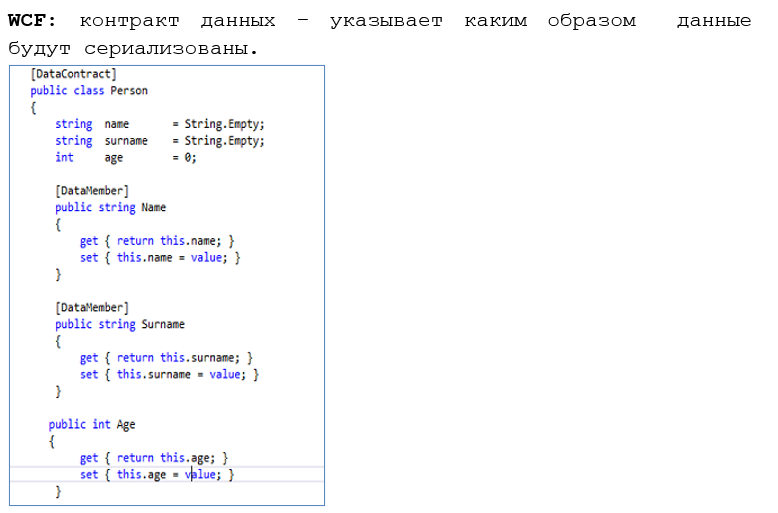
-   сервис должен поддерживать стандарты WS-\*;

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

****

**WCF: контракты**



****

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

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

**привязка** – инкапсулирует все технологии, позволяющие переправить сообщение от одной конечной точке к другой (транспорт, кодировка, протоколы (безопасность,сессия), правила обмена сообщениями (размер, timeout)).

**WCF: привязка: безопасность**: WS-ReliableMessaging;

**WCF: привязка: безопасность**: WS-Security

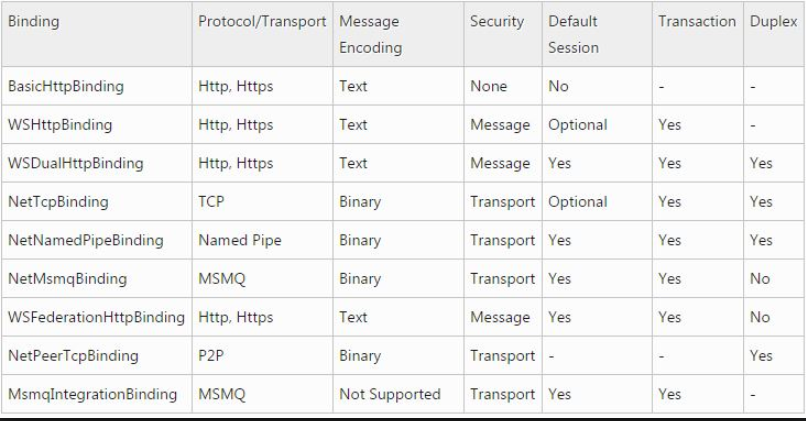
**WCF: привязка: кодирование:**

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

**WCF: привязка: транспорт:**

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

**WCF: привязка: стандартные:**

****

Основные отличия от ASMX-сервисов:

1. Протоколы: ASMX-сервисы используют протоколы SOAP и HTTP, в то время как WCF-сервисы могут использовать различные протоколы, такие как SOAP, HTTP, TCP, и т. д.

2. Гибкость настройки: WCF предоставляет большую гибкость настройки и конфигурирования, чем ASMX. Например, в WCF можно настроить различные протоколы, форматы сообщений, безопасность, маршрутизацию и т. д.

3. Поддержка различных стандартов: WCF поддерживает различные стандарты, такие как WS-\*, WS-Security, WS-ReliableMessaging и другие, что делает его более гибким для разработки сложных интеграционных сценариев.

**Поведение WCF-сервиса**

**Instance Content Mode** определяет, как и когда создаются экземпляры сервиса. Это можно настроить с помощью атрибута ServiceBehavior и свойства InstanceContextMode

**В режиме Per Call** для каждого вызова клиента создается новый экземпляр сервиса. Это означает, что каждый запрос обрабатывается отдельным экземпляром сервиса, который уничтожается после завершения обработки запроса.Этот режим подходит для сценариев, где состояние между вызовами не требуется.

**В режиме Per Session** для каждого клиента создается отдельный экземпляр сервиса, который сохраняется на протяжении всей сессии. Это позволяет сохранять состояние между вызовами в рамках одной сессии.Этот режим требует использования привязок, поддерживающих сессии, таких как NetTcpBinding или WSHttpBinding. Для использования режима Per Session необходимо использовать привязки, поддерживающие сессии. Например, NetTcpBinding и WSHttpBinding поддерживают сессии, в то время как BasicHttpBinding не поддерживаетЭто важно учитывать при настройке сервиса, чтобы обеспечить корректную работу сессий.

**В режиме Single** создается один экземпляр сервиса, который обслуживает все запросы от всех клиентов на протяжении всего времени работы приложенияЭтот режим подходит для сценариев, где требуется общий доступ к состоянию или ресурсам.

**Concurrency Mode** определяет, сколько потоков может одновременно выполнять код в экземпляре сервиса

**1. Single**: Каждый экземпляр службы обрабатывает одно сообщение за раз. Это режим по умолчанию. Он обеспечивает простоту и безопасность, так как не требует синхронизации между потоками.

**2. Multipl**e: Каждый экземпляр службы может обрабатывать несколько сообщений одновременно. Этот режим требует, чтобы реализация службы была потокобезопасной, так как несколько потоков могут обращаться к одному и тому же экземпляру службы одновременно.

**3. Reentrant**: Каждый экземпляр службы обрабатывает одно сообщение за раз, но может принимать повторные вызовы. Это означает, что служба может быть вызвана повторно, пока она выполняет другой вызов, что полезно для сценариев, где служба вызывает другие службы

**serviceThrottling** позволяет ограничивать количество одновременно обрабатываемых запросов, сессий и экземпляров сервиса. Это помогает управлять производительностью и ресурсами сервиса

**binding** определяет, какие протоколы и механизмы безопасности будут использоваться для взаимодействия между клиентом и сервисом. Привязки могут быть настроены для поддержки различных сценариев, таких как HTTP, TCP, и MSM

**Безопасность WCF-сервиса**

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

WCF: **Безопасность транспорта** (TLS/SSL) обеспечивает защиту канала связи, по которому передаются сообщения. Это достигается с помощью протоколов, таких как HTTPS, TCP, IPC и MSMQ.

