Оглавление

[1. Протокол HTTP: клиент-сервер; типы сообщений, структура запроса, структура ответа, статус (серии значений), методы, заголовки, параметры. Понятие stateless-протокола. 2](#_Toc105587078)

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

**SOA** – архитектурный стиль написания распределенных компонентов, каждая компонента представляет собой сервис

**Свойства SOA?**

* - независимость от аппаратной реализации
* - независимость от оперционноый системы
* - независимость от используемого ЯП
* - масштабируемость

**Сервис** – это приложение, которое предоставляет интерфейс для других приложений  
является поставщиком услуг для других приложений

**Свойства сервисов SOA?**

* - сервис ориентирован решение бизнес логики
* - сервис автономен
* - повторно используется
* - инструкция сервиса описана в терминах SLA (Service Level Agreement)
* - сервис виден (доступен)

интерфейсы? (3 на основе которых можно построить сервисы) – **rest, soap, RPC**

**Специальные компоненты SOA**:

* **ESB** (enterprise service bus) **–** компонента, обеспечивающая обмен сообщениями
* - реестр сервисов
* - Workflow Engine - сервис использующий несколько других сервисов
* - Service broker - координирует работу
* - SOA supervisor - мониторит служебные сервисы
* - Identity Service - сервис аутентификации

**Способы клиент-серверного взаимодействия:**

* - симплекс
* - полудуплекс
* - дуплекс
* - потоковая связь
* - издатель-подписчик

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

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

**REST API** – это архитектурный стиль написания серверного web-приложения (способ взаимодействия), который представляет собой набор конечных точек, для передачи данных может использоваться xml либо json формат

**6 основных требований**, которым должно удовлетворять **rest-full приложение**

* - клинет-сервеное взаимодействие
* - отсутсвие состояния
* - кэширование
* - единообразный интерфейс
* - слоистая архитектура
* - код по требованию

**hateaos** – это механизм, который позволяет передавать ссылки на другие сервисы чтобы можно было уточнить ответ (позволяет отправлять не только данные, но связанные действия)

нужен для того, чтобы клиента отвязать от местоположения других сервисов  
чтобы уменьшить связность между клиентом и сервером

На каких платформах мы можем разрабатывать REST API

* - WebAPi
* - сервлеты
* - http-handler

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

**SOAP** – протокол прикладного уровня, по которому web-сервисы взаимодействуют друг с другом в формате xml   
в качестве транспорта может выступать http, tcp, smtp

http: envelope передается в теле  
как указать что в теле envelope (content-type: soap+xml)

tcp: самим нужно продумать, что будет является маркером soap сообщения

SOAP сообщение представляет собой envelope

**Структура envelope**

* soap header (soap header необязателен)
* soap body
* в случае ошибки soap fault

**Узел** – компонент SOA-архитектуры, который может отправлять или получать SOAP-сообщения

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

* Нет роли
* Промежуточный слой
* Конечный получатель

замечательное свойство soap: может расширяться за счет namespace  
на основе soap можно разрабатывать собственные протоколы

**пространства имен soap:**

* soap-envelope
* soap-encoding
* soap-rpc

**Расширения SOAP**

* WS-Security - позволяющий работать с шифрованием и электронными подписями
* WS-Policy

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

**WSDL** - язык описания сервисов, позволяет на его основе сгенерировать клиент (proxy)

**версии WSDL:**

* 1.1 - есть разделение на типы и сообщения
* 2.0 - используются только типы

**Структура WSDL документа**:

* - типы данных
* - типы сообщений
* - порты (методы)
* - биндинги (указывают как обращаться к серверу, какой транспорт использовать)
* - описание местоположения сервиса

**Web-сервис** – это web-приложение, которое предназначено для использования другими web-приложениями

**stub** – класс, находящийся на стороне сервиса, отвечающий за распаковку и отправку сообщений

stub также присутствует на стороне клиента, его называют proxy

**wsdl.exe** - на основе wsdl документа генерирует proxy класс через который происходит обращение к сервису  
с ключом /server будет сгенерирован абстрактный класс

