

1.	Общее описание структуры и принципов Веб. История Интернет. OSI.	3
2.	Структура и принципы WWW. DNS и IP. Протоколы Интернет прикладного уровня	
3.	TCP/IP. Стандартизации в Интернет. RFC с примерами.....	8
4.	Протокол HTTP. Структура запроса клиента. Версии протокола HTTP.....	11
5.	Протокол HTTP. Поля заголовка. Структура ответа сервера. Классы кодов ответа сервера. MIME-тип.	14
6.	Протокол HTTP. Обеспечение безопасности данных при передаче по протоколу HTTP. Cookie.....	19
7.	Общее описание языка разметки XML.	23
8.	Общее описание языка разметки HTML. HTML5.....	26
9.	Объектная модель документа. Область применения. Сравнение XML и HTML	29
10.	Схемы DTD, XDR, XSD с примерами.	33
11.	XSLT и XPath. Применение XSLT	37
12.	Архитектура информационных систем. Централизованная архитектура. Архитектура "файл-сервер". Описание технологии «клиент — сервер». Многоуровневая архитектура клиент-сервер.	40
13.	Архитектура информационных систем. Описание технологии «клиент — сервер». Архитектура распределенных систем. Распределенные системы с репликацией. Архитектура Веб-приложений. SOA.	44
14.	Web-серверы (Apache, Nginx IIS и другие). Примеры, краткая характеристика и сравнительный анализ.....	50
15.	Web-серверы. Конвейер обработки HTTP-запросов в IIS. HTTP.SYS. W3SVC. WAS.	54
16.	Web-серверы. Конвейер обработки HTTP-запросов в IIS. Пул приложений. Домен приложения, приложение.	57
17.	Интеграция и взаимодействие в сети Веб. Веб-сервисы. Спецификация WSDL. Спецификация UDDI.....	58
18.	Шаблон проектирования MVC. Примеры. Достоинства и недостатки.	63
19.	Протокол SOAP. Краткая характеристика. Общая структура SOAP сообщения. Обработка ошибок в SOAP-сообщениях. SOAP-сообщения с вложениями. Сравнительная характеристика SOAP и REST.....	65
20.	REST. Краткая характеристика, методы запроса, архитектурные ограничения, правила REST API. Сравнительная характеристика SOAP и REST. Сравнительная характеристика REST и GraphQL.	69
21.	Обзор серверных фреймворков на основе: Java Script, Python, C#	73
22.	Обзор серверных фреймворков на основе: Java, PHP, Ruby	80

23. Docker. Цель технологии (сравнение с установочным скриптом, виртуальными машинами, облачными сервисами). Основные сведения. Термины и концепции. Файл Dockerfile	87
24. Docker. Цель технологии (сравнение с установочным скриптом, виртуальными машинами, облачными сервисами). Основные сведения. Команды. Работа с данными.	91
25. Docker. Цель технологии (сравнение с установочным скриптом, виртуальными машинами, облачными сервисами). Kubernetes. Сравнение Kubernetes и Docker Swarm.....	94
26. ORM. Парадигма «несоответствия». Варианты реализации ORM.....	98
27. Entity Framework. Подходы для организации взаимодействия EF с базой данных. Архитектура EF.....	102
28. ADO.NET. Основные отличительные особенности ADO.NET. Уровни в объектной модели ADO.NET.	106
29. Основы языка JavaScript. Особенности синтаксиса JavaScript. Отличие от языков C# и Java.	108
30. Обзор клиентских фреймворков: Angular, React, и Vue.js..	112
31. Обзор клиентских фреймворков. Сравнительный анализ Angular, React, и Vue.js.	117
32. CSS. История CSS. Основные определения. Общий синтаксис таблиц стилей. Типы селекторов с примерами. Работа со шрифтами в CSS. Цветовое оформление в CSS.	122
33. Наследование в CSS. Каскадирование. Аппаратно-зависимые стили.	140
Аппаратно-зависимые таблицы стилей.....	142
34. Обзор существующих CSS-фреймворков.	144
35. AJAX. Принцип работы технологии AJAX. Форматы передаваемых данных. Достоинства и недостатки.	157
36. CMS. Основные сведения. Описание распространённых функций CMS. Достоинства и недостатки.	160
37. Идентификация, авторизация, аутентификация в Web.	162

1. *Общее описание структуры и принципов Веб. История Интернет. OSI.*

Структура и принципы Веб (базовые понятия, архитектура, стандарты и протоколы)

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер.

Основная часть приложения, как правило, находится на стороне веб-сервера, который обрабатывает полученные запросы и формирует ответ, отправляемый пользователю. Полученный от сервера ответ преобразуется в графический интерфейс, понятный пользователю с помощью специальных программ – Браузеров.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты (потребители этих функций).

Клиент – это браузер (в тех случаях, когда один веб-сервер (BC1) выполняет запрос к другому (BC2), роль клиента играет веб-сервер BC1). Для того, чтобы пользователь увидел графический интерфейс приложения в окне браузера, последний должен обработать полученный ответ веб-сервера, в котором будет содержаться информация, реализованная с применением HTML, CSS, JS (самые используемые технологии). Именно эти технологии «дают понять» браузеру, как именно необходимо «отрисовать» все, что он получил в ответе.

Веб-сервер — это сетевое приложение, обслуживающее HTTP-запросы от клиентов, обычно веб-браузеров. Веб-сервер принимает запросы и возвращает ответы, обычно вместе с HTML-страницей, изображением, файлом, медиа-поток или другими данными. Веб-серверы — основа Всемирной паутины.

Сеть WWW образуют миллионы веб-серверов, расположенных по всему миру.

Для идентификации ресурсов (зачастую файлов или их частей) в WWW используются идентификаторы ресурсов URI (Uniform Resource Identifier).

Для определения местонахождения ресурсов в этой сети используются локаторы ресурсов URL (Uniform Resource Locator). Такие URL-локаторы представляют собой комбинацию URI и системы DNS.

Доменное имя (или IP-адрес) входит в состав URL для обозначения компьютера (его сетевого интерфейса), на котором работает программа веб-сервер.

Функционирование сервиса обеспечивается четырьмя составляющими:

- URL/URI — унифицированный способ адресации и идентификации сетевых ресурсов;
- HTML — язык гипертекстовой разметки веб-документов;
- HTTP — протокол передачи гипертекста;
- CGI — общий шлюзовый интерфейс, представляющий доступ к серверным приложениям.

Интернет

Интернет - Это самая большая в мире сеть, не имеющая единого центра управления, но работающая по единым правилам и предоставляющая своим пользователям единый набор услуг. Интернет можно рассматривать как «сеть сетей», каждая из которых управляется независимым оператором – поставщиком услуг Интернета (ISP, Internet Service Provider).

С точки зрения пользователей Интернет представляет собой **набор информационных ресурсов, рассредоточенных по различным сетям**, включая ISP-сети, корпоративные сети, сети и отдельные компьютеры домашних пользователей. Каждый отдельный компьютер в данной сети называется *хостом* (от английского термина host).

Хронология развития Интернета

Год	Событие
1966	Эксперимент с коммутацией пакетов управления ARPA
1969	Первые работоспособные узлы сети ARPANET
1972	Изобретение распределенной электронной почты
1973	Первые компьютеры, подключенные к сети ARPANET за пределами США
1975	Сеть ARPANET передана в ведение управления связи министерства обороны США
1980	Начинаются эксперименты с TCP/IP
1981	Каждые 20 дней к сети добавляется новый хост
1983	Завершен переход на TCP/IP
1986	Создана магистраль NSFnet
1990	Сеть ARPANET прекратила существование
1991	Появление Gopher
1991	Изобретение Всемирной паутины. Выпущена система PGP. Появление Mosaic
1995	Приватизация магистрали Интернета
1996	Построена магистраль OC-3 (155 Мбит/с)
1998	Число зарегистрированных доменных имен превысило 2 млн.
2000	Количество индексируемых веб-страниц превысило 1 млрд.

Интернет является децентрализованной сетью, что имеет свои достоинства и недостатки.

Достоинства:

- Легкость наращивания Интернета путем заключения соглашения между двумя ISP.

Недостатки:

- Сложность модернизации технологий и услуг Интернета, поскольку требуются согласованные усилия всех поставщиков услуг.
- Невысокая надежность услуг Интернета.
- Ответственность за работоспособность отдельных сегментов этой сети возлагается на поставщиков услуг Интернета.
- Существуют различные типы поставщиков услуг Интернета:
- поставщик интернет-контента имеет собственные информационно-справочные ресурсы, предоставляя их содержание в виде веб-сайтов;
- поставщик услуг хостинга предоставляет свои помещения, каналы связи и серверы для размещения внешнего контента;

OSI

Сетевая модель OSI (Open Systems Interconnection) — это концептуальная модель, которая описывает и стандартизирует функции компьютерных систем при их взаимодействии друг с другом. Каждый из семи уровней накладывается поверх предыдущего: от физического до прикладного, взаимодействуя с нижним и предоставляя средства для уровня выше.

Модель				
Уровень (layer)	Тип данных (PDU ^[1])	Функции	Примеры	Оборудование
Host layers	7. Прикладной (application)	Доступ к сетевым службам	HTTP , FTP , POP3 , WebSocket	Хосты
	6. Представления (presentation)	Представление и шифрование данных	ASCII , EBCDIC	
	5. Сеансовый (session)	Управление сеансом связи	RPC , PAP , L2TP	
	4. Транспортный (transport)	Сегменты (segment) / Датаграммы (data gram)	Прямая связь между конечными пунктами и надёжность TCP , UDP , SCTP , PORTS	
Media layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация IPv4 , IPv6 , IPsec , AppleTalk	Маршрутизатор
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация PPP , IEEE 802.22 , Ethernet , DSL , ARP , сетевая карта .	Коммутатор , точка доступа
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными USB , кабель («витая пара», коаксиальный, оптоволоконный), радиоканал	Концентратор

2. Структура и принципы WWW. DNS и IP. Протоколы Интернет прикладного уровня

Структура и принципы WWW

Сеть WWW образуют миллионы веб-серверов, расположенных по всему миру. Веб-сервер является программой, запускаемой на подключённом к сети компьютере и передающей данные по протоколу HTTP.

Для идентификации ресурсов (зачастую файлов или их частей) в WWW используются идентификаторы ресурсов URI (Uniform Resource Identifier). Для определения местонахождения ресурсов в этой сети используются локаторы ресурсов URL (Uniform Resource Locator). Такие URL-локаторы представляют собой комбинацию URI и системы DNS.

Доменное имя (или IP-адрес) входит в состав URL для обозначения компьютера (его сетевого интерфейса), на котором работает программа веб-сервер.

На клиентском компьютере для просмотра информации, полученной от веб-сервера, применяется специальная программа — веб-браузер. Основная функция веб-браузера - отображение гипертекстовых страниц (веб-страниц). Для создания гипертекстовых страниц в WWW изначально использовался язык HTML. Множество веб-страниц образуют веб-сайт.

В основе www — взаимодействие между веб-сервером и браузерами по протоколу HTTP (HyperText Transfer Protocol). Веб-сервер — это программа, запущенная на сетевом компьютере и ожидающая клиентские запросы по протоколу HTTP.

Функционирование сервиса обеспечивается четырьмя составляющими:

- [URL/URI](#) — унифицированный способ адресации и идентификации сетевых ресурсов;
- [HTML](#) — язык гипертекстовой разметки веб-документов;
- [HTTP](#) — протокол передачи гипертекста;
- [CGI](#) — общий шлюзовый интерфейс, представляющий доступ к серверным приложениям.

DNS и IP

Понятие DNS расшифровывается как Domain Name System, или, Система Доменных Имен. Данный инструмент используется в качестве средства преобразования доменных имен в IP-адреса в момент отправки запроса на сервер. И в дополнение к основному функционалу, DNS является хранилищем этих обменных данных.

Обобщив, DNS можно рассматривать как регистр посещаемости разных веб-сайтов. В нем хранятся доменные имена и их IP (Internet Protocol).

Когда вы набираете в браузере адрес сайта, браузер посылает запрос на DNS-сервер и проверяет зарегистрировано ли имя в базе данных. Получив положительный результат, сервер отвечает

вам IP-адресом сайта, к которому вы пытаетесь получить доступ. Условный вид IP-адреса: 123.456.789.10.

Протоколы Интернет прикладного уровня

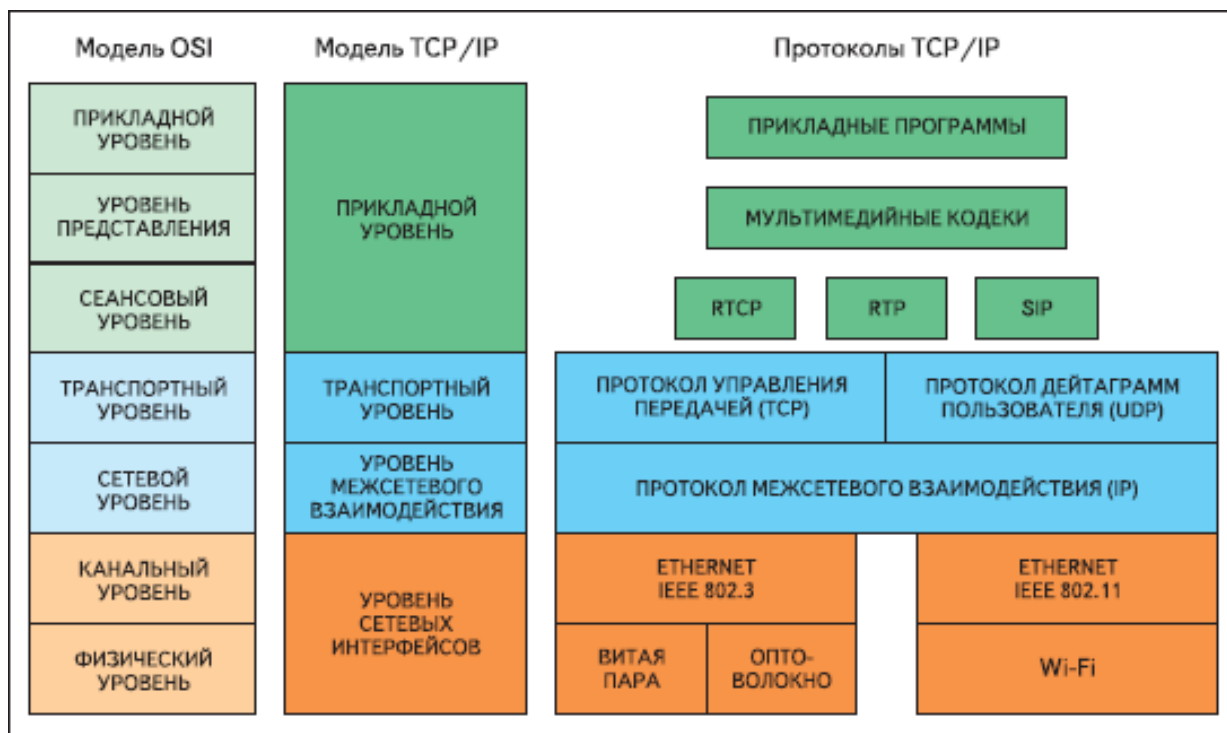
- *DNS* - распределённая система доменных имён, которая по запросу, содержащему доменное имя хоста сообщает IP адрес;
- *HTTP* - протокол передачи гипертекста в Интернет;
- *HTTPS* - расширение протокола HTTP, поддерживающее шифрование;
- *FTP* (File Transfer Protocol - RFC 959) - протокол, предназначенный для передачи файлов в компьютерных сетях;
- *Telnet* (TELEcommunication NETwork - RFC 854) - сетевой протокол для реализации текстового интерфейса по сети;
- *SSH* (Secure Shell - RFC 4251) - протокол прикладного, позволяющий производить удалённое управление операционной системой и передачу файлов. В отличие от Telnet шифрует весь трафик;
- *POP3* – протокол почтового клиента, который используется почтовым клиентом для получения сообщений электронной почты с сервера;
- *IMAP* - протокол доступа к электронной почте в Интернет;
- *SMTP* – протокол, который используется для отправки почты от пользователей к серверам и между серверами для дальнейшей пересылки к получателю;
- *LDAP* - протокол для доступа к службе каталогов X.500, является широко используемым стандартом доступа к службам каталогов;
- *XMPP* (Jabber) - основанный на XML расширяемый протокол для мгновенного обмена сообщениями в почти реальном времени;
- *SNMP* - базовый протокол управления сети Internet.

3. TCP/IP. Стандартизации в Интернет. RFC с примерами.

Стек протоколов TCP/IP

Стек протоколов TCP/IP (Transmission Control Protocol/Internet Protocol, протокол управления передачей/протокол интернета) — сетевая модель, описывающая процесс передачи цифровых данных.

TCP/IP соответствует модели OSI достаточно условно и содержит 4 уровня. Прикладной уровень стека соответствует трем верхним уровням модели OSI: прикладному, представления и сеансовому.

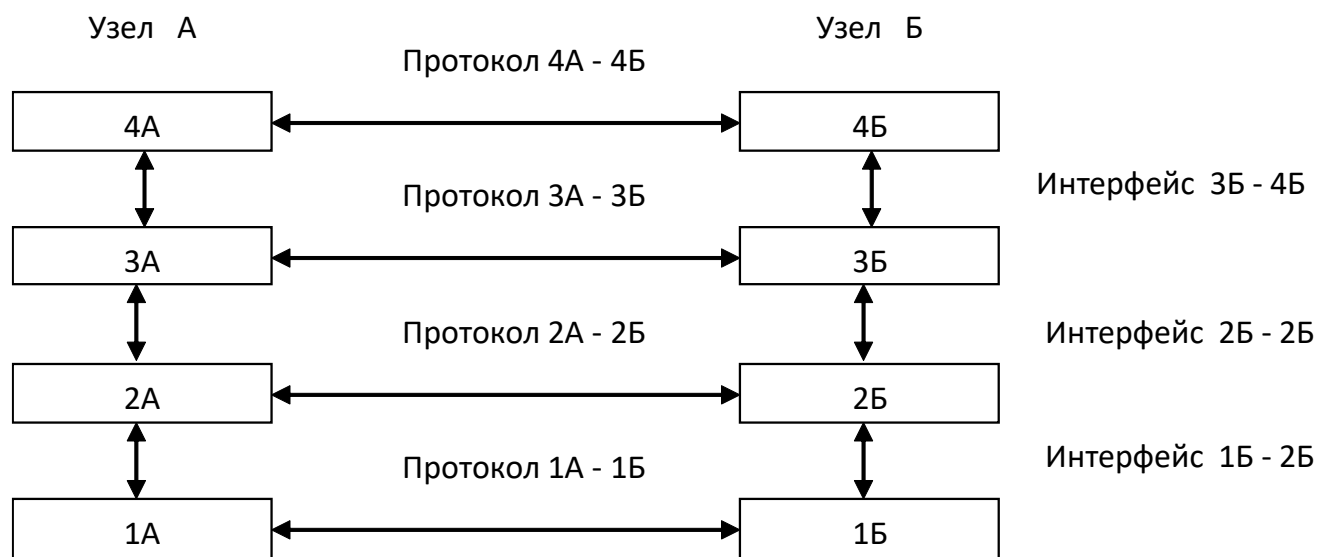


Стандартизации в Интернет. RFC

Интернет является очень сложной сетью, и соответственно такой же сложной является задача организации взаимодействия между устройствами сети. Для решения такого рода задач используется декомпозиция, т.е. разбиение сложной задачи на несколько более простых задач-модулей. Одной из концепций, реализующих декомпозицию, является многоуровневый подход. Такой подход дает возможность проводить разработку, тестирование и модификацию каждого отдельного уровня независимо от других уровней. Иерархическая декомпозиция позволяет, перемещаясь в направлении от более низких к более высоким уровням переходить к более простому представлению решаемой задачи.

Специфика многоуровневого представления сетевого взаимодействия состоит в том, что в процессе обмена сообщениями участвуют как минимум две стороны, для которых необходимо обеспечить согласованную работу двух иерархий аппаратно-программных средств. Каждый из

уровней должен поддерживать интерфейс с выше- и нижележащими уровнями собственной иерархии средств и интерфейс со средствами взаимодействия другой стороны на том же уровне иерархии. Данный тип интерфейса называется протоколом.



Поскольку сеть – это соединение разнородного оборудования, актуальной является проблема совместимости, что в свою очередь, требует согласования всеми производителями общепринятых стандартов. Открытой является система, построенная в соответствии с открытыми спецификациями.

Спецификация представляет собой формализованное описание аппаратных (программных) компонентов, способов их функционирования, взаимодействия с другими компонентами, условий эксплуатации, особых характеристик. Под открытыми спецификациями понимаются опубликованные, общедоступные спецификации, соответствующие стандартам и принятые в результате достижения согласия после всестороннего обсуждения всеми заинтересованными сторонами. Использование открытых спецификаций при разработке систем позволяет третьим сторонам разрабатывать для этих систем аппаратно-программные средства расширения и модификации, а также создавать программно-аппаратные комплексы из продуктов разных производителей.

Если две сети построены с соблюдением принципов открытости, это дает следующие преимущества:

- Возможность построения сети из аппаратных и программных средств различных производителей, придерживающихся стандарта;
- Безболезненная замена отдельных компонентов сети другими, более совершенными;
- Легкость сопряжения одной сети с другой.

В рамках модели OSI средства взаимодействия делятся на семь уровней: прикладной, представления, сеансовый, транспортный, сетевой, канальный и физический. В распоряжение программистов предоставляется прикладной программный интерфейс, позволяющий обращаться с запросами к самому верхнему уровню, а именно, - уровню приложений.

Сеть Интернет строилась в полном соответствии с принципами открытых систем. В разработке стандартов этой сети принимали участие тысячи специалистов-пользователей сети из вузов, научных организаций и компаний. Результат работы по стандартизации воплощается в документах RFC.

RFC (англ. Request for Comments) — документ из серии пронумерованных информационных документов Интернета, содержащих технические спецификации и стандарты, широко применяемые во Всемирной сети.

Номер RFC	Тема
RFC 768	UDP
RFC 791	IP
RFC 793	TCP
RFC 822	Формат электронной почты, заменен RFC 2822
RFC 959	FTP
RFC 1034	DNS — концепция
RFC 1035	DNS — внедрение
RFC 1591	Структура доменных имен
RFC 1738	URL
RFC 1939	Протокол POP версии 3 (POP3)
RFC 2026	Процесс стандартизации в Интернете
RFC 2045	MIME
RFC 2231	Кодировка символов
RFC 2616	HTTP
RFC 2822	Формат электронной почты
RFC 3501	IMAP версии 4 издание 1 (IMAP4rev1)

4. Протокол HTTP. Структура запроса клиента. Версии протокола HTTP.

HTTP (HyperText Transfer Protocol - RFC 1945, RFC 2616) - протокол прикладного уровня для передачи гипертекста. Центральным объектом в HTTP является *ресурс*, на который указывает *URI* в запросе клиента.

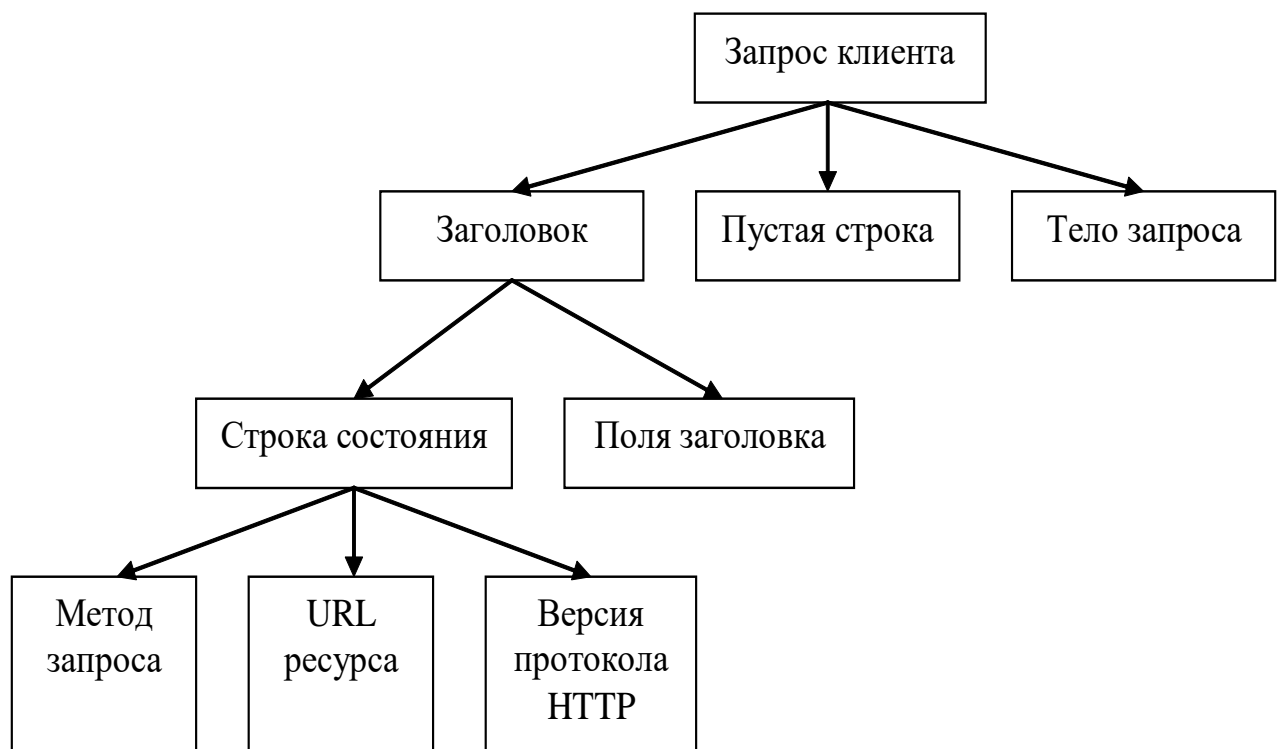
Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя изначально данный протокол предназначен для передачи символьной информации.

В отличие от многих других протоколов, HTTP является протоколом без памяти. Это означает, что протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера.

"Классическая" схема HTTP-сеанса выглядит так.

- Установление TCP-соединения.
- Запрос клиента.
- Ответ сервера.
- Разрыв TCP-соединения.

Структура запроса клиента.



Строка состояния имеет следующий формат:

метод_запроса URL_ресурса версия_протокола_HTTP

Метод, указанный в строке состояния, определяет способ воздействия на ресурс, URL которого задан в той же строке. Метод может принимать значения *GET*, *POST*, *HEAD*, *PUT*, *DELETE* и т.д. Несмотря на обилие методов, для веб-программиста по-настоящему важны лишь два из них: *GET* и *POST*.

Версия протокола HTTP, как правило, задается в следующем формате:

HTTP/версия.модификация

Поля заголовка, следующие за строкой состояния, позволяют уточнять запрос, т.е. передавать серверу дополнительную информацию. Поле заголовка имеет следующий формат:

Имя_поля: Значение

Назначение поля определяется его именем, которое отделяется от значения двоеточием.

Во многих случаях при работе в Веб тело запроса отсутствует. При запуске CGI-сценариев данные, передаваемые для них в запросе, могут размещаться в теле запроса.

Ниже представлен пример HTML-запроса, сгенерированного браузером

- *GET http://oak.oakland.edu/ HTTP/1.0*
- *Connection: Keep-Alive*
- *User-Agent: Mozilla/4.04 [en] (Win95; I)*
- *Host: oak.oakland.edu*
- *Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */**
- *Accept-Language: en*
- *Accept-Charset: iso-8859-1,*,utf-8*

Версии протокола HTTP.

1. Первоначальная версия HTTP не имела номера версии. Позже он был назван 0.9, чтобы отличить его от более поздних версий.

HTTP/0.9 чрезвычайно прост: запросы состоят из одной строки и начинаются с единственно возможного метода *GET*, за которым следует путь к ресурсу.

```
GET /mypage.html
```

Ответ тоже чрезвычайно прост: он состоял только из самого файла.

```
<HTML>
```

```
A very simple HTML page
```

```
</HTML>
```

В отличие от последующих эволюций, заголовков HTTP не было, а это означало, что можно было передавать только файлы HTML. Кодов статуса или ошибок не было: в случае возникновения проблемы возвращался специальный HTML-файл с содержащимся в нем описанием проблемы для человеческого восприятия.

2. HTTP/0.9 был очень ограничен, и как браузеры, так и серверы быстро расширили его, сделав более универсальным:

- Информация о версии теперь отправляется в каждом запросе (**HTTP/1.0** добавляется к строке GET)
- Строка состояния также отправляется в начале ответа, позволяя самому браузеру понять успех или неудачу запроса и, как следствие, адаптировать свое поведение (например, при обновлении или использовании своего локального кеша определенным образом).
- Было введено понятие заголовков HTTP как для запросов, так и для ответов, что позволяет передавать метаданные и делает протокол чрезвычайно гибким и расширяемым.
- С помощью новых HTTP-заголовков добавлена возможность передачи документов, отличных от обычных HTML-файлов (благодаря заголовку Content-Type).

3. **HTTP/1.1** прояснил неясности и внес многочисленные улучшения:

- Соединение можно использовать повторно, что экономит время на многократное повторное открытие для отображения ресурсов.
- Добавлена конвейерная обработка, позволяющая отправить второй запрос до того, как ответ на первый будет полностью передан, что снижает задержку связи.
- Введены дополнительные механизмы управления кешем.
- Было введено согласование контента, включая язык, кодировку или тип, что позволяет клиенту и серверу согласовать наиболее подходящий контент для обмена.

4. Протокол **HTTP/2** имеет несколько основных отличий от версии HTTP/1.1:

- Это бинарный протокол, а не текст.
- Это мультиплексный протокол. Параллельные запросы могут обрабатываться по одному и тому же соединению, что устраняет ограничения порядка и блокировки протокола HTTP/1.1.
- Он сжимает заголовки. Поскольку они часто похожи в наборе запросов, это устраняет дублирование и накладные расходы на передаваемые данные.
- Он позволяет серверу заполнять данные в клиентском кеше до того, как они потребуются, с помощью механизма, называемого проталкиванием сервера.

5. Протокол HTTP. Поля заголовка. Структура ответа сервера. Классы кодов ответа сервера. MIME-тип.

HTTP (HyperText Transfer Protocol - RFC 1945, RFC 2616) - протокол прикладного уровня для передачи гипертекста. Центральным объектом в HTTP является *ресурс*, на который указывает *URI* в запросе клиента.

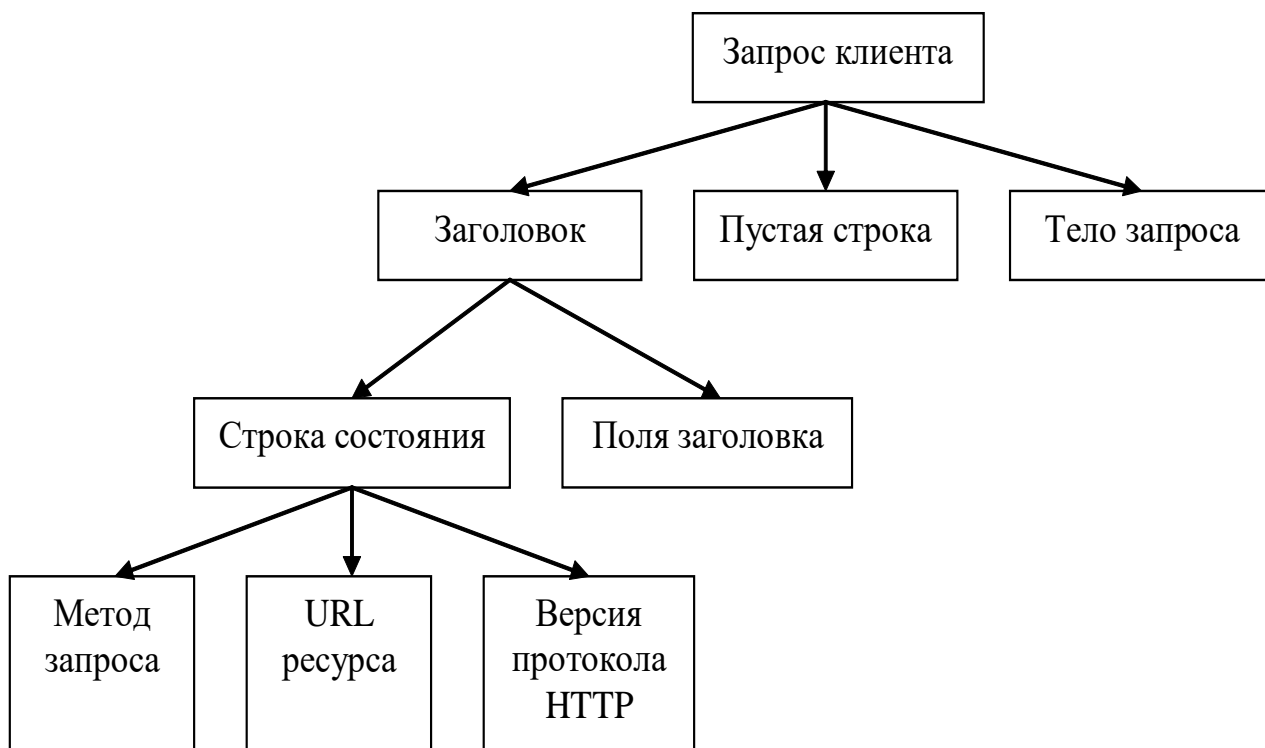
Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя изначально данный протокол предназначен для передачи символьной информации.

В отличие от многих других протоколов, HTTP является протоколом без памяти. Это означает, что протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера.

"Классическая" схема HTTP-сеанса выглядит так.

- Установление TCP-соединения.
- Запрос клиента.
- Ответ сервера.
- Разрыв TCP-соединения.

Поля заголовка



Поля заголовка, следующие за строкой состояния, позволяют уточнять запрос, т.е. передавать серверу дополнительную информацию. Поле заголовка имеет следующий формат:

Имя_поля: Значение

Назначение поля определяется его именем, которое отделяется от значения двоеточием.

Поля заголовка HTTP-запроса	Значение
Host	Доменное имя или IP-адрес узла, к которому обращается клиент
Referer	URL документа, который ссылается на ресурс, указанный в строке состояния
From	Адрес электронной почты пользователя, работающего с клиентом
Accept	MIME-типы данных, обрабатываемых клиентом. Это поле может иметь несколько значений, отделяемых одно от другого запятыми. Часто поле заголовка Accept используется для того, чтобы сообщить серверу о том, какие типы графических файлов поддерживает клиент
Accept-Language	Набор двухсимвольных идентификаторов, разделенных запятыми, которые обозначают языки, поддерживаемые клиентом
Accept-Charset	Перечень поддерживаемых наборов символов
Content-Type	MIME-тип данных, содержащихся в теле запроса (если запрос не состоит из одного заголовка)
Content-Length	Число символов, содержащихся в теле запроса (если запрос не состоит из одного заголовка)
Range	Присутствует в том случае, если клиент запрашивает не весь документ, а лишь его часть
Connection	Используется для управления TCP-соединением. Если в поле содержится Close, это означает, что после обработки запроса сервер должен закрыть соединение. Значение Keep-Alive предлагает не закрывать TCP-соединение, чтобы оно могло быть использовано для последующих запросов
User-Agent	Информация о клиенте

Структура ответа сервера.

Получив от клиента запрос, сервер должен ответить ему. Знание структуры ответа сервера необходимо разработчику веб-приложений, так как программы, которые выполняются на сервере, должны самостоятельно формировать ответ клиенту.

Подобно запросу клиента, ответ сервера также состоит из четырех перечисленных ниже компонентов.

- Строка состояния.

- Поля заголовка.
- Пустая строка.
- Тело ответа.

Ответ сервера клиенту начинается со строки состояния, которая имеет следующий формат:

Версия_протокола Код_ответа Пояснительное_сообщение

Версия_протокола задается в том же формате, что и в запросе клиента, и имеет тот же смысл.

Код_ответа - это трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером.

Пояснительное_сообщение дублирует код ответа в символьном виде. Это строка символов, которая не обрабатывается клиентом. Она предназначена для системного администратора или оператора, занимающегося обслуживанием системы, и является расшифровкой кода ответа.

Из трех цифр, составляющих код ответа, первая (старшая) определяет класс ответа, остальные две представляют собой номер ответа внутри класса. Так, например, если запрос был обработан успешно, клиент получает следующее сообщение:

HTTP/1.0 200 OK

В используемых в настоящее время реализациях протокола HTTP первая цифра не может быть больше 5 и определяет следующие классы ответов.

- **1** - специальный класс сообщений, называемых информационными. Код ответа, начинающийся с 1, означает, что сервер продолжает обработку запроса. При обмене данными между HTTP-клиентом и HTTP-сервером сообщения этого класса используются достаточно редко.
- **2** - успешная обработка запроса клиента.
- **3** - перенаправление запроса. Чтобы запрос был обслужен, необходимо предпринять дополнительные действия.
- **4** - ошибка клиента. Как правило, код ответа, начинающийся с цифры 4, возвращается в том случае, если в запросе клиента встретилась синтаксическая ошибка.
- **5** - ошибка сервера. По тем или иным причинам сервер не в состоянии выполнить запрос.