Основные характеристики:

**Шифрование**: Все данные, передаваемые по каналу, шифруются, обеспечивая конфиденциальность.

**Целостность:** Гарантируется, что данные не были изменены в процессе передачи.

**Аутентификация:** Обеспечивается взаимная аутентификация между клиентом и сервером

WCF: **Безопасность сообщений**, спецификация WS-Security (цифровая подпись и шифрование фрагментов конверта) обеспечивает защиту самих сообщений, а не канала связи. Это достигается с помощью спецификации WS-Security, которая добавляет к сообщениям SOAP элементы для обеспечения конфиденциальности, целостности и аутентификации.

Основные характеристики:

**Конфиденциальность**: Сообщения могут быть зашифрованы, чтобы их содержимое было доступно только получателю.

**Целостность:** Сообщения могут быть подписаны, чтобы гарантировать, что они не были изменены.

**Аутентификация:** Включает в себя аутентификацию отправителя и получателя на уровне сообщений

WCF: **Гибридная модель безопасности**.

Порядок разработки WCF/RPC и WCF/REST-сервисов:

При разработке WCF/RPC-сервисов, разработчик определяет контракт WCF-сервиса с использованием атрибутов, конфигурирует протоколы обмена сообщениями (например, HTTP или TCP) и настраивает поведения и безопасность. Затем реализуется логика операций сервиса.

При разработке WCF/REST-сервисов, разработчик определяет контракт WCF-сервиса с использованием атрибута [WebGet] или [WebInvoke], который указывает на REST-операции, и настраивает протоколы и поведения для работы с REST-запросами. Затем реализуется логика операций сервиса.

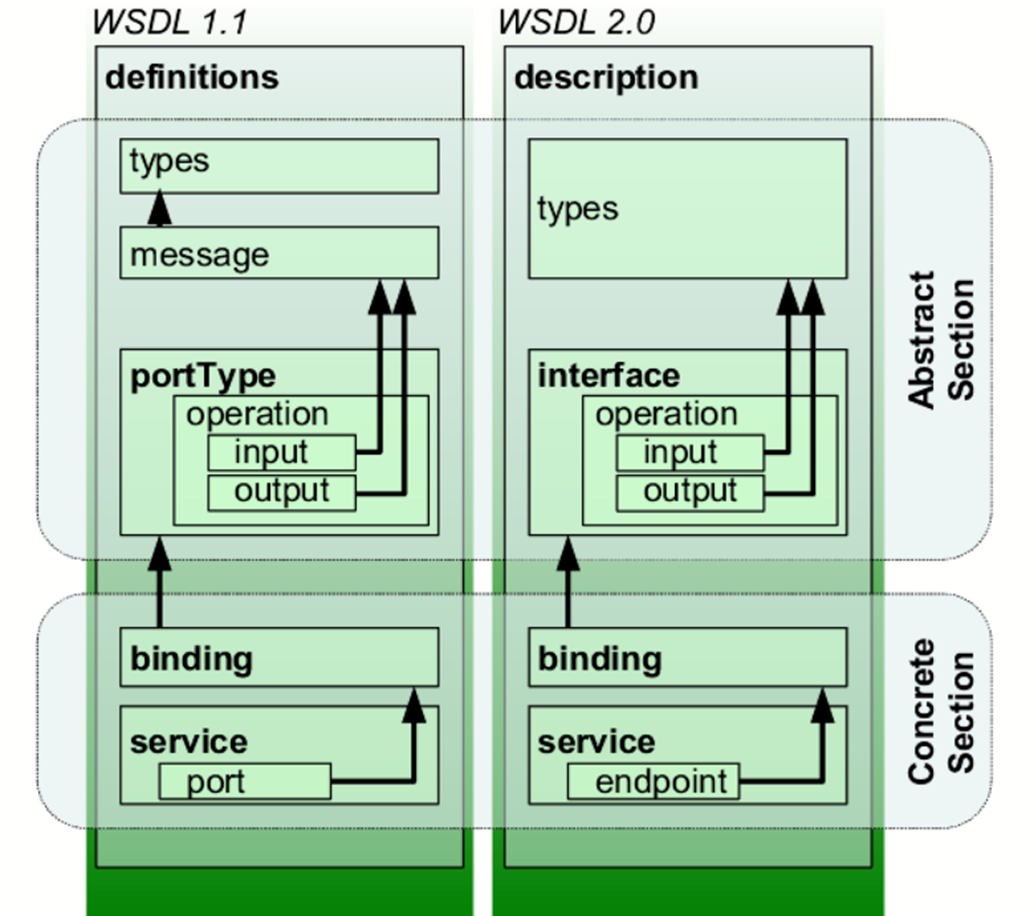
Разработка WCF-сервиса с несколькими конечными точками:

WCF позволяет определить несколько конечных точек (endpoints) для одного и того же сервиса. Каждая конечная точка может использовать разный протокол, формат сообщений, поведение и безопасность. Это позволяет сервису предоставлять различные интерфейсы и протоколы для разных клиентов. Для этого используется конфигурация WCF-сервиса, где задаются различные конечные точки и их свойства.

1. WSDL: определение, назначение, структура WSDL-документа.

**WSDL: Web Services Description Language,** разработанаIBM (NASSL), Microsoft (SCL), webMethods (WIDL), этосредство XML-описания интерфейса для доступа к web-службе; платформонезависимое описание;

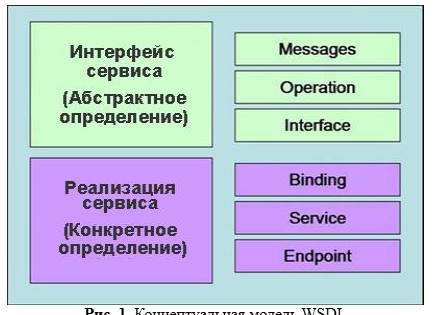
**Стандарты и версии:** **WSDL:** версия 1.1(2001), **версия 2.0**(2007, статус рекомендации).