**пространства имен wsdl:**

* wsdl-instance
* wsdl-extensions
* wsdl/rpc
* wsdl/soap
* wsdl/http

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

**ASMX-сервис** – это web-сервис, построенный в соответствии с RPC, взаимодействие происходит по протоколу SOAP  
является частым случаем wcf сервиса  
в качестве транспорта использует: http  
требуется iis

Порядок разработки:

* Класс необходимо пометить атрибутом WebService (где задается namespace)
* А методы атрибутом WebMethod (где задается MessageName, через которое будет происходить обращение к методу)

**RPC** – это технология, которая позволяет представить сервер в качестве объекта, у которого мы можем вызывать удаленные процедуры

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

**wcf** - платформа для разработки приложений SOA (сервис ориентированной архитектуры), в основе лежит soap протокол

**wcf-сервис** – это web-сервис, который может быть построен в соответствии с RPC или REST, в основе которого лежит soap протокол

**главные особенности:**

* - может иметь несколько точек доступа (за счет нескольких транспортов)
* - поддерживается self-hosting (можно встроить в iis либо в обыкновенное консольное приложение)
* - транспорты: tcp, msmq, nimed pipe, http

Есть **стандартные привязки**, которые представляют собой комбинацию протоколов, как пример (**BasicHttpBinding**)

**Коммуникационная модель:**

клиент обращается к WCF-сервису и получает ответ (для обращения необходимо знать адрес, биндинг(протокол) и контракт точки)

**WCF-контракты:**

* - контракт службы – представляет собой интерфейс
* - контракт данных – указывает каким образом данные будут сериализованы
* - контракт сообщений – указывает структура SOAP сообщения

**хост** - контейнер для сервиса (может быть консольное приложение либо iis)

для каждой конечной точки должен быть задан: **адрес, биндинг и контракт**

**биндинг** отвечают за то, как взаимодействовать с сервисом  
биндинг позволяют указать:

* - протокол взаимодействия
* - кодировку
* - методы безопасности

**Для разработки WCF/REST сервиса также необходимо:**

* определить интерфейс, указать http методы и URI паттерны
* После чего реализовать данный интерфейс
* далее в app.config определить поведение <webHttp>

Также в поведении можно определить какой **scope у контекста сервиса**:

* - свой конекст на каждый вызов
* - на каждую новую сессию
* - один контекс на все вызовы

**Также в поведении можно указать количество параллельных подключений**

**Безопасность WCF-сервиса**

в биндингах можно задать методы безопасности

* - алгоритм шифрования
* - стандарт http аутентификации
* - транспортная аутентификация на основе сертификтов (протокол TLS)

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

**syndication service** - это реализация wcf сервиса для создания новостных каналов, в качестве формата данных применяется atom либо rss

Стандарты **ATOM**: (version 1.0) формат, предназначенные для описания web-ресурсов  
MIME: application/atom+xml  
ATOM более универсальный и применяется чаще  
Структура:

* Ресурс является **каналом**
* Каждый элемент **item**
* Который имеет **title, link и данные**

**RSS**: (version 2.0) формат, предназначенные для описания web-ресурсов  
MIME: application/rss+xml

Структура:

* Ресурс является **feed**
* Каждый элемент **entry**
* Который имеет **title, link и данные**

**RSS-агрегатор** – web-приложение, которое автоматически с заданным интервалом времени проверяет ресурсы

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

**Data services** – это одна из реализаций wcf, которая позволяет использовать протокол OData

**Open Data Protocol** - протокол, позволяющий выполнять операции над ресурсами и получать результаты в формате xml и json

Начиная с версии 4.0, OData — открытый стандарт, одобренный организацией OASIS.