MIME-тип

Спецификация *MIME* (Multipurpose Internet Mail Extension — многоцелевое почтовое расширение Internet) первоначально была разработана для того, чтобы обеспечить передачу различных форматов данных в составе электронных писем. Однако применение MIME не исчерпывается электронной почтой. Средства MIME успешно используются в WWW и, по сути, стали неотъемлемой частью этой системы.

В соответствии со спецификацией *MIME*, для описания формата данных используются *тип* и *подтип*. *Тип* определяет, к какому классу относится формат содержимого HTTP-запроса или HTTP-ответа. *Подтип* уточняет формат. Тип и подтип отделяются друг от друга косой чертой:

- *тип/подтип*

Поскольку в подавляющем большинстве случаев в ответ на запрос клиента сервер возвращает исходный текст HTML-документа, то в поле *Content-type* ответа обычно содержится значение *text/html*.

Тип/подтип	Расширение файла	Описание
application/pdf	.pdf	Документ, предназначенный для обработки Acrobat Reader
application/msexcel	.xls	Документ в формате Microsoft Excel
application/postscript	.ps, .eps	Документ в формате PostScript
application/x-tex	.tex	Документ в формате TeX
application/msword	.doc	Документ в формате Microsoft Word
application/rtf	.rtf	Документ в формате RTF, отображаемый с помощью Microsoft Word
image/gif	.gif	Изображение в формате GIF
image/jpeg	.jpeg, .jpg,	Изображение в формате JPEG
image/tiff	.tiff, .tif	Изображение в формате TIFF
image/x-xbitmap	.xbm	Изображение в формате XBitmap
text/plain	.txt	ASCII-текст
text/html	.html, .htm	Документ в формате HTML
audio/midi	.midi, .mid	Аудиофайл в формате MIDI
audio/x-wav	.wav	Аудиофайл в формате WAV
message/rfc822		Почтовое сообщение
message/news		Сообщение в группы новостей
video/mpeg	.mpeg, .mpg, .mpe	Видеофрагмент в формате MPEG
video/avi	.avi	Видеофрагмент в формате AVI

6. Протокол HTTP. Обеспечение безопасности данных при передаче по протоколу HTTP. Cookie.

HTTP (HyperText Transfer Protocol - RFC 1945, RFC 2616) - протокол прикладного уровня для передачи гипертекста. Центральным объектом в HTTP является *ресурс*, на который указывает *URI* в запросе клиента.

Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя изначально данный протокол предназначен для передачи символьной информации.

В отличие от многих других протоколов, HTTP является протоколом без памяти. Это означает, что протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера.

"Классическая" схема HTTP-сеанса выглядит так.

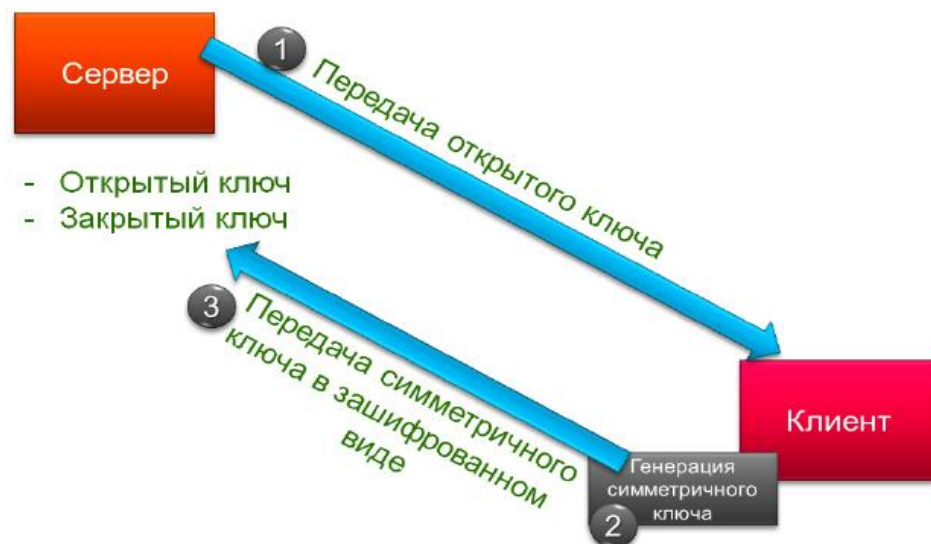
- Установление TCP-соединения.
- Запрос клиента.
- Ответ сервера.
- Разрыв TCP-соединения.

HTTPS

Данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол *SSL* или *TLS*, тем самым обеспечивая защиту этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443. Чтобы подготовить веб-сервер для обработки HTTPS соединений, администратор должен получить и установить в систему сертификат для этого веб-сервера.

SSL (Secure Sockets Layer) - криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет. При его использовании создаётся защищённое соединение между клиентом и сервером.

Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший название TLS. Этот протокол использует шифрование с открытым ключом для подтверждения подлинности передатчика и получателя.



Cookie

Поскольку HTTP-сервер не помнит предыстории запросов клиентов, то каждый запрос обрабатывается независимо от других, и у сервера нет возможности определить, исходят ли запросы от одного клиента или разных клиентов.

Если сервер будет проверять TCP-соединения и запоминать IP-адреса компьютеров-клиентов, он все равно не сможет различить запросы от двух браузеров, выполняющихся на одной машине. И даже если допустить, что на компьютере работает лишь одна клиент-программа, то никто не может утверждать, что в промежутке между двумя запросами она не была завершена, а затем запущена снова уже другим пользователем.

Механизм *cookie* позволяет серверу хранить информацию на компьютере клиента и извлекать ее оттуда.

Инициатором записи *cookie* выступает сервер. Если в ответе сервера присутствует поле заголовка *Set-cookie*, клиент воспринимает это как команду на запись *cookie*. В дальнейшем, если клиент обращается к серверу, от которого он ранее принял поле заголовка *Set-cookie*, помимо прочей информации он передает серверу данные *cookie*. Для передачи указанной информации серверу используется поле заголовка *Cookie*.

Для того чтобы в общих чертах представить себе, как происходит обмен данными *cookie*, рассмотрим следующий пример. Предположим, что клиент передает запросы на серверы *A*, *B* и *C*. Предположим также, что сервер *B*, в отличие от *A* и *C*, передает клиенту команду записать *cookie*. Последовательность запросов клиента серверу и ответов на них будет выглядеть приблизительно следующим образом.

- Передача запроса серверу *A*.
- Получение ответа от сервера *A*.
- Передача запроса серверу *B*.

- Получение ответа от сервера **B**. В состав ответа входит поле заголовка *SetCookie*. Получив его, клиент записывает *cookie* на диск.
- Передача запроса серверу **C**. Несмотря на то что на диске хранится запись *cookie*, клиент не предпринимает никаких специальных действий, так как значение *cookie* было записано по инициативе другого сервера.
- Получение ответа от сервера **C**.
- Передача запроса серверу **A**. В этом случае клиент также никак не реагирует на тот факт, что на диске хранится *cookie*.
- Получение ответа от сервера **A**.
- Передача запроса серверу **B**. Перед тем как сформировать запрос, клиент определяет, что на диске хранится запись *cookie*, созданная после получения ответа от сервера **B**. Клиент проверяет, удовлетворяет ли данный запрос некоторым требованиям, и, если проверка дает положительный результат, включает в заголовок запроса поле *Cookie*.

Поле *Set-cookie* имеет следующий формат:

Set-cookie: имя = значение; expires = дата; path = путь; домен = имя_домена, secure

- Пара *имя = значение* – именованные данные, сохраняемые с помощью механизм *cookie*. Эти данные должны храниться на клиент-машине и передаваться серверу в составе очередного запроса клиента.

- Дата, являющаяся значением параметра *expires*, определяет время, по истечении которого информация *cookie* теряет свою актуальность. Если ключевое слово *expires* отсутствует, данные *cookie* удаляются по окончании текущего сеанса работы браузера.

- Значение параметра *domain* определяет домен, с которым связываются данные *cookie*. Чтобы узнать, следует ли передавать в составе запроса данные *cookie*, браузер сравнивает доменное имя сервера, к которому он собирается обратиться, с доменами, которые связаны с записями *cookie*, хранящимися на клиент-машине. Результат проверки будет считаться положительным, если сервер, которому направляется запрос, принадлежит домену, связанному с *cookie*. Если соответствие не обнаружено, данные *cookie* не передаются.

- Путь, указанный в качестве значения параметра *path*, позволяет выполнить дальнейшую проверку и принять окончательное решение о том, следует ли передавать данные *cookie* в составе запроса.

- Последний параметр, *secure*, указывает на то, что данные *cookie* должны передаваться по защищенному каналу.

Для передачи данных *cookie* серверу используется поле заголовка *Cookie*. Формат этого поля достаточно простой:

- ***Cookie: имя=значение; имя=значение; ...***

С помощью поля *Cookie* передается одна или несколько пар *имя = значение*. Каждая из этих пар принадлежит записи *cookie*, для которой URL запрашиваемого ресурса соответствуют имени домена и пути, указанным ранее в поле *Set-cookie*.

7. Общее описание языка разметки XML.

XML (eXtensible Markup Language – расширенный язык разметки) - текстовый формат, предназначенный для хранения структурированных данных, для обмена информацией между программами, а также для создания на его основе специализированных языков разметки.

Разработка стандартов и работы, связанные с развитием и использованием XML, начали появляться в первой половине 90-х годов, и уже к концу 90-х годов она получила достаточно широкое распространение.

Язык XML является подмножеством языка разметки SGML (Standard Generalized Markup Language – Стандартный Обобщенный Язык Разметки). SGML был разработан в начале 60-х годов прошлого века фирмой IBM.

Язык SGML – это мощный язык разметки документов, обладающий очень большими возможностями. Однако, как это часто бывает, именно из его достоинств вытекают его недостатки. Мощь SGML обуславливает сложность его использования, наличие большой по объему документации. Он не эффективен для описания простых документов.

Рабочая группа по разработке XML была сформирована в 1996 году и при разработке нового языка должна была обеспечить решение набора требований:

- XML должен быть пригоден как для использования в Интернет, так и поддерживать широкий круг возможных других применений;
- XML должен быть простым языком, но при этом совместимым с SGML;
- XML документы должны легко создаваться, читаться и быть понятны людям, просто обрабатываться программно;
- процедура построения XML документа должна быть формальной и точной, количество факultatивных свойств должно быть минимально, а возможно вообще сведено к нулю.
- Условие обеспечения краткости описания перед XML не ставилась.

Язык XML имеет следующие достоинства:

- Это человеко-ориентированный формат документа, он понятен как человеку, так и компьютеру.
- Поддерживает Юникод.
- В формате XML могут быть описаны основные структуры данных - такие как записи, списки и деревья.
- Это самодокументируемый формат, который описывает структуру и имена полей также как и значения полей.
- Имеет строго определённый синтаксис и требования к анализу, что позволяет ему оставаться простым, эффективным и непротиворечивым.
- Широко используется для хранения и обработки документов;
- Это формат, основанный на международных стандартах;
- Иерархическая структура XML подходит для описания практически любых типов документов;
- Представляет собой простой текст, свободный от лицензирования и каких-либо ограничений;

- Не зависит от платформы;

К известным недостаткам языка можно отнести следующие:

- Синтаксис XML избыточен.
 - Размер XML документа существенно больше бинарного представления тех же данных (порядка 10 раз).
 - Размер XML документа существенно больше, чем документа в альтернативных текстовых форматах передачи данных (например, *JSON*) и особенно в форматах данных, оптимизированных для конкретного случая использования.
 - Избыточность XML может повлиять на эффективность приложения. Возрастает стоимость хранения, обработки и передачи данных.
 - Для большого количества задач не нужна вся мощь синтаксиса XML, и можно использовать значительно более простые и производительные решения.
- Пространства имён XML сложно использовать и их сложно реализовывать в XML парсерах.
- XML не содержит встроенной в язык поддержки типов данных. В нём нет понятий «целых чисел», «строк», «дат», «булевых значений» и т. д.
- Иерархическая модель данных, предлагаемая XML, ограничена по сравнению с реляционной моделью и объектно-ориентированными графами.

XML документ имеет следующую структуру:

- Первая строка XML документа называется *объявлением XML*. Это необязательная строка, указывающая версию стандарта XML (обычно это 1.0). Также здесь может быть указана кодировка символов и внешние зависимости.
- Комментарий может быть размещен в любом месте дерева. XML комментарии размещаются внутри пары тегов `<!--` и заканчиваются `-->`. Два знака дефис (`--`) не могут быть применены ни в какой части внутри комментария.
- Остальная часть этого XML-документа состоит из вложенных элементов, некоторые из которых имеют атрибуты и содержимое.
- Элемент обычно состоит из *открывающего* и *закрывающего* тегов, обрамляющих текст и другие элементы.
- *Открывающий* тег состоит из имени элемента в угловых скобках;

- *Закрывающий* тег состоит из того же имени в угловых скобках, но перед именем ещё добавляется косая черта.

- *Содержимым* элемента называется всё, что расположено между открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы.

- Кроме *содержания* у элемента могут быть атрибуты - пары *имя=значение*, добавляемые внутрь открывающего тега после названия элемента.

- Значения атрибутов всегда заключаются в кавычки (одинарные или двойные), одно и то же имя атрибута не может встречаться дважды в одном элементе.

- Не рекомендуется использовать разные типы кавычек для значений атрибутов одного тега.

- Для обозначения *элемента без содержания*, называемого *пустым* элементом, необходимо применять особую форму записи, состоящую из одного тега, в котором *после имени элемента* ставится косая черта «/».

К сожалению, описанные выше правила позволяют контролировать только формальную правильность XML документа, но не содержательную. Для решения второй задачи используются так называемые *схемы*.

Схема четко определяет *имя* и *структуру* корневого элемента, включая *спецификацию* всех его *дочерних элементов*. Программист может задать, какие элементы и в каком количестве *обязательны*, а какие – *необязательны*. Схема также определяет, какие элементы содержат *атрибуты*, *допустимые значения* этих атрибутов, в т.ч. *значения по умолчанию*.

Чаще всего для описания схемы используются следующие спецификации:

- DTD (*Document Type Definition*) - язык определения типа документов.
- XDR (*XML Data Reduced*) – диалект XML, разработанный Майкрософт.
- XSD (*язык определения схем XML*) – рекомендована консорциумом W3C.

8. Общее описание языка разметки HTML. HTML5.

HTML (*HyperText Markup Language* — «язык гипертекстовой разметки») — стандартизированный язык разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в графическое представление.

Язык гипертекстовой разметки HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1986—1991 годах в стенах ЦЕРНа в Женеве в Швейцарии. HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. HTML успешно справлялся с проблемой сложности SGML путём определения небольшого набора структурных и семантических элементов — дескрипторов. Дескрипторы также часто называют «тегами». С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Мультимедийные возможности были добавлены позже.

HTML является фундаментальной, базовой технологией Интернета, позволяет формировать на странице сайта текстовые блоки, включать в них изображения, организовывать таблицы, управлять отображением цвета документа и текста, добавлять в дизайн сайта звуковое сопровождение, организовывать гиперссылки с контекстным переходом в другие разделы сервера или обращаться к иным ресурсам Сети и компоновать все эти элементы между собой. Файлы, содержащие гипертекстовый код, имеют расширение или .html.

Преимущества HTML:

- HTML-документ стал стандартным строительным блоком в Интернете. Он работает в разных системах и сервисах, и практически каждая страница или приложение в настоящее время представляет собой конгломерат HTML-тегов.
- HTML легко понять и изучить. Фактически, это один из первых языков, которые новички используют при обучении программированию.
- Он касается того, как выглядят данные и как отображается контент. Таким образом, разумное использование тегов, объектов и других элементов упрощает создание веб-сайта. Используя HTML, вы можете отображать данные в презентабельном виде.

Недостатки HTML

- HTML — язык заранее определенных тегов, за каждым из которых стандартом закреплена определенная роль. Набор этих тегов громоздок и при этом нерасширяем.
- HTML предназначен для визуализации передаваемой информации и не имеет достаточных средств для описания сущности этой информации.

HTML – это *теговый* язык разметки документов. Чтобы браузер понимал, что имеет дело не с простым текстом, а с особым элементом, который задает его форматирование, применяются *теги*. Теги являются основой HTML и берутся в угловые скобки. Общий синтаксис написания тегов следующий.

<тег параметр1="значение" параметр2="значение">

<тег параметр1="значение" параметр2="значение">...</тег>

Теги бывают одиночными и парными (контейнеры). Одиночный тег используется самостоятельно, а парный может включать внутри себя другие теги или текст. У тегов допустимы различные параметры, которые разделяются между собой пробелом. Впрочем, есть теги, без всяких дополнительных атрибутов. Параметры условно можно подразделить на обязательные, они непременно должны присутствовать, и необязательные, их добавление зависит от цели применения.

Свойства тегов:

- теги не чувствительны к регистру;
- если для тега не добавлен какой-либо допустимый параметр, браузер будет подставлять значение, заданное по умолчанию;
- внутри тега между его параметрами допустимо ставить перенос строк;
- все параметры тегов рекомендуется брать в двойные или одинарные кавычки (хотя в современном браузере отсутствие кавычек не приведет к ошибкам);
- если какой-либо тег или его параметр был написан неверно, то браузер проигнорирует подобный тег;
- существует определенная иерархия вложенности тегов (например, метатеги должны находиться внутри контейнера <HEAD>), причем если теги между собой равноценны в иерархии, то их последовательность не имеет значения;
- Теги бывают парными и одиночными. Для парных нужен закрывающийся тег, для одиночных – нет.
- порядок параметров в любом теге не имеет значения и на результат отображения элемента не влияет.

Обычные веб-страницы состоят из двух разделов – заголовка (<HEAD>) и тела документа (<BODY>). Раздел заголовка может содержать текст и теги, но содержимое этого раздела не показывается напрямую на странице. Тело документа предназначено для размещения тегов и содержательной части.

HTML5

Семантические элементы HTML5 доступно описывают свой смысл или назначение как для браузеров, так и для веб-разработчиков.

До появления стандарта HTML5 вся разметка страниц осуществлялась преимущественно с помощью элементов <div>, которым присваивали классы `class` или идентификаторы `id` для наглядности разметки (например, <div id="header">). С их помощью в HTML-документе размещали верхние и нижние колонтитулы, боковые панели, навигацию и многое другое.

Стандарт HTML5 предоставил новые элементы для структурирования, группировки контента и разметки текстового содержимого. Новые семантические элементы позволили улучшить структуру веб-страницы, добавив смысловое значение заключенному в них содержимому (было <div id="header">, стало <header>). Для отображения внешнего вида элементов не задано никаких правил, поэтому элементы можно стилизовать по своему усмотрению. Для всех элементов доступны глобальные атрибуты.

Согласно спецификации HTML5, каждый элемент принадлежит к определенной (ноль или более) категории. Каждая из них группирует элементы со схожими характеристиками.

Выделяют следующие общие категории:

- Метаданные
- Потокное содержимое
- Секционное содержимое
- Заголовочное содержимое
- Текстовое содержимое
- Встроенное содержимое
- Интерактивное содержимое

В HTML5 появилось множество семантических элементов, а также тегов, позволяющих вставлять аудио и видео на сайт.

[<article>](#) [<aside>](#) [<audio>](#) [<canvas>](#) [<command>](#) [<datalist>](#) [<details>](#) [<figcaption>](#) [<figure>](#)
[<footer>](#) [<header>](#) [<hgroup>](#) [<keygen>](#) [<main>](#) [<mark>](#) [<menu>](#) [<meter>](#) [<nav>](#) [<output>](#) [<pro-](#)
[gress>](#) [<rp>](#) [<rt>](#) [<ruby>](#) [<section>](#) [<source>](#) [<summary>](#) [<time>](#) [<video>](#) [<wbr>](#)

9. Объектная модель документа. Область применения. Сравнение XML и HTML

DOM (Document Object Model) – представляет собой стандарт консорциума W3C для программного доступа к документам HTML или XML. Фактически это платформно- и языково-нейтральный интерфейс, позволяющий программам и сценариям динамически обращаться и обновлять содержимое, структуру и стиль документа.

В рамках данного стандарта можно выделить 3 части:

- Core DOM – стандартная модель любого структурированного документа
- XML DOM - стандартная модель XML документа
- HTML DOM - стандартная модель HTML документа

При использовании DOM для работы с текстовым файлом в формате XML она анализирует файл, разбивает его на индивидуальные элементы, атрибуты, комментарии и т.д. Затем в памяти создается представление файла XML в виде дерева узлов в котором каждый объект в документе рассматривается в виде узла: элементы, атрибуты, комментарии, команды обработки и даже составляющий атрибуты обыкновенный текст.

После этого разработчик может обращаться к содержанию документа, используя дерево узлов, и при необходимости вносить в него изменения, например, чтобы добавить новый элемент (для этого достаточно просто создать новый узел и прикрепить его в качестве потомка к нужному узлу).

В W3C DOM для различных составляющих DOM объектов определены интерфейсы, облегчающие манипулирование с деревом узлов, но для этих интерфейсов не предлагается никакой специфической реализации, и ее можно осуществить на любом языке программирования. Определяемая консорциумом W3C модель DOM не зависит от платформы, т.е. W3C определяет, какие методы и свойства должны быть сделаны доступными в реализациях, специфических для конкретных систем, но не подробности осуществления этих реализаций. Реализуя приложение использующие модель DOM, а не спецификации W3C, разработчики должны ссылаться на документацию, специфическую для данной реализации, которую можно найти в библиотеках.

В качестве метода доступа к файлам XML всегда следует выбирать модель DOM. По сравнению с такими доступными механизмами генерации документов XML, как запись непосредственно в поток, этот метод имеет ряд преимуществ:

1. Модель DOM гарантирует правильную грамматику и правильное оформление документов.
2. Модель DOM абстрагирует содержание от грамматики.
3. Модель DOM упрощает внутреннее манипулирование документом.

4. Модель DOM напоминает структуры иерархических и реляционных баз данных.

В браузерах Internet Explorer 5 и выше находятся встроенные библиотеки DOM и поддержка XSL. Для сценариев на стороне клиента доступно множество объектов для работы с XML-документом, самые важные из них, объекты **XMLDOMDocument**, **XMLDOMNode**, **XMLDOMNodeList**, **XMLDOMParseError** представляющие интерфейс для доступа ко всему документу, отдельным его узлам и поддеревьям, предоставляющие необходимую для отладки информацию о произошедших ошибках анализатора соответственно.

Объект **XMLDOMNode**, реализует базовый DOM интерфейс Node, предназначен для манипулирования с отдельным узлом дерева документа. Его свойства и методы позволяют получать и изменять полную информацию о текущем узле - его тип (является ли текущий узел элементом, комментарием, текстом и т.д.), название, полное название (вместе с Namespace префиксом), его содержимое, список дочерних элементов и т.д.

Объект **XMLDOMDocument** представляет верхний уровень объектной иерархии и содержит методы для работы с документом: его загрузки, анализа, создания в нем элементов, атрибутов, комментариев и т.д. Многие свойства и методы этого объекта реализованы также в классе Node, т.к. документ может быть рассмотрен как корневой узел с вложенными в него поддеревьями.

Объект **XMLDOMNodeList** представляет собой список узлов - поддеревья и содержит методы, при помощи которых можно организовать процедуру обхода дерева.

Объект **XMLDOMParserError** позволяет получить всю необходимую информацию об ошибке, произошедшей в ходе разбора документа. Все свойства этого объекта доступны только для чтения.

Язык XML предоставляет идеальный механизм обмена информацией между различными базами данных. По своей природе базы данных являются закрытыми - в каждой базе используется своя структура имен элементов, свой уровень нормализации и даже свои методы описания перечисляемой информации. С помощью модели DOM можно упростить передачу информации между различными **базами данных**.

Модель DOM — это модель с произвольной выборкой, т.е. узел может быть создан или прикреплен в любом месте дерева XML в любой момент времени. Эта особенность очень полезна при создании документов XML на основе информации иерархической или реляционной базы данных.

Создать документ XML с помощью модели DOM значительно проще, чем записывать информацию в текстовый файл. Вместо того чтобы все время перемещаться между таблицами в поисках требуемой информации, всю информацию из каждой таблицы можно записать одновременно. С увеличением глубины дерева узлов первый метод становится все более и более запутанным, в то время как второй легко масштабируется. Кроме того, генерация документа с помощью модели DOM гарантирует его правильное оформление.

Для программной обработки XML документов используется модель XML DOM, которая определяет объекты и свойства всех XML элементов и методы (интерфейс) для доступа к ним.

Иначе говоря, XML DOM описывает каким образом необходимо получать, изменять, добавлять и удалять XML элементы.

Согласно DOM:

- все, что содержится внутри XML документа, является узлом;
- весь документ представляется узлом документа;
- каждый XML элемент – узел элемента;
- текст внутри XML элементов - текстовый узел;
- каждый атрибут - узел атрибута;
- комментарии - узлы комментариев.

XML документ в соответствии с моделью XML DOM представляется как дерево из узлов, при этом:

- Все узлы дерева находятся в определенных отношениях друг с другом.
- Все узлы доступны через дерево. Их содержимое может быть изменено, удалено; новые элементы могут быть добавлены в дерево.
- Дерево начинается с корневого узла и разветвляется вниз вплоть до текстовых узлов на самом низшем уровне дерева.
- Все узлы находятся в иерархических отношениях между собой.
- Эти отношения описываются с помощью понятий родитель, дочерний и потомок (все дочерние на одном уровне).

Сравнение XML и HTML

Между HTML и XML существует определённое сходство:

1. Оба являются языками разметки.
2. Описание разметки выполняется при помощи тегов.
3. Теги могут иметь атрибуты.
4. Документы имеют иерархическую структуру (корневой элемент, вложенные элементы).

Сравнение HTML и XML:

- XML чувствителен к регистру тегов и атрибутов, HTML – нет
- HTML оперирует фиксированным набором тегов, XML – нет
- В XML каждый открывающий тег должен быть закрыт, а в HTML существуют теги, которые не нужно закрывать
- Атрибуты XML записываются в виде пар `имя="значение"`. В HTML есть логические атрибуты (указывается только имя)

При желании любой HTML-документ можно превратить в правильный XML-документ:

- закрыть все теги
- соблюдать регистр (нижний),
- логические атрибуты перекодировать в `атрибут="атрибут"`
- использовать спецсимволы XML (`<`), где необходимо.

Такой «исправленный» HTML называется XHTML (стандартизирован, последняя версия 1.1).

10. Схемы DTD, XDR, XSD с примерами.

Правила, описывающие структуру документа, позволяют контролировать только формальную правильность XML документа, но не содержательную. Для решения второй задачи используются так называемые *схемы*.

Схема четко определяет *имя* и *структуру* корневого элемента, включая *спецификацию* всех его *дочерних элементов*. Программист может задать, какие элементы и в каком количестве *обязательны*, а какие – *необязательны*. Схема также определяет, какие элементы содержат *атрибуты*, *допустимые значения* этих атрибутов, в т.ч. *значения по умолчанию*.

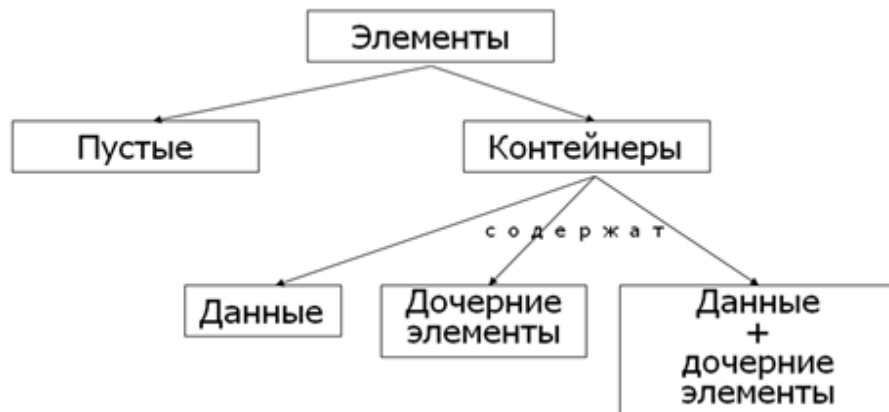
Чаще всего для описания схемы используются следующие спецификации:

- DTD (*Document Type Definition*) - язык определения типа документов.
- XDR (*XML Data Reduced*) – диалект XML, разработанный Майкрософт.
- XSD (*язык определения схем XML*) – рекомендована консорциумом W3C.

DTD схема

Схема DTD предоставляет *шаблон* разметки документа, в котором указываются *наличие*, *порядок следования* и *расположение элементов* и их *атрибутов* в документе XML.

В рамках DTD модель содержимого XML документа можно описать следующим образом:



Содержание	Синтаксис	Комментарий
Данные	<!ELEMENT имя (#PCDATA)>	Содержит только текстовые данные
Другие элементы	<!ELEMENT имя (дочерний элемент 1, дочерний элемент 2)>	Содержит только дочерние элементы
Смешанное	<!ELEMENT имя (#PCDATA, дочерний элемент)*>	Содержит комбинацию текстовых данных и дочерних элементов
EMPTY	<!ELEMENT имя EMPTY>	Ничего не содержит
ANY	<!ELEMENT имя ANY>	Может содержать текстовые данные или дочерние элементы

Атрибуты, находящиеся внутри тэгов документа, описываются отдельно с помощью синтаксиса:

<!ATTList

имя_элемента имя_атрибута1 (тип) значение_по_умолчанию

.....

имя_элемента имя_атрибутаN (тип) значение_по_умолчанию >

При этом атрибут в DTD может иметь один из трех типов:

- Строка
- Маркированные атрибут
- Атрибута с перечислением

Кроме типа атрибута можно также задавать и его модальность:

Значение	Описание
#REQUIRED	Атрибут обязательно должен быть указан
#FIXED	Значение атрибута не должно отличаться от указанного
#IMPLIED	Необязательное значение

Рассмотрим в качестве примера описание атрибутов *строкового* типа для элемента, описывающего некоторое сообщение:

<!ATTLIST message

| | | |
|--------|-------|-----------|
| number | CDATA | #REQUIRED |
| date | CDATA | #REQUIRED |
| from | CDATA | #FIXED |
| status | CDATA | #IMPLIED> |

DTD схема, описывающая структуру электронного почтового ящика:

```
<!ELEMENT mailbox (message*)>
<!ELEMENT message (head, body)>
<!ATTLIST message uid CDATA #REQUIRED>
<!ELEMENT head ( from to+, subject?, CC*, notify?) >
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ELEMENT notify EMPTY>
<!ELEMENT body (#PCDATA)>
```

Исходный XML документ, удовлетворяющий данной схеме, может выглядеть, например, так

```
<?xml version="1.0" ?>
<!DOCTYPE mailbox SYSTEM "mailbox.dtd">
<mailbox>
  <message uid="1">
    <head>
      <from>user1@myhp.edu</from>
      <to>user2@myhp.edu</to>
      <subject>Re:</subject>
    </head>
    <body>
      What's up!
    </body>
  </message>
  <message uid="2">
    <head>
      <from>user3@myhp.edu</from>
      <to>user2@myhp.edu</to>
      <subject>Remind</subject>
      <CC> user1@myhp.edu </CC>
      <notify/>
    </head>
    <body>
      Remind me about meeting.
    </body>
  </message>
</mailbox>
```

Недостатки DTD схем:

- Не являются экземплярами XML. Требуется изучение совершенно другого языка.
- Не предоставляют контроль за типами данных, за исключением самых простых текстовых данных.
- Не являются экземплярами XML, поэтому их нельзя легко расширить или преобразовать к другим языкам разметки – HTML или DHTML.
- Не обеспечивают поддержки пространств имен XML.

XDR схема

XML-Data – полное имя языка описания схем, предложенного Майкрософт, а *XML-Data Reduced* – это “часть” полной рекомендации. Схема XDR – это экземпляр XML, т.е. соответствует всем синтаксическим правилам и стандартам XML.

Реализуя проверки данных на уровне документа с помощью схемы, приложения, генерирующие и принимающие транзакции, можно оптимизировать для обеспечения максимального быстродействия. Соответствие полей и правильность записей проверяются на уровне экземпляров XML.

Корневым элементом в схеме XDR всегда является элемент *Schema*.

В отличие от DTD схем XDR поддерживает типы данных. Элемент *Schema* имеет следующий атрибут:

Xmlns:dt=“urn=schemas-microsoft-com:datatypes”

XSD схема

Можно использовать схему XSD с заметками, которые описывают сопоставление с базой данных, запрашивают базу данных, а затем возвращают результаты в форме XML-документа. Заметки служат для сопоставления схемы XSD с таблицами и столбцами базы данных. Можно указывать запросы XPath к представлениям XML, созданным на основе схемы XSD, для запроса базы данных и получения результатов в виде XML.

Примечание

В Microsoft SQLXML 4.0 язык схем XSD поддерживает заметки, введенные в языке схем XDR в SQL Server 2000 (8.x). Схемы XDR с заметками в SQLXML 4.0 считаются устаревшими.

В контексте реляционной базы данных полезно сопоставить произвольную схему XSD с реляционным хранилищем. Один из способов достижения этого состоит в создании аннотированной схемы XSD. Схема XSD с заметками называется *схемой сопоставления*, которая предоставляет сведения о том, как данные XML должны сопоставляться с реляционным хранилищем. По сути, схема сопоставления

является XML-представлением реляционных данных. Эти сопоставления позволяют получать реляционные данные в виде XML-документа.

11. XSLT и XPath. Применение XSLT

XSLT (eXtensible Stylesheet Language Transformations) - расширяемый язык преобразования листов стилей.

Спецификация XSLT является рекомендацией W3C.

В результате применения таблицы стилей XSLT, состоящей из набора шаблонов, к XML-документу (исходное дерево) образуется конечное дерево, которое может быть другой *XML-структурой*, *HTML-документом* или обычным *текстом*. Правила выбора данных из исходного дерева записываются на языке запросов *XPath*. XSLT применяется в основном в веб-программировании и для генерации отчётов.

Благодаря XSLT реализуется отделение данных от их представления в рамках парадигмы *MVC (Model-view-controller)*.

Язык XSLT состоит из множества инструкций, записанных в виде тегов. Имя каждой инструкции обычно начинается с символов *xsl*. Для выполнения трансформации документ стиля XSLT должен являться корректным документом XML.

Достоинства применения XSLT

1. Плюсом является то, что шаблоны внешне идентичны выводимому коду, то есть они наглядны и проще для понимания.
2. Шаблоны легче читать, редактировать, отлаживать и выполнять другие манипуляции. Также относительно легко можно производить поиск нужного участка генерируемого кода в шаблонах. При генерации без применения шаблонов сделать это не так легко.
3. Язык XSLT уже поддерживает работу с одинарными и двойными кавычками и другими специальными символами, что облегчает применение этих символов в генерации кода.
4. Можно заново генерировать код, не выполняя перекомпиляцию приложения, а только изменив шаблон.
5. В отличие от T4 технология XSLT является стандартом, который поддерживают все производители программного обеспечения. Таким образом, при его применении сохраняется независимость от производителя.
6. Так как XSLT является декларативным языком, в ней гораздо легче простым декларированием можно выполнять многие непростые с точки зрения стандартного программирования операции, вроде прохода по всем узлам документа, их фильтрации, сортировки, подстановки соответствующих значений. Для выполнения их на стандартном языке программирования потребуется трудоемкое программирование, тогда как в XSLT они могут быть реализованы всего лишь несколькими строчками объявлений, а процессор XSLT выполнит все необходимые преобразования. Данный факт является также преимуществом шаблонов XSLT перед шаблонами T4.

7. Поддерживается разделение метаданных и шаблонов.

8. Так как шаблоны можно изменять динамически, есть возможность применять XSLT для генерации комбинированно, в сочетании с языками программирования.

Недостатки и ограничения применения XSLT

1. Шаблоны малоэффективны, когда в генерируемом коде мало стандартных участков и слишком много параметров, от задания которых зависит результирующий код. В некоторых таких случаях шаблон может стать малопонятным и очень сложным для поддержки.

2. В шаблонах XSLT по сравнению с T4 затруднено программирование и выполнение некоторых операций. Так как XSLT является декларативным языком, в ней трудно реализовывать сложные алгоритмы.

3. Изучить XSLT на профессиональном уровне не так легко. Выработка навыков его уверенного применения потребует значительных усилий и практики. Поэтому при планировании разработки генератора нужно учесть этот факт.

XPath

XPath (XML Path Language) - язык запросов к элементам XML-документа. XPath был разработан для организации доступа к частям документа XML в файлах трансформации XSLT и является стандартом консорциума W3C. В языке XPath используется компактный синтаксис, отличный от принятого в XML.

Начиная с версии 2.0, XPath является составной частью языка *XQuery*. XPath призван помочь обходить всевозможные деревья, получать необходимые элементы из другой ветви относительно точки обхода, распознавать предков, потомков, атрибуты элементов. Это полноценный язык навигации по дереву.

Для нахождения элемента(ов) в дереве документа используются пути адресации.

Каждый шаг адресации состоит из трех частей:

- оси, например *child::* ;
- условия проверки узлов, например имена элементов документа *body*, *html*;
- предиката, например *attribute::class* .

Дополнением к ядру языка является набор функций, которые делятся на 5 групп: *системные* функции, функции с *множествами*, *строковые* функции, *логические* функции, *числовые* функции.

Применение XSLT

Модель XSLT включает в себя такие части как:

- документы XML,
- стили XSLT,
- процессор XSLT,
- выходные документы.