****

**Концептуальная модель**:

 Описание Web-сервиса можно разделить на две части. В **абстрактной части** описания Web-сервис описывается в языке WSDL с помощью системы типов, обычно W3C XML-схемы, в терминах сообщений, которые этот сервис отправляет и получает. Шаблоны обмена сообщениями определяют последовательность и количество сообщений. Элемент **operation** связывает шаблоны обмена сообщениями с одним или несколькими сообщениями. Элемент **interface** группирует операции (элементы **operation**) независимо от транспорта и способа доставки.

В **конкретной части** описания элементы **binding** задают транспорт и формат доставки для интерфейсов (элементов **interface**). Элемент сервиса (элемента **service**) **endpoint** связывает сетевой адрес в соответствие со связыванием (элементом **binding**). Наконец, элемент **service** группирует конечные точки (элементы **endpoint**), которые реализуют общий интерфейс (элемент **interface**). На рисунке 1 изображена концептуальная модель компонентов WSDL.



**Пространства имён**:

1. **wsdl** (WSDL 1.1)
   * **Описание**: Основное пространство имен для элементов языка WSDL версии 1.1. Оно определяет структуру документа WSDL, включая элементы <definitions>, <types>, <message>, <portType>, <binding>, <service>, и т. д.
2. **wsdli** (WSDL Instance)
   * **Описание**: Это пространство имен используется для атрибутов, которые указывают на экземплярные данные WSDL-документа. Оно может включать информацию, связанную с реализацией конкретного WSDL-документа.
3. **wsdlx** (WSDL Extensions)
   * **Описание**: Пространство имен, определенное для расширений WSDL. Оно позволяет добавлять новые конструкции и элементы в WSDL, которые могут быть специфичны для определенных реализаций или функциональных расширений.
4. **wrpc** (Web RPC)
   * **Описание**: Пространство имен для WSDL-описаний, использующих RPC (Remote Procedure Call) стиль взаимодействия. Оно определяет дополнительные элементы и атрибуты для поддержки вызовов удаленных процедур в WSDL, например, для SOAP-сообщений в RPC-стиле.
5. **soap** (SOAP Binding)
   * **Описание**: Пространство имен для элементов, которые описывают привязку SOAP в WSDL, например, <soap:binding>, <soap:operation>, <soap:address>. Используется для определения, как операции и сообщения должны быть упакованы и отправлены с использованием протокола SOAP.
6. **http** (HTTP Binding)
   * **Описание**: Пространство имен для описания привязки HTTP в WSDL. Определяет элементы для передачи данных с использованием простого HTTP-протокола.
7. **xsd** (XML Schema Definition)
   * **Описание**: Пространство имен для определения схем XML. Используется в разделе <types> WSDL для описания структуры данных, которые обрабатываются веб-сервисом.

**Структура**:

1. **Definitions (Определения)** — корневой элемент WSDL, содержащий все определения.
2. **Types (Типы)** — определяет схемы данных, используемые в сообщениях.
3. **Message (Сообщения)** — определяет сообщения, которые могут содержать несколько частей.
4. **PortType (Интерфейсы)** — описывает набор операций. Для каждого метода сервиса в WSDL-описании будет присутствовать соответствующий раздел <operation>. Внутри раздела <operation> вы найдете разделы <input> и <output>, которые описывают входное и выходное сообщения для соответствующего метода. Входные и выходные сообщения могут ссылаться на определенные типы данных из раздела <types>.
5. **Binding (Связи)** — описывает, как операции будут связаны с конкретным протоколом (например, SOAP).
6. **Service (Сервис)** — определяет конечные точки веб-сервиса.

В **разделе <binding>** WSDL-описания вы найдете информацию о протоколе обмена данных, который используется сервисом. Например, это может быть SOAP (Simple Object Access Protocol) или другой протокол. Вы можете исследовать атрибут binding раздела <binding> для получения информации о протоколе.

**Раздел <service>** в WSDL-описании содержит информацию о самом сервисе. Вы можете исследовать атрибуты и элементы внутри раздела <service>, такие как имя сервиса, адрес (URL) сервиса, используемые протоколы и т.д.

1. JSON-RPC: определение, назначение, структуры запросов и ответов, обработка ошибок, пакетная обработка запросов.

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

* последняя версия 2.0.
* транспорт HTTP/HTTPS или TCP Socket.

**Объект запроса**:

* jsonrpc (версия протокола)
* method (имя вызываемого метода)
* params (параметры вызываемого метода (может быть массивом или объектом))
* id

**Объект ответа**:

* jsonrpc (версия протокола)
* result (результат выполнения метода)
* id
* error (объект ошибки, если метод вызывает исключение)

**Объект ответа (ошибка):**

* code (код ошибки)
* message (описание ошибки)
* data (дополнительные данные об ошибке)

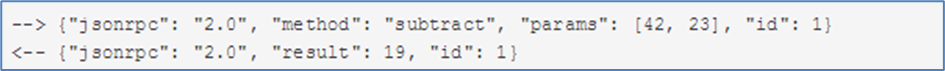
**Пакеты запросов**:

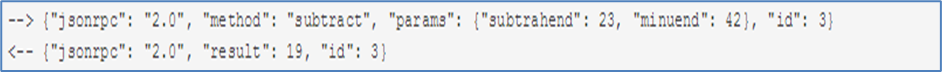
JSON-RPC поддерживает отправку нескольких запросов в одном HTTP-запросе. Каждый запрос в пакете обрабатывается независимо, и в ответе для каждого запроса указывается соответствующий идентификатор.

**Реализация JSON-RPC**:

1. Создание контроллера, наследующего ApiController
2. Определение классов запроса и ответа: ReqJsonRpc и ResJsonRpc

**В запросах** rpc с позиционными параметрами, rpc с именованными параметрами, rpc с уведомлением (запрос без ответа нет id потому что)





**https://lh7-rt.googleusercontent.com/docsz/AD_4nXfkDkvyIaE4rc2QJVTZV4i718i8RZ7d_BKZUq6rRkYw2nn88MfCshPSHegmmaw1dv4hXSfAA7KKWIW0OADKHo2AFQqYeXjARujgwc0hhHhxFOtwdjGR6gVHT7H4tJ778Ugm7JsZ?key=3ZMcOXNgOONnP4IJ5OasaAqK**



1. Микросервис: определение, назначение, основные принципы разработки, оркестрация и хореография сервисов; DevOps, CI/CD.