данный протокол позволяет реализовать доступ к реалиционной базе данных через rest интерфейс  
использую язык запросов, который по мощности близок к sql-запросам (dml и даже ddl возможности реализованы в полной мере)

**Порядок разработки wcf data service:**

* - установить расширение wcf data service
* - сформировать класс contextDB для обращения к базе данных
* - добавить элемент wcfDataService
* - в качестве параметра указать сформированный класс для обращения к бд

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

**json-rpc** - протокол удаленного вызова процедур, в качестве формата передачи данных используется json  
запрос на удаленный сервер выполняется посредством HTTP или TCP-IP(начиная с версии 2.0 – актуальная версия)

**Формат RPC-запроса:**

* - jsonrpc (строка, указывающая на версию JSON-RPC протокола)
* - method (строка с именем вызываемого метода)
* - params (объект или массив параметров)
* - id (числовое значение, которые используется для установки соответствия между запросом и ответом)

**Формат RPC-ответа:**

* - jsonrpc (строка, указывающая на версию JSON-RPC протокола)
* - result (результат вызова удаленной функции)
* - error (объект ошибки, в случае если она произошла)
* - id (тот же числовой идентификатор что и в запросе)

**RPC-уведомление** - отличается от запроса отсутствием свойства id

**Пакет RPC-запросов** – это массив объектов запроса, на который будет получен массив объектов ответа

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

**Nancy** - один из способов реализации сервисов, nancy позволяет создавать отдельные модули, которые независисы друг от друга (аналог OSGi) позволяет в горячем режиме менять модули

**OWIN** – это интерфейс между web-сервером и web-приложением, позволяющие разрабатывать отдельно web-сервер и web-приложение  
представляет из себя конвеер, включающий в себя:

* host
* web-сервер
* middleware
* web-приложение

**Katana** – OWIN-совместимый host

**Хост** – процесс операционной системы управляющий жизненным циклом web-сервера  
**Kestrel** – web-сервер реализующий интерфейс OWIN

**Middleware** – промежуточный слой, который обрабатывает запрос  
цепочка middleware представляет собой конвеер обработки, при отправке ответа формируется обратная петля, и middleware-ы могут изменять response  
каждый middleware может заблокировать дальнейшую обработку и отправить ответ

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

Event-Storing – это подход который зачастую применятся к сервис-ориентированной архитектуре еще чаще в микросервисной архитектуре

есть спец базы данных в которых запоминается само событие

а позже в асинхронном режиме это событие преобразовывается в запрос к реалиционной или нереалиционной базе данных

используется две базы данных: event-storing (хранит события) и база данных которая используется только для чтения,   
для получения текущего состояние необходимо пройтись по событиям и собрать информацию  
могут использоваться агрегаты для промежуточного подведения итогов  
(часто используется с CQRS архитектурой)

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

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

**в чем особенность**

* - устойчивость к сбоям   
  (1. при выходе из строя одного микросервиса остальная часть приложения продолжает работать  
  2. Благодаря брокеру сообщений (шины), упавший сервис сможет обработать поступившие задачи позже
* - горизонтальная масштабируемость  
  (может быть запущено несколько экземпляров одного микросервиса)
* - микросервисы могут быть написаны на разных ЯП
* - микросервисы позволяет разделить одну большую функцию на простейшие
* - низкий порог вхождения   
  (новый разработчик на проекте быстро сможет разобраться в отдельном микросервисе)

**Главная проблема – работа с распределенными транзакциями**

**Архитектурные решения**: REST, PRC

**Популярные паттерны**: **Tolerant Reader**, **Consumer Driver**, **API Gateway**  
**Saga (предназначен для управления распределенными транзакциями в микросервисной архитектуре)**

Для координации транзакций существует два основных способа:

* **Хореография**. Децентрализованная координация, при которой каждый микросервис прослушивает события/сообщения другого микросервиса и решает, следует предпринять действие или нет.
* **Оркестровка**. Централизованная координация, при которой отдельный компонент (оркестратор) сообщает микросервисам, какое действие необходимо выполнить далее.

**DevOps** — это методология по автоматизации сборки и развертывания программного обеспечения  
Одна из практик DevOps — непрерывная интеграция и доставка, т.е. настройка CI/CD пайплайна (Continuous integration и Continuous delivery или CI/CD pipeline) — это методы, которые автоматизируют процесс сборки и развертывания.