Документы XML являются входными данными, которые нужно преобразовать в другие документы. Документ стиля XSLT является корректным (well formed) документом XML и содержит набор правил для выполнения преобразования. Иными словами, документ стиля является шаблоном.

Процессор XSLT является приложением, которое принимает в качестве входных данных документы XML и стили XSLT. Он выполняет трансформацию, то есть применение набора правил в стилях XSLT к документам XML. Результатом этой работы являются выходные документы.

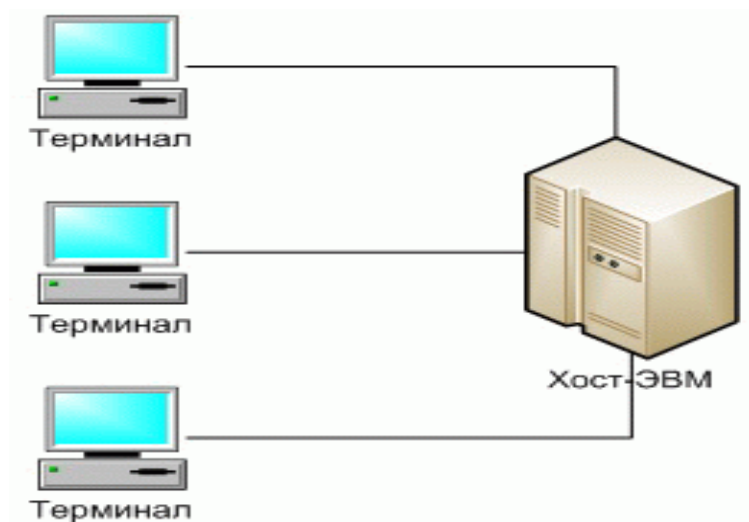
Процессоры XSLT имеют множество реализаций и встроены во многие браузеры вроде Internet Explorer, Firefox, Mozilla, Opera и другие. В Internet Explorer используется инструмент MSXML, разработанный Microsoft. XSLT-процессор встроен в Internet Explorer, начиная с версии 4.5. Сгенерированный результат примеров данной лекции можно просматривать путем открытия XML-файлов в одном из браузеров. В конце лекции мы рассмотрим возможности запуска трансформации программным путем, используя соответствующие классы языка программирования.

Язык XSLT состоит из множества инструкций, записанных в виде тегов. Имя каждой инструкции обычно начинается с символов xsl. Для выполнения трансформации документ стиля XSLT должен являться корректным документом XML.

12. *Архитектура информационных систем. Централизованная архитектура. Архитектура "файл-сервер". Описание технологии «клиент — сервер». Многоуровневая архитектура клиент-сервер.*

Архитектура информационной системы – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы

Централизованная архитектура



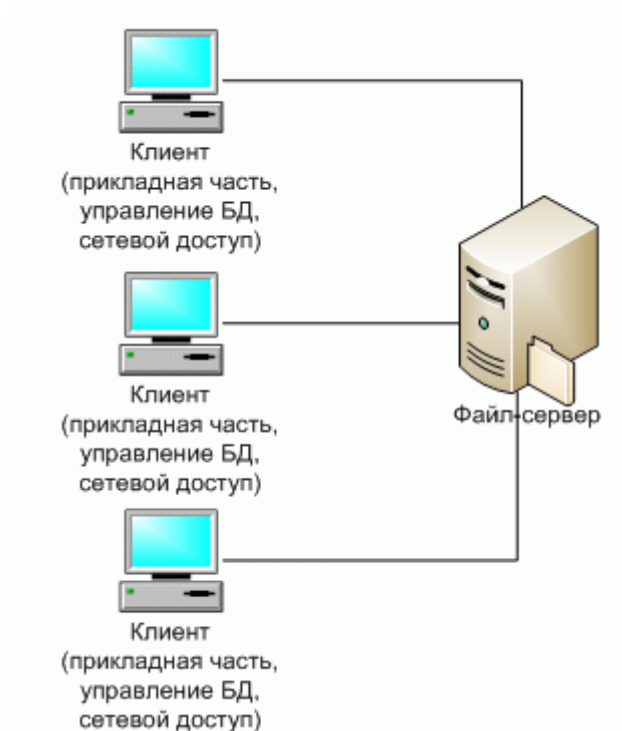
Характерная особенность такой архитектуры – полная "неинтеллектуальность" терминалов. Их работой управляет хост-ЭВМ.

Достоинства такой архитектуры:

- пользователи совместно используют дорогие ресурсы ЭВМ и дорогие периферийные устройства;
- централизация ресурсов и оборудования облегчает обслуживание и эксплуатацию вычислительной системы;
- отсутствует необходимость администрирования рабочих мест пользователей;

Главным недостатком для пользователя является то, что он полностью зависит от администратора хост-ЭВМ. Пользователь не может настроить рабочую среду под свои потребности – все используемое программное обеспечение является коллективным. Использование такой архитектуры является оправданным, если хост-ЭВМ очень дорогая, например, супер-ЭВМ.

Архитектура "файл-сервер"



Файл-серверные приложения – приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения программы и данных.

- Функции сервера: хранения данных и кода программы.
- Функции клиента: обработка данных происходит исключительно на стороне клиента.

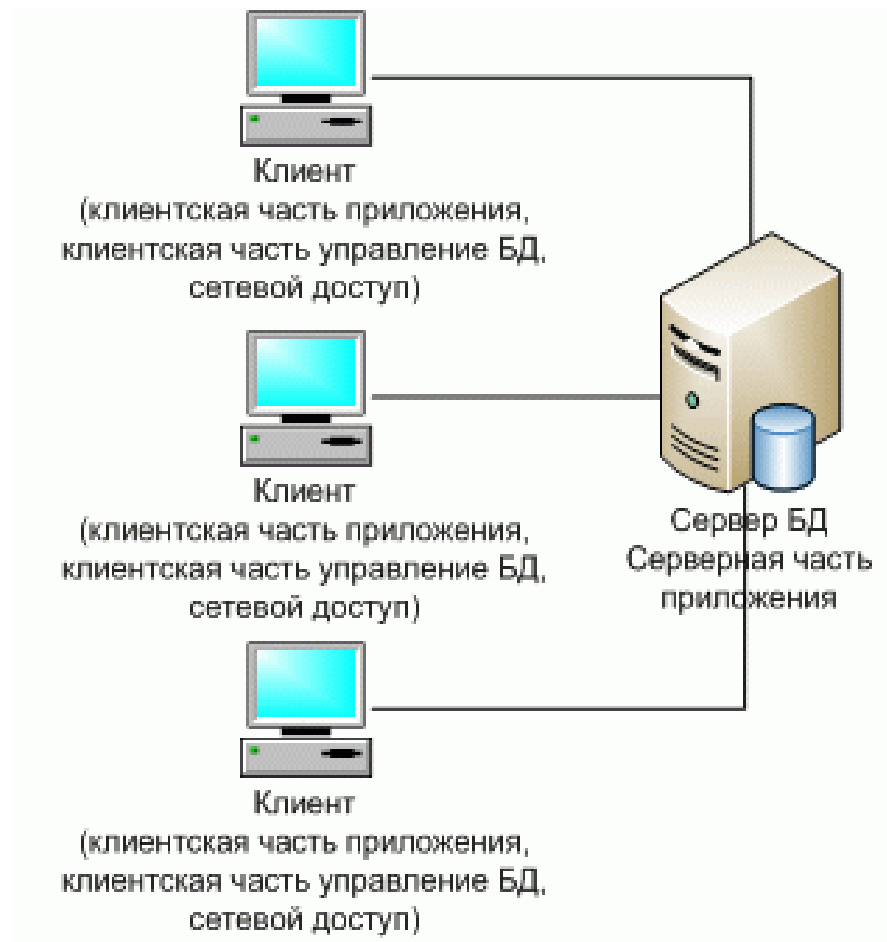
Преимущества:

- возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети;
- все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
- поддержка многопользовательской работы;
- гарантия целостности данных.

Недостатки:

- неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;
- администрирование данной системы требует квалифицированного профессионала;
- высокая стоимость оборудования;
- бизнес логика приложений осталась в клиентском ПО.

Архитектура "клиент-сервер"



Клиент-сервер (Client-server) – архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами. Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture). Под клиент-серверным приложением в этом случае понимается информационная система, основанная на использовании серверов баз данных.

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции.

Клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления базами данных.

Наконец, клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу баз данных, передавая ему текст оператора языка SQL.

- Сервер производит компиляцию полученного оператора.
- Далее (если компиляция завершилась успешно) происходит выполнение оператора.

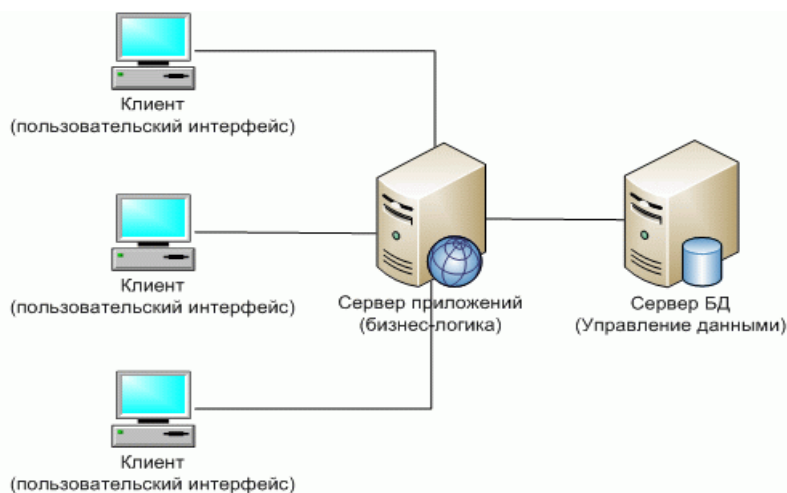
Преимуществами данной архитектуры являются:

- возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети;
- все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
- поддержка многопользовательской работы;
- гарантия целостности данных.

Недостатки:

- неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;
- администрирование данной системы требует квалифицированного профессионала;
- высокая стоимость оборудования;
- бизнес логика приложений осталась в клиентском ПО.

Многоуровневая архитектура «клиент — сервер»



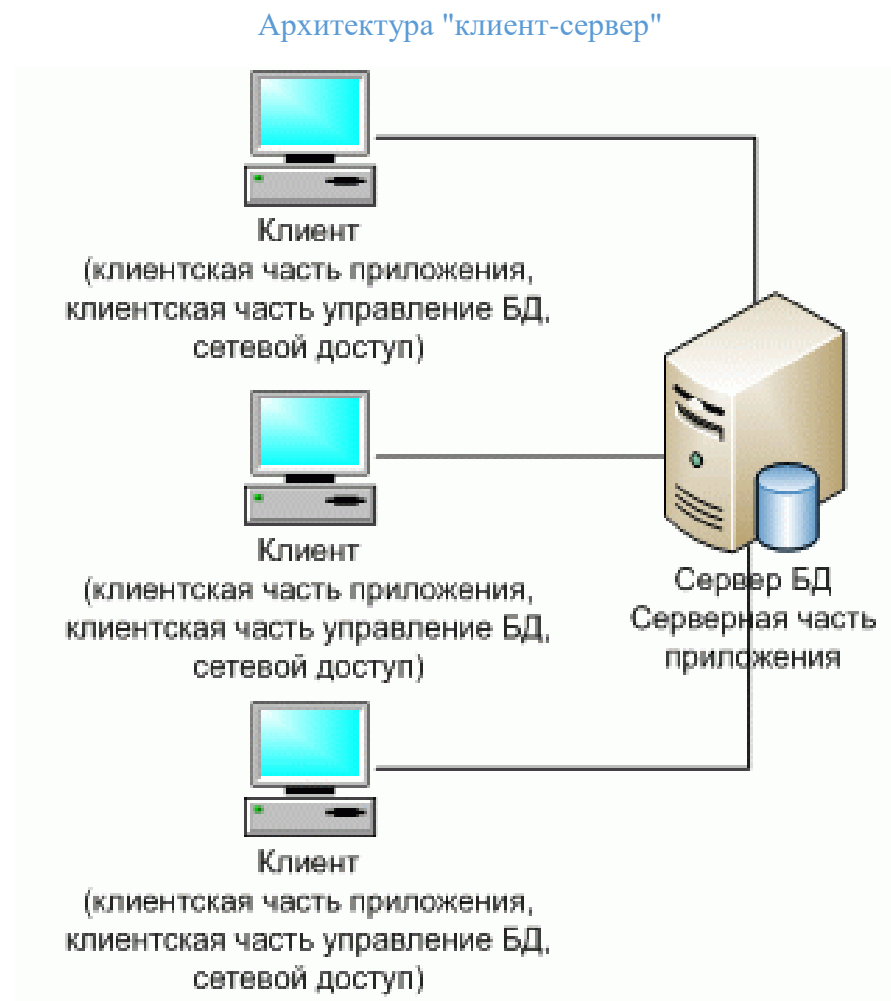
Многоуровневая архитектура «клиент — сервер» — разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

В трехуровневой архитектуре "тонкий" клиент не перегружен функциями обработки данных, а выполняет свою основную роль системы представления информации, поступающей с сервера приложений. Это уменьшает объем данных, передаваемых между клиентом и сервером приложений, что позволяет подключать клиентские компьютеры даже по медленным линиям типа телефонных каналов.

Трехуровневая архитектура клиент-сервер позволяет более точно назначать полномочия пользователей, так как они получают права доступа не к самой базе данных, а к определенным функциям сервера приложений. Это повышает защищенность системы (по сравнению с обычно архитектурой) не только от умышленного нападения, но и от ошибочных действий персонала.

13. *Архитектура информационных систем. Описание технологии «клиент — сервер». Архитектура распределенных систем. Распределенные системы с репликацией. Архитектура Веб-приложений. SOA.*

Архитектура информационной системы – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.



Клиент-сервер (Client-server) – архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами. Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture). Под клиент-серверным приложением в этом случае понимается информационная система, основанная на использовании серверов баз данных.

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции.

Клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления базами данных.

Наконец, клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу баз данных, передавая ему текст оператора языка SQL.

- Сервер производит компиляцию полученного оператора.
- Далее (если компиляция завершилась успешно) происходит выполнение оператора.

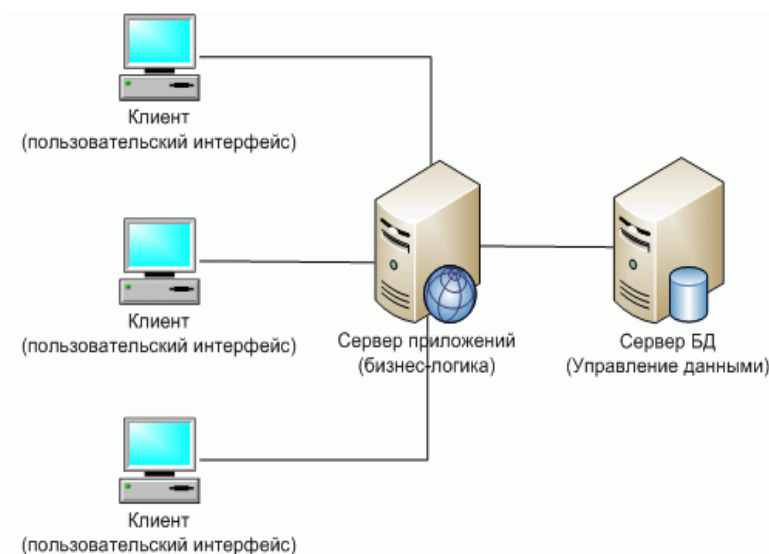
Преимущества данной архитектуры являются:

- возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети;
- все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
- поддержка многопользовательской работы;
- гарантия целостности данных.

Недостатки:

- неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;
- администрирование данной системы требует квалифицированного профессионала;
- высокая стоимость оборудования;
- бизнес логика приложений осталась в клиентском ПО.

Многоуровневая архитектура «клиент — сервер»



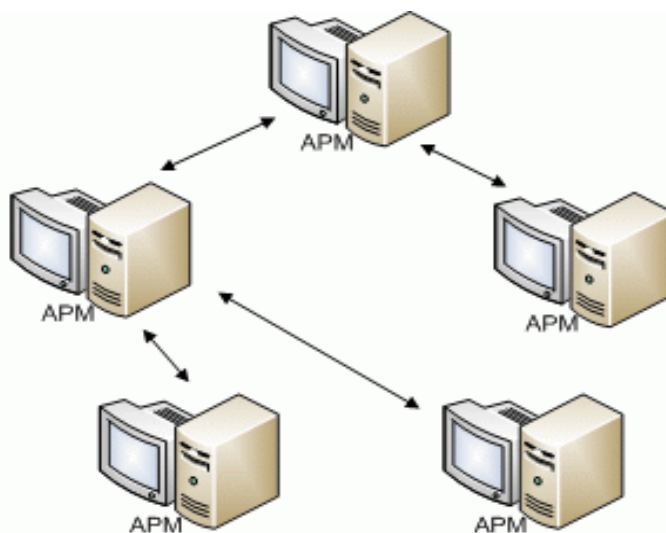
Многоуровневая архитектура «клиент — сервер» — разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

В трехуровневой архитектуре "тонкий" клиент не перегружен функциями обработки данных, а выполняет свою основную роль системы представления информации, поступающей с сервера

приложений. Это уменьшает объем данных, передаваемых между клиентом и сервером приложений, что позволяет подключать клиентские компьютеры даже по медленным линиям типа телефонных каналов.

Трехуровневая архитектура клиент-сервер позволяет более точно назначать полномочия пользователей, так как они получают права доступа не к самой базе данных, а к определенным функциям сервера приложений. Это повышает защищенность системы (по сравнению с обычной архитектурой) не только от умышленного нападения, но и от ошибочных действий персонала.

Архитектура распределенных систем.



Такой тип систем является более сложным с точки зрения организации системы. Суть распределенной системы заключается в том, чтобы хранить локальные копии важных данных.

Более 95 % данных, используемых в управлении предприятием, могут быть размещены на одном персональном компьютере, обеспечив возможность его независимой работы. Поток исправлений и дополнений, создаваемый на этом компьютере, ничтожен по сравнению с объемом данных, используемых при этом. Поэтому если хранить непрерывно используемые данные на самих компьютерах, и организовать обмен между ними исправлениями и дополнениями к хранящимся данным, то суммарный передаваемый трафик резко снизится. Это позволяет понизить требования к каналам связи между компьютерами и чаще использовать асинхронную связь, и благодаря этому создавать надежно функционирующие распределенные информационные системы, использующие для связи отдельных элементов неустойчивую связь типа Интернета, мобильную связь, коммерческие спутниковые каналы. А минимизация трафика между элементами сделает вполне доступной стоимость эксплуатации такой связи. Конечно, реализация такой системы не элементарна, и требует решения ряда проблем, одна из которых своевременная синхронизация данных.

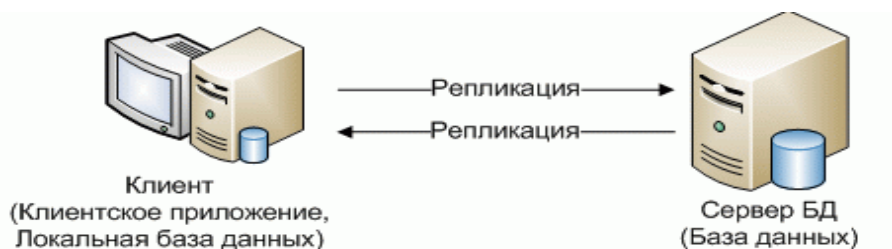
Достоинства:

- Каждый АРМ независим, содержит только ту информацию, с которой должен работать, а актуальность данных во всей системе обеспечивается благодаря непрерывному обмену сообщениями с другими АРМами. Обмен сообщениями между АРМами может быть реализован различными способами, от отправки данных по электронной почте до передачи данных по сетям.
- Обеспечение возможности персональной ответственности за сохранность данных. Так как данные, доступные на конкретном рабочем месте, находятся только на этом компьютере,

при использовании средств шифрования и личных аппаратных ключей исключается доступ к данным посторонних, в том числе и IT администраторов.

- Такая архитектура системы также позволяет организовать распределенные вычисления между клиентскими машинами. Например, расчет какой-либо задачи, требующей больших вычислений, можно распределить между соседними АРМаи благодаря тому, что они, как правило, обладают одной информацией в своих БД и, таким образом, добиться максимальной производительности системы.

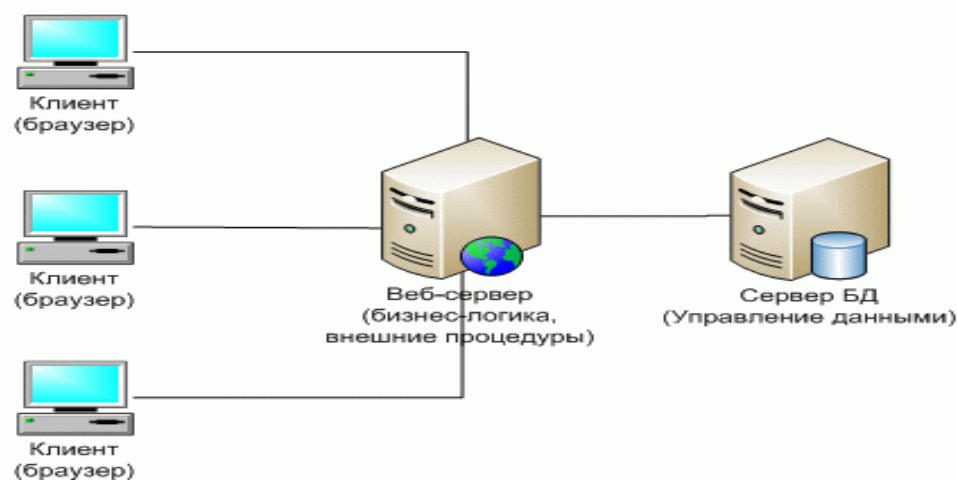
Распределенные системы с репликацией.



Репликация — механизм синхронизации содержимого нескольких копий объекта.

Данные между различными рабочими станциями и централизованным хранилищем данных, передаются репликацией. При вводе информации на рабочих станциях – данные также записываются в локальную базу данных, а лишь затем синхронизируются.

Архитектура Веб-приложений



Internet Server API (ISAPI). Интерфейс ISAPI – это особенность Microsoft Internet Information Server. ISAPI-приложения представляют собой динамические загружаемые библиотеки (DLL), которые выполняются в адресном пространстве Веб-сервера. У других Веб-серверов через некоторое время также появилась возможность выполнять приложения, реализованные в виде библиотек.

Естественно, что при использовании ISAPI-приложений стало появление нового, высокоуровневого интерфейса, который упростил задачи генерации HTML-кода, позволил обращаться к компонентам и использовать базы данных. Таким интерфейсом стала объектная модель Active Server Pages (ASP), построенная на основе ISAPI-фильтра.

Основной идеей ASP с точки зрения создания интерфейса приложения является то, что на Веб-странице присутствуют фрагменты кода, который интерпретируется Веб-сервером и вместо которого пользователь получает результат выполнения этих фрагментов кода.

Active Server Pages – ASP .NET, являющаяся ключевой в архитектуре Microsoft .NET Framework. С помощью ASP .NET можно создавать Веб-приложения и Веб-сервисы, которые не только позволяют реализовать динамическую генерацию HTML-страниц, но и интегрируются с серверными компонентами и могут использоваться для решения широкого круга бизнес-задач, возникающих перед разработчиками современных Веб-приложений.

Особенности веб-архитектуры:

- отсутствие необходимости использовать дополнительное ПО на стороне клиента – это позволяет автоматически реализовать клиентскую часть на всех платформах;
- возможность подключения практически неограниченного количества клиентов;
- благодаря единственному месту хранения данных и наличия системы управления базами данных обеспечиваются минимальные требования для поддержания целостности данных;
- доступность при работоспособности сервера и каналов связи;
- недоступность при отсутствии работоспособности сервера или каналов связи;
- достаточно низкая скорость Веб сервера и каналов передачи данных;
- относительно объема данных – архитектура Веб систем не имеет существенных ограничений.

Сервис-ориентированная архитектура (SOA)

Сервис-ориентированная архитектура (SOA, service-oriented architecture) – модульный подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами

Сервисно-ориентированная архитектура – это парадигма организации и использования распределенных информационных ресурсов таких как: приложения и данные, находящихся в сфере ответственности разных владельцев, для достижения желаемых результатов потребителем, которым может быть: конечный пользователь или другое приложение

В основе SOA лежат принципы многократного использования функциональных элементов ИТ, ликвидации дублирования функциональности в ПО, унификации типовых операционных процессов, обеспечения перевода операционной модели компании на централизованные процессы и функциональную организацию на основе промышленной платформы интеграции.

Компоненты программы могут быть распределены по разным узлам сети, и предлагаются как независимые, слабо связанные, заменяемые сервисы-приложения. Программные комплексы, разработанные в соответствии с SOA, часто реализуются как набор веб-сервисов, интегрированных при помощи известных стандартных протоколов (SOAP, WSDL, и т. п.)

Основными целями применения SOA для крупных информационных систем, уровня предприятия, и выше являются:

- сокращение издержек при разработке приложений, за счет упорядочивания процесса разработки;

- расширение повторного использования кода;
- независимость от используемых платформ, инструментов, языков разработки;
- повышение масштабируемости создаваемых систем;
- улучшение управляемости создаваемых систем.

Принципы SOA:

- архитектура, как таковая, не привязана к какой-то определенной технологии;
- независимость организации системы от используемой вычислительной платформы (платформ);
- независимость организации системы от применяемых языков программирования;
- использование сервисов, независимых от конкретных приложений, с единообразными интерфейсами доступа к ним;
- организация сервисов как слабосвязанных компонентов для построения систем.

14. *Web-серверы (Apache, Nginx IIS и другие). Примеры, краткая характеристика и сравнительный анализ.*

Информационные службы интернета (Internet Information Services, IIS) представляют собой пакет приложений для интернета, выпускаемый компанией Microsoft. Входит в поставку некоторых версий Windows. Как и все продукты от Microsoft, легок в установке и управлении.

Apache

Apache HTTP web – сервер – полное название платформы, распространяемой организацией Apache Software Foundation как открытое программное решение или проще говоря «open-source». Программное обеспечение сервера распространяется абсолютно бесплатно, и его лицензия позволяет конечному пользователю редактировать исходный код, чтобы адаптировать Apache под свои нужды, а также, внести вклад в будущее развитие серверной платформы.

Веб – сервер Apache может работать на всех популярных операционных системах, но чаще всего он используется в рамках Linux. Именно в паре с СУБД MySQL и PHP – скриптами образуется известный комплекс программного обеспечения LAMP Web – сервер (Linux, Apache, MySQL, PHP), который повсеместно используется в сети интернет.

Многие функции реализуются как совместимые модули, расширяющие базовый функционал, диапазон которых варьируется от поддержки языков программирования до обеспечения различных схем аутентификации.

Многие другие функции, такие как Secure Sockets Layer (SSL) или TLS (Transport Layer Security) так же обеспечиваются модульной системой.

Apache поддерживает возможность развернуть несколько web – сайтов, или графических интерфейсов приложений.

Веб – сервер сжимает страницы, чтобы уменьшить их размер, что обеспечивает высокую скорость их загрузки.

Архитектура

- Apache состоит из ядра и динамической модульной системы. Параметры системы изменяются с помощью конфигурационных файлов. Для запуска на сервере нескольких веб-проектов одновременно используется механизм виртуальных хостов.

Ядро

- Ядро Apache разработано Apache Software Foundation на языке C. Основные функции — обработка конфигурационных файлов, протокол HTTP/HTTPS и загрузка модулей. Ядро может работать без модулей, но будет иметь ограниченный функционал.

Модульная система

- Модуль – отдельный файл, подключение которого расширяет изначальный функционал ядра. Они могут включаться в состав ПО при первоначальной установке или подгружаться позже через изменение конфигурационного файла.

Конфигурация

- Система конфигурации Apache работает на текстовых файлах с прописанными настройками. Она подразделяется на три условных уровня, для каждого из которых имеется свой конфигурационный файл:
- Уровень конфигурации сервера (файл **httpd.conf**) – основной конфигурационный файл. Действие распространяется на весь механизм веб-сервера.
- Уровень каталога (файл **.htaccess**) – дополнительный конфигурационный файл. Его директивы охватывают только каталог, где расположен файл, а также вложенные подкаталоги.
- Уровень виртуального хоста (файл **httpd.conf** или **extra/httpd-vhosts.conf**).

Достоинства:

- **Доступность.** Это программное обеспечение с открытым исходным кодом. Значит, его может бесплатно использовать или модифицировать любой желающий.
- **Удобство и гибкость настройки.** Приложение легко настраивается через текстовые конфигурационные файлы. Apache способен обрабатывать большой объем трафика и быстро масштабироваться, даже без сложного дополнительного конфигурирования.
- **Функциональность.** У Apache динамическая модульная структура. Можно быстро подключать дополнительный функционал в виде скачиваемых модулей, даже без обращения к внешним источникам. Это позволяет решать целый комплекс важнейших задач в области безопасности, кэширования, редактирования URL, распределения нагрузки.
- **Кроссплатформенность.** Сервер работает как на Windows и MacOS, так и на всех Unix-подобных системах.
- **Совместимость.** Apache работает на базе скриптовых или веб-ориентированных языков (PHP, Python, Tcl, Ruby, Perl, ASP), что делает его совместимым с самым широким спектром баз данных и серверного ПО.
- **Масштабируемость.** Подходит для веб-ресурсов любого масштаба. Apache хорошо работает как на одностраничном сайте (лендинге), так и на многостраничном сайте с ежедневной аудиторией в десятки тысяч посетителей.
- **Поддержка пользователей.** Apache удерживает первенство популярности среди веб-серверов с 1996 года. За прошедшее время для него создана обширнейшая база документации – как официальной, так и созданной сторонними разработчиками. Готовые, подробно описанные руководства можно найти практически на любой сценарий.

Недостатки:

- **Трафик влияет на производительность.** На сайтах с большой посещаемостью производительность веб-сервера может значительно снижаться, что замедляет работу самого ресурса. Это связано с тем, что работа Apache основана на модели «процессов» и обрабатывает каждый пользовательский запрос отдельно.
- **Сложная конфигурация повышает уязвимость.** Возможность подключать модули в Apache это не всегда преимущество. Чем больше модулей, тем сложнее становятся

настройки. Соответственно, больше шансов допустить критические пробелы в контуре безопасности.

- **Неудобное редактирование.** В операционных системах семейства Unix/Linux конфигурационные файлы Apache приходится редактировать вручную. Это связано с отсутствием встроенного графического интерфейса для настройки. Решение проблемы – бесплатный инструмент Apache GUI, позволяющий настраивать функции веб-сервера прямо из браузера.

- **Излишний функционал.** Даже без дополнительных модулей Apache предоставляет пользователям массу возможностей. Правда, большинство использует лишь небольшую часть базового функционала приложения. Поэтому часто после установки приходится тратить время на отключение «лишних» модулей.

NGINX

Nginx — мощный инструмент для развертывания веб-сервера, который при правильной настройке превосходит Apache. Области применения Nginx весьма обширны — от кэширования HTTP до создания инвертированного прокси-сервера.

В отличие от обычного веб-сервера, Nginx не создаёт один поток под каждый запрос, а разделяет его на меньшие однотипные структуры, называемые рабочими соединениями. Каждое такое соединение обрабатывается отдельным рабочим процессом, а после выполнения они сливаются в единый блок, возвращающий результат в основной процесс обработки данных. Одно рабочее соединение может обрабатывать до 1024 запросов одного вида одновременно.

Nginx имеет родительский процесс, который загружает конфигурацию и запускает дочерние процессы (т.н. воркеры), каждый воркер может обслуживать очень много http соединений потому, что он работает по модели `event_loop`, т.е. он в очень быстром цикле проходит каждое соединение и обрабатывает событие, которые возникают в этих соединениях. Пр. пришло новое http соединение, он ему что-нибудь сообщил (привет, соединение установлено) и пойдет обрабатывать следующее соединение, потом он понимает, что внутри этого соединения нужно запросить какие-то данные о файловой системе, он их запрашивает и идет обрабатывать следующее, данные из файловой системы пришли и он перенаправляет эти данные в http соединение наружу, в браузер. Таким образом каждый воркер(поток) в nginx может обслуживать очень много http соединений.

Сравнение

Особенность	Apache	Nginx
Простота	Легко разрабатывать и внедрять инновации благодаря своей модели «одно соединение на процесс»	Сложный в разработке, поскольку он имеет сложную архитектуру для одновременной обработки нескольких соединений.
Производительность - Статический контент	Медленно в отображении статического контента	В 2,5 раза быстрее чем Apache и потребляет меньше памяти
Производительность - Динамический контент	Отличная производительность для динамического контента	Отличная производительность для динамического контента
Поддержка операционной системы	Поддерживает все ОС - Unix, как и Windows	Поддерживает все ОС - как Unix, так и Windows, однако производительность в Windows сравнительно менее стабильна.
Безопасность	Это безопасный веб-сервер. Понимание и настройка функций безопасности важны	Это безопасный веб-сервер. Понимание и настройка функций безопасности важны
Гибкость	Можно настроить, добавив модули. Apache имел динамическую загрузку модулей дольше всего.	Nginx версии 1.11.5 и Nginx Plus Release R11 представили совместимость для динамических модулей.
Поддержка и документация	Отличная поддержка и документация доступны, как это было на рынке в течение очень долгого времени.	Несмотря на слабое начало поддержки и документации для Nginx, он быстро рос, поэтому теперь у него есть отличная поддержка ресурсов и доступная документация.

Опция	Apache	IIS
Поддерживаемая ОС	Windows, Linux, Unix, Mac OS	Windows
Техническая поддержка	Сообщество	Корпоративная
Стоимость	Полностью бесплатно	Покупается в комплекте с Windows
Разработка	«open-source»	Проприетарное решение
Безопасность	Хорошо	Отлично
Производительность	Хорошо	Хорошо

15. *Web-серверы. Конвейер обработки HTTP-запросов в IIS. HTTP.SYS. W3SVC. WAS.*

Web-серверы.

Выше, в вопросе 14. Видимо, повторяется полностью.

Конвейер обработки HTTP-запросов в IIS

Ключевую роль при обработке запросов к приложениям ASP.NET играет веб-сервер IIS. IIS прослушивает входящие запросы, определяет, как они должны обрабатываться, и отправляет пользователю определенный ответ.

IIS определяет число запросов, которые могут обрабатываться одновременно на одном ядре процессора компьютера. По умолчанию число одновременно обрабатываемых запросов равно 5000. Если количество входящих запросов превышает это число, то все дополнительные запросы помещаются в очередь ожидания, которая также имеет свой лимит. Если же и данная очередь уже заполнена, то входящий запрос отвергается, а браузеру в ответ посылается статусный код ошибки 503. Но это в целом, теперь рассмотрим более детально обработку запросов.

Но прежде чем рассмотреть, как IIS обрабатывает запросы, познакомимся с отдельными компонентами, которые играют ключевую роль в обработке запросов в веб-сервере IIS.

HTTP.sys

Драйвер уровня ядра HTTP.sys является посредником между приложением и операционной системой. В архитектуре IIS он выполняет две задачи:

- HTTP.sys является протокольным слушателем (protocol listeners) по умолчанию в IIS. То есть HTTP.sys прослушивает все запросы, которые, используют протоколы HTTP и HTTPS, и затем передает эти запросы другим компонентам IIS для дальнейшей обработки.
- HTTP.sys также представляет стек протокола HTTP. А именно HTTP.sys выполняет кэширование, постановку запросов в очередь, предобработку и ряд других функций.

Порт завершения ввода-вывода

Для максимизации числа одновременно обрабатываемых запросов IIS использует специальное API, которое называется портом завершения ввода-вывода (Input/Output Completion Port). IIS имеет выделенный пул потоков для обработки операций ввода/вывода и управления очередью запросов. При обработке запросов этот API взаимодействует с драйвером HTTP.sys.

World Wide Web Publishing Service (W3SVC)

World Wide Web Publishing Service или Служба веб-публикации является адаптером над драйвером HTTP.sys и выполняет следующие функции:

- Конфигурирование HTTP.sys
- Мониторинг показателей производительности

До версии IIS 7.0 служба веб-публикации также управляла рабочими процессами приложений. Но начиная с IIS 7.0 за эту функцию отвечает Windows Process Activation Service.

Данная служба была выделена начиная с IIS 7.0. Ее предназначение - управление рабочими процессами. WAS состоит из трех основных компонентов:

- менеджер конфигурации: считывает конфигурационную информацию о веб-приложениях и пулах приложений из файла `applicationhost.config`
- менеджер процессов: выполняет сопоставление между пулами приложений и рабочими процессами. Если соответствующий пулу приложения рабочий процесс еще не запущен, то менеджер процессов его запускает.
- интерфейс неуправляемых адаптеров прослушивателей (`unmanaged listener adaptor interface`): предоставляет интерфейс для прослушивателей запросов, которые не относятся к протоколу HTTP

Обработка запросов в IIS

Теперь рассмотрим поэтапно работу конвейера IIS по обработке входящих запросов.