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

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

***Основные принципы разработки*** - сервисы должны быть легкими (сервисы маленькие и протоколы легковесные). Легкий сервис – сервис, который может быть переписан за 2 недели (Real Estate)

1) устойчивость к сбоям;

2) облегчить понимание и поддержку кода;

3) усилить работу команды программистов.

***Паттерны разработки:*** популярные паттерны ***Tolerant Reader***, ***Consumer Driver***.

***DevOps для микросервисов:*** требуется ***DevOps*** (Development & Operation)- 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.

**Микросервисы:** автоматизация, продукты DevOps: ***Continuous Delivery***, ***Continuous Integration***.

**Микросервисы:** клиенты работающие с микросервисом могут реализованы с помощью ***оркестрового*** (прямые вызовы сервисов) или ***хореографического*** (сервисы подписываются на события клиента) принципов;

автоматизация, продукты DevOps: ***Continuous Delivery***, ***Continuous Integration***.

CI/CD

 CI:

Компиляция кода

Запуск автоматических тестов

Анализ кода (статический и динамический)

Создание артефактов сборки

CD:

Деплой на staging-среду

Проверка и тестирование на staging

Автоматический или полуавтоматический деплой на production

1. DDD/CQRS: определение, назначение, применение для проектирования приложений микросервисной архитектуры. Event Sourcing.

**Domain-Driven Design (DDD)**

**DDD: Domain-Driven Design** -  предметно(проблемно)-ориентированное проектирование: набор принципов направленных на создание  систем объектов со сложной предметной областью.

**DDD:** цель–ускоритьпроцесс проектирование программного обеспечения.

**DDD: Domain** – предметная область. **Эксперт домена** – специалист в предметной области.

**DDD:** 1) создание единого языка домена (Ubiquitous Language).Язык - набор однозначно определенных терминов. **Тезаурус**.

**DDD:** 2) выделение ограниченных контекстов (имена).

**DDD:** 3) определение контрактов (имена) (интерфейсов взаимодействия контекстов); контракты описывают команды (приводит к изменению состояния контекста, могут быть отклонены) и события (сообщение системы об изменении состояния, на них могут подписываться другие контексты, могут быть проигнорированы, но не могут быть отклонены).

8. **DDD:** 4) описание контекстов.

**DDD:** система имеет 4уровня:

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).

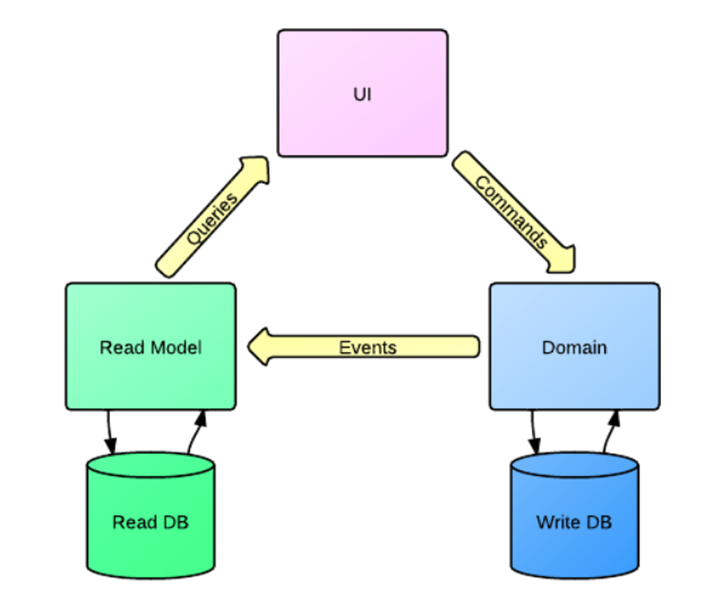
**DDD:** рекомендует паттерны: Value Object, Entity Object, Aggregate (граница транзакции, граница Entity), Repository,  Factory,  Service.

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

**CQRS**: разделение ответственности на команды и запросы ; CQS – command-query separation(Б. Майер) = принцип императивного программирования (команда-память-команда-память, ассемблер – типичный императивный язык). Основной  принцип – команда-чтение не вызывает побочного эффекта(не изменяет состояние  объекта).

контрактное программирование (Б.Майер): спецификация методов (интерфейсы), предусловие, постусловие, инвариант (обязательные соглашения в методе), исключения.

16. CQRS: перенос основных принципов CQS на архитектуру (М.Фаулер)



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

**Event Sourcing (ES)**

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

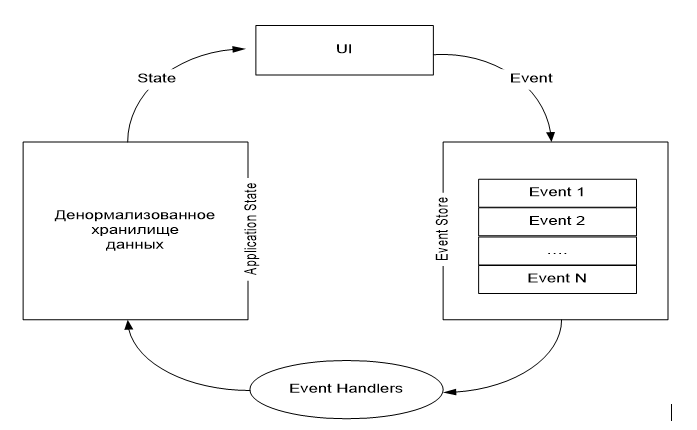
19. **ES:** каждое событие имеет имя.

20. **ES:** произошедшие события неизменны.

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

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

23. **ES:** обычно применяется в архитектуре CQRS для реализации WriteDB.



**EventStore**

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

1. gRPC: определение, назначение, протоколы, Protobuf.