**Отличия от SOA:**

* SOA про переиспользование сервисов (повышается связность между сервисами)
* SOA имеет ESB через которую происходит общение между сервисами
* MSA: разбиение на сервисы по контекстам
* MSA: избегание повторного использования (лучше дублирование, а не зависимость между сервисами)
* MSA: общение через брокер сообщений
* MSA: каждый микросервис может быть независимо запущен от других

1. Docker: назначение, архитектура, основные команды. Docker-Compose: назначение, основные команды.

**Docker –** технология для создания и управления контейнерами, которые изолированны друг от друга  
**Docker** –платформа для разработки, доставки и эксплуатации приложений. Основное назначение – упростить развертывание приложения

В распоряжении у докера имеются **образы** и **контейнеры**  
**Образ** – шаблон, из которого создается контейнер,

**Основные команды:**

* docker build – создает образ на основе Dockerfile
* docker run – создает контейнер на основе image
* docker start – запускает остановленный контейнер
* docker stop – останавливает контейнер
* docker rm – удаляет контейнер
* docker rmi – удаляет image

**docker-compose** – это инструмент для работы с мультиконтейнерным приложением.

**Основные команды:**

* docker-compose up – запускает мультиконтейнерное приложением на основе docker-compose.yml файла
* docker-compose down

**Оркестрация контейнеров** – набор действий, направленный на обеспечение доступности, масштабируемости контейнеров и распределении нагрузки между ними.  
Все это предоставляет **Kubernetes** программное обеспечение с открытым исходным кодом

1. Протокол HTTP: клиент-сервер; типы сообщений, структура запроса, структура отвта, статус (серии значений), методы, заголовки, параметры. Понятие stateless-протокола.

**ИНТЕРНЕТ** –сеть построенная на стеке протоколов tcp/ip

**Web-приложение** – это клиент-серверное приложение, взаимодействие в котором осуществляется по протоколу http

**HTTP** – это протокол прикладного уровня, работающий поверх tcp (начиная с версии http3 поверх udp) для передачи данных между клиентом и сервером, характеризуется полудуплексным каналом связи

**Полудуплексный канал связи** – это канал в котором в один момент времени данные могут идти в одном направлении, либо от клиента к серверу, либо от сервера к клиенту

**Типы сообщений**

* **request**
* **response**

**Структура request и response** сообщений одинаковая, включает в себя:  
- \*стартовую строку  
- заголовки  
- \*пустую строку  
- и тело

**request и response отличаются стартовой строкой.  
Стартовая строка request** состоит из:  
- метода  
- URI  
- и версии протокола Http  
**Стартовая строка response** состоит из:  
- версии протокола  
- кода состояния  
- и пояснения к коду состояния

**Заголовки –** это пары ключ-значение, которые хранят мета-информацию о сообщении(пакете)

**Типы заголовков:**

* **General** – используются в запросах и ответах (примеры: Cache-Control, Upgrade, Connection)
* **Request** – используются в запросах (примеры: Accept, Host, User-Agent)
* **Response** – используются в ответах (примеры: WWW-Authenticate, Location)
* **Entity** – используются для описания тела сообщения (примеры: Content-Type, Content-Language, Content-Length)

**Коды состояний**

* 1xx – информационные сообщения
* 2xx – успешные
* 3xx – переадресация
* 4xx – ошибка клиента
* 5xx – ошибка сервера

**http-методы**

* **Get** – используется для получения данных с сервера
* **Post** – для добавления данных
* **Put** – для обновления данных
* **Patch** – для частичного обновления данных
* **Delete** – для удаления данных
* **Options** – для описания параметров соединения с целевым ресурсом

**Понятие Stateless протокола**

Данное понятие говорит о том, что сервер не сохраняет никаких данных клиента между запросами, каждый запрос является изолированным друг от друга

1. ASP.NET: публикация ASP.NET-приложения, структура и параметры узла IIS, реальный и виртуальный каталоги, процедура настройки web-узла.