1. Пользователь посылает HTTP-запрос, обращаясь через браузер к определенному ресурсу на сервере. Этот запрос перехватывается драйвером HTTP.sys.
2. Драйвер HTTP.sys обращается к WAS для получения конфигурационных данных для запрошенного адреса URL
3. Менеджер конфигурации WAS считывает данные из файла `applicationhost.config`, в частности пул приложения и конфигурационные настройки сайта
4. Считанные данные WAS передает службе веб-публикации IIS (служба W3SVC)
5. Служба веб-публикации использует полученные от WAS данные для конфигурации HTTP.sys. Затем драйвер HTTP.sys помещает пришедший запрос в очередь порта завершения ввода-вывода (I/O completion port), которую обрабатывает WAS.
6. WAS использует выделенный пул потоков для обработки очереди. По умолчанию данный пул потоков может использовать до 250 потоков на одно ядро компьютера. Если к данному моменту не был запущен рабочий процесс, ассоциированный с запрошенным URL, то WAS запускает приложение `w3wp.exe`, в рамках которого работает рабочий процесс обработки запросов.
7. В рамках запущенного рабочего процесса ASP.NET проверяет, сколько запросов обрабатывается в текущий момент времени. Если их число превышает лимит по умолчанию в 5000 запросов, то новый запрос помещается в очередь. Однако если очередь достигла своего лимита в 1000 запросов, то данный запрос отвергается, и в ответ клиенту посылается статусный код ошибки 503.

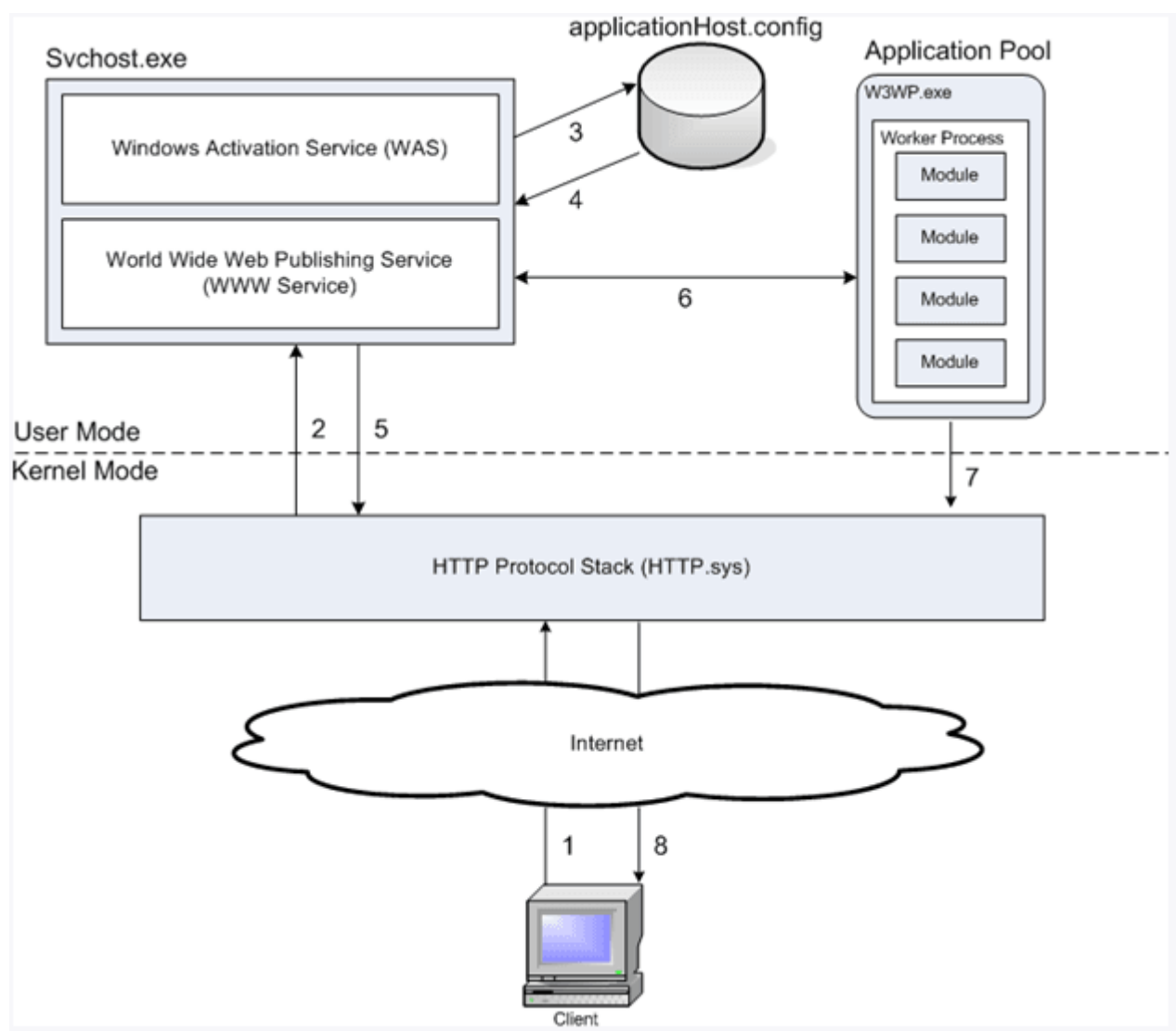
Если запрос направлен к статическому файлу, который не содержит кода .NET-языков, то ASP.NET посылает содержимое этого файла в порт завершения ввода-вывода IIS, а оттуда - пользователю, сделавшему запрос. В остальных случаях ASP.NET отправляет запрос в пул потоков CLR.

CLR обрабатывает запрос. На этой стадии запускается непосредственно код нашего веб-приложения или веб-сайта, размещенного на веб-сервере.

После того, как среда CLR закончит обработку запроса, она посылает результаты драйверу HTTP.sys, а тот - на порт завершения ввода-вывода IIS.

8. Пользователь получает результат обработки

Схематично весь процесс можно представить следующим образом:



16. *Web-серверы. Конвейер обработки HTTP-запросов в IIS. Пул приложений. Домен приложения, приложение.*

Web-серверы.

Выше, в вопросе 14. Видимо, повторяется полностью.

Конвейер обработки HTTP-запросов в IIS

Выше, в вопросе 15. Видимо, повторяется полностью.

Пул приложений.

Пулы приложений позволяют вам изолировать ваши приложения друг от друга, даже если они работают на одном сервере. Таким образом, если в одном приложении будет ошибка, другие приложения не будут удалены.

Кроме того, пулы приложений позволяют вам разделять разные приложения, которые требуют разных уровней безопасности.

IIS запускает любой настроенный вами веб-сайт в процессе с именем w3wp.exe. Пул приложений IIS - это функция в IIS, которая позволяет каждому веб-сайту или его части работать под соответствующим процессом w3wp.exe. Таким образом, вы можете запустить 100 сайтов в одном файле w3wp.exe или в 100 разных файлах w3wp.exe.

Домен приложения, приложение.

На сервере может быть много сайтов asp.net, работающих вместе. Каждый сайт - это **домен приложения**.

Вы должны назначить каждому из них по одному **пулу приложений**. Многие домены приложений (сайты) могут иметь один и тот же пул приложений, и поскольку они имеют один и тот же пул приложений, они запускаются под одними и теми же процессами и под одной учетной записью - и у них одинаковые настройки пула. Если этот пул перезапускается, то перезапускаются все сайты в этом пуле.

Теперь в каждом пуле может быть один или несколько **рабочих процессов**. Каждый рабочий процесс - это отдельная программа, которая запускает ваш сайт, имеет свои статические переменные, разные вызовы запуска и остановки и т. Д. Различные рабочие процессы не взаимодействуют друг с другом, и единственный способ обмена данными - из общих файлов или общей базы данных. Если у вас несколько рабочих процессов и один из них выполняет длительные вычисления, то другой может позаботиться об обработке интернет-вызовов и отображении контента.

17. *Интеграция и взаимодействие в сети Веб. Веб-сервисы. Спецификация WSDL. Спецификация UDDI.*

Интеграция и взаимодействие в сети Веб.

Задача веб-интеграции заключается в том, чтобы объединить разнородные веб-приложения и системы в единую среду на базе сети Веб.

Практикуются следующие подходы к веб-интеграции:

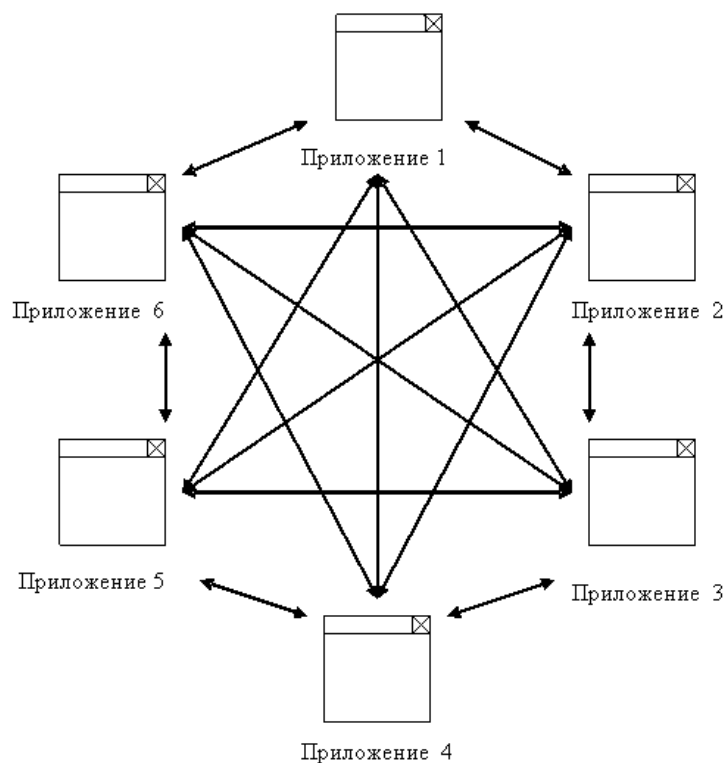
- Интеграция на уровне представления. Данный уровень позволяет пользователю взаимодействовать с приложением. Интеграция на уровне представления дает доступ к пользовательскому интерфейсу удаленных приложений.
- Интеграция на уровне функциональности. Данная интеграция подразумевает обеспечение прямого доступа к бизнес-логике приложений. Это достигается непосредственным взаимодействием приложений с *API* (программному интерфейсу приложений) или же взаимодействием посредством *веб-сервисов*.
- Интеграция на уровне данных. В данном случае предполагается доступ к одной или нескольким базам данных, используемых удаленным приложением.
- Комплексная интеграция. Коммерческие решения по веб-интеграции, как правило, включают все три типа интеграции

Использование веб-интеграции выгодно по многим причинам:

- Веб-интеграция позволяет развертывать информационные системы на базе сторонних приложений без необходимости разбираться в их родительских системах, программных средах и архитектурах баз данных.
- SOA и веб-сервисы используют программный язык и платформено-независимые интерфейсы между приложениями корпоративной инфраструктуры ИТ. Это дает очевидные преимущества в поддержке, управляемости, развертывании информационных сетей.
- Веб-интеграция позволяет конструировать комплексную функциональность, комбинируя разнородные компоненты посредством протоколов веб-сервисов.
- Веб-интеграция позволяет использовать веб-сервисы разработчиков.
- Веб-интеграция позволяет развивать программные интерфейсы приложений через протоколы веб-сервисов без программирования.

Интеграция на основе XML

Большое количество систем, стандартов и технологий приводит к тому, что эффективно связать разные источники данных в одну систему не получается. Даже такие, на первый взгляд однородные источники, как системы управления базами данных, применяют языки запросов и форматы представления выбираемой информации, которые редко полностью совместимы между собой. Как следствие, проекты интеграции в таких условиях требуют больших усилий - требуется вникать в детали различных баз данных, протоколов, операционных систем и так далее. В результате *интеграция* нескольких приложений или систем реализуется *по* схеме, представленной ниже:

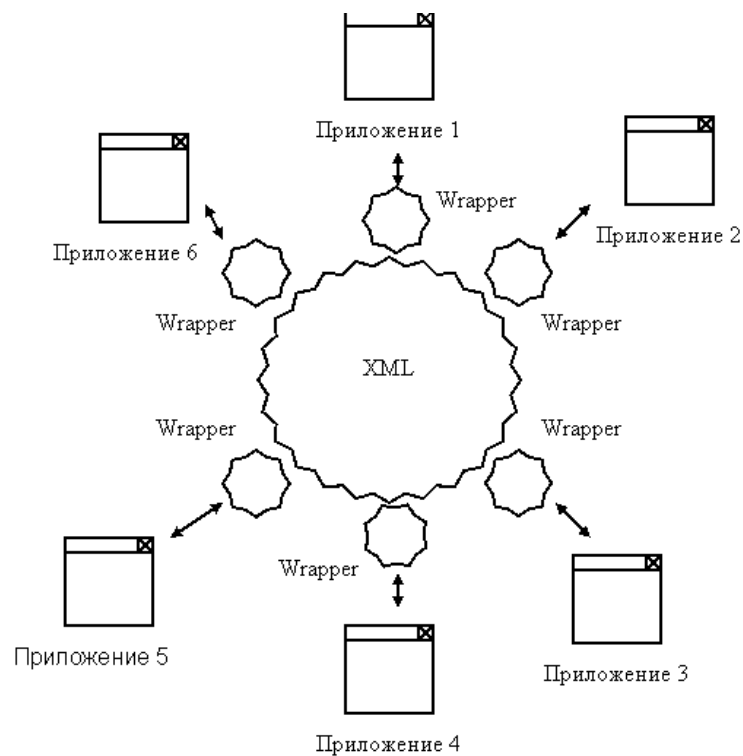


Заставить разные системы работать вместе - чрезвычайно трудоемкая задача. Идея использования XML в интеграции информационных систем сводится к созданию общего XML-языка, которым могла бы пользоваться каждая из них.

Такое решение сразу же намного упрощает проект. Вместо реализации взаимодействия между каждой парой систем следует всего лишь научить каждую из них "говорить" на XML языке. Иначе говоря, все сводится к разработке нескольких wrappers (wrapper - упаковщик, программное средство создания системной оболочки для стандартизации внешних обращений и изменения функциональной ориентации действующей системы), которые будут переводить со стандартного XML-языка интегрированной системы на язык, понятный каждой системе в отдельности.

В принципе, *интеграция по XML-схеме* не отличается коренным образом от интеграции на основе любого другого общего стандарта. Вместе с тем, она имеет *целый* ряд весомых преимуществ:

- XML языки не зависят от аппаратных и программных платформ, что позволяет связывать разнородные системы;
- выразительная мощность XML достаточно велика для того, чтобы описать данные практически любой сложности;



Веб-сервисы

Веб-сервис (web service) — программная система, имеющая *идентификатор URI*, и общедоступные интерфейсы которой определены на языке *XML*. Описание этой программной системы может быть найдено другими приложениями, которые могут взаимодействовать с ней в соответствии с этим описанием посредством сообщений, основанных на *XML*, и передаваемых с помощью интернет-протоколов. Веб-служба является единицей модульности при использовании *сервис-ориентированной архитектуры* приложения.

Веб-сервисы *.NET* имеют следующие достоинства:

- **Открытость стандартов.** В веб-сервисах отсутствуют какие-либо скрытые или недоступные элементы. Каждый аспект технологии, от способа поиска веб-сервиса до ее описания и организации связи с ней, определен общедоступными стандартами.
- **Межплатформенность.** Язык программирования, который позволяет создавать XML-документы и отправлять информацию посредством HTTP, позволяет взаимодействовать с любым веб-сервисом. Можно получать веб-услугу из системы, отличной от *.NET*.
- **Простота.**
- **Поддержка сообщений на понятном человеку языке.** Переход от двоичных стандартов, применяемых в COM и CORBA, к XML-тексту позволил упростить исправление ошибок и обеспечил возможность осуществлять взаимодействие с веб-сервисами по обычным каналам HTTP.

Реализация веб-сервисов *.NET* осуществляется так же просто, как и активизация удаленной веб-сервиса или *вызов метода* локального класса. Это достигается за счет применения инструментов, предоставляемых системой *.NET Framework*, которые позволяют создать полноценный *веб-сервис*. При этом выполняются следующие действия:

1. Веб-сервис разрабатывается как .NET-класс с атрибутами, которые идентифицируют его как веб-сервис с некоторыми функциями.
2. В среде .NET автоматически создается документ WSDL, где описывается, как клиент должен взаимодействовать с веб-сервисом.
3. Потребитель находит созданный веб-сервис и может добавить соответствующую веб-ссылку в проект Visual Studio .NET.
4. В среде .NET осуществляется автоматическая *проверка документа WSDL* и генерируется прокси-класс, который позволяет потребителю взаимодействовать с веб-сервисом.
5. Потребитель вызывает один из методов вашего класса веб-сервиса. С его точки зрения этот вызов внешне ничем не отличается от вызова метода любого другого класса, хотя взаимодействие происходит на самом деле с прокси-классом, а не с веб-сервисом.
6. Прокси-класс преобразует, переданные параметры в сообщение SOAP и отправляет его веб-сервису.
7. Затем прокси-класс получает SOAP-ответ, преобразует его в соответствующий тип данных и возвращает его как обычный тип данных .NET.
8. Потребитель использует полученные данные.

При работе веб-сервисов *.NET* используется технология *ASP .NET*, являющаяся частью системы *.NET Framework*. Она также требует поддержки со стороны сервера *Microsoft IIS*.

Работа веб-сервисов построена на использовании нескольких открытых стандартов:

- *XML* - расширяемый язык разметки, предназначенный для хранения и передачи структурированных данных;
- *SOAP* - протокол обмена сообщениями на базе XML;
- *WSDL* - язык описания внешних интерфейсов веб-сервисов на базе XML;
- *UDDI* - универсальный интерфейс распознавания, описания и интеграции (Universal Discovery, Description, and Integration). Каталог веб-сервисов и сведений о компаниях, предоставляющих веб-сервисы во всеобщее пользование или конкретным компаниям.

Спецификация WSDL

Каждый *веб-сервис* предоставляет документ *WSDL* (*Web Service Description Language* - язык описания веб-сервиса), в котором описывается все, что клиенту необходимо для работы с этим сервисом. *WSDL*-документ предоставляет простой и последовательный способ задания разработчиком синтаксиса вызова любого веб-метода. Более того, этот документ позволяет использовать инструменты автоматического генерирования прокси-классов, подобные включенным в среды *Visual Studio .NET* и *.NET Framework*. Благодаря указанным средствам использование веб-сервиса является таким же простым, как и применение локального класса.

WSDL-документ имеет основанный на *XML* формат, в соответствии с которым *информация* подразделяется на пять групп. Первые три группы представляют собой абстрактные определения, не зависящие от особенностей платформы, сети или языка, а оставшиеся две группы включают конкретные описания.

Спецификация *UDDI* (*Universal Description, Discovery, and Integration* - универсальное описание, поиск и интеграция) использует хранилище (репозиторий), где предприятия и организации могут размещать данные о предоставляемых ими сервисах. Инициаторами создания технологии *UDDI* стали более 100 компаний, включая *Sun* и *Microsoft*. Объединив свои усилия, эти компании разработали проект спецификации *UDDI*, которая по истечении 18 месяцев была стандартизирована.

Информация в этом репозитории должна обновляться вручную. С этой целью некоторые "узловые операторы" хранят идентичные копии репозитория *UDDI*. Эти компании обеспечивают хранение указанного репозитория и бесплатный доступ к нему для популяризации веб-сервисов. Кроме того, Майкрософт включила версию *UDDI* в программное обеспечение сервера *Windows .NET* для использования в корпоративных сетях интранета.

В хранилище *UDDI* содержатся сведения о предприятиях, предоставляющих веб-сервисы, о типе каждого сервиса и связях с информацией и спецификациями, относящимися к этим сервисам. Интерфейс *UDDI* сам по себе представляет собой веб-сервис. Для регистрации или поиска службы следует отправить *SOAP*-сообщение.

18. *Шаблон проектирования MVC. Примеры. Достоинства и недостатки.*

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

- **Модель** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
- **Представление** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- **Контроллер** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений

Пример:

Предположим, нам надо разработать онлайн-книжный магазин. Пользователь может выполнять следующие действия: просматривать книги, регистрироваться, покупать, добавлять пункты к текущему заказу, создавать или удалять книги (если он администратор).

У нас есть определённый контроллер для обработки всех действий, связанных с книгами (просматривать, редактировать, создавать и так далее). Давайте назовем его `books_controller` в нашем примере. Также нам нужна модель, например, `book_model`, которая обрабатывает данные и логику, связанные с позицией в магазине. В заключение, нам нужно несколько видов для представления данных, например, список книг, страница для редактирования и так далее.

Контроллер (`books_controller`) получает запрос пользователя (запрос HTTP GET или POST), проверяет запрос и параметры, а затем вызывает модель (`book_model`), запрашивая у неё список доступных книг по теме фэнтези.

Модель получает данные из базы (или из другого источника, в котором хранится информация), применяет фильтры и необходимую логику, а затем возвращает данные, которые представляют список книг.

Контроллер использует подходящий вид для представления данных пользователю. Если запрос приходит с мобильного телефона, используется вид для мобильного телефона; если пользователь использует определённое оформление интерфейса, то выбирается соответствующий вид, и так далее.

Достоинства:

- *единая концепция приложения/системы.* Ярким плюсом концепции MVC является единая глобальная архитектура приложения. Даже в сложных системах, разработчики (как те, которые разрабатывали систему, так и вновь присоединившиеся) могут легко ориентироваться в программных блоках. Например, если возникла ошибка в логике обработки данных, разработчик сразу отбрасывает 2-блока программы (`controller` и `view`) и занимается исследованием 3-го (`model`);
- *упрощенный механизм отладки приложения.* В рамках разработки web-приложения механизм визуализации теперь сконцентрирован в одном программном блоке, упростились механизмы опционального вывода графических элементов и их отладка;

- *увеличение переиспользования кода.* Механизм наследования от родительских классов позволяет с одной стороны наделять новые модели, контроллеры и представления широким кастомным функционалом, с другой – уменьшает количество одинакового кода.

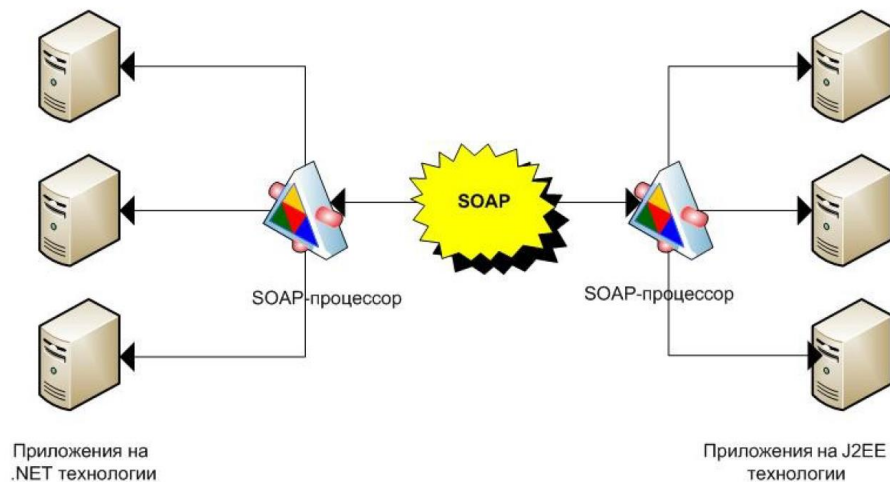
Недостатки:

- *необходимость использования большего количества ресурсов.* Отрицательный момент обусловлен тем, что все три фундаментальных блока являются абсолютно независимыми и взаимодействуют между собой исключительно путем передачи данных. Controller должен всегда загрузить (и при необходимости создать) все возможные комбинации переменных и передать их в Model. Model, в свою очередь, должен загрузить все данные для визуализации и передать их во View;
- *усложнение механизма разделения программы на модули.* В концепции MVC наличие трех блоков (Model, View, Controller) прописано жестко. Соответственно каждый функциональный модуль должен состоять из трех блоков, что в свою очередь, несколько усложняет архитектуру функциональных модулей программы. Это условие, конечно, не всегда выполняется на 100%, так как могут быть ситуации, когда может отсутствовать модель или представление, но это частные, достаточно редкие случаи;
- *Усложнение процесса расширения функционала.* Проблема проистекает из вышеописанной. Недостаточно просто написать функциональный модуль и подключить его в одном месте программы. Каждый функциональный модуль должен состоять из трех частей, и каждая из этих частей должна быть подключена в соответствующем блоке.

19. *Протокол SOAP. Краткая характеристика. Общая структура SOAP сообщения. Обработка ошибок в SOAP-сообщениях. SOAP-сообщения с вложениями. Сравнительная характеристика SOAP и REST.*

SOAP - протокол обмена структурированными сообщениями в распределённой вычислительной среде. Поддерживается консорциумом W3C.

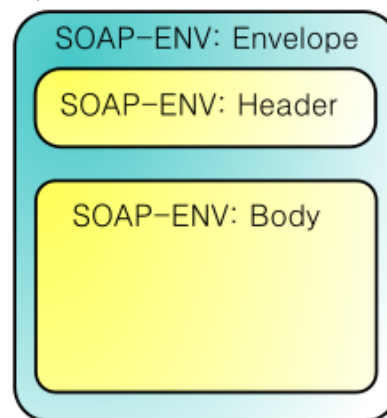
SOAP основан на языке XML и расширяет некоторый протокол прикладного уровня — HTTP, FTP, SMTP и т.д. Как правило чаще всего используется HTTP. Вместо использования HTTP для запроса HTML-страницы, которая будет показана в браузере, SOAP отправляет посредством HTTP-запроса XML-сообщение и получает результат в HTTP-отклике.



Общая структура SOAP сообщения.

Сообщение SOAP выглядит так:

- **Envelope** — корневой элемент XML-документа, который определяет сообщение и пространство имен, использованное в документе.
- **Header** — содержит атрибуты сообщения, например, информация о безопасности или о сетевой маршрутизации.
- **Body** — содержит сообщение, которым обмениваются приложения.
- **Fault** — необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений.



```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
  xmlns:t="www.example.com">
  <SOAP-ENV:Header>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <t:CurrentDate>
      <Year>2011</Year>
      <Month>February</Month>
      <Day>12</Day>
      <Time>18:02:00</Time>
    </t:CurrentDate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

В SOAP-сообщениях могут передаваться данные различных типов (числа, даты, массивы, строки и т.п.). Определение этих типов данных выполняется в схемах XML (обычно — XSD). Типы, определенные в схеме, заносятся в пространство имен, идентификатор которого служит значением атрибута encodingStyle.

Атрибут actor задает целевой SOAP-узел — тот, который расположен в конце цепочки и будет обрабатывать заголовок полностью. Атрибут actor может встречаться в отдельных блоках заголовка, указывая узел-обработчик этого блока. После обработки блок удаляется из SOAP-сообщения.

Атрибут mustUnderstand. Тип данных — boolean. По умолчанию 0. Если значение равно 1, то SOAP-узел при обработке элемента обязательно должен учитывать его синтаксис, определенный в схеме документа, или совсем не обрабатывать сообщение. Это повышает точность обработки сообщения.

Обработка ошибок в SOAP-сообщениях

Если SOAP-сервер, обрабатывая поступившее SOAP-сообщение, обнаружит ошибку, то он прекратит обработку и отправит клиенту SOAP-сообщение, содержащее один элемент Fault с сообщением об ошибке.

В версии 1.1 элемент Fault имел 4 дочерних элемента:

- код ошибки faultcode — предназначено для программы, обрабатывающей ошибки;
 - описание ошибки faultstring – словесное описание типа ошибки, предназначено для человека;
 - место обнаружения ошибки faultactor – адрес URI сервера, заметившего ошибку.
- Промежуточные SOAP-узлы обязательно записывают этот элемент, целевой SOAP-сервер не обязан это делать;

- Детали ошибки detail — описывают ошибки, встреченные в теле Body послания, но не в его заголовке. Если при обработке тела ошибки не обнаружены, то этот элемент отсутствует.

В версии SOAP 1.2 содержание элемента Fault изменилось. В него входят два обязательных элемента и три необязательных элемента.

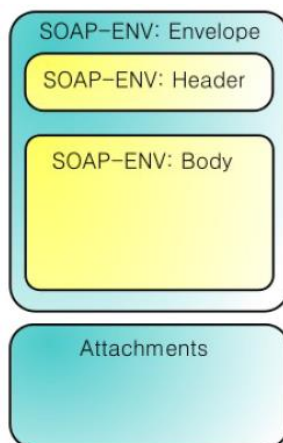
Обязательные элементы:

- Код ошибки code. Он содержит обязательный вложенный элемент value с кодом ошибки и необязательный вложенный элемент subcode, также содержащий элемент value с уточняющим кодом ошибки и элемент subcode, и далее все повторяется рекурсивно.
- Причина ошибки Reason. Содержит необязательный атрибут xml:lang, указывающий язык сообщения, и произвольное число вложенных элементов с описанием ошибки.

Необязательные элементы:

- Node — адрес URI промежуточного SOAP-узла, заметившего ошибку.
- Role — роль SOAP-узла, заметившего ошибку.
- Detail — описание ошибки, замеченной при обработке тела Body послания, но не его заголовка.

SOAP сообщения с вложениями



Двоичные данные включаются в сообщение в виде «вложения». В 2000 году консорциумом W3C выпущена спецификация «SOAP-сообщения с вложениями» (SOAP with Attachments), опирающаяся на версию SOAP 1.1.

Этот протокол определяет пересылку SOAP-сообщения внутри MIME-сообщения, состоящего из нескольких частей. Первая часть MIME-сообщения - часть SOAP - содержит XML: конверт SOAP с вложенными в него заголовком и телом сообщения.

Остальные части - вложения - содержат данные в любом формате, двоичном или текстовом. Каждая часть предваряется MIME-заголовком, описывающим формат данных части и содержащим идентификатор части (Content-ID). По этому идентификатору тело SOAP- сообщения может ссылаться на вложения (href). Приведенный ниже фрагмент должен дать представление о структуре SOAP-сообщения с вложениями. Жирным шрифтом в нем выделен идентификатор вложения и ссылка на него из основной части сообщения.

Нет прямого сравнения между SOAP и REST API. Но есть несколько моментов, которые будут перечислены ниже, что поможет вам выбрать между этими двумя веб-сервисами.

- Поскольку SOAP является протоколом, он следует строгому стандарту, чтобы разрешить связь между клиентом и сервером, тогда как REST — это архитектурный стиль, который не следует никакому строгому стандарту, но следует шести ограничениям, определенным Роем Филдингом в 2000 году. Эти ограничения — Унифицированный Интерфейс, клиент-сервер, без сохранения состояния, кэшируемая, многоуровневая система, код по требованию.

- SOAP использует только XML для обмена информацией в своем формате сообщений, тогда как REST не ограничивается XML и его выбор реализатора, какой Media-Type использовать, например, XML, JSON, Plain-text. Более того, REST может использовать протокол SOAP, а SOAP не может использовать REST.

- От имени сервисов взаимодействует с бизнес-логикой, SOAP использует @WebService, тогда как REST вместо использования интерфейсов использует URI, такой как @Path.

- SOAP сложно реализовать и требует большей пропускной способности, тогда как REST прост в реализации и требует меньшей пропускной способности, например, для смартфонов

- Преимущества SOAP по сравнению с REST, поскольку SOAP поддерживает транзакции с жалобами ACID. Некоторые из приложений требуют возможности транзакции, которая принимается SOAP, тогда как в REST ее нет.

- На основе безопасности, SOAP имеет SSL (S ecurе S ocket L Эйер) и WS-безопасности, тогда как REST имеет SSL и HTTPS. В случае пароля банковского счета, номера карты и т. Д. SOAP предпочтительнее, чем REST. Проблема безопасности связана с требованиями вашего приложения, вы должны создать систему безопасности самостоятельно. Речь идет о том, какой тип протокола вы используете.

- Обычно, SOAP используется в крупных корпоративных системах со сложной логикой, когда требуются четкие стандарты, подкрепленные временем. XML-RPC, пожалуй, устарел и не имеет смысла ввиду наличия собрата JSON-RPC. RPC-протоколы подойдут для совсем простых систем с малым количеством единиц информации и API-методов.

- Если же вы разрабатываете публичное API и логика взаимодействия во многом покрывается четверкой методов CRUD — смело выбирайте REST. Он наиболее популярен в WEB. Яндекс, Google и другие используют именно его для своего API.

20. *REST. Краткая характеристика, методы запроса, архитектурные ограничения, правила REST API. Сравнительная характеристика SOAP и REST. Сравнительная характеристика REST и GraphQL.*



REST представляет собой архитектурный стиль, который определяет набор ограничений, которые будут использоваться для создания веб — служб. **REST API** — это простой и гибкий способ доступа к веб-сервисам без какой-либо обработки.

Технология REST, как правило, предпочтительнее более надежной технологии SOAP, поскольку REST использует меньшую пропускную способность, прост и гибок, что делает ее более пригодной для использования в Интернете. Он используется для получения или предоставления некоторой информации из веб-сервисов. Все общение, выполненное через REST API, использует только HTTP-запрос.

- В **HTTP** есть пять методов, которые обычно используются в архитектуре на основе REST, т.е. POST, GET, PUT, PATCH и DELETE. Они соответствуют операциям создания, чтения, обновления и удаления (или CRUD) соответственно. Есть и другие методы, которые используются реже, такие как OPTIONS и HEAD.

- **GET:** метод HTTP GET используется для **чтения** (или получения) представления ресурса. В безопасном пути GET возвращает представление в XML или JSON и код ответа HTTP 200 (OK). В случае ошибки чаще всего возвращается 404 (НЕ НАЙДЕНО) или 400 (ПЛОХОЙ ЗАПРОС).

- **POST:** глагол POST чаще всего используется для **создания** новых ресурсов. В частности, он используется для создания подчиненных ресурсов. То есть подчиняться какому-то другому (например, родительскому) ресурсу. При успешном создании верните HTTP-статус 201, вернув заголовок Location со ссылкой на вновь созданный ресурс с HTTP-статусом 201. **ПРИМЕЧАНИЕ:** POST не является ни безопасным, ни идемпотентным.

- **PUT:** используется для **обновления** возможностей. Однако PUT также можно использовать для **создания** ресурса в случае, когда идентификатор ресурса выбирается клиентом, а не сервером. Другими словами, если PUT относится к URI, который содержит значение несуществующего идентификатора ресурса. При успешном обновлении верните 200 (или 204, если не возвращаете никакого содержимого в теле) из PUT. Если для создания используется PUT, верните HTTP-статус 201 при успешном создании. PUT не безопасная операция, но она идемпотентна..

- **DELETE:** используется для **удаления** ресурса, идентифицируемого URI. При успешном удалении верните HTTP-статус 200 (OK) вместе с телом ответа.

Унифицированный интерфейс

это ключевое ограничение, которое различает REST API и Non-REST API. Он предполагает, что должен быть единый способ взаимодействия с данным сервером независимо от устройства или типа приложения (веб-сайт, мобильное приложение). Существует четыре основных принципа единого интерфейса:

- **На основе ресурсов:** индивидуальные ресурсы идентифицируются в запросах. Например: API / пользователи.
- **Управление ресурсами с помощью представлений:** у клиента есть представление ресурса, и оно содержит достаточно информации для изменения или удаления ресурса на сервере, если у него есть разрешение на это. Пример: Обычно пользователь получает идентификатор пользователя, когда пользователь запрашивает список пользователей, а затем использует этот идентификатор для удаления или изменения этого конкретного пользователя.
- **Сообщения с самоописанием.** Каждое сообщение содержит достаточно информации, чтобы описать, как обрабатывать сообщение, чтобы сервер мог легко проанализировать запрос.
- **Гипермедиа как движок состояния приложения (HATEOAS).** Необходимо включать ссылки для каждого ответа, чтобы клиент мог легко находить другие ресурсы.

Отсутствие состояния

это означает, что необходимое состояние для обработки запроса содержится в самом запросе, и сервер не будет хранить ничего, связанного с сеансом. В REST клиент должен включать всю информацию, необходимую серверу для выполнения запроса, как часть параметров запроса, заголовков или URI. Отсутствие состояния обеспечивает большую доступность, поскольку серверу не нужно поддерживать, обновлять или передавать это состояние сеанса. Недостатком является то, что клиенту необходимо отправить слишком много данных на сервер, что снижает объем оптимизации сети и требует большей пропускной способности.

Кэшируемый

каждый ответ должен включать в себя, является ли ответ кэшируемым или нет, и на сколько времени ответы могут быть кэшированы на стороне клиента. Клиент вернет данные из своего кэша для любого последующего запроса, и нет необходимости отправлять запрос снова на сервер. Хорошо управляемое кэширование частично или полностью исключает некоторые взаимодействия клиент-сервер, что еще больше повышает доступность и производительность. Но иногда есть вероятность, что пользователь может получить устаревшие данные.

Клиент-сервер

приложение REST должно иметь архитектуру клиент-сервер. Клиент — это тот, кто запрашивает ресурсы и не занимается хранением данных, которое остается внутренним для каждого сервера, а сервер — это тот, кто хранит ресурсы и не имеет отношения к пользовательскому интерфейсу или состоянию пользователя. Они могут развиваться независимо. Клиенту не нужно ничего знать о бизнес-логике, а серверу не нужно ничего знать о интерфейсе внешнего интерфейса.

Правила REST API

Существуют определенные правила, которые следует учитывать при создании конечных точек REST API.

- REST основан на ресурсе или существительном, а не на действии или глаголе. Это означает, что URI API REST всегда должен заканчиваться существительным. Пример: / api / users — хороший пример, но / api? Type = users — плохой пример создания REST API.

- HTTP глаголы используются для определения действия. Некоторые из HTTP-глаголов — GET, PUT, POST, DELETE, UPDATE, PATCH.
- Веб-приложение должно быть организовано в ресурсы, подобные пользователям, и затем использовать HTTP-глаголы, такие как — GET, PUT, POST, DELETE, чтобы изменить эти ресурсы. И как разработчику должно быть ясно, что нужно сделать, просто взглянув на конечную точку и используемый метод HTTP.

URI	HTTP verb	Description
api/users	GET	Get all users
api/users	POST	Add a user
api/users/1	PUT	Update a user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

- Всегда используйте множественное число в URL, чтобы поддерживать постоянный URI API во всем приложении.
- Отправьте правильный HTTP-код, чтобы указать на успех или статус ошибки

Сравнительная характеристика SOAP и REST

Нет прямого сравнения между SOAP и REST API. Но есть несколько моментов, которые будут перечислены ниже, что поможет вам выбрать между этими двумя веб-сервисами.

- Поскольку SOAP является протоколом, он следует строгому стандарту, чтобы разрешить связь между клиентом и сервером, тогда как REST — это архитектурный стиль, который не следует никакому строгому стандарту, но следует шести ограничениям, определенным Роем Филдингом в 2000 году. Эти ограничения — Унифицированный Интерфейс, клиент-сервер, без сохранения состояния, кэшируемая, многоуровневая система, код по требованию.

- SOAP использует только XML для обмена информацией в своем формате сообщений, тогда как REST не ограничивается XML и его выбор реализатора, какой Media-Type использовать, например, XML, JSON, Plain-text. Более того, REST может использовать протокол SOAP, а SOAP не может использовать REST.

- От имени сервисов взаимодействует с бизнес-логикой, SOAP использует @WebService, тогда как REST вместо использования интерфейсов использует URI, такой как @Path.

- SOAP сложно реализовать и требует большей пропускной способности, тогда как REST прост в реализации и требует меньшей пропускной способности, например, для смартфонов

- Преимущества SOAP по сравнению с REST, поскольку SOAP поддерживает транзакции с жалобами ACID. Некоторые из приложений требуют возможности транзакции, которая принимается SOAP, тогда как в REST ее нет.

- На основе безопасности, SOAP имеет SSL (Secure Socket Layer) и WS-безопасности, тогда как REST имеет SSL и HTTPS. В случае пароля банковского счета, номера карты и т. Д. SOAP предпочтительнее, чем REST. Проблема безопасности связана с требованиями вашего приложения, вы должны создать систему безопасности самостоятельно. Речь идет о том, какой тип протокола вы используете.

- Обычно, SOAP используется в крупных корпоративных системах со сложной логикой, когда требуются четкие стандарты, подкрепленные временем. XML-RPC, пожалуй, устарел и не имеет смысла ввиду наличия собрата JSON-RPC. RPC-протоколы подойдут для совсем простых систем с малым количеством единиц информации и API-методов.
- Если же вы разрабатываете публичное API и логика взаимодействия во многом покрывается четверкой методов CRUD — смело выбирайте REST. Он наиболее популярен в WEB. Яндекс, Google и другие используют именно его для своего API.

Сравнительная характеристика REST и GraphQL

1. Получение данных

У REST есть проблемы с комплектацией: получаемые из конечной точки данные часто бывают избыточными или неполными. У GraphQL этого недостатка нет.

2. Структура ответа

В REST-архитектуре структуру ответа определяет сервер, а в GraphQL – клиентский код.

3. Автоматическое кэширование

REST использует кэширование ответов по умолчанию, в то время как GraphQL не имеет системы автоматического кэширования. Однако существуют различные клиенты GraphQL (Apollo, Relay), которые предоставляют эту возможность.

4. Обработка ошибок

В REST обработка ошибок намного проще. GraphQL обычно возвращает статус 200 OK даже с ошибкой, но при использовании специальных клиентов эта проблема легко решается.

JavaScript

Node.js - событийно-ориентированная платформа, применяемая для создания веб-приложений, которая обеспечивает возможность использовать JavaScript на стороне сервера.

Так как до появления Node.js язык JavaScript использовался только на стороне клиента, появление такой платформы было встречено разработчиками с энтузиазмом. По сути, это открывало новые возможности для создания приложений с высокой производительностью и масштабируемостью.

Сообщество Node.js растет и развивается. Регулярно генерируются новые идеи, а в результате появляются новые инструменты и библиотеки. Благодаря таким темпам развития разработчики получили в свое распоряжение широкий ассортимент фреймворков:

- **Express**, как самый гибкий, простой и быстрый фреймворк
- **Koa**, как версию *Express* нового поколения, созданную той же командой разработчиков
- **Sails**, как созданный для быстрой разработки приложений на принципах *Ruby on Rails* и *Express*.

Express

Express используется для разработки приложений достаточно давно и благодаря своей стабильности прочно занимает позицию одного из самых [популярных фреймворков Node.js](#).

Основная особенность этого фреймворка заключается в том, что для *Express* характерен **небольшой объем базового функционала**. Все остальные нужные вам функции нужно будет добирать за счет внешних модулей. По сути, *Express* в чистом виде – это сервер и у него может не быть ни одного модуля.

Благодаря такому минимализму разработчик изначально получает в свое распоряжение **легкий и быстрый инструмент**, который он может расширять и развивать.

Достоинства:

- простота
- гибкость
- хорошая масштабируемость
- развитое сообщество
- подробная документация
- широкий выбор подключаемых модулей

Недостатки:

- большой объем ручной работы
- используется устаревший подход *callbacks* функций

Кoa

Кoa был создан командой разработчиков, как **вариант фреймворка *Express* в новом поколении**. Такая улучшенная версия разрабатывалась для создания веб-приложений и API с повышенной производительностью. Соответственно, создатели стремились учесть все недостатки предшественника и сделать его более современным и удобным в использовании.

- *Кoa* обладает практически таким же функционалом и превосходит *Express* по легкости.
- Характерной особенностью *Кoa* является использование генераторов ES6.
- **Генераторы** - тип функций, которые могут быть запущены, остановлены и возобновлены независимо от того, на каком этапе выполнения они находятся, и при этом сохраняют свой контент.
- Применение генераторов ES6 в *Кoa* позволяет исключить обратные вызовы, **уменьшает для разработчиков объем работы с кодом и снижает вероятность ошибок**.
- Благодаря тому, что создателями *Кoa* уже были учтены минусы, выявленные в процессе работы с *Express*, этот фреймворк может похвастаться тем, что его применение **существенно упрощает адаптацию под конкретные запросы клиентов (кастомизацию)**.

Достоинства:

- легкий
- гибкий
- быстрый
- генераторы ES6
- лучшая кастомизация

Недостатки:

- недостаточная поддержка сообщества

SAILS

Этот фреймворк разработан как **полный готовый продукт**, который уже включает в себя достаточный функционал для того, чтобы можно было начать работу, и при этом использует минимальное количество внешних модулей.

Соответственно, такой фреймворк изначально будет **тяжелее, чем два предыдущих**.

С одной стороны, от разработчика требуется минимальное количество усилий, так как для создания приложения используется собственный функционал фреймворка. Нет необходимости вникать в тонкости процесса - можно просто взять готовое проверенное решение.

Отличительной особенностью фреймворка является встроенная технология программирования *Waterline* ORM (англ. Object-Relational Mapping), которая используется для обеспечения связи с различными базами данных.

Waterline не поддерживает транзакции, новые фичи и исправления ошибок вносятся несвоевременно.

Достоинства:

- богатый функционал
- поддержка Socket.io
- документация в одном месте
- легче найти специалиста с опытом работы на Sails

Недостатки:

- тяжелый
- медленный
- ограничения Waterline
- недостаточно подробная документация

Python

Django

Django — это бесплатная среда разработки веб-приложений Python с открытым исходным кодом, которая следует шаблону **Model-Template-View (MTV)**. Он был создан осенью 2003 года Адрианом Головатым и Саймоном Уиллисоном.

Django был создан для упрощения процесса разработки сайта. Основное внимание уделяется повторно используемым компонентам, меньшему количеству кода и быстрой разработке.

Flask

Flask — это микрофреймворк с удивительным началом. Это фактически началось, как первоапрельская шутка. Перед разработкой **Flask** Армин Роначер, создатель Flask, написал два других решения:

- Werkzeug (серверная среда)
- Jinja2 (библиотека шаблонов).

Армин подумал, что было бы интересно взять эти два решения и собрать их в zip-файл, поэтому он написал **Denied Framework** (как он называл его прежде, и назвал его Flask). Когда разработчик устанавливает этот Denied Framework, программа установки автоматически разархивирует файл и запускает эти два решения одновременно.

Основные различия между Django и Flask

Django предоставляет свой собственный **Django ORM** (объектно-реляционное отображение) и использует модели данных, в то время как **Flask** вообще не имеет моделей данных. Модели данных позволяют разработчикам связывать таблицы базы данных с классами на языке программирования, чтобы они могли работать с моделями так же, как ссылки на базы данных. Почему у Flask нет модели данных? Потому что философия Flask отличается от философии Django.

Django связывает все вместе, а Flask более модульный.

Основное различие между Django и Flask в том, что **Django предоставляет полнофункциональную среду Model-View-Controller.**

Цель Django — упростить процесс разработки сайта. Он основан на меньшем количестве кода, повторно используемых компонентах и быстрой разработке.

Flask, с другой стороны, представляет собой **микрофреймворк, основанный на концепции «хорошо выполнять одну вещь»**. Он не предоставляет ORM и поставляется только с базовым набором инструментов для веб-разработки.

Приложения для **Flask** в основном представляют собой одностраничные приложения (SPA). Это хороший выбор для небольших и средних сайтов, таких как форумы и личные блоги. **Django** идеально подходит для крупных проектов, таких как сайты электронной коммерции и CMS.

Преимущества и недостатки

Производительность

- Если учесть, что производительность **Django vs Flask** имеет хорошие результаты и используется на сайтах с большим трафиком, что является отличным показателем их эффективности.

Пакеты

- **Flask** минималистичен и не имеет ограничений, то есть разработчики могут реализовать именно то, что они хотят, используя внешние библиотеки. Это делает **Flask** гибким и расширяемым.
- **Django**, с другой стороны, имеет огромное количество встроенных пакетов. Если быть точным, то на сентябрь 2019 года было 4046 пакетов Django. Это означает, что вы, скорее всего, найдете пакет для сборки и запуска своего приложения с меньшими усилиями.

Сообщество

- В случае сравнения **Python Django с Flask**, Django имеет огромное и активное сообщество разработчиков. Если у вас есть какие-либо вопросы, вы можете задать их на различных веб-порталах и форумах, таких как **Stack Overflow**, и, скорее всего, получите ответ. Кроме того, не трудно найти работу Django.
- Сообщество Flask не такое большое, как сообщество Django.

Случаи использования

- **Django** был разработан для быстрой разработки сложных веб-приложений. Он предоставляет разработчикам необходимые инструменты для реализации масштабируемой и поддерживаемой функциональности. С другой стороны, простота Flask позволяет разработчикам быстрее создавать небольшие приложения.

C#

.NET (ASP.NET/ASP.NET Core)

ASP.NET (Active Server Pages для .NET) — технология создания веб-приложений и веб-сервисов от компании Майкрософт. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP. На данный момент последней версией этой технологии является ASP.NET 5.

Платформа ASP.NET MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Платформа Web API представляет иной способ построения приложения ASP.NET несколько отличный от ASP.NET MVC. Web API представляет собой веб-службу, которая может взаимодействовать с различными приложениями. При этом приложение может быть веб-приложением ASP.NET, либо может быть мобильным или обычным десктопным приложением.

Платформа Web API не является частью фреймворка ASP.NET MVC и может быть задействована как в связке с MVC, так и в соединении с Web Forms. Поэтому в Web API имеется своя система версий. Так, первая версия появилась с .net 4.5. А вместе с .NET 4.5.1 и MVC 5 вышла Web API 2.0.

Платформа ASP.NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

Платформа ASP.NET Core построен на основе кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS X, Linux. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS.

.NET Core — это горизонтальное развитие программной платформы .NET в другие операционные системы, в которых она обеспечивает возможность использования приложений, разработанных для Windows.

ОТКРЫТОСТЬ

Как уже было сказано, платформа .NET является разработкой и собственностью компании Microsoft, но была и остаётся бесплатной в использовании.

Программный код платформы был размещён в публичном репозитории [GitHub](#) и теперь распространяется как продукт с открытым исходным кодом.

КРОССПЛАТФОРМЕННОСТЬ

Сегодня на потребительском рынке наиболее распространены три: Windows, macOS и Linux (с вариациями).

Разработчик удачного, востребованного пользователями приложения заинтересован в его распространении в разных операционных системах. Однако в большинстве случаев просто так запустить приложение, разработанное для другой операционной системы, невозможно — требуется его «портировать». В каких-то ситуациях можно просто перекомпилировать исходный текст программы для новой операционной системы. Но в большинстве случаев такой трюк не проходит, и требуется переписывать программу целиком или какие-то её части. Именно так пришлось действовать компании Microsoft, когда в середине 90-х она захотела сделать свой Office доступным в MacOS.

Microsoft портировала не текущую версию платформы .NET, существующую в ОС Windows, а некую производную от неё, которую назвала [.NET Core](#).

В значительной степени её функциональность совпадает с функциональностью исходной .NET. В целом, можно считать, что .NET и .NET Core совместимы друг с другом.

Сейчас платформа .NET Core существует для трёх операционных систем: [Windows](#), [Linux](#) и [macOS](#). То есть приложение, разработанное на базе .NET Core, может быть без изменения запущено во всех операционных системах, в которых имеется указанная платформа.

В настоящее время на платформе .NET Core не реализованы Windows Forms, Windows Presentation Foundation (WPF), WebForms. Это означает, что приложения с развитым графическим интерфейсом портироваться таким способом в настоящее время не могут. Однако в информационных системах существует много задач, решение которых не требует пользовательского графического интерфейса.

Формально модульная архитектура снижает быстродействие системы в целом. Но при этом увеличивается её гибкость, повышается надёжность, сокращается срок её разработки, особенно при использовании уже готовых модулей.

- .NET Core выпускается через NuGet в небольших сборочных пакетах.
- .NET Framework — это одна большая сборка, которая содержит большинство основных функций.
- .NET Core доступен в виде небольших функционально-ориентированных пакетов.
- Такой модульный подход позволяет разработчикам оптимизировать свое приложение, включая в него только те пакеты NuGet, которые им необходимы.

- Преимущества меньшей площади приложения включают более строгую безопасность, снижение уровня обслуживания, повышение производительности и снижение затрат в модели с платой за использование.

Платформа .NET Core

.NET Core Platform содержит следующие основные части —

- **.NET Runtime** — предоставляет систему типов, загрузку сборок, сборщик мусора, встроенное взаимодействие и другие базовые сервисы.
- **Фундаментальные библиотеки** — набор библиотек фреймворков, которые предоставляют примитивные типы данных, типы композиций приложений и фундаментальные утилиты.
- **SDK & Compiler** — набор инструментов SDK и языковых компиляторов, которые обеспечивают базовый опыт разработчика, доступны в .NET Core SDK.
- **Хост приложения 'dotnet'** — используется для запуска приложений .NET Core. Он выбирает среду выполнения и размещает среду выполнения, предоставляет политику загрузки сборок и запускает приложение.

Java

Spring MVC

[Spring MVC](#) — один из самых популярных каркасов для web-сайтов на данный момент. Поэтому он обладает развитой экосистемой — если вам не хватает чего-то при разработке, вы легко сможете найти дополнение для фреймворка, которое добавит необходимую возможность. Огромное сообщество разработчиков всегда будет готово прийти вам на помощь, если у вас возникнут вопросы. С помощью Spring MVC вы сможете без особого труда разрабатывать большие и быстроразвивающиеся сайты. Тем не менее, есть у него и несколько минусов: данный фреймворк довольно сложен для изучения, поэтому, несмотря на [хорошую документацию](#), новички, скорее всего, испытают трудности в его освоении. Кроме того, он не предоставляет удобных инструментов для создания пользовательского интерфейса.

Spring Framework может быть задействован на всех архитектурных слоях, применяемых при разработке web-приложений

- Позволяет свободно связывать модули и легко их тестировать
- Поддерживает декларативное программирование
- Избавляет от самостоятельного создания фабричных и синглтон-классов
- Поддерживает различные способы конфигурации

Vaadin

базируется на [Google Web Toolkit](#) (он используется для вывода элементов пользовательского интерфейса и взаимодействия с сервером на стороне клиента), что добавляет архитектуре этого фреймворка сложности. Тем не менее, знание Google Web Toolkit не является обязательным для разработки на Vaadin.

Этот фреймворк один из самых простых для изучения, а главной его «фичей» является «режим дизайна» — в нем можно строить пользовательский интерфейс в режиме WYSIWIG и писать логику отдельно для каждого компонента.

Документация. Кроме [множества гайдов и демонстрационных примеров](#) существует [«Книга Vaadin»](#) — полный справочник по фреймворку. Книга бесплатно доступна онлайн. Vaadin идеально подходит как для новичков, так и для профессионалов. Из минусов: не самая развитая экосистема и плохая масштабируемость для больших сайтов.

Одним из главных преимуществ использования фреймворка Vaadin является возможность использовать только Java, избегая прикруток языков веба (HTML, JS, XML). Но есть и обратная сторона. Во-первых, Vaadin — это надстройка на GWT, что в конечном счете обяжет вас изучить и этот фреймворк. Во-вторых, он медлителен.

GWT

Vaadin и GWT чрезвычайно похожи и всячески связаны, но у последнего есть одно важное отличие — он компилируется в JavaScript, с которым сервер и работает. Писать вам здесь тоже придется на чистом и немного ограниченном Java, но в отличие от Vaadin здесь больше самостоятельности в обработке запросов. С одной стороны, это позволяет увеличить быстродействие, наладить работу с памятью, с другой — потребует от вас больше знаний и навыков в разработке.

В остальном одни плюсы: множество API, виджетов, отличная поддержка, отсутствие необходимости безупречно знать Java.

JSF

[JSF](#) официально поддерживается Oracle. Хотя этот фреймворк и не очень подходит для быстрой разработки, он легок в использовании благодаря наличию отличной документации, отсутствию каких-либо внешних зависимостей (пока вы остаетесь в экосистеме Java EE) и богатству возможностей. Экосистема фреймворка крайне развита и представляет из себя множество библиотек на все случаи жизни, в том числе и инструменты для удобной разработки пользовательского интерфейса. Главная фишка JSF в том, что, он является частью Java EE — из этого следует отличное взаимодействие с IDE и официальная поддержка от Oracle. Все это в совокупности значительно облегчает разработку. Недостаток фреймворка в сложности его устройства.

Но есть и обратная сторона. Для работы на профессиональном уровне потребуется и отличное знание Java, и понимание веб-технологий, и опыт работы в вебе без всяких фреймворков.

PHP — один из самых популярных и востребованных языков программирования. Его активно используют крупные проекты, например, Facebook и «ВКонтакте». На PHP написаны популярные системы управления контентом (CMS), в том числе WordPress. На этом движке работает около трети всех сайтов в интернете и около 60 % сайтов на CMS.

Основным фактором в процессе выбора фреймворка должно стать предназначение проекта. Например, для простого приложения нет необходимости использовать продукт промышленного уровня, который требует массу лишних ресурсов

Чем выше популярность PHP-фреймворка, тем проще получить обратную связь от его создателей и найти дополнительные библиотеки или внешние пакеты. Имеет значение и ваше местоположение: в русскоговорящих странах предпочитают Yii, тогда как сообщество Laravel составляют по большей части англоговорящие разработчики.

Разрабатывая сайты на PHP, нельзя не уделять особенное внимание проблеме безопасности, ведь любой веб-сервис потенциально уязвим. Советуем всегда использовать такие фреймворки, которые в этом плане обладают продвинутыми современными функциями.

Благодаря подходящей подробной документации программистам легче ориентироваться в исходном коде, а значит, и заниматься самой разработкой. Фреймворки бывают разного уровня сложности: изучить Yii, к примеру, нетрудно, а вот для освоения Symfony придется приложить заметно больше усилий. Поэтому, прежде чем сделать выбор инструмента, обязательно оцените качество и количество имеющейся документации.

Топ-4 PHP-фреймворка:

- Symfony;
- Laravel;
- Yii 2;
- Zend;

Они дают возможность делать как образцовые веб-приложения, так и качественное корпоративное ПО.

Symfony

Фреймворк появился в 2005 году. Сейчас он имеет встроенную систему тестирования, позволяет взаимодействовать с компонентами, а также многократно применять код. К другим важным преимуществам Symfony относятся:

- Быстрая загрузка. Для потребителей производительность приложения стоит далеко не на последнем месте. Именно от нее зависит, будут ли вообще им пользоваться. В версии 4.2 загрузка REST API происходит за 2 мс, т.е. Symfony на сегодняшний день является самым скоростным PHP-фреймворком.
- Подробная документация. Детальная документация данного инструмента пригодится не только начинающим, но и опытным программистам. В ней понятно и с примерами описан каждый компонент фреймворка.

- Гибкость. Symfony полностью конфигурируем за счет Event Dispatcher, он способен приспособиться к самым разным требованиям проекта. Плюс ко всему, компоненты фреймворка управляются независимо один от другого.
- Надежность. Компания-разработчик занимается поддержкой Symfony уже больше 13 лет. Так что в случае возникновения проблем вам будет к кому обратиться

Laravel

Laravel изначально проектировался под создание многофункциональных веб-приложений. Он может похвастать отличным движком шаблонов, который выполняет множество обычных задач. Что касается функционала, нужно отметить Restful Routing, аутентификацию и кэширование. Благодаря этим опциям Laravel делает разработку более быстрой. Но есть и другие причины, по которым мы советуем применять этот фреймворк:

- Высокий уровень безопасности. Мы уже акцентировали внимание на том, что для PHP-фреймворка безопасность является очень важной характеристикой. В данном случае вы будете работать в безрисковой среде, потому как Laravel блокирует все вредоносные активности и обладает прекрасной защитой от случайных и скрытых внедрений SQL-кода.
- Встроенные почтовые сервисы. За счет интегрированного почтового сервиса разработанное вами веб-приложение сможет отправлять пользователям SMS и Slack уведомления. Фреймворк имеет простой в использовании API, а также драйвера для Mailgun, Amazon SES, SMTP и Mandrill.
- Обширный функционал. Разработчики могут внедрять разные виды готового функционала благодаря пакетам, плагинам, компонентам и модулям, что позволяет существенно сократить время разработки.
- Улучшенная аутентификация. Фреймворк позволяет пользователям без особых усилий создавать логику аутентификации. А это, как известно, далеко не самая простая задача.

Yii2

Высокая скорость загрузки и усиленная безопасность – вот чем известен Yii2. В этом инструменте также заложен принцип Don't repeat yourself, т.е. не повторения кода. А с помощью Ajax и JQuery фреймворк облегчает создание высоко масштабируемых веб-приложений. Достоинствами Yii2, помимо прочего, являются:

- Улучшенная безопасность. Yii2 способен защищать сайты и приложения от разного рода атак, в том числе подделки cookie, внедрения SQL-кода, CSRF и межсайтового скриптинга.
- Ускоренная разработка. Фреймворк генерирует базовые функции CRUD, поэтому программисты работают быстрее, что не только экономит время, но и уменьшает стоимость разработки.
- Простая установка. Когда каждая минута на счету, Yii радует пользователей шаблоном инсталляции. В результате установка и настройка продукта происходят в разы быстрее.

Zend Framework

Этот продукт создан по методологии agile и ориентирован на разработку корпоративных веб-приложений. Zend позволяет внедрять в проект любые специфичные фичи. Пятилетняя поддержка дает возможность делать стабильные бизнес-программы с длительным жизненным циклом.

Кроме безопасности и великолепной производительности Zend обладает такими преимуществами:

- Набор компонентов аутентификации. Процесс аутентификации усложняет PHP-разработку. Решением этой проблемы во фреймворке выступает коллекция инструментов для тестирования форм. Таким образом, этот функционал не приходится разрабатывать с нуля.
- Zend Studio. Это проприетарная интегрированная среда, в которой разработчики могут писать код намного быстрее. Отладка становится более удобной благодаря интеграции Xdebug, Z-Ray и Zend Debugger. Также есть возможность делать развертку на разных серверах, в частности поддерживается Microsoft Azure и Amazon Web Services.
- Улучшенная документация. В руководстве по фреймворку описано более 500 примеров на одной тысяче страниц. Там вы точно сможете найти подходящие решения для своих проектов. В качестве бонуса Zendcasts содержит множество обучающих роликов, посвященных функционалу этого инструмента.

С помощью PHP-фреймворков можно быстро создавать безопасные веб-приложения, надевая их при этом богатым функционалом. Сейчас есть огромное количество разных фреймворков, поэтому выбрать среди них лучший непросто.

Прежде всего, вам нужно определить нужды своего проекта. Затем оцените безопасность продукта, наличие качественной документации и большого активного сообщества.

Ruby

Ruby — это язык программирования, который был создан в 1990-х годах Юкиhiro «Мац» Мацумото. Основная задача Ruby — быстрое создание новых проектов с высокой производительностью. Это язык программирования общего назначения, такой же, как C++ или PHP. Язык программирования Ruby постоянно развивается и окружен большим сообществом.

Обширная поддержка сообщества, удобство для пользователя, простота, удобочитаемость, гибкость и жемчужины сообщества, которые являются скриптами кодирования, которые упрощают процесс разработки, являются основными преимуществами языка программирования Ruby, которые называют внутренние разработчики.

Однако создание нового программного обеспечения с использованием только Ruby — утомительная работа. Именно поэтому была создана специальная структура для оптимизации процесса разработки.

Ruby on Rails

Rails часто называют главной причиной популярности Ruby. Rails, или Ruby on Rails, является средой с открытым исходным кодом, написанной на языке программирования Ruby и основанной в 2003 году Дэвидом Хайнмайером Ханссоном, который также известен под именем DHH.

Платформа Ruby on Rails значительно упрощает процесс создания веб-сайтов и приложений, снимая общие повторяющиеся задачи с плеч разработчиков, такие как создание форм, таблиц и меню. Разработчикам не нужно создавать новый веб-сайт или веб-приложение с нуля, поскольку они могут использовать готовые решения для повторяющихся задач.

Популярность Ruby on Rails

Статус Open Source — это первое, что нужно учитывать при выборе правильной серверной среды. Это означает, что Ruby on Rails является бесплатным и может использоваться бесплатно. Любой желающий может пойти и скачать исходный код для дальнейшего использования в своих проектах.

Сообщество разрабатывает множество бесплатных дополнений, которые можно интегрировать в приложения. Они очень полезны для стартапов, которые хотят запустить новое multifunctional приложение в кратчайшие сроки.

Преимущества:

- **Обширная экосистема.** Именно его экосистема делает Ruby on Rails превосходным по сравнению со многими другими фреймворками.
- **Ruby on Rails MVC.** Ruby on Rails MVC допускает параллельную разработку и позволяет программистам ускорить процесс разработки в три раза.
- **Согласованность и чистый код.** Разработчики Ruby on Rails могут использовать готовые к использованию части кода, что упрощает реализацию многих функций. В результате код приложения является чистым и имеет высокую читаемость.
- **DRY (не повторяй себя)** — еще один принцип, на котором основан Ruby on Rails. Если есть повторяющаяся задача, то при разработке на Ruby on Rails вам не нужно писать один и тот же код снова и снова. Фреймворк воспринимает повторяющиеся задачи таким образом, что фоновые разработчики могут использовать их неограниченное количество раз.

- **Высокая масштабируемость.** Масштабируемость Ruby on Rails — еще одно преимущество. Приложение, построенное на RoR, может быть масштабировано для обработки тысяч запросов в секунду, отправленных несколькими пользователями. Отличным примером высокой производительности Ruby on Rails является платформа электронной коммерции Shopify, которая обрабатывает до 80 000 запросов в секунду.
- **Безопасность.** Ruby on Rails не рискует вопросами безопасности. Безопасность Ruby on Rails — еще одно преимущество. В инфраструктуру встроены некоторые ориентированные на безопасность функции, которые делают приложения защищенными от SQL-инъекций и XSS-атак. Кроме того, вы можете найти много драгоценных камней, которые направлены на другие угрозы безопасности.
- **Время и эффективность затрат.** Время часто является основным препятствием для стартапов. Все перечисленные выше функции в совокупности делают Ruby on Rails экономичным по времени и затратам.
- **Тестовая среда.** В Ruby on Rails есть три среды по умолчанию, а именно: производство, разработка и тестирование. Весь цикл разработки оптимизирован, и вы можете протестировать продукт, который разрабатывается на каждом этапе. Это приводит к меньшему количеству ошибок и ошибок, о которых вы должны знать и отлаживать. Это важный фактор, который необходимо учитывать при попытке определить, для чего используется Ruby on Rails.
- **Соглашение по конфигурации.** Соглашение о конфигурации является одним из ключевых принципов разработки Ruby on Rails. Это сокращает время, которое программисты тратят на настройку файлов. Платформа Ruby on Rails имеет набор правил, облегчающих начинающим разработчикам Ruby on Rails использование платформы. Благодаря соглашениям код становится читабельным и лаконичным и позволяет легко перемещаться по веб-приложению Ruby on Rails.

Недостатки:

- **Документация.** В среде Ruby on Rails документацию иногда сложно найти. Эта проблема особенно актуальна при использовании драгоценных камней, так как немногие разработчики склонны документировать все.
- **Скорость выполнения.** Ruby и Rails работают быстро, но не так быстро, как подавляющее большинство других объектно-ориентированных языков программирования. Скорость выполнения часто называют главным аргументом против Ruby on Rails. По сравнению со скоростью исполнения Ruby on Rails против Node.JS отстает. Тем не менее, когда мы рассматриваем Java-фреймворк Spring, RoR побеждает в этой битве. Вы вряд ли будете иметь узкие места в производительности Ruby on Rails.
- **Скорость загрузки.** Разработчики часто называют скорость загрузки одним из разочаровывающих аспектов среды Ruby on Rails. В зависимости от количества используемых файлов и драгоценных камней запуск платформы может занять значительное время. Spring, предварительный загрузчик приложения Rails, решил эту проблему как-то; Тем не менее, еще предстоит проделать определенную работу, чтобы полностью решить проблему.
- **Хостинг сайтов.** Не все хосты сайта поддерживают Ruby on Rails. Фреймворк требует гораздо больше ресурсов, чем PHP, и низкоуровневые хостинг-провайдеры не могут обеспечить необходимые вычислительные мощности. Однако вы можете найти множество провайдеров хостинга веб-сайтов, которые поддерживают приложения Ruby on Rails.

23. *Doker. Цель технологии (сравнение с установочным скриптом, виртуальными машинами, облачными сервисами). Основные сведения. Термины и концепции. Файл Dockerfile*

Проблемы

Первая проблема — как передать продукт клиенту.

Предположим у вас есть серверный проект, который вы закончили и теперь его необходимо передать пользователю. Вы готовите много разных файлов, скриптов и пишете инструкцию по установке. А потом тратите уйму времени на решения проблем клиента вроде: «у меня ничего не работает», «ваш скрипт упал на середине — что теперь делать», «я перепутал порядок шагов в инструкции и теперь не могу идти дальше» и т. п.

Всё усугубляется если продукт тиражируемый и вместо одного клиента у вас сотни или тысячи покупателей. И становится еще сложнее, если вспомнить о необходимости установки новых версий продукта.

Вторая проблема — тиражируемость. Пусть вам нужно поднять 5 (или 50) почти одинаковых серверов. Делать это вручную долго, дорого и подвержено ошибкам.

Наконец, третья проблема — переиспользуемость. Предположим у вас есть отдел, который делает браузерные игры. Предположим, что их у вас уже несколько. И все они используют один и тот же технологический стек (например — java-tomcat-nginx-postgre). Но при этом, чтобы поставить новую игру вы вынуждены заново подготавливать на новом сервере почти одинаковую конфигурацию. Вы не можете просто так взять и сказать — «хочу сервер, как в игре странники но только с другим веб архивом»

Установочный скрипт

Первый подход я уже упомянул — вы можете написать скрипт, который установит всё, что вам нужно и запускать его на всех нужных серверах. (Скрипт может быть как простым sh файлом, так и чем-то сложным, созданным с использованием специальных инструментов).

Недостатки этого подхода — хрупкость и неустойчивость к ошибкам. Как бы хорошо не был написан скрипт, рано или поздно на какой-то машине он упадёт. И после этого падения машина фактически окажется «испорченной» — просто так «откатить» те действия, которые скрипт успел выполнить, у вашего клиента не получится.

Облачные сервисы

Второй подход — использование облачных сервисов. Вы вручную устанавливаете на виртуальный сервер всё, что вам нужно. Затем делаете его image. И далее клонируете его столько раз, сколько вам надо.

Недостатка здесь два.

- Во-первых, vendor-lock-in. Вы не можете запускать свое решение вне выбранного облака, что не всегда удобно и может привести к потерям несогласных с этим выбором клиентов.
- Во-вторых, облака медленны. Виртуальные (и даже «bare-metal») сервера предоставляемые облаками на сегодняшний день сильно уступают по производительности dedicated серверам.

Виртуальные машины

Третий подход — использование виртуальных машин. Здесь тоже есть недостатки:

- Размер — не всегда удобно качать образ виртуальной машины, который может быть довольно большим. При этом, любое изменение внутри образа виртуальной машины требует скачать весь образ заново.
- Сложное управление совместным использованием серверных ресурсов — не все виртуальные машины вообще поддерживают совместное использование памяти или CPU. Те что поддерживают, требуют тонкой настройки.

Подход докера — контейнеризация

И вот тут появляется docker, в котором

- есть контролируемая среда (как в виртуальных машинах)
- есть эффективное управление серверными ресурсами
- и нет vendor lock-in

Подобно виртуальной машине докер запускает свои процессы в собственной, заранее настроенной операционной системе. Но при этом все процессы докера работают на физическом host сервере деля все процессоры и всю доступную память со всеми другими процессами, запущенными в host системе. Подход, используемый докером находится посередине между запуском всего на физическом сервере и полной виртуализацией, предлагаемой виртуальными машинами. Этот подход называется контейнеризацией.

Docker — это платформа, которая предназначена для разработки, развёртывания и запуска приложений в контейнерах. Слово «Docker» в последнее время стало чем-то вроде синонима слова «контейнеризация».

Контейнер Docker обладает следующими характеристиками:

- В нём можно что-то хранить. Нечто может находиться либо в контейнере, либо за его пределами.
- Его можно переносить. Контейнер Docker можно использовать на локальном компьютере, на компьютере коллеги, на сервере поставщика облачных услуг (вроде AWS).
- В контейнер удобно что-то класть и удобно что-то из него вынимать. У контейнеров Docker есть нечто подобное, представляющее их интерфейс, то есть — механизмы, позволяющие им взаимодействовать с внешним миром.
- В случае с контейнерами Docker то, что можно сравнить с пресс-формой, а именно — образ контейнера, хранится в специальном репозитории. Если вам нужен некий контейнер, вы можете загрузить из репозитория соответствующий образ, и, используя его, этот контейнер создать.

Механизмы Docker:

- Платформа Docker — это программа, которая даёт возможность упаковывать приложения в контейнеры и запускать их на серверах. Платформа Docker позволяет помещать в контейнеры код и его зависимости.

- Движок Docker — это клиент-серверное приложение. Компания Docker разделила движок Docker на два продукта. Docker Community Edition (CE) — это бесплатное ПО, во многом основанное на open-сурсных инструментах.

- Клиент Docker — это основное средство, которое используют для взаимодействия с Docker. Так, при работе с интерфейсом командной строки Docker, в терминал вводят команды, начинающиеся с ключевого слова `docker`, обращаясь к клиенту. Затем клиент использует API Docker для отправки команд демону Docker.

- Демон Docker — сервер Docker, отвечающий за управление ключевыми механизмами системы.

- Тома Docker — это сервер Docker, который ожидает запросов к API Docker. Демон Docker управляет образами, контейнерами, сетями и томами.

- Реестр Docker представляет собой удалённую платформу, используемую для хранения образов Docker. В ходе работы с Docker образы отправляют в реестр и загружают из него. Подобный реестр может быть организован тем, кто пользуется Docker.

- Хаб Docker — это самый крупный реестр образов Docker. Кроме того, именно этот реестр используется при работе с Docker по умолчанию. Пользоваться хабом Docker можно бесплатно.

- Репозиторий. Репозиторием Docker (Docker Repository) называют набор образов Docker, обладающих одинаковыми именами и разными тегами. Теги — это идентификаторы образов. Обычно в репозиториях хранятся разные версии одних и тех же образов. Например, Python — это имя популярнейшего официального репозитория Docker на хабе Docker. А вот `Python:3.7-slim` — это версия образа с тегом `3.7-slim` в репозитории Python. В реестр можно отправить как целый репозиторий, так и отдельный образ.

Масштабирование:

- Сетевая подсистема Docker. Сетевые механизмы Docker (Docker Networking) позволяют организовывать связь между контейнерами Docker. Соединённые с помощью сети контейнеры могут выполняться на одном и том же хосте или на разных хостах.

- Docker Compose- Инструмент, который упрощает развёртывание приложений, для работы которых требуется несколько контейнеров Docker. Docker Compose позволяет выполнять команды, описываемые в файле `docker-compose.yml`.

- Docker Swarm — это решение, предназначенное для управления контейнерными развёртываниями (то есть, как говорят, для оркестрации контейнеров).

- Сервисы Docker — это различные части распределённого приложения. Сервисы Docker позволяют масштабировать контейнеры в пределах нескольких демонов Docker, благодаря им существует и технология Docker Swarm.

Файл Dockerfile

- **Образы Docker**

контейнер Docker — это образ Docker, вызванный к жизни. Это — самодостаточная операционная система, в которой имеется только самое необходимое и код приложения.

Образы Docker являются результатом процесса их сборки, а контейнеры Docker — это выполняющиеся образы. В самом сердце Docker находятся файлы Dockerfile. Подобные файлы сообщают Docker о том, как собирать образы, на основе которых создаются контейнеры.

Контейнеры состоят из слоёв. Каждый слой, кроме последнего, находящегося поверх всех остальных, предназначен только для чтения. Dockerfile сообщает системе Docker о том, какие слои и в каком порядке надо добавить в образ.

Базовый образ — это то, что является исходным слоем (или слоями) создаваемого образа. Базовый образ ещё называют родительским образом. Когда образ загружается из удалённого репозитория на локальный компьютер, то физически скачиваются лишь слои, которых на этом компьютере нет. Docker стремится экономить пространство и время путём повторного использования существующих слоёв.

- **Файлы Dockerfile**

В файлах Dockerfile содержатся инструкции по созданию образа. С них, набранных заглавными буквами, начинаются строки этого файла. После инструкций идут их аргументы. Инструкции, при сборке образа, обрабатываются сверху вниз. Вот как это выглядит:

```
FROM Ubuntu: 18.04
COPY ./app
```

Инструкции Dockerfile

- FROM — задаёт базовый (родительский) образ.
- LABEL — описывает метаданные. Например — сведения о том, кто создал и поддерживает образ.
- ENV — устанавливает постоянные переменные среды.
- RUN — выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.
- COPY — копирует в контейнер файлы и папки.
- ADD — копирует файлы и папки в контейнер, может распаковывать локальные .tar-файлы.
- CMD — описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать лишь одна инструкция CMD.
- WORKDIR — задаёт рабочую директорию для следующей инструкции.
- ARG — задаёт переменные для передачи Docker во время сборки образа.
- ENTRYPOINT — предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются.
- EXPOSE — указывает на необходимость открыть порт.
- VOLUME — создаёт точку монтирования для работы с постоянным хранилищем.

24. *Doker. Цель технологии (сравнение с установочным скриптом, виртуальными машинами, облачными сервисами). Основные сведения. Команды. Работа с данными.*

Команды

Общая схема команд для управления контейнерами выглядит так:

```
docker container my_command
```

Вот команды, которые могут быть подставлены туда, где мы использовали my_command:

- create — создание контейнера из образа.
- start — запуск существующего контейнера.
- run — создание контейнера и его запуск.
- ls — вывод списка работающих контейнеров.
- inspect — вывод подробной информации о контейнере.
- logs — вывод логов.
- stop — остановка работающего контейнера с отправкой главному процессу контейнера сигнала SIGTERM, и, через некоторое время, SIGKILL.
- kill — остановка работающего контейнера с отправкой главному процессу контейнера сигнала SIGKILL.
- rm — удаление остановленного контейнера.

Команды для управления образами

```
docker image my_command
```

Вот некоторые из команд этой группы:

- build — сборка образа.
- push — отправка образа в удалённый реестр.
- ls — вывод списка образов.
- history — вывод сведений о слоях образа.
- inspect — вывод подробной информации об образе, в том числе — сведений о слоях.
- rm — удаление образа.

Разные команды

- `docker version` — вывод сведений о версиях клиента и сервера Docker.
- `docker login` — вход в реестр Docker.
- `docker system prune` — удаление неиспользуемых контейнеров, сетей и образов, которым не назначено имя и тег.

Начало существования контейнера

На начальном этапе работы с контейнерами используются команды `create`, `start` и `run`. Они применяются, соответственно, для создания контейнера, для его запуска, и для его создания и запуска.

Вот команда для создания контейнера из образа:

```
docker container create my_repo/my_image:my_tag
```

Три важнейших команды:

- Создание и запуск контейнера: `docker container run my_image`
- Сборка образа: `docker image build -t my_repo/my_image:my_tag .`
- Отправка образа в удалённый репозиторий: `docker image push my_repo/my_image:my_tag`

Для того чтобы посмотреть справку по командам Docker, можете выполнить в терминале команду `docker`.

Работа с данными

- Временное хранение данные

В контейнерах Docker организовать работу с временными данными можно двумя способами.

По умолчанию файлы, создаваемые приложением, работающим в контейнере, сохраняются в слое контейнера, поддерживающем запись. Однако после того как контейнер перестанет существовать, исчезнут и данные, сохранённые таким вот нехитрым способом.

Для хранения временных файлов в Docker можно воспользоваться ещё одним решением, вы можете подключить к контейнеру `tmpfs` — временное хранилище информации, которое использует оперативную память хоста. Это позволит ускорить выполнение операций по записи и чтению данных.

Часто бывает так, что данные нужно хранить и после того, как контейнер прекратит существовать. Для этого нам пригодятся механизмы постоянного хранения данных.

- Постоянное хранение данных

Существуют два способа, позволяющих сделать срок жизни данных большим срока жизни кон-

тейнера. Один из способов заключается в использовании технологии bind mount. При таком подходе к контейнеру можно примонтировать, например, реально существующую папку. Работать с данными, хранящимися в такой папке, смогут и процессы, находящиеся за пределами Docker.

Минусы использования технологии bind mount заключаются в том, что её использование усложняет резервное копирование данных, миграцию данных, совместное использование данных несколькими контейнерами. Гораздо лучше для постоянного хранения данных использовать тома Docker.

Тома Docker

Том — это файловая система, которая расположена на хост-машине за пределами контейнеров. Созданием и управлением томами занимается Docker. Вот основные свойства томов Docker:

- Они представляют собой средства для постоянного хранения информации.
- Они самостоятельны и отделены от контейнеров.
- Ими могут совместно пользоваться разные контейнеры.
- Они позволяют организовать эффективное чтение и запись данных.
- Тома можно размещать на ресурсах удалённого облачного провайдера.
- Их можно шифровать.
- Им можно давать имена.
- Контейнер может организовать заблаговременное наполнение тома данными.
- Они удобны для тестирования.
-

Как видите, тома Docker обладают замечательными свойствами. Давайте поговорим о том, как их создавать.

Создание томов

Тома можно создавать средствами Docker или с помощью запросов к API.

Вот инструкция в Dockerfile, которая позволяет создать том при запуске контейнера.

```
VOLUME /my_volume
```

При использовании подобной инструкции Docker, после создания контейнера, создаст том, содержащий данные, которые уже имеются в указанном месте.

Кроме того, тома можно создавать средствами командной строки во время работы контейнера.

25. *Doker. Цель технологии (сравнение с установочным скриптом, виртуальными машинами, облачными сервисами). Kubernetes. Сравнение Kubernetes и Docker Swarm.*

Kubernetes

Kubernetes — это инструмент с открытым исходным кодом, который заботится о оркестровке контейнеров.

Он автоматизирует развертывание контейнера, непрерывное масштабирование и удаление, балансировку нагрузки контейнеров и т. д.

Google изначально разработал Kubernetes на языке программирования Go.

В настоящее время это лидер на рынке оркестрации контейнеров. Kubernetes — это идеальная платформа для размещения приложений микросервисов, которые динамически масштабируются.

Когда вы работаете в производственной среде, Kubernetes является наиболее предпочтительной и безопасной платформой для оркестрации контейнеров.

Поскольку в производственной среде вы ожидаете нулевого времени простоя, ваш кластер всегда должен быть в рабочем состоянии.

Почему Kubernetes

Ранее борьба между разработчиками и тестировщиками была обычным делом. Среды раньше были разными для всех; то, что работало в системе разработчика, не работало в системе тестеров. Теперь, когда большинство организаций используют контейнеры, проблемы, возникающие из-за различий в среде, больше не возникают. Но организация и запуск нескольких контейнеров также не простая задача. Когда вы работаете с динамическими приложениями, масштабируя их вверх / вниз, количество контейнеров является обычным делом. Выполнение таких задач вручную может быть сложным и рискованным. Следовательно, необходим инструмент оркестрации контейнеров, и именно поэтому нужен Kubernetes.

Особенности

Их много, но ниже приведены некоторые из лучших.

Автоматическая упаковка

- Kubernetes упаковывает ваше приложение и автоматически размещает контейнеры в соответствии с их требованиями и доступными ресурсами

Обнаружение службы и балансировка нагрузки

- Kubernetes автоматически устанавливает систему хранения по вашему выбору.

Оркестровка хранилищ

- Это может быть локальное хранилище или провайдер общедоступного облака, такой как AWS.

Самовосстановление

- Всякий раз, когда Kubernetes обнаруживает, что один из ваших контейнеров вышел из строя, он самостоятельно перезапускает этот контейнер и создает новый контейнер вместо сломанного.

Горизонтальное масштабирование

- Вы можете быстро масштабировать свои приложения с помощью простой команды. Эта команда может быть запущена на CLI или через панель управления Kubernetes. Также возможно автоматическое масштабирование, в зависимости от загрузки процессора, ваши контейнеры будут автоматически увеличиваться или уменьшаться.

Автоматический откат

- Всякий раз, когда происходит обновление вашего приложения, Kubernetes постепенно разворачивает эти изменения и обновления для приложения или его конфигураций. Не все экземпляры обновляются одновременно, что обеспечивает высокую доступность. Если что-то пойдет не так, то Kubernetes немедленно откатит то, что изменится случае сбоя вашего узла контейнеры, работающие на этом узле, будут запущены на другом работающем узле в кластере.

Концепции

Node Узел кластера Kubernetes

Cluster Состоит из группы узлов, выполняющих контейнерные приложения в Kubernetes.

Annotation Метка для хранения данных, необходимых для ресурсов.

Volume Это каталог данных, который содержит контейнеры в вопросе доступа к поду.

ReplicaSet Несколько копий запущенных копий.

Label Предоставление имени объектам Kubernetes, чтобы его можно было идентифицировать в системе.

Kubelet Это агент, который работает на каждом узле и проверяет, работают ли контейнеры в подах.

Kubectl Утилита командной строки для взаимодействия с сервером API Kubernetes.

Kube-proxy Сетевой прокси, который содержит все сетевые правила на каждом узле в кластере.

Архитектура Kubernetes

Работающий кластер Kubernetes включает в себя агента, запущенного на нодах (kubelet) и компоненты мастера (APIs, scheduler, etc), поверх решения с распределённым хранилищем. Приведённая схема показывает желаемое, в конечном итоге, состояние, хотя все ещё ведётся работа над некоторыми вещами, например: как сделать так, чтобы kubelet (все компоненты, на самом деле) самостоятельно запускался в контейнере, что сделает планировщик на 100% подключаемым.

Нода Kubernetes

При взгляде на архитектуру системы мы можем разбить его на сервисы, которые работают на каждой ноде и сервисы уровня управления кластера. На каждой ноде Kubernetes запускаются сервисы, необходимые для управления нодой со стороны мастера и для запуска приложений. Конечно, на каждой ноде запускается Docker. Docker обеспечивает загрузку образов и запуск контейнеров.

Kubelet

Kubelet управляет pod'ами их контейнерами, образами, разделами, etc.

Kube-Proxy

Также на каждой ноде запускается простой проху-балансировщик. Этот сервис запускается на каждой ноде и настраивается в Kubernetes API. Kube-Proxy может выполнять простейшее перенаправление потоков TCP и UDP (round robin) между набором бэкендов.

Компоненты управления Kubernetes

Система управления Kubernetes разделена на несколько компонентов. В данный момент все они запускаются на мастер-ноде, но в скором времени это будет изменено для возможности создания отказоустойчивого кластера. Эти компоненты работают вместе, чтобы обеспечить единое представление кластера.

etcd

Состояние мастера хранится в экземпляре etcd. Это обеспечивает надёжное хранение конфигурационных данных и своевременное оповещение прочих компонентов об изменении состояния.

Kubernetes API Server

Kubernetes API обеспечивает работу api-сервера. Он предназначен для того, чтобы быть CRUD сервером со встроенной бизнес-логикой, реализованной в отдельных компонентах или в плагинах. Он, в основном, обрабатывает REST операции, проверяя их и обновляя соответствующие объекты в etcd (и событийно в других хранилищах).

Scheduler

Scheduler привязывает незапущенные pod'ы к нодам через вызов /binding API. Scheduler подключаем; планируется поддержка множественных scheduler'ов и пользовательских scheduler'ов.

Kubernetes Controller Manager Server

Все остальные функции уровня кластера представлены в Controller Manager. Например, ноды обнаруживаются, управляются и контролируются средствами node controller. Эта сущность в итоге может быть разделена на отдельные компоненты, чтобы сделать их независимо подключаемыми.

ReplicationController — это механизм, основывающийся на pod API. В конечном счете планируется перевести её на общий механизм plug-in, когда он будет реализован.

Kubernetes vs. Docker Swarm

Kubernetes	Docker Swarm
Установка сложна	Установка проста
Большое комьюнити	Маленькое комьюнити
Может легко организовать сотни контейнеров	Хорош для оркестровки 10-20 контейнеров
Kubernetes Dashboard предоставляет графический интерфейс	Нет графического интерфейса (Можно конечно доставить Portainer)
Поддерживает автоматическое масштабирование	Не поддерживает автоматическое масштабирование
Поддерживает непрерывные обновления с автоматическим откатом	Поддерживает непрерывные обновления, но не автоматический откат
Ведение журналов и мониторинг доступны и встроенные	Требуется сторонние приложения, такое как ELK для ведения журнала и мониторинга
Можно разделить хранилище контейнерам в одном поде	Может разделить хранилище любым контейнерам в кластере

ORM (англ. *object-relational mapping*, рус. *объектно-реляционное отображение*)

Технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

В объектно-ориентированном программировании объекты в программе представляют объекты из реального мира. В качестве примера можно рассмотреть адресную книгу, которая содержит список людей с нулем или более телефонов и нулем или более адресов.

Суть задачи состоит в преобразовании таких объектов в форму, в которой они могут быть сохранены в файлах или базах данных, и которые легко могут быть извлечены в последующем, с сохранением свойств объектов и отношений между ними.

Парадигма «несоответствия»

Использование ORM решает проблему так называемой парадигмы «несоответствия», которая гласит о том, что объектные и реляционные модели не очень хорошо работают вместе.

Основные проблемы:

- реляционная модель может быть намного детальнее, чем объектная, т.е. для хранения одного объекта в реляционной базе данных используется несколько таблиц;
- реляционные СУБД не имеют ничего похожего на наследование — естественную парадигму объектно-ориентированных языков программирования;
- в СУБД определен только один параметр для сравнения записей — первичный ключ. В то время как ООП предоставляет как проверку идентичности объектов ($a=b$), так и их равенства ($a.equals(b)$);
- для связи объектов СУБД использует понятие внешних ключей, в объектно-ориентированных языках связь между объектами может быть только однонаправленной. Если же нужно организовать двунаправленные отношения, то придется определить две однонаправленные ассоциации. Кроме того, нет возможности определить кратность отношения, глядя на модель предметной области;
- принцип доступа к данным в ООП кардинально отличается от доступа к данным в БД. Для доступа к данным в ООП используются последовательные переходы от родительского объекта к свойствам дочерних элементов и инициализации объектов по необходимости. Такой подход считается не эффективным способом извлечения данных из реляционных баз данных. Как правило, количество запросов к БД должно быть сведено к минимуму, необходимые сущности должны по возможности загружаться сразу с использованием JOIN-ов.

Решения:

Разработано множество пакетов, устраняющих необходимость в преобразовании объектов для хранения в реляционных базах данных.

Некоторые пакеты решают эту проблему, предоставляя библиотеки классов, способных выполнять такие преобразования автоматически. Имея список таблиц в базе данных и объектов в программе, они автоматически преобразуют запросы из одного вида в другой. В результате запроса объекта «человек» (из примера с адресной книгой) необходимый SQL-запрос будет сформирован и выполнен, а результаты «волшебным» образом преобразованы в объекты «номер телефона» внутри программы.

С точки зрения программиста система должна выглядеть как постоянное хранилище объектов. Он может просто создавать объекты и работать с ними как обычно, а они автоматически будут сохраняться в реляционной базе данных.

Принцип работы ORM

Ключевой особенностью ORM является отображение, которое используется для привязки объекта к его данным в БД. ORM как бы создает «виртуальную» схему базы данных в памяти и позволяет манипулировать данными уже на уровне объектов. Отображение показывает, как объект и его свойства связаны с одной или несколькими таблицами и их полями в базе данных. ORM использует информацию этого отображения для управления процессом преобразования данных между базой и формами объектов, а также для создания SQL-запросов для вставки, обновления и удаления данных в ответ на изменения, которые приложение вносит в эти объекты.

Преимущества и недостатки использования ORM

Достоинства

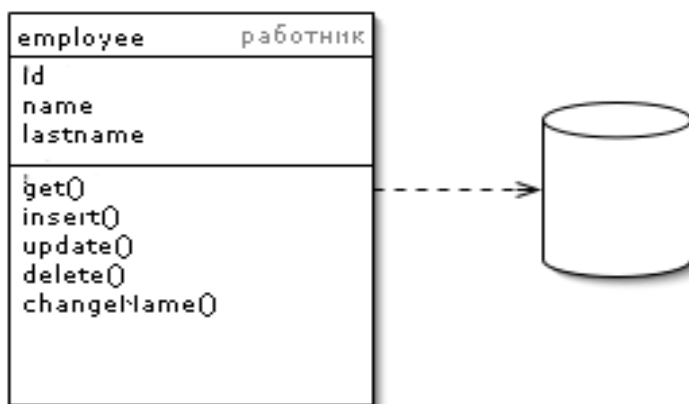
Использование ORM в проекте избавляет разработчика от необходимости работы с SQL и написания большого количества кода, часто однообразного и подверженного ошибкам. Весь генерируемый ORM код предположительно хорошо проверен и оптимизирован, поэтому не нужно в целом задумываться о его тестировании.

Недостатки:

Потеря производительности. Это происходит потому, что большинство ORM предназначены для обработки широкого спектра сценариев использования данных, гораздо большего, чем любое отдельное приложение когда-либо сможет использовать.

Варианты реализации ORM

Active Record (Активная запись)



Один объект управляет и данными, и поведением. Большинство этих данных постоянны и их надо хранить в БД. Этот паттерн использует наиболее очевидный подход - хранение логики доступа к данным в объекте сущности.

Объект является "обёрткой" одной строки из БД или представления, включает в себя доступ к БД и логику обращения с данными.

Пример: объект "Работник" содержит данные об одном работнике и методы: добавить, обновить или удалить. Помимо прочего, отдельным методом вынесена *смена имени*.

Примеры

- Examples of AR:
- Ruby on Rails
- Laravel's Eloquent
- Propel (Symfony)
- Yii Active Record
- Django's ORM

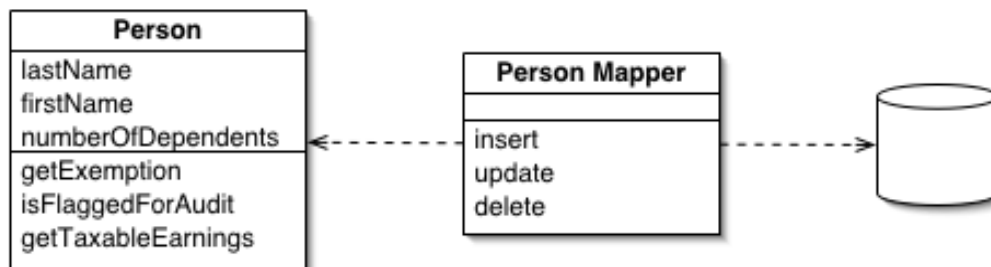
Преимущества Active Record

- Писать код с Active Record получается быстро и легко, в том случае, когда свойства объекта прямо соотносятся с колонками в базе данных.
- Сохранение происходит в одном месте, что позволяет легко изучить, как это работает.

Недостатки Active Record

- Модели Active Record нарушают принципы SOLID. В частности, принцип единой ответственности (SRP — «S» в принципах SOLID). Согласно принципу, доменный объект должен иметь только одну зону ответственности, то есть только свою бизнес-логику. Вызывая его для сохранения данных, вы добавляете ему дополнительную зону ответственности, увеличивая сложность объекта, что усложняет его поддержку и тестирование.
- Реализации сохранения данных тесно связаны с бизнес-логикой, а это означает, что если вы позже захотите использовать другую абстракцию для сохранения данных (например для хранения данных в XML-файле, а не в базе данных), то вам придется делать рефакторинг кода.

Data Mapper



Data Mapper — это программная прослойка, разделяющая объект и БД. Его обязанность — пересылать данные между ними и изолировать их друг от друга. При использовании Data Mapper'a

объекты не нуждаются в знании о существовании БД. Они не нуждаются в SQL-коде, и (естественно) в информации о структуре БД. Так как Data Mapper - это разновидность паттерна [Mapper](#), сам объект-Mapper неизвестен объекту.

Примеры

- Examples of Data Mapper:
- Java Hibernate
- Doctrine2
- SQLAlchemy in Python
- EntityFramework for Microsoft .NET
- Golang Prisma
- Ecto Elixir

Преимущества Data Mapper-а

•Каждый объект имеет свою зону ответственности, тем самым следуя принципам SOLID и сохраняя каждый объект простым и по существу.

•Бизнес- логика и сохранение данных связаны слабо, и если вы хотите сохранять данные в XML-файл или какой-нибудь другой формат, вы можете просто написать новый Mapper, не при-трагиваясь к доменному объекту.

Недостатки, Data Mapper-а

- Вам придется гораздо больше думать, перед тем как написать код.
- В итоге у вас больше объектов в управлении, что немного усложняет код и его отладку.

Что выбрать:

Объекты Active Record исторически были очень популярны из-за того, что они проще, легче в понимании и быстрее в написании, поэтому многие фреймворки и ORM используют Active Record по умолчанию.

Если вы уверены, что вам никогда не понадобится менять слой сохранения данных (если вы имеете дело с объектом, который представляет из себя INI-файл, например), или вы имеете дело с очень простыми объектами, в которых не так много бизнес-логики, или просто предпочитаете держать все в небольшом количестве классов, тогда шаблон Active Record это то, что вам нужно.

Использование Data Mapper-а хотя и ведет к более чистому, простому в тестировании и поддержке коду, и обеспечивает большую гибкость, — цена этому, — повышение сложности.

27. *Entity Framework. Подходы для организации взаимодействия EF с базой данных. Архитектура EF.*

Entity Framework (EF) — это среда ORM с открытым исходным кодом для ADO.NET, которая является частью .NET Framework.

Entity Framework является продолжением технологии Microsoft ActiveX Data и предоставляет возможность работы с базами данных через объектно-ориентированный код C#. Этот подход предоставляет ряд существенных преимуществ: вам не нужно беспокоиться о коде доступа к данным, вам не нужно знать деталей работы СУБД SQL Server и синтаксиса языка запросов T-SQL, вместо этого вы работаете с таблицами базы данных как с классами C#, с полями этих таблиц - как со свойствами классов, а синтаксис SQL-запросов, который в ADO.NET раньше нужно было вставлять в код C# в виде команд, заменен на более удобный подход с LINQ. Entity Framework берет на себя обязанности по преобразованию кода C# в SQL-инструкции.

Entity Framework был впервые выпущен в 2008 году, основным средством взаимодействия Microsoft между приложениями .NET и реляционными базами данных. Entity Framework — это Object Relational Mapper (ORM), который представляет собой тип инструмента, который упрощает сопоставление между объектами в вашем программном обеспечении и таблицами и столбцами реляционной базы данных.

- Entity Framework (EF) — это среда ORM с открытым исходным кодом для ADO.NET, которая является частью .NET Framework.
- ORM заботится о создании соединений с базой данных и выполнении команд, а также о получении результатов запроса и автоматической материализации этих результатов как объектов вашего приложения.
- ORM также помогает отслеживать изменения в этих объектах, и при получении инструкции он также сохраняет эти изменения обратно в базу данных для вас.

Entity Framework — это ORM, и ORM направлены на повышение производительности труда разработчика за счет сокращения избыточной задачи сохранения данных, используемых в приложениях.

- Entity Framework может генерировать необходимые команды базы данных для чтения или записи данных в базу данных и выполнять их для вас.
- Если вы запрашиваете, вы можете выразить свои запросы к объектам вашего домена, используя LINQ для сущностей.
- Entity Framework выполнит соответствующий запрос в базе данных, а затем материализует результаты в экземпляры ваших доменных объектов для работы в вашем приложении.

Подходы для организации взаимодействия EF с базой данных

Начиная с версии Entity Framework 4.1 вам предоставляется три подхода по проектированию базы данных, из которых вы можете выбрать для себя подходящий:

- Database-First

- Model-First
- Code-First

Подход Code-First, который впервые появился в Entity Framework 4.1, обычно используется, когда у вас есть уже существующее приложение, содержащее модель данных. Эта модель, как правило, описывается с помощью нескольких классов и кода взаимодействия между этими классами. Например, вы можете создать класс по имени Customer, который будет содержать данные покупателя в интернет-магазине:

```
public class Customer
{
    // Определить поля, которые используются в базе данных
    public int CustomerId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string City { get; set; }
    public int Age { get; set; }
    public DateTime BirthDate { get; set; }
}
```

Вы можете использовать этот класс модели в своем приложении без создания базы данных. Добавив дополнительный код, вы могли бы хранить объекты этого класса, например, на диске в формате XML или в памяти рабочего процесса программы. Однако когда ваше приложение расширяется и становится крупным, эти данные необходимо будет хранить в базе данных. Эта та точка, в которой и начинает работать подход Code-First – вы можете использовать существующий код для создания базы данных не беспокоясь о деталях реализации базы данных (этим займется Entity Framework), а вы можете сфокусировать свое внимание на коде.

При проектировании приложений с подходом Code-First, вы сначала создаете классы модели данных не обращая никакого внимания на Entity Framework. После того, как вам понадобилось работать с базой данных, вы используете различные инструменты, которые проецируют структуру базы данных из созданной модели классов. После этого вы можете вернуться к этой модели в коде и, например, изменить ее. Эти изменения затем можно будет отразить в базе данных используя все те же инструменты.

Подход Model-First, впервые появившийся в версии Entity Framework 4, применяется разработчиками, которые не хотят использовать инструменты СУБД для создания и управления базами данных, а также не хотят вручную настраивать классы модели EDM. Фактически это самый простой подход при работе с Entity Framework. Проектирование модели происходит в графическом дизайнера EDM среды Visual Studio.

Рабочий процесс создания модели при подходе Model-First начинается в тот момент, когда вы проектируете базу данных. При этом вам необходимы минимальные знания устройства баз данных, например, для настройки отношений между таблицами

Как и в случае подхода Code-First, вся работа строится вокруг класса контекста базы данных. Фактически, взаимодействие с базой данных в этих подходах одинаковое. Например, для вставки объекта, используется следующая последовательность действий:

- Создать объект модели и заполнить его данными.

- Создать класс контекста, унаследованный от DbContext (в подходе Code-First это делается вручную, в Model-First этот класс генерируется автоматически вместе с сущностными классами).
- Добавить объект в базу данных, используя класс контекста.
- Сохранить изменения.

Подход Database-First, появившийся вместе с Entity Framework, позволяет писать приложения для существующих баз данных. Базы данных в реальных приложениях довольно быстро становятся сложными и пытаться создать модель для существующей базы данных, которую могут понять разработчики, довольно трудно. Еще тяжелее написать код использования модели, в котором происходит взаимодействие с базой данных. Во многих отношениях, подход Database-First является противоположностью подходу Model-First. При подходе Database-First база данных уже существует, поэтому разработчик должен знать, где расположена база данных, а также иметь информацию об имени базы данных.

При этом подходе, рабочий процесс создания модели начинается с создания и проектирования базы данных. После генерации сущностных классов модели из существующей базы данных, работа с Entity Framework аналогична подходам Code-First и Model-First. Это означает создание объекта класса контекста и использование этого объекта для выполнения необходимых задач.

Архитектура EF

Архитектура Entity Framework, снизу-вверх, состоит из следующего:

Поставщики данных

- Это поставщики, зависящие от источника, которые абстрагируют интерфейсы ADO.NET для подключения к базе данных при программировании на основе концептуальной схемы.
- Он переводит распространенные языки SQL, такие как LINQ, через дерево команд в собственное выражение SQL и выполняет его для конкретной системы СУБД.

Entity Client

- Этот уровень выставляет уровень сущности на верхний уровень. Сущностный клиент предоставляет разработчикам возможность работать с сущностями в форме строк и столбцов, используя запросы сущностных SQL, без необходимости создавать классы для представления концептуальной схемы. Entity Client показывает уровни структуры сущности, которые являются основными функциями. Эти слои называются Entity Data Model.
- Уровень **хранения** содержит всю схему базы данных в формате XML.
- **Слой сущностей**, который также является файлом XML, определяет сущности и отношения.

- Уровень **отображения** — это XML-файл, который отображает сущности и отношения, определенные на концептуальном уровне, с фактическими отношениями и таблицами, определенными на логическом уровне.
- **Службы метаданных**, которые также представлены в Entity Client, предоставляют централизованный API для доступа к слоям Entity, Mapping и Storage, которые хранятся в метаданных.

Слой Object Services — это Object Context, который представляет сеанс взаимодействия между приложениями и источником данных.

- Основное использование контекста объекта заключается в выполнении различных операций, таких как добавление, удаление экземпляров объектов и сохранение измененного состояния обратно в базу данных с помощью запросов.
- Это уровень ORM Entity Framework, который представляет результат данных для экземпляров объектов.
- Эти сервисы позволяют разработчику использовать некоторые богатые функции ORM, такие как сопоставление первичных ключей, отслеживание изменений и т. Д., Путем написания запросов с использованием LINQ и Entity SQL.

28. *ADO.NET. Основные отличительные особенности ADO.NET. Уровни в объектной модели ADO.NET.*

В платформе .NET определено множество типов (организованных в соответствующие пространства имен) для взаимодействия с локальными и удаленными хранилищами данных. Общее название пространств имен с этими типами - *ADO.NET*(ActiveX Data Objects).

***ADO.NET* - это технология доступа к базам данных, специально оптимизированная для нужд построения рассоединенных (*disconnected*) систем на платформе .NET.**

Технология *ADO.NET* ориентирована на приложения *N-tier* - архитектуру многоуровневых приложений, которая в настоящее время стала фактическим стандартом для создания распределенных систем.

Отличительные особенности ADO.NET:

- *ADO* расширяет концепцию объектов-наборов записей в базе данных новым типом *DataSet*, который представляет локальную копию сразу множества взаимосвязанных таблиц. При помощи объекта *DataSet* пользователь может локально производить различные операции с содержимым базы данных, будучи физически рассоединен с СУБД, и после завершения этих операций передавать внесенные изменения в базу данных при помощи соответствующего "адаптера данных" (*data adapter*);
- в *ADO.NET* реализована полная поддержка представления данных в *XML* -совместимых форматах. В *ADO.NET* сформированные для локальной обработки наборы данных представлены в формате *XML* (в этом же формате они и передаются с сервера баз данных). Данные в форматах *XML* очень удобно передавать при помощи обычного *HTTP*, решает многие проблемы с установлением соединений через брандмауэры;
- *ADO.NET* - это библиотека управляемого кода и взаимодействие с ней производится как с обычной сборкой .NET. Типы *ADO.NET* используют возможности управления памятью *CLR* и могут использоваться во многих .NET - совместимых языках. При этом обращение к типам *ADO.NET* (и их членам) производится практически одинаково вне зависимости от того, какой язык используется.

Уровни в объектной модели ADO.NET.

- *Уровень данных.* Это по сути дела базовый уровень, на котором располагаются сами данные (например, таблицы базы данных *MS SQL Server*). На данном уровне обеспечивается физическое хранение информации на магнитных носителях и манипуляция с данными на уровне исходных таблиц (*выборка, сортировка, добавление, удаление, обновление* и т. п.).
- *Уровень бизнес-логики.* Это набор объектов, определяющих, с какой базой данных предстоит установить *связь* и какие действия необходимо будет выполнить с содержащейся в ней информацией. Для установления связи с базами данных используется *объект* *DataConnection*. Для хранения команд, выполняющих какие либо действия над данными, используется *объект* *DataAdapter*. И, наконец, если выполнялся процесс выборки информации из базы данных, для хранения результатов выборки используется *объект* *DataSet*. *Объект* *DataSet* представляет собой набор данных "вырезанных" из таблиц

основного хранилища, который может быть передан любой программе-клиенту, способной либо отобразить эту информацию конечному пользователю, либо выполнить какие-либо манипуляции с полученными данными.

- *Уровень приложения.* Это набор объектов, позволяющих хранить и отображать данные на компьютере конечного пользователя. Для хранения информации используется уже знакомый нам *объект* DataSet, а для отображения данных имеется довольно большой набор элементов управления (DataGrid, TextBox, ComboBox, Label и т. д.).
- *Обмен данными* между приложениями и уровнем бизнес-логики происходит с использованием формата *XML*, а средой передачи данных служат либо локальная *сеть (Интранет)*, либо глобальная *сеть (Интернет)*.

29. Основы языка JavaScript. Особенности синтаксиса JavaScript. Отличие от языков C# и Java.

JavaScript — это относительно простой объектно-ориентированный язык, предназначенный для создания небольших клиентских и серверных приложений для Internet. Программы, написанные на языке JavaScript, включаются в состав HTML-документов и распространяются вместе с ними.

Язык JavaScript, будучи схожим по синтаксису с языком Java, за исключением объектной модели, в то же время не обладает такими свойствами, как статические типы данных и строгой типизацией. В JavaScript, в отличие от Java, понятие классов не является основой синтаксических конструкций языка.

Такой основой является небольшой набор предопределенных типов данных, поддерживаемых исполняемой системой: числовые, булевские и строковые; функции, которые могут быть как самостоятельными, так и методами объектов.

Структурно JavaScript состоит из трех частей:

- **ядро** (ECMAScript),

Ядром JavaScript является спецификация *ECMAScript*, описывающая типы данных, инструкции, ключевые и зарезервированные слова, операторы, объекты, регулярные выражения.

- **объектная модель браузера** (Browser Object Model или BOM)

Каждое из окон браузера представляется объектом **window**. Браузеры управляют окнами, фреймами, адресом открытой страницы, поддерживают работу с cookie

- **объектная модель документа** (Document Object Model или DOM).

Объектная модель документа - *интерфейс программирования приложений* для HTML и XML-документов. Согласно DOM документу можно поставить в соответствие дерево объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции.

Области применения языка программирования JavaScript

AJAX

- JavaScript используется в AJAX, когда web-страница не перезагружается полностью при обновлении данных и интерфейс web-приложения становится быстрее, чем при традиционном подходе. В качестве примера можно привести Gmail, некоторые картографические сервисы.

Comet

- Comet - механизм работы веб-приложений, использующих постоянные HTTP-соединения, когда web-сервер отправляет данные браузеру без дополнительного запроса со стороны браузера.

Браузерные операционные системы

- Исходный код IndraDesktop WebOS на 73 % состоит из JavaScript, код браузерной операционной системы IntOS написан на JavaScript на 70 %. Доля JavaScript в исходном коде eyeOS - 21 %.

Букмарклеты

- JavaScript используется для создания небольших программ, размещаемых в закладки браузера. При этом используются URL-адреса со спецификатором javascript.

Пользовательские скрипты в браузере

- Пользовательские скрипты в браузере - это программы, позволяющие автоматически заполнять формы, переформатировать страницы, скрывать нежелательное содержимое и встраивать желательное для отображения содержимое, изменять поведение клиентской части веб-приложений, добавлять элементы управления на страницу и т. д.

Серверные приложения

- JavaScript на стороне сервера используется в проектах Google.

Мобильные приложения

- JavaScript использовался при разработке операционной системы Palm webOS.

Виджеты

- Виджет - вспомогательная программа, использующаяся для украшения рабочего стола или быстрого получения информации из интернета без помощи web-браузера. При помощи JavaScript созданы Apple Dashboard, Microsoft Gadgets, Yahoo!_Widgets, Google Desktop Gadgets и Klipfolio Dashboard.

Прикладное программное обеспечение

- 57 % исходного кода Mozilla Firefox написано на JavaScript.

Возможности языка JavaScript

С его помощью можно динамически управлять отображением и содержимым HTML-документов. Можно записывать в отображаемый на экран документ произвольный HTML-код в процессе синтаксического анализа загруженного браузером документа.

JavaScript позволяет взаимодействовать с содержимым документов.

JavaScript обеспечивает взаимодействие с пользователем. Важной особенностью этого языка является реализованная в нем возможность определять обработчики событий — произвольные порции кода, которые выполняются при наступлении конкретных событий (обычно действий пользователя).

JavaScript	Java
Исходный код программ встраивается непосредственно в HTML-документ либо загружается из независимых файлов.	Исходный код программ не распространяется с приложением -апплетом. Апплеты загружаются с сервера из независимых файлов.

Программа выкладывается на сервер в виде исходного кода в текстовой форме и в дальнейшем интерпретируется (без предварительной компиляции) браузером после загрузки ее с сервера.	Программа компилируется в машинно-независимый байтовый код Java-код, после чего выкладывается на сервер. Браузер (виртуальная Java-машина) исполняет Java-код.
Объектный. Можно программировать как без использования объектного программирования, так и используя предопределенные встроенные классы. Имеется теоретическая возможность расширения этих классов, но реально она никогда не используется.	Объектно-ориентированный. Программировать без использования объектного программирования нельзя. Апплеты состоят из классов с возможностью иерархического наследования по традиционной схеме наследования. Использование наследования и полиморфизма – основа программирования в Java.
В отличие от подавляющего большинства языков объектного программирования в JavaScript структура объектов не задается однозначно устройством класса, а является динамической и может меняться на этапе выполнения программы. Объекты могут динамически получать новые поля и методы или изменять любые параметры старых.	Структура объектов полностью задается на этапе компиляции их классов.
Свободная типизация: элементарные типы данных переменных не описываются, при присваивании тип левой части всегда определяется по результату присваивания (т.е. по правой части)	Строгая типизация: типы данных любых переменных должны быть описаны перед использованием, тип левой части должен совпадать с типом правой (за редкими исключениями, когда работает автоматическое приведение типа результата к типу левой части)
Динамическое связывание кода с объектами: ссылки на объекты проверяются во время выполнения программы.	Статическое связывание кода с объектами: ссылки на объекты должны существовать на момент компиляции

Отличие от языков C# и Java.

- **Windows vs open-source.** Хотя существуют реализации с открытым исходным кодом, C# в основном используется в разработке для платформ Microsoft – .NET Framework CLR и является наиболее широко используемой реализацией CLI. На другом конце спектра Java имеет огромную экосистему с открытым исходным кодом и у него открылось второе дыхание отчасти благодаря тому, что Google использует JVM для Android.
- **Поддержка обобщений (Generics):** Generics улучшает проверку типов с помощью компилятора, в основном удаляя приведения из исходного кода. В Java средства обобщений реализуются с использованием стираний. Параметры общего типа «стираются», а при компиляции в байт-код добавляются приведения. C# также использует обобщения, интегрируя его в CLI и предоставляя информацию о типе во время выполнения, что дает небольшое увеличение производительности.

- **Поддержка делегатов (указателей):** В C# есть делегаты, которые по существу служат в качестве методов, которые могут быть вызваны без знания целевого объекта. Для достижения такой же функциональности в Java вам необходимо использовать интерфейс с одним методом или другим способом обхода, который может потребовать нетривиального количества дополнительного кода, в зависимости от приложения.
- **Проверяемые исключения:** Java различает два типа исключений – проверяемые и непроверяемые. C# выбрал более минималистский подход, имея только один тип исключения. Хотя способность ловить исключения может быть полезна, она также может отрицательно влиять на масштабируемость и контроль версий.
- **Полиморфизм:** C# и Java используют очень разные подходы к полиморфизму. Java допускает полиморфизм по умолчанию, C# же должен вызывать ключевое слово «virtual» в базовом классе и ключевое слово «override» в производном классе.
- **Перечисления (Enums):** в C# перечисления представляют собой простые списки именованных констант, где базовый тип должен быть целым. Java представляет перечисления более глубоко, рассматривая его как именованный экземпляр типа, что упрощает добавление пользовательского поведения к отдельным перечислениям.

Angular

Angular хорош для создания single-page apps (SPA). Браузер не перезагружает всю страницу, а только ее фрагменты, по мере того как пользователь взаимодействует с приложением (жмет на кнопки, ссылки и т. д.). У Ангуляра есть Router для навигации внутри приложения. Внутри темплейта компонента с помощью тега `<router-outlet>` можно выделить одну или больше областей для отображения разных фрагментов страницы.

Модуляризация приложения

- Приложение можно и нужно разбивать на модули. У приложения должен быть один *корневой модуль* с минимальной функциональностью, а фичи можно имплементировать в отдельных модулях, например, Billing, Shipping и т. д.
- Более того, эти модули могут подгружаться «лениво», чтобы минимизировать размер базовой страницы приложения.

Dependency injection

- Компоненты — это классы, у которых есть UI, а сервисы — это классы, содержащие аппликационную логику. Вы пишете такой класс, например, ProductService, а Ангуляр создает его экземпляр и вставляет (injects) его в ProductComponent. Для этого нужно просто указать сервис аргументом конструктора компонента и не нужно создавать экземпляр класса оператором `new:@`

В последнее время стали популярными реактивные (RX) библиотеки, которые построены на работе с observable потоками данных, которые асинхронно генерируются каким-либо источником, например событиями мышки, сенсорами, сокетами и т. д. Такие библиотеки включают большой набор операторов, которые легко собирать в цепочку для манипуляции данными по дороге от источника данных до подписчика. Ангуляр поставляется с библиотекой RxJS и вы можете либо подписываться на готовые observable streams предоставляемые разными APIs, либо создавать такие потоки сами.

Многие веб-приложения используют популярную библиотеку Bootstrap для стилизации и разметки (layout) страниц. Вы можете пользоваться этой библиотекой и в приложениях на Ангуляре. Bootstrap поддерживает responsive web design, чтобы расположение UI-компонентов автоматически адаптировалось к ширине экрана устройства пользователя.

Есть и другие библиотеки, сделанные специально для Ангуляра. Angular Material — это библиотека, включающая 30 современных компонентов, сделанных согласно спецификации Google Material Design. Если вам нужно больше компонентов, используйте одну из сторонних библиотек. Так, например, PrimeNG включает 75 UI компонентов, сделанных специально для Ангуляра.

Достоинства

1. Большое комьюнити
2. Декларативный стиль кода

3. Использование директив
4. Высокая скорость разработки
5. MVC из коробки
6. Полезные фичи для SPA
7. Модульность
8. Наличие готовых решений
9. Двустороннее связывание данных
10. Простота тестирования

Недостатки

1. **Сложность освоения.** Трудности обычно возникают у тех, кто раньше использовал библиотеку jQuery, ведь в отличие от Angular.js она полагается на выполнение манипуляций с деревом DOM.
2. **Замедление работы при использовании более 2000 вотчеров** (или слушателей событий).
3. **Отсутствие обратной совместимости со второй версией.** Разумеется, вы можете начать подготовку своего кода к миграции уже сейчас, но гарантий, что она пройдет гладко, нет.

React

React (иногда React.js или ReactJS) — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

React был создан Джорданом Валке, разработчиком программного обеспечения из Facebook. На него оказал влияние XHP — компонентный HTML-фреймворк для PHP. Впервые React использовался в новостной ленте Facebook в 2011 году и позже в ленте Instagram в 2012 году. Исходный код React открыт в мае 2013 года на конференции «JSConf US».

Особенности

Однонаправленная передача данных

- Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых (англ. immutable) значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через callback-функции. Такой механизм называют «свойства вниз, события вверх».

Виртуальный DOM

- React использует виртуальный DOM (англ. virtual DOM). React создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера. Таким образом программист

может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

Методы жизненного цикла

- Методы жизненного цикла позволяют разработчику запускать код на разных стадиях жизненного цикла компонента. Например:
- `shouldComponentUpdate` — позволяет предотвратить перерисовку компонента с помощью возврата `false`, если перерисовка не нужна.
- `componentDidMount` — вызывается после первой отрисовки компонента. Часто используется для запуска получения данных с удаленного источника через API.
- `render` — важнейший метод жизненного цикла. Каждый компонент должен иметь этот метод. Обычно вызывается при изменении данных компонента для перерисовки данных в интерфейсе.

Не только отрисовка HTML в браузере

- React используется не только для отрисовки HTML в браузере. Например, Facebook имеет динамические графики, которые отрисовываются в теги `<canvas>`. Netflix и PayPal используют изоморфные загрузки для отрисовки идентичного HTML на сервере и клиенте.

Одной из сильнейших сторон React является тот факт, что эта библиотека не принуждает разработчика к использованию классов. В Angular же все компоненты должны быть реализованы в виде классов. Это приводит к чрезмерному усложнению кода, не давая никаких преимуществ.

В React все компоненты пользовательского интерфейса могут быть выражены в виде наборов чистых функций. Использование чистых функций для формирования UI можно сравнить с глотком чистого воздуха.

Vue

Vue (произносится как `view` - вью) - это прогрессивный фреймворк для создания пользовательских интерфейсов. Vue позволяет строить приложения с применением архитектурного паттерна MVVM (Model-View-ViewModel).

Особенность `Vue.js` в том, что его можно начинать с легкостью использовать в уже существующем приложении, внедряя постепенно и используя вместе с другими JavaScript библиотеками. Это очень отличная особенность, когда нужно переписать приложение постепенно, а не все сразу.

Создатель фреймворка `Vue.js` - Эван Ю (Evan You), бывший сотрудник Google. Его фреймворк используют множество компаний, например, Alibaba, Baidu & Tencent, Xiaomi. Разработка фреймворка ведется Эваном вместе с сообществом разработчиков вне всяких компаний.

Достоинства

- достаточно быстрая разработка
- небольшой вес фреймворка
- хороша документация

- большое сообщество разработчиков
- быстрое взаимодействие с виртуальным DOM
- подключение плагинов и создание своих плагинов
- компонентная разработка приложения
- реализует современные подходы к разработке
- поддержка серверного рендинга приложения с помощью Nuxt.js

Основными концепциями Vue являются:

- реактивность

В этом фреймворке используются идеи - реактивного программирования. Реактивное программирование - это когда разработка основана на потоках статических и динамических данных, а также распространении изменений благодаря потоку этих данных.

- компоненты

Компоненты - это разделение кода, его инкапсулирование, на независимые части позволяющие повторно использовать код.

Vue.js позволяет разделять весь код приложения на компоненты и даже асинхронно загружать необходимый компонент в нужный момент.

- Можно представить 3 вида разделения кода приложения:
- постраничное разделение
- разделение вне видимости
- разделение по условию
- директивы

Директивы - специальные атрибуты с префиксом v-. В качестве значения атрибуты принимают одно выражение JavaScript (кроме v-for). Директива реактивно применяет к DOM изменения при обновлении значения этого выражения.

`<p v-if="seen">Сейчас меня видно</p>`

В примере директива v-if удалит или вставит элемент `<p>` в зависимости от истинности значения выражения seen.

- анимация и переходы

Vue позволяет анимировать переходы при добавлении, обновлении и удалении объектов DOM.

- автоматически использовать CSS-классы для анимаций и переходов
- интегрировать сторонние библиотеки CSS-анимаций, например, animate.css

- использовать JavaScript для работы с DOM напрямую в transition-хуках
- интегрировать сторонние библиотеки JavaScript-анимаций, например, Velocity.js

Система анимирования переходов Vue предоставляет много простых методов для анимации появления и исчезновения элементов и списков, а также анимации самих данных. Например:

- чисел и арифметических выражений
- отображаемых цветов
- позиции SVG-элементов
- размеров и прочих свойств элементов

31. Обзор клиентских фреймворков. Сравнительный анализ Angular, React, и Vue.js.

Angular vs React vs Vue: Популярность в 2020 году

Тренды NPM: React - самый высоко оцениваемый и скачиваемый фреймворк, за которым следует Vue.

В соответствии с трендами NPM, React является наиболее скачиваемым фреймворком. Однако это не означает, что он является самым лучшим. Чтобы сделать вывод о том, какой вариант является лучшим, необходимо провести множество опросов и голосований

Тренды Stack Overflow: React находится в топе по трендам, обгоняя Angular. При сравнении Angular vs React с популярностью Vue в 2020 году, Stack Overflow Trends показывает, что React получает наибольший процент, за которым следует Angular. Но популярность Vue постоянно растет по сравнению с последними несколькими годами.

GitHub тренды: Vue - это наиболее высоко оцениваемый фреймворк, за которой следует React.

В репозиториях Github есть несколько удивительных, но захватывающих статистик. С одной стороны, у React больше всего Fork-ов, а у Vue больше всего звёзд.

Angular vs React vs Vue: Производительность

Angular

Angular - популярный JavaScript фреймворк, который использует реальный DOM и лучше всего подходит для одностраничных приложений (SPA), в которых контент время от времени обновляется по AJAX.

Во-вторых, Angular использует двусторонний процесс связывания данных, который воспроизводит все изменения, при обновлении Модели, в представления безопасным, эффективным и интуитивно понятным способом.

Недостаток: Из-за большого количества функций этого фреймворка, при переводе вашего проекта в тяжелые приложения, он снизит производительность по сравнению с React и Vue.

React

React - это front-end библиотека, использующая Virtual DOM и обеспечивающая повышение производительности приложений любого размера, нуждающихся в регулярном обновлении контента. Например, Instagram.

React основана на одностороннем управлении данными. Это обеспечивает лучший контроль над всем проектом.

Недостатки: Поскольку React постоянно меняется, разработчикам, работающим с React, необходимо регулярно обновлять свои навыки, чтобы безупречно идти в ногу со временем.

Vue

Vue - самый молодой JS-фреймворк, разработанный с невероятными возможностями для решения проблем, с которыми сталкиваются разработчики при работе с Angular and React. Он состоит из всех хороших вещей React и Angular, поэтому в нём используется виртуальный DOM для обеспечения высокой производительности и распределения памяти.

Недостатки: Будучи новым членом семьи, Vue имеет самое маленькое сообщество, по сравнению с React и Angular.

Замечание: По сравнению с другими фреймворками Javascript, Angular, React и Vue являются тремя самыми быстрыми фреймворками для построения веб-приложений.

Angular vs React vs Vue: Миграция

Angular

Angular обычно выпускает мажорные обновления каждые шесть месяцев. Кроме того, существует период ещё в шесть месяцев до того, как основное API будут считаться устаревшим. В конечном итоге это означает, что разработчики имеют два цикла выпуска обновлений по шесть месяцев для внесения необходимых изменений.

React

Когда речь идёт о Vue vs Angular vs React в 2020 году, обновления версий в React, как правило, гораздо более простое, чем в Angular и Vue. Такие скрипты, как React codemod, обеспечивают плавный переход и стабильность при переходе с одной версии на другую. Кроме того, Facebook также считает, что стабильность является приоритетной задачей, так как некоторые известные компании, такие как Twitter и Airbnb, используют React.

Vue

Vue имеет самые умные и разнообразные варианты миграции для разработчиков. Если вы нанимаете веб-разработчиков, то им просто нужно использовать инструмент хелпера по миграции для внесения изменений на сайте, так как 90% API остаются неизменными, если вы решите перейти с 1.x на 2.

Angular vs React vs Vue: Кривая обучения

Кривая обучения любого фреймворка не ограничивается только программированием. На самом деле, то, насколько легко отлаживать и тестировать код, также является реальной проблемой, особенно когда компания, разрабатывающая веб-приложения, работает с огромным проектом с большим количеством кода и разработчиков в приложении.

С одной стороны, Angular and React требует глубоких знаний и опыта работы с JavaScript для принятия решений относительно сторонних библиотек. С другой стороны, многие компании быстро переходят на Vue, так как он имеет самую простую кривую обучения по сравнению с Angular и React.

Тем не менее, большая часть разработчиков предпочитает React, поскольку это позволяет создавать приложения новейшими современными способами, в отличие от Vue, который все еще следует старинному стилю разработки веб-приложений на JavaScript.

Несмотря на то, что популярность Angular падает, он находится в тройке лучших JS-фреймворков, так как предоставляет разработчикам чёткие, информативные сообщения для быстрого исправления проблем в разработке приложений, при возникновении ошибок.

Angular vs React vs Vue: Размер

Angular

Angular поставляется с широким набором возможностей и инструментов по расширению возможностей разработчиков от шаблонов до тестовых утилит. Если вы наняли индийского разработчика для создания крупного веб-сайта, многофункционального приложения, то Angular - идеальный вариант для вас.

React

React не является фреймворком, предоставляющим вам большой набор функциональных возможностей, архитектуры, как Angular. Поэтому React - это подходящий каркас для создания современной веб-разработки, и проектирования большей части архитектуры "под себя".

Vue

Vue является наименьшим из остальных фреймворков и библиотек, как упоминалось выше, и является идеальным вариантом для легковесной веб-разработки и одностраничных SPA приложений. Если вы хотите выбрать фреймворк, который легко установить, запустить и работать, и который, к тому же, меньше по размеру, то Vue - лучший вариант.

Angular vs React vs Vue: Скорость деплоя

Angular

Поскольку Angular - это масштабный фреймворк, который охватывает все этапы - от создания проекта до оптимизации кода, он является самым сложным фреймворком для развертывания в целом. Однако, поскольку Angular предоставляет широкий выбор возможностей, разработчики могут получить полностью оптимизированное приложение в билде при развёртывании на любом статическом хостинге с помощью одной команды.

React

Хотя React не поставляется с теми же инструментами, как Angular или Vue, он допускает компромисс в отношении гибкостью. С помощью React вы можете комбинировать и использовать любую библиотеку. С ростом экосистемы у нас появились возможность использовать такие инструменты, как: CLI, Create React App и Next.js.

Vue

По сравнению с Angular и React, структура программирования на Vue позволяет быстро развернуть приложение без ущерба для его производительности. С помощью простой команды вы можете использовать именно то, что вам нужно для разработки. Разработка приложения происходит быстро и легко с помощью Vue, поэтому это идеальный вариант для стартапов.

Angular

С момента запуска Angular в 2010 году он постоянно поддерживается компанией Google и выходит с частыми обновлениями каждые шесть месяцев. Более того, согласно отчету Medium, Angular - это фреймворк, который используют такие известные компании, как Microsoft, Autodesk, Apple, Adobe, Freelancer, Upwork, Telegram и так далее.

React

Он был представлен Facebook в 2013 году как JavaScript-библиотека, которая предоставляет вам богатый функционал фреймворка. Основными компаниями, которые используют React, являются Instagram, Netflix, New York Times, Yahoo, Whatsapp, Microsoft, Airbnb, Dropbox и так далее.

Vue

Он может быть и является новым фреймворком и не имел такого активного комьюнити, как Angular и React, но всё же, он используется ведущими компаниями, включая EuroNews, Alibaba, WizzAir, Xiaomi, Gitlab, Grammarly и многие другие.

Советы по выбору

Выберите Angular, если вам необходимо:

- При разработке большого и многофункционального приложения.
- Надежный и масштабируемый фреймворк.
- Для разработки приложений в реальном времени, таких как чат-приложения или приложения для работы с сообщениями.
- Для разработки нативных приложений, гибридных приложений или веб-приложений, которые являются долгосрочными и существенными инвестиционными проектами.
- Программирование на TypeScript.
- Объектно-ориентированное программирование.

Выберите React, если вам необходимо:

- Разработка лёгких современных приложений корпоративного уровня в сжатые сроки.
- Гибкий фреймворк, обеспечивающий безопасные решения для разработки веб-сайтов.
- Для разработки кросс-платформенных или одностраничных приложений.
- Расширение функциональности существующего приложения.
- Мощная поддержка и решения сообщества.

Выберите Vue, если вам необходимо:

- Умные, быстрые и высокопроизводительные приложения.
- Для разработки небольшого и легкого приложения, по типу как Grammarly.

- Выбор фреймворка, обеспечивающего скорейший выпуск продукта на рынок.
- Переход от существующего проекта к современному фреймворку, но с ограниченными ресурсами.
- Использовать фреймворк поддерживаемый сообществом, а не компанией.

32. *CSS. История CSS. Основные определения. Общий синтаксис таблиц стилей. Типы селекторов с примерами. Работа со шрифтами в CSS. Цветовое оформление в CSS.*

Основные определения

CSS (Cascading Style Sheets – каскадные таблицы стилей) – язык описания внешнего вида web-документа, созданного при помощи языка разметки ([HTML](#))

CSS — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML).

Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL.

Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

История

В самом начале 1990 года, для того, чтобы отображать веб-страницы, разные браузеры обладали своими собственными стилями. Развитие HTML было очень быстрым, и он был способен удовлетворять на тот момент все существовавшие потребности по оформлению информации, именно поэтому тогда и не получил широкого признания CSS.

И лишь Хокон Виум Ли в 1994 году предложил для HTML документов использование концепции «каскадные таблицы стилей». Один из основателей Netscape, Марк Андреесен, в 1994 году 13 октября сообщил, что доступна для тестирования от Netscape Navigator первая версия. И за три дня до проведения тестирования, норвежский программист, сейчас он является сотрудником компании Opera Software, Хокон Виум Ли публикует черновой вариант CSS.

В ноябре 1994 года в Чикаго на Веб-конференции, как и планировалось, был предоставлен первый черновик CSS. Дебаты различного политического характера и разрешение некоторых технических вопросов продолжались в течение двух лет, но 1996 года 17 декабря W3C официально рекомендовал CSS1.

Уровень развития первый (CSS1).

На данном этапе имеется возможность задавать гарнитуру и размер шрифта, а еще изменять его стиль: обычный, курсив или полужирный. Благодаря спецификации имеется возможность определять рамки, фоны, цвета текста и другие элементы страницы. Самый первый браузер для использования являлся Microsoft Internet Explorer 3. Вышел 1996 года в августе, он то и поддерживал новый принятый стандарт.

Уровень развития второй (CSS2).

Рекомендация от консорциума W3C, которую приняли 1998 году 12 мая. Она основана на CSS1, а также сохранила обратную совместимость с добавлением некоторых функций, а именно:

- 1) Возникло фиксированное, абсолютное и относительное позиционирование.

- 2) Для разных носителей возможность устанавливать разными стилями.
- 3) Для звуковых носителей появилась возможность определять громкость и голос.
- 4) Позволяет устанавливать на нечетных и четных страницах во время печати различные элементы.
- 5) Расширился механизм селекторов.
- 6) Возможность добавлять содержимое, которое не содержится в исходном документе.

Уровень третий (CSS3).

Очень сильно расширены функции. Введены нововведения, начиная от мелочей, заканчивая трансформацией, а также введение новых переменных.

Основные определения.

- *Каскадные таблицы стилей (Cascading Style Sheets, CSS)* — это стандарт, определяющий представление данных в браузере. Если HTML предоставляет информацию о структуре документа, то таблицы стилей сообщают как он должен выглядеть.
- *Стиль* — это совокупность правил, применяемых к элементу гипертекста и определяющих способ его отображения. Стиль включает все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице.
- *Таблица стилей* — это совокупность стилей, применимых к гипертекстовому документу.
- *Каскадирование* — это порядок применения различных стилей к веб-странице. Браузер, поддерживающий таблицы стилей, будет последовательно применять их в соответствии с приоритетом: сначала связанные, затем внедренные и, наконец, встроенные стили. Другой аспект каскадирования — *наследование (inheritance)*, — означает, что если не указано иное, то конкретный стиль будет применен ко всем дочерним элементам гипертекстового документа. Например, если вы примените определенный цвет текста в теге <div>, то все теги внутри этого блока будут отображаться этим же цветом.

Общий синтаксис таблиц стилей



Объявление стиля состоит из двух частей: **селектора** и **объявления**. В HTML имена элементов нечувствительны к регистру, поэтому `<h1>` работает так же, как и `<H1>`. Объявление состоит из двух частей: имя свойства (например, `color`) и значение свойства (`grey`). Селектор сообщает

браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) перечисляются формирующие команды — свойства и их значения.

Типы селекторов

1. Универсальный селектор

Соответствует любому HTML-элементу. Например, `* {margin: 0;}` обнулит внешние отступы для всех элементов сайта. Также селектор может использоваться в комбинации с псевдоклассом или псевдоэлементом: `*:after {CSS-стили}`, `*:checked {CSS-стили}`.

2. Селектор элемента

Селекторы элементов позволяют форматировать все элементы данного типа на всех страницах сайта. Например, `h1 {font-family: Lobster, cursive;}` задаст общий стиль форматирования всех заголовков `h1`.

3. Селектор класса

Селекторы класса позволяют задавать стили для одного и более элементов с одинаковым именем класса, размещенных в разных местах страницы или на разных страницах сайта. Например, для создания заголовка с классом `headline` необходимо добавить атрибут `class` со значением `headline` в открывающий тег `<h1>` и задать стиль для указанного класса. Стили, созданные с помощью класса, можно применять к другим элементам, не обязательно данного типа.

```
<h1 class="headline">Инструкция пользования персональным компьютером</h1>
```

HTML

```
.headline {  
text-transform: uppercase;  
color: lightblue;  
}
```

CSS

Если элемент имеет несколько атрибутов класса, их значения объединяются с пробелами.

```
<h1 class="headline post-title">Инструкция пользования персональным компьюте-  
ром</h1>
```

HTML

4. Селектор идентификатора

Селектор идентификатора позволяет форматировать **один** конкретный элемент. Значение `id` должно быть уникальным, на одной странице может встречаться только один раз и должно содержать хотя бы один символ. Значение не должно содержать пробелов.

Нет никаких других ограничений на то, какую форму может принимать `id`, в частности, идентификаторы могут состоять только из цифр, начинаться с цифры, начинаться с подчеркивания, состоять только из знаков препинания и т. д.

Уникальный идентификатор элемента может использоваться для различных целей, в частности, как способ ссылки на конкретные части документа с использованием идентификаторов фрагментов, как способ нацеливания на элемент при создании сценариев и как способ стилизации конкретного элемента из CSS.

```
<div id="sidebar"></div>
```

HTML

```
#sidebar {  
width: 300px;  
float: left;  
}
```

CSS

5. Селектор потомка

Селекторы потомков применяют стили к элементам, расположенным внутри элемента-контейнера. Например, `ul li {text-transform: uppercase;}` — выберет все элементы `li`, являющиеся потомками всех элементов `ul`.

Если нужно отформатировать потомки определенного элемента, этому элементу нужно задать стилевой класс:

- `p.first a {color: green;}` — данный стиль применится ко всем ссылкам, потомкам абзаца с классом `first`;
- `p .first a {color: green;}` — если добавить пробел, то будут стилизованы ссылки, расположенные внутри любого тега класса `.first`, который является потомком элемента `<p>`;
- `.first a {color: green;}` — данный стиль применится к любой ссылке, расположенной внутри другого элемента, обозначенного классом `.first`.

6. Дочерний селектор

Дочерний элемент является прямым потомком содержащего его элемента. У одного элемента может быть несколько дочерних элементов, а родительский элемент у каждого элемента может

быть только один. Дочерний селектор позволяет применить стили только если дочерний элемент идёт сразу за родительским элементом и между ними нет других элементов, то есть дочерний элемент больше ни во что не вложен.

Например, `p > strong` — выберет все элементы `strong`, являющиеся дочерними по отношению к элементу `p`.

7. Сестринский селектор

Сестринские отношения возникают между элементами, имеющими общего родителя. Селекторы сестринских элементов позволяют выбрать элементы из группы элементов одного уровня:

- `h1 + p` — выберет все первые абзацы, идущие непосредственно за любым тегом `<h1>`, не затрагивая остальные абзацы;
- `h1 ~ p` — выберет все абзацы, являющиеся сестринскими по отношению к любому заголовку `h1` и идущие сразу после него.

8. Селектор атрибута

Селекторы атрибутов выбирают элементы на основе имени атрибута или значения атрибута:

- `[атрибут]` — все элементы, содержащие указанный атрибут, `[alt]` — все элементы, для которых задан атрибут `alt`;
- `селектор[атрибут]` — элементы данного типа, содержащие указанный атрибут, `img[alt]` — только картинки, для которых задан атрибут `alt`;
- `селектор[атрибут="значение"]` — элементы данного типа, содержащие указанный атрибут с конкретным значением, `img[title="flower"]` — все картинки, название которых содержит слово `flower`;
- `селектор[атрибут~="значение"]` — элементы частично содержащие данное значение, например, если для элемента задано несколько классов через пробел, `p[class~="feature"]` — абзацы, имя класса которых содержит `feature`;
- `селектор[атрибут|= "значение"]` — элементы, список значений атрибута которых начинается с указанного слова, `p[class|= "feature"]` — абзацы, имя класса которых `feature` или начинается на `feature`;
- `селектор[атрибут^="значение"]` — элементы, значение атрибута которых начинается с указанного значения, `a[href^="http://"]` — все ссылки, начинающиеся на `http://`;
- `селектор[атрибут$="значение"]` — элементы, значение атрибута которых заканчивается указанным значением, `img[src$=".png"]` — все картинки в формате `png`;
- `селектор[атрибут*="значение"]` — элементы, значение атрибута которых содержит в любом месте указанное слово, `a[href*="book"]` — все ссылки, название которых содержит `book`.

9. Селектор псевдокласса

Псевдоклассы — это классы, фактически не прикрепленные к HTML-тегам. Они позволяют применить CSS-правила к элементам при совершении события или подчиняющимся определенному правилу.

Псевдоклассы характеризуют элементы со следующими свойствами:

- `:link` — не посещенная ссылка;
- `:visited` — посещенная ссылка;
- `:hover` — любой элемент, по которому проводят курсором мыши;
- `:focus` — интерактивный элемент, к которому перешли с помощью клавиатуры или активировали посредством мыши;
- `:active` — элемент, который был активизирован пользователем;
- `:valid` — поля формы, содержимое которых прошло проверку в браузере на соответствие указанному типу данных;
- `:invalid` — поля формы, содержимое которых не соответствует указанному типу данных;
- `:enabled` — все активные поля форм;
- `:disabled` — заблокированные поля форм, т.е., находящиеся в неактивном состоянии;
- `:in-range` — поля формы, значения которых находятся в заданном диапазоне;
- `:out-of-range` — поля формы, значения которых не входят в установленный диапазон;
- `:lang()` — элементы с текстом на указанном языке;
- `:not(селектор)` — элементы, которые не содержат указанный селектор — класс, идентификатор, название или тип поля формы — `:not([type="submit"])`;
- `:target` — элемент с символом `#`, на который ссылаются в документе;
- `:checked` — выделенные (выбранные пользователем) элементы формы.

10. Селектор структурных псевдоклассов

Структурные псевдоклассы отбирают дочерние элементы в соответствии с параметром, указанным в круглых скобках:

- `:nth-child(odd)` — нечётные дочерние элементы;
- `:nth-child(even)` — чётные дочерние элементы;
- `:nth-child(3n)` — каждый третий элемент среди дочерних;
- `:nth-child(3n+2)` — выбирает каждый третий элемент, начиная со второго дочернего элемента `(+2)`;
- `:nth-child(n+2)` — выбирает все элементы, начиная со второго;
- `:nth-child(3)` — выбирает третий дочерний элемент;
- `:nth-last-child()` — в списке дочерних элементов выбирает элемент с указанным местоположением, аналогично с `:nth-child()`, но начиная с последнего, в обратную сторону;

- `:first-child` — позволяет оформить только самый первый дочерний элемент тега;
- `:last-child` — позволяет форматировать последний дочерний элемент тега;
- `:only-child` — выбирает элемент, являющийся единственным дочерним элементом;
- `:empty` — выбирает элементы, у которых нет дочерних элементов;
- `:root` — выбирает элемент, являющийся корневым в документе — элемент `html`.

11. Селектор структурных псевдоклассов типа

Указывают на конкретный тип дочернего тега:

- `:nth-of-type()` — выбирает элементы по аналогии с `:nth-child()`, при этом берёт во внимание только тип элемента;
- `:first-of-type` — выбирает первый дочерний элемент данного типа;
- `:last-of-type` — выбирает последний элемент данного типа;
- `:nth-last-of-type()` — выбирает элемент заданного типа в списке элементов в соответствии с указанным местоположением, начиная с конца;
- `:only-of-type` — выбирает единственный элемент указанного типа среди дочерних элементов родительского элемента.

12. Селектор псевдоэлемента

Псевдоэлементы используются для добавления содержимого, которое генерируется с помощью свойства `content`:

- `:first-letter` — выбирает первую букву каждого абзаца, применяется только к блочным элементам;
- `:first-line` — выбирает первую строку текста элемента, применяется только к блочным элементам;
- `:before` — вставляет генерируемое содержимое перед элементом;
- `:after` — добавляет генерируемое содержимое после элемента.

Работа со шрифтами

1. Семейство шрифтов: свойство `font-family`

Свойство `font-family` используется для выбора начертания шрифта. Поскольку невозможно предсказать, установлен тот или иной шрифт на компьютере посетителя вашего сайта, рекомендуется прописывать все возможные варианты однотипных шрифтов. В таком случае браузер будет проверять их наличие, последовательно перебирая предложенные варианты.

Если в названии шрифта имеются пробелы или символы (например, #, \$, %), то оно заключается в кавычки. Это делается для того, чтобы браузер мог понять, где начинается и заканчивается название шрифта.

Свойство наследуется.

font-family

Значения:

family-name Название (имя) семейства шрифтов, например, Times, Courier, Arial. Рекомендуется указывать вместе с базовым семейством.

generic-family Базовое семейство. CSS определяет пять базовых семейств шрифтов:
Шрифты с засечками — Serif (Times New Roman, Times, Garamond, Georgia)
Рубленые шрифты — Sans-serif (Helvetica, Geneva, Arial, Verdana, Trebuchet, Univers)
Моноширинные шрифты — Monospace (Courier, Courier New, Andale Mono)
Рукописные шрифты — Cursive (Comic Sans, Gabriola, Monotype Corsiva, Author, Zapf Chancery)
Аллегорические шрифты (Western, Woodblock, Klingon)

`initial` Устанавливает значение свойства в значение по умолчанию.

`inherit` Наследует значение свойства от родительского элемента.

2. Насыщенность шрифта: свойство font-weight

Свойство `font-weight` задаёт насыщенность шрифта.

Свойство наследуется.

font-weight

Значения:

`normal` Значение по умолчанию, устанавливает нормальную насыщенность шрифта. Эквивалентно значению насыщенности, равной 400.

`bold` Делает шрифт текста полужирным. Эквивалентно значению насыщенности, равной 700.

`bolder` Насыщенность шрифта будет больше, чем у предка.

`lighter` Насыщенность шрифта будет меньше, чем у предка.

`100, 200, 300, 400, 500, 600, 700, 800, 900` Значение 100 соответствует самому легкому варианту начертания шрифта, а 900 — самому плотному. При этом, эти числа не опре-

600, 700, 800,
900

деляют конкретной плотности, т.е. 100, 200, 300 и 400 могут соответствовать одному и тому же варианту слабой насыщенности начертания шрифта; 500 и 600 — средней насыщенности, а 700, 800 и 900 могут выводить одинаковое очень насыщенное начертание. Распределение плотности так же зависит от количества уровней насыщенности, определенных в конкретном семействе шрифтов.

`initial`

Устанавливает значение свойства в значение по умолчанию.

`inherit`

Наследует значение свойства от родительского элемента.

3. Ширина шрифта: свойство `font-stretch`

Свойство `font-stretch` позволяет выбрать нормальное, сжатое или расширенное начертание символа из семейства шрифтов. Свойство не работает на любом шрифте, а только на шрифтах, для которых разработаны различными начертания, соответствующими определенным размерам.

Свойство наследуется.

Абсолютные значения ключевых слов имеют следующий порядок, от самого узкого до самого широкого:

`font-stretch`

Значения:

`ultra-condensed`

Указывает на наиболее сжатый шрифт.

`extra-condensed`

Указывает на второй по сжатости шрифт.

`condensed`

Указывает на сжатый шрифт.

`semi-condensed`

Указывает на немного сжатый шрифт.

`normal`

Значение по умолчанию.

`semi-expanded`

Слегка расширенный шрифт.

`expanded`

Расширенный шрифт.

`extra-expanded`

Второй по расширенности шрифт.

`ultra-expanded`

Максимально расширенный шрифт.

`initial`

Устанавливает значение свойства в значение по умолчанию.

`inherit`

Наследует значение свойства от родительского элемента.

4. Начертание шрифта: свойство `font-style`

Свойство `font-style` позволяет выбрать стиль начертания для шрифта. При этом разница между курсивом и наклонным начертанием заключается в том, что курсив вносит небольшие изменения в структуру каждого символа, в то время как наклонное начертание представляет собой наклонную версию прямого шрифта.

Свойство наследуется.

`font-style`

Значения:

`normal`

Значение по умолчанию, устанавливает для текста обычное начертание шрифта.

`italic`

Выделяет текст курсивом.

`oblique`

Устанавливает наклонное начертание шрифта.

`initial`

Устанавливает значение свойства в значение по умолчанию.

`inherit`

Наследует значение свойства от родительского элемента.

5. Размер шрифта: свойство `font-size`

Свойство `font-size` указывает желаемую высоту глифов из шрифта.

Свойство наследуется.

`font-size`

Значения:

`absolute-size`

`xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`. В качестве стандартного размера принимается `medium`. В CSS1 предложенный коэффициент масштабирования между соседними индексами составлял 1.5, что для пользователя оказалось слишком большим. В CSS2 предложенный коэффициент масштаби-

рования для экрана компьютера между смежными индексами составлял 1.2, что все еще создавало проблемы для небольших размеров. Новый коэффициент масштабирования варьируется между каждым индексом, чтобы обеспечить лучшую читаемость.

relative-size

`smaller`, `larger`. Относительные размеры обуславливают изменение размера шрифта элемента относительно родителя. При этом размер шрифта может выйти за рамки размеров, предполагаемых для `xx-small` и `xx-large`.

длина

Размер шрифта устанавливается с помощью положительных значений единиц длины, например, `px`, `em`, как целых, так и дробных.

`%`

Относительное значение, вычисляется на основании любого размера, унаследованного от родительского элемента. Обеспечивает более точную настройку вычисляемого размера шрифта. Задание размеров шрифта с помощью `em` эквивалентно процентному значению.

`initial`

Устанавливает значение свойства в значение по умолчанию.

`inherit`

Наследует значение свойства от родительского элемента.

CSS

Значения absolute-size

<code>xx-small</code>	<code>x-small</code>	<code>small</code>	<code>medium</code>	<code>large</code>	<code>x-large</code>	<code>xx-large</code>	
-----------------------	----------------------	--------------------	---------------------	--------------------	----------------------	-----------------------	--

Коэффициент масштабирования

3/5	3/4	8/9	1	6/5	3/2	2/1	3/1
-----	-----	-----	---	-----	-----	-----	-----

HTML заголовки

<code>h6</code>		<code>h5</code>	<code>h4</code>	<code>h3</code>	<code>h2</code>	<code>h1</code>	
-----------------	--	-----------------	-----------------	-----------------	-----------------	-----------------	--

HTML размер шрифта

1		2	3	4	5	6	7
---	--	---	---	---	---	---	---

6. 6. Относительный размер шрифта: свойство font-size-adjust

Свойство `font-size-adjust` — способ сохранить читабельность текста при использовании резервных шрифтов. Это достигается путем настройки размера шрифта таким образом, чтобы `x-height` была одинаковой независимо от используемого шрифта.

Свойство наследуется.

font-size-adjust	
Значения:	
<code>none</code>	Не сохраняет <code>x-height</code> шрифта.
число	Задаёт значение аспекта, используемое в приведенных ниже расчетах для расчета скорректированного размера шрифта: $c = (a / a')$ где: <code>s</code> = значение размера шрифта <code>a</code> = значение аспекта, указанное в свойстве <code>font-size-adjust</code> <code>a'</code> = значение аспекта фактического шрифта <code>c</code> = скорректированный размер шрифта для использования Отрицательные значения недействительны.
<code>initial</code>	Устанавливает значение свойства в значение по умолчанию.
<code>inherit</code>	Наследует значение свойства от родительского элемента.

7. Сокращенная запись свойств шрифта: свойство font

Свойство `font` за исключением описанного ниже, является сокращенным свойством для установки `font-style`, `font-variant`, `font-weight`, `font-stretch`, `font-size/line-height`, `font-family`. Также могут быть включены значения для свойства `font-variant`, которые поддерживаются CSS 2.1 — `normal` или `small-caps`.

Все подсвойства свойства `font` сначала сбрасываются на свои начальные значения, включая перечисленные выше, плюс `font-size-adjust`, `font-kerning`, все подсвойства `font-variant` и настройки шрифтов, за исключением `font-synthesis`. Затем этим свойствам присваиваются те значения, которые указаны в свойстве `font`. Для свойства `font-size-adjust` невозможно установить значение, отличное от его начального значения, поэтому следует использовать вместо этого индивидуальное свойство. Если явное значение какого-либо свойства не нужно, то оно опускается.

Свойство наследуется.

Следующие значения относятся к системным шрифтам:

- `caption` — шрифт, используемый для элементов управления с субтитрами (например, кнопок, раскрывающихся списков и т.д.).
- `icon` — шрифт, используемый для обозначения значков.
- `menu` — шрифт, используемый в меню (например, раскрывающиеся меню и списки меню).
- `message-box` — шрифт, используемый в диалоговых окнах.
- `small-caption` — шрифт, используемый для маркировки подписи элементов управления.
- `status-bar` — шрифт, используемый в строке состояния окна.

Системные шрифты могут быть установлены только целиком; то есть семейство шрифтов, размер, вес, стиль и т.д. задаются одновременно. Эти значения затем могут быть изменены индивидуально, если это необходимо. Ключевые слова, используемые для системных шрифтов, перечисленных выше, обрабатываются как ключевые слова только в том случае, если они находятся в начальной позиции, в других позициях эта же строка обрабатывается как часть имени семейства шрифтов. Системные шрифты могут быть указаны только с этим свойством, но не с самим `font-family`.