**gRPC:** технология Google RPC (Remote Call Procedure), фреймворк (2015).(транспорт HTTP/2)

**gRPC:** ориентирован для применения в приложениях микросервисной архитектуры, для межсервисного взаимодействия.

**Межсервисное взаимодействие**: Обеспечивает эффективное взаимодействие между микросервисами в распределенных системах.

**Высокая производительность**: Использует HTTP/2 для многопоточности и сжатия данных, что повышает скорость передачи.

**Поддержка различных языков**: Подходит для систем, написанных на различных языках программирования.

**Реализация удаленных вызовов (RPC)**: Предоставляет средства для вызова процедур и обмена данными между клиентом и сервером.

**Бинарная сериализация**: Использует Protobuf для эффективной передачи данных, снижая нагрузку на сеть и ускоряя обработку.

(мб смелик хочет это)

**gRPC:** синхронный и асинхронный вызов удалённых процедур.

**gRPC:** мультиплексирование запросов и ответов (binary framing).

**gRPC:** применение метаданных (ключ-значение).

Протоколы:

**gRPC:** IDL (Interface Definition Language) ( язык спецификаций для описания интерфейсов, синтаксически похожий на описание классов в языке), Protobuf (Protocol Buffer).

Файл .proto, используемый в gRPC, компилируется с помощью компилятора Protocol Buffers (protoc) в код, который содержит как серверные, так и клиентские интерфейсы. Это значительно упрощает процесс разработки, так как автоматически генерируется значительная часть необходимого кода для взаимодействия между клиентом и сервером.

### Что генерирует компилятор Protobuf:

1. **Серверный интерфейс**:
   * Создаются абстрактные классы или интерфейсы, которые разработчики могут реализовать для предоставления серверной логики.
   * В сгенерированном коде будет класс, который нужно наследовать и переопределить методы для обработки запросов.
2. **Клиентский интерфейс**:
   * Создаются классы и методы для клиента, которые позволяют легко отправлять запросы к gRPC-серверу.
   * В сгенерированном клиенте предоставляются методы для вызова удалённых процедур, указанных в файле .proto.

Protobuf:

**Protobuf** (Protocol Buffer) (Буферы протоколов) — это независимый от языка и платформы расширяемый механизм сериализации структурированных данных

это механизм сериализации данных, разработанный Google. Protobuf используется для описания схем данных и их сериализации/десериализации при передаче между системами.



1. AWS: определение, назначение, сервисы: EC2, S3, лямбда-функции, API Gateway.

**AWS :** Облачная платформа, предлагающая. широкий набор инструментов и сервисов для разработки, развертывания и хранения данных.

AWS — это облачная платформа, предоставляемая Amazon, которая предлагает широкий набор облачных услуг, включая вычисления, хранение данных, базы данных, аналитику, машинное обучение, сетевую инфраструктуру и инструменты для разработки приложений. (ChatGPT)

**AWS**: виртуальные серверы

**AWS**: платформа (базы данных, бессерверные вычисления, средства разработки,…)

EC2:

**AWS:** **EC2** - Elastic Compute Cloud (служба аренды виртуального сервера (хостинг)).

спотовые инстансы – инстансы, не сохраняющие состояние.

**EKS** - Elastic Kubernetes Service(облачный Kubernetes).

**EFS** – Elastic File System (подсоединяемые к виртуальным машинам файловые системы).

1. **Route 53** - облачный DNS-сервер.
2. **Kinesis** - облачный аналог Kafka.

S3:

**AWS: S3 -** объектное хранилище, рассчитанное на хранение и извлечение любых объемов данных из любого расположения в Интернете.

**AWS: S3 -** хранилище объектов на основе ключа, ключ – любое строковое значение, объекты могут описаны с помощью тегов.

для доступа к S3 применяются два программных интерфейса: REST, SOAP.

Лямбда-функции:

**AWS: Lambda** – бессерверные вычисления (функция как услуга). Event + Req Res

**AWS Lambda:** сервис AWS (сервис бессерверных вычислений), который запускает программный код в ответ на определенные события: HTTP-запрос(**API Gateway**), изменение объектов в **Amazon S3**, обновление таблиц **Amazon Dynamo DB**, ...

API GATEWAY:  
**AWS API Gateway:** сервис AWS, предназначенный для создания, публикации и мониторинга доступа к объектам AWS(данным, Lambda-функциям,…). Позволяет создавать RESTful и WebSocket API.

**AWS API Gateway:** создание API Gateway (с именем APIsmwLambda) для доступа к Lambda-функции (с именем smwLambda).

4 шага:

1) имя;

2)конфигурирование типов маршрутов;

3) стадия разработки;

4)просмотр перед созданием.

1. Kafka: определение, назначение, основные компоненты, принципы работы.

**программный брокер сообщений**, распределенная, горизонтально масштабируемая  система; применяется в приложениях  микросервисной архитектурой; осуществляет централизованную координацию сообщений; надежная очередь,    pub-sub, потоковая обработка, кластерная архитектура, репликация в кластере, транзакции, горизонтальная масштабируемость,  протоколы работы, механизмы обеспечения надежности**.**



**topic:**тема:именованныйпоток сообщений.

**producer:**издатели сообщений.

**consumer –** подписчики**.**

**partition: раздел темы –** логическаяточка подключения consumer, для распараллеливания обработки сообщений темы.

**смещение (offset)** - номер(индекс) сообщения в partition.

**commit –** фиксация чтения; продвижение offset**.**

**consumer group** – группа подписчиков; для разделения (распараллеливания) запросов, для каждой группы свое смещение. **partitions > consumers(group), partitions = consumers(group), partitions < consumers(group).**

**streams:** механизм обработки потока сообщений; фильтрация (filter), преобразования (map), агрегация (count, sum, avg),запись с другую тему (out-topic).