**ASP.NET** – это фреймворк, позволяющий разрабатывать web-приложения на платформе .NET Framework

**Платформа** – это набор библиотек и инструментов

**Типы ASP.NET приложений**

* http-handler
* mvc
* webApi
* asmx

**Публикация ASP.NET приложения**

Подразумевает под собой, настройку web-узла на web-сервере,   
для узла мы должны указать **протокол, port, ip**, так как на одном компьютере может быть несколько сетевых карт, а соответственно несколько ip адресов  
Также мы должны указать физический путь к web-приложению

1. ASP.NET: ASMX-сервисы, WSDL, SOAP, прокси, порядок разработки, принципы применения. Пример.

**ASMX-сервис** – это web-сервис, построенный в соответствии с RPC взаимодействие происходит по протоколу SOAP

**Web-сервис** – это web-приложение, которое предназначено для использования другими web-приложениями

**RPC** – это технология, которая позволяет представить сервер в качестве объекта, у которого мы можем вызывать удаленные процедуры

**Особенности ASMX-сервиса**

* в качестве транспорта используется http (полудуплексный канал связи)
* одна точка доступа
* сервис построен в соответствии с RPC
* в качестве web-сервера может использоваться только IIS web-сервер

**WCF-сервис** – это web-сервис, который может быть построен в соответствии с RPC или REST

**Особенности WCF-сервиса**

* транспорт: http, tcp, named pipe, smtp (соответственно дуплексный, полудуплексный, симплексный и потоковый каналы связи)
* может быть несколько точек доступа (за счет разных транспортов)
* сервис может быть построен в соответствии с RPC или REST
* поддерживается self-hosting

**WCF-сервис**, представляет собой класс; этот класс не может существовать самостоятельно, а должен находиться под управлением некоторого процесса Windows, называемого хостовым процессом.

В качестве хоста может выступать консольное или графическое NET-приложение (автохостинг), IIS.

**WSDL** – это язык описания web-сервисов  
**Структура WSDL**

WSDL-документ можно разбить на три части:

* определение типов данных
* список операций
* способ, которым сообщение будет доставлено

**SOAP** – это протокол, по которому web-сервисы взаимодействуют друг с другом в формате xml  
**SOAP сообщение состоит из:**

*envelope {  
- header  
- body  
- fault (необязательный компонент)  
}*

**Proxy** – класс, упрощающий обращению к web-сервису  
**Задача proxy** – создать иллюзию локальной работы с сервисом  
proxy принимает параметры преобразует их в xml формат генерирует envelope, по сути формирует soap сообщение, отправляте его на сервер, затем принимает soap-сообщение ответа, распаковывает его и выдает ответ

**proxy создается на стороне клиента**

**proxy класс создается на основе WSDL**

на стороне сервера есть аналог proxy называемый **stub**

1. ASP.NET CORE: программная платформа, принципы работы, архитектура. Пример.

**OWIN** – это интерфейс между web-сервером и web-приложением, позволяющие разрабатывать отдельно web-сервер и web-приложение  
представляет из себя конвеер, включающий в себя:

* host
* web-сервер
* middleware
* web-приложение

**web-сервер не может существовать самостоятельно**, а должен находиться под управлением некоторого процесса Windows, называемого хостовым процессом.

**Katana** – OWIN-совместимый host

**Хост** – процесс операционной системы управляющий жизненным циклом web-сервера  
**Kestrel** – web-сервер реализующий интерфейс OWIN

ASP.NET Core приложения могут работать под IIS (Windows) или под web-сервером **Kestrel**

**Middleware** – промежуточный слой, который обрабатывает запрос  
цепочка middleware представляет собой конвеер обработки, при отправке ответа формируется обратная петля, и middleware-ы могут изменять response  
каждый middleware может заблокировать дальнейшую обработку и отправить ответ

**Session** – серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении (SessionID передается в куки для идентификация клиента)

**Куки (Cookies)** – данные, которые хранит клиент в виде ключ-значение, создается по инициативе сервера (заголовок Set-Cookie), пересылается клиентом (заголовок Cookie)

**CLR** – программа интерпретатор, является COM компонентом, имеет JIT компилятор

**FCL** – библиотека классов

**Каждый узел** – это отдельное web-приложение

**Переадресация**

**301 и 308 – постоянные перенаправления**

**302, 303 и 307 – временные перенаправления**

**307 и 308 не изменяют метод и тело запроса**