**брокер: узел:** системадляпредоставления опубликованных данных. 

**leader –** узел(сервер), для чтения/записи в раздел**.**

**follower –** узел(сервер), для репликации данных.

1. OSGi: определение, назначение, основные компоненты, принципы работы.

**Open Services Gateway initiative** – инициатива доступа к открытым службам.

**OSGi Alliance** – организация, разрабатывающая спецификацию платформы OSGi

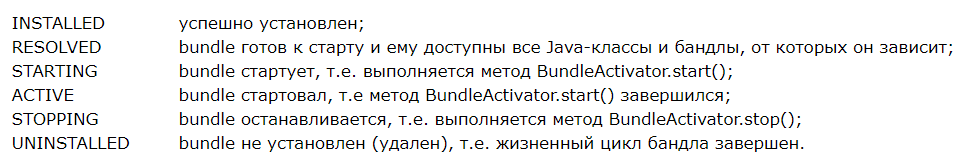
- спецификация динамической модульной системы и сервисной платформы для Java-приложений. Позволяет построить приложение из компонентов, которые можно динамически переинсталлировать компоненты (модули).

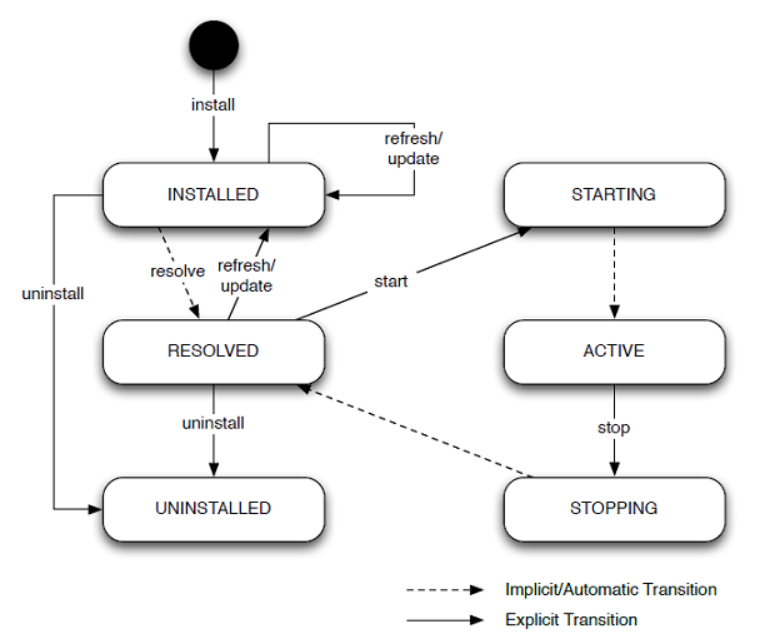
- изначально спецификация разрабатывалась для встроенных систем (BMW, Siemens), затем распространилась на многофункциональные приложения.

**OSGi:** процесс

* разрабатывается ***модуль*** (bundle, jar);
* модуль может быть ***динамически подсоединен*** к системе модулей, работающей на платформе OSGi;
* модуль может быть ***динамически отсоединен*** от системы модулей, работающей на платформе OSGi.

**жизненный цикл OSGi bundle** (интерфейс **BundleActivator**).

****

****

**OSGi-bundle** – это jar-архив, включающий дополнительные данные.

**взаимодействие OSGi-модулей**:

* ***ресурсные bundle*** – централизованное хранилище, предназначенное для хранения данных;
* ***сервисные bundle*** – классы, с методами, которые можно вызывать (должен быть зарегистрирован в реестре сервисов, позднее связывание);
* ***издатель/подписчик***.

**Felix:** фреймворк, реализующий стандарт OSGi.

основные компоненты

**Bundles** –  это компоненты OSGi, создаваемые разработчиками. Они представляют собой JAR-файлы с дополнительными метаданными, описывающими их зависимости и возможности.

**Services** – Слой сервисов связывает бандлы динамическим образом, предлагая модель публикации, поиска и привязки для обычных Java-объектов.

**Life-Cycle** –  API для установки, запуска, остановки, обновления и удаления бандлов.

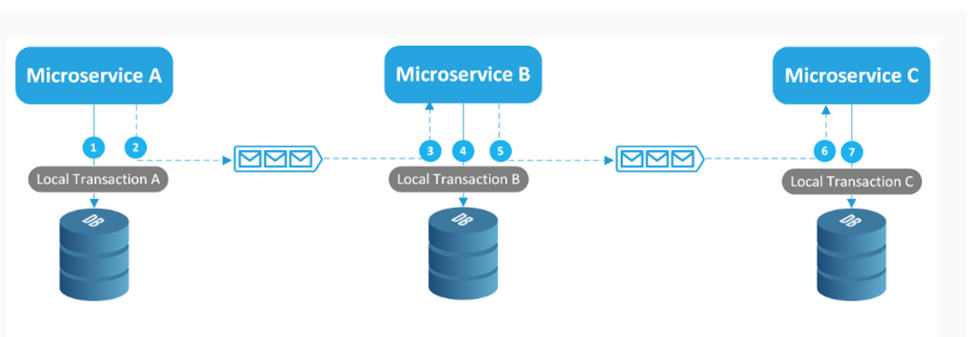
**Modules** –  Слой, определяющий, как бандл может импортировать и экспортировать код.

**Security** –  Слой, отвечающий за аспекты безопасности.

**Execution Environment** – Определяет, какие методы и классы доступны в конкретной платформе

1. Паттерн Saga: определение, назначение, применение.

**Saga:** Сага, для распределенной транзакции, изменение по очереди в локальных базах данных, сообщение в другой сервис, выполнение изменение в другой базе данных, ошибка – компенсирующие операции (откат). **Хореография** – децентрализованная координация, микросервис ждет-принимает сообщение (событие), выполняет (или нет) соответствующее действие. **Оркестровка** – централизованная координация, центральный компонент (сервис) отсылает сообщение другим сервисам команды-сообщения, сервисы выполняет соответствующее действие.

****

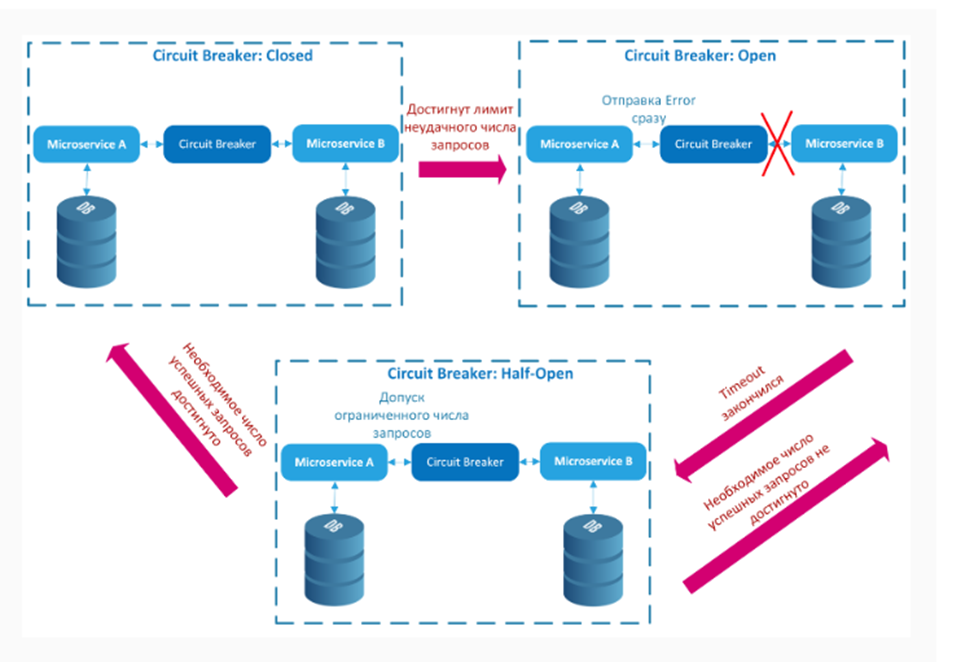
1. Паттерн Circuit Breaker: определение, назначение, применение.

**Circuit Beaker**: автоматический выключатель- промежуточный сервис, для снижения непроизводительной нагрузки, три режима:

**Closed:** передача запросов между сервисами, подсчет  сбоев; если число сбоев  превышает порог, выключатель  переводится в состояние Open;

**Open:** запросы немедленно возвращаются с ошибкой; через тайм-аут выключатель переводится в состояние Half-Open.

**Half-Open:** выключатель пропускает ограниченное количество запросов подсчитывает число успешных запросов; если необходимое количество достигнуто, выключатель переходит в состояние Closed, если нет — возвращается в Open.



1. Паттерн Log Aggregation: определение, назначение, применение.

**Log Aggregation** — это паттерн в архитектуре микросервисов, который организует сбор, централизованное хранение и анализ логов, генерируемых различными микросервисами. Вместо того чтобы изучать логи каждого сервиса отдельно, они агрегируются в одном месте для удобства обработки и анализа.

**Назначение:**

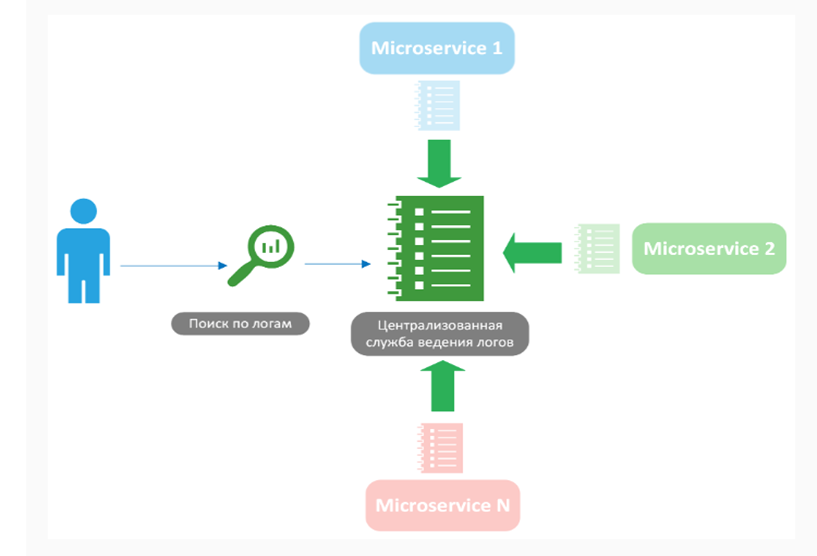
* **Централизация логов:** Обеспечивает сбор логов из всех микросервисов в единую систему для удобного доступа и управления.
* **Мониторинг и отладка:** Упрощает отслеживание ошибок, исключений и производительности за счёт единого хранилища данных.
* **Корреляция событий:** Позволяет связывать логи из различных сервисов для анализа цепочки запросов или выявления взаимосвязанных проблем.
* **Анализ и визуализация:** Помогает отслеживать метрики, такие как время отклика, частота ошибок и загрузка системы.
* **Соблюдение требований к хранению данных:** Выполняет задачи по архивации и ретенции логов для соответствия регуляторным требованиям.

**Применение:**

1. **Процесс Log Aggregation:**
   * **Сбор логов:** Логи собираются из различных источников, таких как текстовые файлы, стандартные потоки вывода (stdout/stderr) или системные журналы.
   * **Транспортировка логов:** Логи отправляются в централизованную систему с помощью агентов, таких как Fluentd, Logstash или Filebeat.
   * **Обогащение и нормализация:** Логи дополняются метаданными (таймстампами, идентификаторами сервисов, уровнем логирования) и форматируются для унификации.
   * **Хранение:** Логи сохраняются в базе данных или специализированном хранилище, например, Elasticsearch.
   * **Анализ и визуализация:** С использованием инструментов, таких как Kibana или Grafana, логи анализируются и визуализируются.
2. **Реализация:**
   * **Агент логирования:** На каждом узле микросервиса запускается агент (например, Filebeat или Fluent Bit), который отправляет логи в центральную систему.
   * **Хранилище логов:** Используются базы данных, оптимизированные для обработки логов (Elasticsearch, Splunk, OpenSearch).
   * **Интерфейс анализа:** Веб-интерфейсы (например, Kibana) предоставляют доступ для поиска, анализа и визуализации логов.
3. **Инструменты для Log Aggregation:**
   * **Logstash:** Инструмент для обработки, фильтрации и доставки логов в хранилище.
   * **Elasticsearch:** Поисковая и аналитическая система, которая часто используется для хранения агрегированных логов.
   * **Kibana:** Инструмент для визуализации и анализа логов.
   * **Fluentd/Fluent Bit:** Инструменты для сбора и отправки логов.
   * **Graylog:** Платформа управления логами с широким функционалом.
   * **Splunk:** Коммерческая система анализа логов с мощными аналитическими функциями.
4. **Пример конфигурации Log Aggregation с ELK Stack:**
   * Микросервисы генерируют логи в формате JSON.
   * Filebeat передаёт логи из контейнеров Docker на Logstash.
   * Logstash обрабатывает, фильтрует и обогащает логи.
   * Логи сохраняются в Elasticsearch.
   * Kibana используется для анализа и визуализации логов.
5. **Преимущества Log Aggregation:**
   * Упрощённая диагностика и устранение неисправностей.
   * Централизованный доступ к логам, независимо от масштабов системы.
   * Возможность анализа сложных сценариев, затрагивающих несколько сервисов.
   * Удобство мониторинга метрик и их визуализации.
6. **Практическое применение:**
   * **Анализ производительности:** Отслеживание узких мест в цепочке запросов.
   * **Устранение ошибок:** Быстрый поиск корневой причины сбоя или исключения.
   * **Инцидент-менеджмент:** Использование логов для восстановления системы после инцидентов.
   * **Аудит:** Хранение логов для выполнения требований безопасности и соответствия.

**В лекции**

**Log Aggregation:** агрегация журналов, централизованный журнал для всех сервисов.



1. Паттерн Health Check: определение, назначение, применение.

**Health Check** — это паттерн в архитектуре микросервисов, который обеспечивает мониторинг состояния и работоспособности микросервисов. Health Check представляет собой механизм проверки здоровья сервиса, позволяющий определить, доступен ли он, правильно ли функционирует и готов ли обрабатывать запросы.

**Назначение:**

* **Поддержание стабильности системы:** Обеспечение актуальной информации о состоянии микросервисов для своевременного реагирования на сбои.
* **Балансировка нагрузки:** Предоставление информации о доступности сервиса балансировщикам нагрузки для предотвращения направления запросов на неисправные сервисы.
* **Автоматическое масштабирование:** Помощь в принятии решений о необходимости добавления или удаления инстансов сервиса в кластерных системах.
* **Обеспечение высокой доступности:** Выявление и устранение проблем в работе системы до того, как они приведут к отказу.
* **Интеграция с системой мониторинга:** Обеспечение данных для мониторинга метрик производительности и доступности микросервисов.

**Применение:**

1. **Типы проверок здоровья:**
   * **Liveness Check:** Проверяет, жив ли сервис и способен ли продолжать работу.
   * **Readiness Check:** Проверяет, готов ли сервис принимать входящие запросы. Это важно при запуске или обновлении сервиса.
   * **Startup Check:** Проверяет успешность инициализации сервиса, что используется для задержки обработки запросов до полной готовности.
2. **Реализация:**
   * В микросервис добавляется эндпоинт (например, /health), который возвращает информацию о состоянии сервиса.
   * Ответ эндпоинта может быть в формате JSON с информацией об отдельных компонентах или в виде простого статуса ("healthy", "unhealthy").
   * Инструменты оркестрации, такие как Kubernetes, используют эти проверки для определения состояния подов.
3. **Интеграция:**
   * **Балансировщики нагрузки:** Применяются для перенаправления запросов только на "здоровые" инстансы.
   * **Системы мониторинга:** Взаимодействие с такими инструментами, как Prometheus, Grafana или ELK Stack, для визуализации и анализа состояния системы.
   * **Автоматическое устранение неисправностей:** Использование информации о здоровье для перезапуска или замены неисправных сервисов.
4. **Пример реализации Health Check:**В gRPC или REST-сервисах часто создают эндпоинты /health или /ready. Например, в Spring Boot это может выглядеть так:

@RestController

 @RequestMapping("/health")

 public class HealthCheckController {

@GetMapping

 public ResponseEntity<String> healthCheck() {

return ResponseEntity.ok("healthy");

 }

 }

**Назначение**

* Обеспечивает информацию о текущем состоянии сервиса.
* Помогает быстро обнаружить и устранить проблемы.
* Поддерживает высокую доступность и надежность системы.

**Применение**

1. **Определение состояния**: Сервисы предоставляют API-эндпоинт (например, /health), который возвращает информацию о доступности и статусе ключевых компонентов.
2. **Проверки**: Включают проверки:
   * Подключения к базе данных.
   * Состояния очередей сообщений.
   * Доступности внешних API.
3. **Мониторинг**: Используется инструментами вроде Kubernetes, Consul, или Load Balancer для автоматического исключения некорректных узлов.

**Пример**

Эндпоинт /health:

json

Копировать код

{

  "status": "UP",

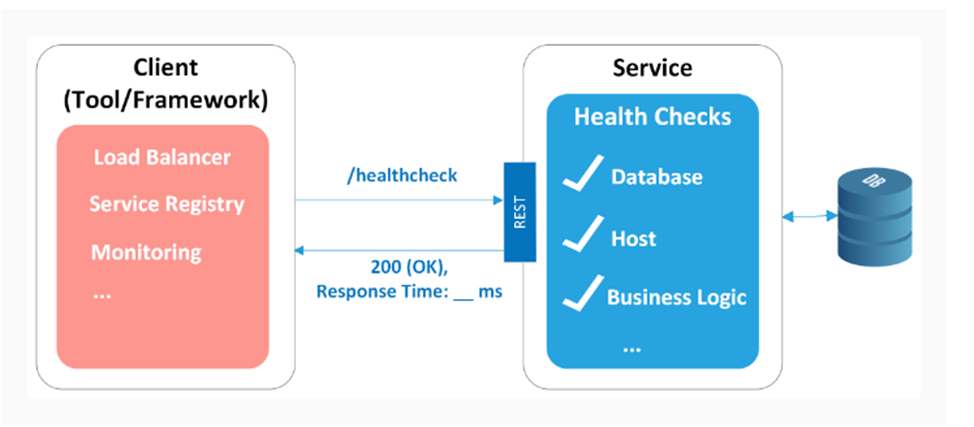
  "db": "connected",

  "queue": "healthy"

}

**В лекции**

**Health Check:** проверка здоровья



Доцент кафедры программной инженерии В.В. Смелов