```
font: menu;          /* используются настройки шрифта для системных меню */
font: large menu;    /* используется семейство шрифтов под названием "menu"
*/
```

CSS

8. Управление синтезом шрифтов: свойство `font-synthesis`

Свойство `font-synthesis` определяет, разрешено ли пользовательским агентам (браузерам) синтезировать полужирное или наклонное начертание шрифтов, когда они отсутствуют в семействе шрифтов. Если `weight` не указан, пользовательские агенты не должны синтезировать полужирное начертание, а если `style` не указан, пользовательские агенты не должны синтезировать курсив.

Свойство наследуется.

`font-synthesis`

Значения:

<code>none</code>	Запрещает синтез начертаний.
<code>weight</code> и/или <code>style</code>	По умолчанию свойство принимает значение <code>font-synthesis: weight style;</code> . Если указано только <code>weight</code> , это говорит браузеру, что жирный шрифт может быть синтезирован при необходимости. Если только <code>style</code> — синтезируется только курсив.
<code>initial</code>	Устанавливает значение свойства в значение по умолчанию.
<code>inherit</code>	Наследует значение свойства от родительского элемента.

Цветовое оформление

1. Приоритетные цвета: свойство color

Свойство задаёт цвет шрифта с помощью различных систем цветопередачи. Свойство описывает цвет текстового содержимого элемента. Кроме того, оно используется для предоставления потенциального косвенного значения (`currentColor`) для любых других свойств, которые принимают значения цвета.

Свойство наследуется.














color	
Значения:	
цвет	Задаётся с помощью значений цвета.
<code>inherit</code>	Наследует значение свойства от родительского элемента.

2. Значения цвета

2.1. Основные ключевые слова

Список основных ключевых слов включает в себя следующие значения:

Название	HEX	RGB	Цвет
----------	-----	-----	------

black	#000000	0,0,0	
silver	#C0C0C0	192,192,192	
gray	#808080	128,128,128	
white	#FFFFFF	255,255,255	
maroon	#800000	128,0,0	
red	#FF0000	255,0,0	
purple	#800080	128,0,128	
fuchsia	#FF00FF	255,0,255	
green	#008000	0,128,0	
lime	#00FF00	0,255,0	
olive	#808000	128,128,0	
yellow	#FFFF00	255,255,0	
navy	#000080	0,0,128	
blue	#0000FF	0,0,255	
teal	#008080	0,128,128	
aqua	#00FFFF	0,255,255	

Названия цветов не чувствительны к регистру.

Синтаксис

```
color: teal;
```

CSS

2.2. Числовые значения цвета

2.2.1. Цвета модели RGB

Формат значения RGB в шестнадцатеричном формате — это знак #, за которым сразу следуют три или шесть шестнадцатеричных символов. Трехзначная запись RGB #rgb преобразуется в шестизначную форму #rrggbb путем копирования цифр, а не путем добавления нулей. Например, #fb0 расширяется до #ffbb00. Это гарантирует, что белый #ffffff может быть указан в короткой записи #fff, и удаляет любые зависимости от глубины цвета дисплея.

Формат значения RGB в функциональной нотации — rgb(, за которым следует разделенный запятыми список из трех числовых значений (либо трех целочисленных значений, либо трех процентных значений), за которыми следует символ). Целочисленное значение 255 соответствует 100% и F или FF в шестнадцатеричной записи:

rgb(255,255,255) = rgb(100%, 100%, 100%) = #FFF

Символы пробела допускаются вокруг числовых значений.

Все цвета RGB указываются в цветовом пространстве sRGB. Пользовательские агенты могут различаться в точности, с которой они представляют эти цвета, но использование sRGB дает однозначное и объективно измеримое определение того, каким должен быть цвет.

Значения за пределами диапазона устройства должны быть обрезаны или отображены в известном диапазоне: значения красного, зеленого и синего необходимо изменить, чтобы они попадали в диапазон, поддерживаемый устройством. Некоторые устройства, например принтеры, имеют диапазоны, отличные от sRGB, поэтому некоторые цвета за пределами диапазона 0..255 sRGB будут представимы (внутри диапазона устройства) и будут отображаться.

Синтаксис

```
color: #fb0;  
color: #ffbb00;  
color: rgb(255,0,0);  
color: rgb(100%, 0%, 0%);
```

CSS

2.2.2. Цвета модели RGBA

Цветовая модель RGB расширена в этой спецификации, чтобы включить alpha, которая управляющая непрозрачностью цвета. В отличие от значений RGB, для значения RGBA нет шестнадцатеричной записи.

Формат значения RGBA в функциональной нотации — rgba(за которым следует разделенный запятыми список из трех числовых значений (либо трех целочисленных значений, либо трех процентных значений), за которыми следует значение непрозрачности, а затем). Целочисленное значение 255 соответствует 100%, rgba(255,255,255,0.8)

= `rgba(100%,100%,100%,0.8)`. Символы пробела допускаются вокруг числовых значений.

Параметр непрозрачности применяется ко всему объекту. Любые значения за пределами диапазона от `0.0` (полностью прозрачный) до `1.0` (полностью непрозрачный) будут ограничены этим диапазоном.

Синтаксис

```
color: rgba(0,0,255,0.5);  
color: rgba(100%, 50%, 0%, 0.1);
```

CSS

2.2.3. Ключевое слово transparent

Это ключевое слово можно считать сокращением для прозрачного черного цвета `rgba(0,0,0,0)`, которое является его вычисленным значением.

Синтаксис

```
color: transparent;
```

CSS

2.2.4. HSL-цвета

Цвета RGB не интуитивно понятны. CSS3 добавляет числовые цвета hue-saturation-lightness (HSL) в дополнение к числовым цветам RGB. HSL-цвета симметричны свету и темноте, и преобразование HSL в RGB максимально просто.

Цвета HSL кодируются как тройка (оттенок, насыщенность, яркость). Оттенок представлен как угол цветного круга (то есть радуга, представленная в круге). Этот угол обычно измеряется в градусах, так что эта единица измерения неявна в CSS; синтаксически дается только число. По определению красный = 0 = 360, а остальные цвета распределены по кругу, поэтому зеленый = 120, синий = 240 и т.д. Насыщенность и яркость представлены в процентах. 100% — это полное насыщение, а 0% — это оттенок серого. Яркость 0% — черная, 100% — белая, а 50% — нормальная.

Синтаксис

```
color: hsl(0, 100%, 50%);  
color: hsl(120, 100%, 50%);
```

CSS

2.2.5. HSLA-значения цвета

Так же, как функциональная нотация `rgb()` имеет альфа-аналог `rgba()`, функциональная нотация `hsl()` имеет альфа-аналог `hsla()`.

Формат значения цвета HSLA в функциональной нотации — `hsla()`, за которым следуют оттенок в градусах, насыщенность и яркость в процентах, и значение непрозрачности, после которого следует символ `)`. Символы пробела допускаются вокруг числовых значений.

Синтаксис

```
color: hsla(240, 100%, 50%, 0.5);  
color: hsla(30, 100%, 50%, 0.1);
```

CSS

2.4. Ключевое слово `currentColor`

CSS1 и CSS2 определили начальное значение свойства `border-color` как значение свойства `color`, но не определили соответствующее ключевое слово. Это упущение было распознано SVG, поэтому SVG 1.0 ввел значение `currentColor` для свойств `fill`, `stroke`, `stop-color`, `flood-color` и `lighting-color`. CSS3 расширяет значение цвета, добавляя ключевое слово `currentColor`, чтобы разрешить его использование со всеми свойствами, которые принимают значение `color`. Это упрощает определение этих свойств в CSS3.

Используемое значение ключевого слова `currentColor` — это вычисленное значение свойства `color`. Если ключевое слово установлено в самом свойстве `color`, оно рассматривается как `color: inherit`.

Пример

```
body {color: grey;}  
div {border: 1px solid currentColor;}
```

5. Наследование и каскад

Наследование и каскад — два фундаментальных понятия в CSS, которые тесно связаны между собой.

Наследование заключается в том, что элементы наследуют свойства от своего родителя (элемента, их содержащего).

Каскад проявляется в том, как разные виды таблиц стилей применяются к документу, и как конфликтующие правила переопределяют друг друга.

5.1. Наследование

Наследование является механизмом, с помощью которого определенные свойства передаются от предка к его потомкам. Спецификацией CSS предусмотрено наследование свойств, относящихся к текстовому содержимому страницы, таких как `color`, `font`, `letter-spacing`, `line-height`, `list-style`, `text-align`, `text-indent`, `text-transform`, `visibility`, `white-space` и `word-spacing`. Во многих случаях это удобно, так как не нужно задавать размер шрифта и семейство шрифтов для каждого элемента веб-страницы.

Свойства, относящиеся к форматированию блоков, не наследуются. Это `background`, `border`, `display`, `float` и `clear`, `height` и `width`, `margin`, `min-max-height` и `-width`, `outline`, `overflow`, `padding`, `position`, `text-decoration`, `vertical-align` и `z-index`.

Принудительное наследование

С помощью ключевого слова `inherit` можно принудить элемент наследовать любое значение свойства родительского элемента. Это работает даже для тех свойств, которые не наследуются по умолчанию.

Как задаются и работают CSS-стили

Стили могут наследоваться от родительского элемента (наследуемые свойства или с помощью значения `inherit`).

Стили, расположенные в таблице стилей ниже, отменяют стили, расположенные в таблице выше.

К одному элементу могут применяться стили из разных источников. Проверить, какие стили применяются, можно в режиме разработчика браузера. Для этого над элементом нужно щёлкнуть правой кнопкой мыши и выбрать пункт «Посмотреть код» (или что-то аналогичное). В правом столбце будут перечислены все свойства, которые заданы для этого элемента или наследуются от родительского элемента, а также файлы стилей, в которых они указаны, и порядковый номер строки кода.

При определении стиля можно использовать любую комбинацию селекторов — селектор элемента, псевдокласса элемента, класса или идентификатора элемента.

```
<div id="wrap" class="box clear"></div>
```

HTML

```
div {border: 1px solid #eee;}
#wrap {width: 500px;}
.box {float: left;}
.clear {clear: both;}
```

CSS

5.2. Каскад

Каскадирование — это механизм, который управляет конечным результатом в ситуации, когда к одному элементу применяются разные CSS-правила. Существует три критерия, которые определяют порядок применения свойств — правило `!important`, специфичность и порядок, в котором подключены таблицы стилей.

Правило `!important`

Вес правила можно задать с помощью ключевого слова `!important`, которое добавляется сразу после значения свойства, например, `span {font-weight: bold!important;}`. Правило необходимо размещать в конец объявления перед закрывающей скобкой, без пробела. Такое объявление будет иметь приоритет над всеми остальными правилами. Это правило позволяет отменить значение свойства и установить новое для элемента из группы элементов в случае, когда нет прямого доступа к файлу со стилями.

Специфичность

Для каждого правила браузер вычисляет **специфичность селектора**, и если у элемента имеются конфликтующие объявления свойств, во внимание принимается правило, имеющее наибольшую специфичность. Значение специфичности состоит из четырех частей: `0, 0, 0, 0`. Специфичность селектора определяется следующим образом:

- для `id` добавляется `0, 1, 0, 0`;
- для `class` добавляется `0, 0, 1, 0`;
- для каждого элемента и псевдоэлемента добавляется `0, 0, 0, 1`;
- для встроенного стиля, добавленного непосредственно к элементу — `1, 0, 0, 0`;
- универсальный селектор не имеет специфичности.

```
h1 {color: lightblue;} /*специфичность 0, 0, 0, 1*/
em {color: silver;} /*специфичность 0, 0, 0, 1*/
h1 em {color: gold;} /*специфичность: 0, 0, 0, 1 + 0, 0, 0, 1 = 0, 0, 0, 2*/
div#main p.about {color: blue;} /*специфичность: 0, 0, 0, 1 + 0, 1, 0, 0 + 0, 0, 0, 1 + 0, 0, 1, 0 = 0, 1, 1, 2*/
.sidebar {color: grey;} /*специфичность 0, 0, 1, 0*/
#sidebar {color: orange;} /*специфичность 0, 1, 0, 0*/
```

```
li#sidebar {color: aqua;} /*специфичность: 0, 0, 0, 1 + 0, 1, 0, 0 = 0, 1, 0, 1*/
```

CSS

В результате к элементу применяются те правила, специфичность которых больше. Например, если на элемент действуют две специфичности со значениями `0, 0, 0, 2` и `0, 1, 0, 1`, то выиграет второе правило.

Порядок подключённых таблиц

Вы можете создать несколько внешних таблиц стилей и подключить их к одной веб-странице. Если в разных таблицах будут встречаться разные значения свойств одного элемента, то в результате к элементу применится правило, находящееся в таблице стилей, идущей в списке ниже.

Аппаратно-зависимые таблицы стилей

Одной из важнейших особенностей таблиц стилей является то, что они могут задавать способ представления документа для различных устройств вывода: монитора, синтезатора речи, устройства печати азбуки **Брайля**, печатных устройств и т.д. Ниже представлен перечень типов устройств, определяемых в **CSS**:

- *all*
Для всех устройств.
- *aural*
Для речевых синтезаторов.
- *braille*
Для устройств чтения азбуки **Брайля**.
- *embossed*
Для печати азбуки **Брайля**.
- *handheld*
Для переносных небольших портативных устройств.
- *print*
Для страничных непрозрачных материалов и для документов, просматриваемых на экране в режиме предварительного просмотра печати.
- *projection*
Для настенных презентаций, например, для проекторов.
- *screen*
Для цветных дисплеев.
- *tty*
Для устройств, использующих набор символов с фиксированной шириной, например, телетайпов, терминалов или портативных устройств с ограниченными возможностями отображения.
- *tv*
Для телевизионных приставок (низкое разрешение, цветное изображение, ограниченная прокрутка на экране, возможность передачи звука).

Существует несколько способов задания аппаратно-зависимых таблиц стилей:

- При помощи правила **@import**

```
<STYLE>  
@import URL("файл.css") print;  
</STYLE>
```

- При помощи правила **@media**

```
<STYLE>  
@media print{...};  
</STYLE>
```

- Задать аппаратно-зависимый стиль непосредственно в элементе **<LINK>**
`<LINK rel="stylesheet" type="text/css" media="print, handheld" href="file.css">`
- Задать аппаратно-зависимый стиль непосредственно в элементе **<STYLE>**

```
<STYLE  
селектор {атрибут: значение} media=print>  
</STYLE>
```

34. Обзор существующих CSS-фреймворков.

CSS framework дает веб-разработчикам базовую структуру, которая включает сетку, интерактивные шаблоны пользовательского интерфейса, веб-типографику, всплывающие подсказки, кнопки, элементы форм, значки. Эта структура помогает веб-разработчикам быстро и эффективно приступить к разработке веб-сайта или веб-приложений.

Это означает, что разработчики могут освободиться от необходимости начинать все с нуля. CSS framework создаст для них прочную основу. Кроме того, разработчики также могут повторно использовать код во всех проектах, над которыми они работают.

Bootstrap

- Bootstrap - один из самых популярных фреймворков CSS.
- Bootstrap построена с использованием SASS, а это означает, что Bootstrap теперь поддерживается как LESS, так и SASS.
- Достоинства:

1) Мощный адаптивный дизайн

- Bootstrap обеспечивает адаптивный дизайн с помощью системы сеток. Его легко использовать, и вы можете быстро создать адаптивную сетку, которая будет хорошо работать во всех браузерах. Ваш дизайн будет отлично смотреться на всех экранах и разрешениях.

2) Встроенные библиотеки ресурсов

- Bootstrap предоставляет большие библиотеки для интерфейсных разработчиков, например макеты веб-сайтов, шаблоны веб-сайтов, темы Bootstrap, панели администратора и огромную коллекцию компонентов пользовательского интерфейса.
- Компоненты включают кнопки, формы, карточки, индикаторы выполнения, предупреждения. Это предварительно созданные компоненты, которые могут сэкономить разработчикам продуктов много времени.

3) Низкая кривая обучения

- Фреймворк Bootstrap хорош для новичков в сети. Используя этот инструмент, вы можете присоединиться к области фронтенд-разработки. Существует множество полезной документации и руководств, на которые вы можете положиться, когда у вас возникнут вопросы.

4) Быстрое создание прототипов

Использование готовых компонентов - один из самых быстрых способов имитировать или прототипировать решение. Благодаря переменным и миксинам, адаптивной системе сеток, богатым компонентам и многим другим мощным инструментам вы можете легко создавать прототипы.

Дополнительные возможности Bootstrap :

- Использует Flexbox

- Хорошая документация
- Включает компоненты HTML и JavaScript

Sass

Sass – это своего рода расширение, созданное для упрощения каскадных таблиц стилей (CSS). Все те, кто начинает заниматься разработкой и администрированием сайтов, первым делом сталкиваются с такими понятиями, как HTML и CSS, и только спустя время, получив опыт в создании таблиц стилей, начинают понимать, чем же так хорош и удобен язык Sass.

Sass (или Syntactically Awesome Stylesheets) – это скриптовый метаязык (т.е. язык, описывающий другой язык), разработанный для упрощения файлов CSS. Этот модуль входит в Haml (HTML abstraction markup language), который используется для упрощения HTML.

Sass компилируется в обычные CSS-стили.

- Выпуск Sass состоялся в ноябре 2006 года.
- В зависимости от выбранного синтаксиса Sass-файлы могут иметь расширения:
- .sass - известный как синтаксис с отступами, через которые реализованы вложенные элементы; фигурных скобок нет;
- .scss - Sassy CSS, где используются фигурные скобки.
- Sass унаследовал Ruby-подобный синтаксис, который значительно отличается от CSS. Поэтому в 2010 году был презентован синтаксис SCSS, который приблизил синтаксис Sass к CSS и сделал его более совместимым.
- Каждый из этих синтаксисов имеет свои особенности и преимущества.
- Этот синтаксис короче, в нем отсутствуют скобки и точки с запятой, поэтому набирать его проще.
- Отступы имеют логическое значение, поэтому крайне важно следить за ними – неправильный отступ может сломать таблицу стилей. Отступ у блока селектора определяет вложенный селектор. Если случайно сместить элемент вправо, он внезапно может оказаться дочерним элементом другого элемента, что сильно изменит результат.
- Такой синтаксис определенно понравится тем, кто в разработке использует Ruby или Python.

Преимущества

- **Использование переменных**

Sass дает возможность назначать переменные. Их доступность определяет уровень вложенности селекторов. Если они определяются вне вложенного селектора, то доступны глобально.

Переменные удобно использовать, если одно значение применяется несколько раз. В этом случае при помощи переменной все необходимые значения можно задать в начале кода и далее

просто ссылаться на них. И если будет необходимо изменить значение переменной, то оно изменится и во всех остальных местах, где она предоставлена.

Вложенные правила

Sass дает возможность вкладывать правила CSS друг в друга. Благодаря этому становится намного проще редактировать стили.

Вложенные правила нужны не только для минимизации кода, но и для структурирования кода (наподобие HTML).

Использование дополнений

Дополнения позволяют следовать прекрасному правилу DRY: Don't Repeat Yourself. Вместо того чтобы копировать и множить куски одинакового кода, Sass предлагает сохранить его в переменную, а затем использовать там, где это необходимо. При компиляции переменная будет преобразована в нужную часть кода.

Наследование

Sass дает возможность создать правило, а затем использовать его внутри другого. При этом все свойства класса будут переданы наследуемому элементу.

Flexbox

Flexbox (сокращение от *flexible box* - "гибкий / резиновый блок") — это современный нативный метод верстки страниц сайта с помощью каскадных таблиц стилей. **Flexbox** был введен в составе отдельного модуля - CSS Flexible Box Layout Module, этот модуль (спецификация) описывает модель CSS, оптимизированную для дизайна пользовательского интерфейса и позволяет быстро решать такие важные задачи при построении макета как:

- вертикальное выравнивание блока внутри родительского элемента.
- размещение колонок макета одинаковой высоты независимо от наполнения их содержимым.
- расположение дочерних элементов контейнера в необходимом нам направлении, распределяя при этом между собой доступную ширину, или высоту, независимо от её доступного количества, не вызывая при этом переполнения родительского элемента.
- изменять порядок элементов независимо от порядка их описания в объектной модели документа *DOM*.

Foundation

Foundation и Bootstrap - это широко используемые CSS-фреймворки. Но Foundation - это гораздо более сложная структура. Он очень гибкий и легко настраиваемый.

Это полезный инструмент для создания адаптивных веб-сайтов и веб-приложений, особенно для предприятий. Facebook, eBay, Mozilla, Adobe, HP, Cisco и Disney используют Foundation в своих продуктах.

Достоинства

1) Создайте адаптивный дизайн

Подобно Bootstrap, Foundation также представляет собой очень гибкую интерфейсную среду, которая помогает веб-разработчикам создавать адаптивные веб-сайты, приложения и электронные письма, которые отлично смотрятся на любом устройстве.

2) Мощная структура электронной почты

Помимо веб-сайтов и приложений, Foundation также можно использовать для создания великолепных адаптивных электронных писем в формате HTML. Вы можете создавать электронные письма в формате HTML, используя Foundation для электронных писем. Нет необходимости использовать сложную разметку таблиц или другие вещи. Это отличный помощник для компаний электронного маркетинга.

3) Поддержка обучающих онлайн-семинаров

Zurb (компания, которая разработала Foundation) открыла онлайн-семинары и профессиональные консультации, чтобы научить вас и вашу команду ценным навыкам. Но обучение платное.

4) Легко настроить

Foundation намного гибче, чем Bootstrap. Интерфейсный разработчик имеет полный контроль над пользовательскими интерфейсами. Однако из-за этого новичкам может быть сложно начать с Foundation.

Дополнительные возможности Foundation:

- Вертикальный макет временной шкалы
- Адаптивные HTML-шаблоны и компоненты пользовательского интерфейса
- Полезные инструменты, которые могут решить множество интерфейсных задач

Модульная структура Foundation основана на руководстве по языку стиля, которое агентство веб-дизайна ZURB первоначально использовало только для проектов клиентов. Позже ZURB начал комбинировать и публиковать многочисленные компоненты HTML, CSS и JavaScript в среде с открытым исходным кодом. Центральная часть состоит из **гибкой сетки**, разделенной на двенадцать столбцов, что позволяет настраивать гибкую компоновку, которая автоматически адаптируется к различным размерам дисплея и разрешениям конечных устройств. Рядом с сеткой система также предлагает следующие модели:

- Ползунок
- Кнопки
- Книгопечатание
- Строки меню и различные меню
- Медиаконтейнеры
- Интегрированные классы «float» и «visibility»

Особенности Foundation

- Фреймворк состоит из CSS-файлов (сгенерированных из SASS-файлов, также доступных для скачивания) и нескольких плагинов JQuery.
- Базовые загрузочные файлы не включают в себя HTML-код (за исключением очень простой демо страницы), вы получаете возможность написать все это сами.
- Разметка и классы очень просты, в том числе и для настройки макета под размер экрана.
- Foundation содержит в себе все компоненты, необходимые для быстрого прототипирования, которое является основной целью использования CSS-фреймворка.
- Возможность быстрого создания сайтов выступает наивысшим приоритетом для фронт-энд разработчиков, ограниченных жесткими сроками сдачи проектов.
- Foundation был разработан для того, чтобы позволить вам быстро собирать прототипы и формировать код для современных сайтов и приложений, которые будут прекрасно смотреться на любых устройствах.

Возможности Foundation

- Создание веб-страниц с перестановкой блоков в зависимости от размера экрана пользователя и способа управления
- Простая и гибкая настройка визуального представления для разных устройств
- Использование графических акселераторов для быстрой анимации
- Готовый современный дизайн веб-форм
- Возможность использования SASS для кастомизации css кода foundation под себя

Pure

Pure – это набор CSS-модулей, разработанных компанией Yahoo, которые вы можете использовать как основу в любом веб-проекте.

В середине 2013 года Yahoo сделала Pure.CSS общедоступным, базовую структуру для разработки передних концов для Интернета, которая по праву не только считается хорошей альтернативой Bootstrap, но также может использоваться в сочетании с Twitter. Pure. CSS основан на SMACSS, акрониме «Масштабируемая и модульная архитектура для CSS», архитектура для CSS, которая заставляет повторяющиеся элементы, такие как таблицы, кнопки или контактные формы, отделяться от основного дизайна (шрифт, макет, и т. д.) и что они отвечают на их соглашения.

Фреймворк интерфейса, который может быть загружен как ответный вариант, так и как невосприимчивый вариант, содержит следующие разделы, которые в сжатой или заархивированной форме имеют **размер всего 4 КБ** (выдержки становятся 16 КБ):

- **Base** (base-min.css): базовая структура, включая регулирование;
- **Сетки** (grid-responsive-min.css): гибкая и гибкая система сетки жидкостей;
- **Формы** (forms-min.css / forms-nr-min.css): контактная форма;

- **Кнопки** (buttons-min.css): несколько кнопок;
- **Таблицы** (tables-min.css): таблицы;
- **Меню** (меню-min.css / menus-nr-min.css): меню.

Особенности

Pure.css является очень маленьким, всего 4.5KB minified + gzipped.

Pure построен на Normalize.css, который стандартизирует производительность фреймворка в браузерах.

Сайт Pure построен на собственном фреймворке, поэтому его минималистский, хорошо разработанный код приводит к простому ориентируемому и удобному руководству.

Вы можете добавить Pure на свою страницу через бесплатный CDN Yahoo. Просто добавьте следующий элемент `<link>` в `<head>` вашей страницы, перед таблицами стилей вашего проекта.

На странице Макетов Pure предлагает выборку загрузок для разных страниц примеров для общих приложений. Вы можете просматривать и загружать любые из них, с которыми вы хотите поэкспериментировать или добавить в приложение. Они включают:

- Блог
- Эл. адрес
- Фотогалерея
- Целевая страница
- Ценовая галерея
- Отзывчивое боковое меню
- Отзывчивое горизонтально-вертикальное меню
- Отзывчивое горизонтально-прокручиваемое меню

Сетки — ключевой аспект макетов веб-сайта, который Pure делает довольно простым.

Pure поддерживает выравнивание полей ввода в формах и создание специализированных полей, таких как:

- Встроенные формы
- Сложенные формы
- Двухколонные формы
- Много-колонные формы (*показано выше*)
- Сгруппированные инпуты
- Обязательные инпуты
- Отключенные инпуты

- Инпуты только для чтения
- Закругленные инпуты
- Флажки и радио

Возможности

- настраиваемая, responsive сетка
- встроенные вертикальные и горизонтальные меню, включая выпадающие меню
- кнопки, которые работают с элементами `<a>` и `<button>`
- гибкие выравнивания форм
- общие стили таблиц
- чистый, минималистский вид, который можно легко расширить

Преимущества	Недостатки
Минималистский дизайн	Мало шаблонов
Отличная совместимость с браузерами	Отсутствует фрагмент кода JavaScript
Архитектура SMACSS	Поддержка Less / Sass отсутствует

Bulma

- Bulma - это бесплатная CSS-структура с открытым исходным кодом, основанная на модели макета Flexbox. Это легкий, отзывчивый, чистый CSS и ориентированный на мобильные устройства.
- Все эти функции сделали Bulma одним из самых популярных фреймворков CSS наряду с Bootstrap и Foundation. У Bulma более 150 000 пользователей.

Достоинства

1) Читаемые и памятные названия классов

Bulma предоставляет разработчикам удобочитаемые имена классов CSS и готовые компоненты для создания мобильных интерфейсов. Распознать и запомнить имена классов CSS невероятно легко, потому что все они названы логически.

2) Чистый CSS, без JavaScript

Булма создана на чистом CSS. Всякий раз, когда вы используете фреймворк, вам нужен только один файл .css, а .js не требуется. Кроме того, разработчики могут легко добавить индивидуальный вид для всех компонентов с помощью классов-модификаторов и переменных.

3) Сообщество

У Булмы большое сообщество. Их поклонники могут общаться друг с другом, задавать вопросы и получать ответы.

4) Легко учиться

Низкая кривая обучения - еще одно преимущество Bulma. Это отличный фреймворк для новичков.

Дополнительные возможности Bulma:

- На основе Flexbox
- Создан на Sass
- Легко учиться, легко использовать

Единственный минус в том, что в нем нет JS. «Из-за этого функциональность некоторых компонентов такие как открытие модальных окон приходится реализовывать самостоятельно»

Semantic UI

В 2013 году программист Джек Лукич выпустил свое рамочное решение для разработки интерфейса под названием Semantic UI. Центральным моментом этой коллекции кода является намерение облегчить написание HTML-кода благодаря простым и интуитивным соглашениям для пользователя. Для этого Semantic UI (семантический пользовательский интерфейс) содержит более 3000 семантических классов CSS, которые легко применять и которые предназначены для оптимизации процесса разработки.

Семантический пользовательский интерфейс - это гибкая интерфейсная среда, использующая удобный для человека HTML. Вы можете создавать красивые, адаптивные макеты с более чем 3000 тематических переменных и 50+ компонентами пользовательского интерфейса.

Он также интегрирован со многими сторонними библиотеками, включая React, Angular, Meteor, Ember и многими другими фреймворками. Все это помогает вам организовать слой пользовательского интерфейса вместе с логикой приложения.

Отдельные компоненты упорядочены в следующих шести полях:

- **Globals** : определения стиля, основанные на Normalize.css; основы типографии и дизайна;
- **Элементы** : общие элементы переднего конца, такие как кнопки, значки, контейнеры и т. Д. ;
- **Коллекции** : структурное содержимое, такое как сетка, меню, таблицы или контактные формы;
- **Просмотры** : интерактивные элементы, такие как поля комментариев, доска объявлений или рекламные баннеры;
- **Модули** : виджетов, таких как раскрывающееся меню, всплывающее окно или флажок;
- **Поведение** : интерфейсы для программирования JavaScript.

Для новичков и пользователей, которые приходят из других систем, Semantic UI может показаться странным и, безусловно, **потребуется много времени, чтобы войти в динамику этой структуры**. В конце концов, это обязательство окупиться, потому что HTML вашего веб-интерфейса будет более интуитивно понятен, чем другие платформы, такие как Bootstrap.

Преимущества	Недостатки
Более 3000 семантических классов CSS	Очень сложный
Поддержка Sass и Less	Большая часть компонентов CSS работает только с JavaScript
Отличные возможности интеграции (React, Ember, Meteor, PHP-Paketmanager, Gulp)	

UI Kit

UI Kit - это легкий фреймворк для дизайна CSS и веб-интерфейса, который предлагает почти все функции других фреймворков.

Вы можете создавать простые, понятные и модульные веб-интерфейсы с набором значков SVG, множеством компонентов, отзывчивостью, унифицированными стилями и параметрами настройки. Кроме того, вы также можете разрабатывать сложные макеты на основе Flexbox с помощью UI Kit, используя простой HTML.

Основные компоненты делятся на следующие категории:

- **Значения по умолчанию** : основа для нормализации HTML-элементов, через которые реализована совместимость между браузерами и некоторые основные принципы стиля;
- **Макет** : инструменты для создания внешнего интерфейса, такие как: сетка, поле содержимого или полезные классы CSS для маркировки повторяющегося содержимого;
- **Навигация** : все элементы, которые поддерживают посетителя при просмотре веб-интерфейса; среди них есть формы для разбивки на страницы (нумерация страниц), а также классические навигационные панели;
- **Элементы** : стиль для блоков контента, которые закрыты сами по себе, такие как таблицы, списки, формы контактов;
- **Common** : компоненты, которые обычно используются в содержимом, например кнопки, значки, значки или анимации;
- **JavaScript** : в основном модули, составленные из JavaScript для реализации интерактивных элементов.

Что отличает наборы пользовательского интерфейса от других фреймворков CSS?

1) Минимализм

Наборы пользовательского интерфейса могут помочь веб-разработчикам создавать понятные и современные интерфейсы. Он предлагает мощные функции, но когда дело доходит до дизайна, он становится очень чистым.

2) Полезные компоненты пользовательского интерфейса

В комплекты пользовательского интерфейса входят предварительно созданные компоненты, такие как Accordion, Alert, Drop, Iconnav, анимация, Padding и т. Д. Каждый компонент показывает шаблон использования, параметры и методы компонентов.

Другие особенности комплектов пользовательского интерфейса:

- Совместим с Sass.
- Включает JavaScript
- Работает практически в любом современном браузере.

Преимущества	Недостатки
Дополнительные компоненты для сложных поверхностей	Немного известно
Поддержка Sass и Less	---
Настройка тем	---

Materialize

Materialize - это основа для CSS, основанная на принципах Material Design, введенная в 2015 году компанией Google и используемая ею в большинстве своих приложений. Концепция дизайна состоит из идеи поверхностей, похожих на карты, которые на графическом уровне спроектированы в основном минималистским образом: они следуют эстетической линии так называемого «плоского дизайна», но при этом поддерживают интерес благодаря многочисленным анимациям и теням. Созданные таким образом эффекты глубины помогают пользователю веб-интерфейса легко обнаруживать информацию и важные элементы взаимодействия. Интерфейс пользовательского интерфейса с лицензией MIT был разработан Альвином Вангом, Аланом Чангом, Алексом Марком и Кевином Луи, четырьмя студентами Университета Карнеги-Меллона в Пенсильвании.

Альтернатива Bootstrap, которая, подобно структуре Twitter, имеет систему с двенадцатью столбцами, содержит несколько компонентов CSS и JavaScript, более 700 официальных символов Material Design в иконочных шрифтах, а также Roboto, стандартный шрифт концепции дизайна от Google. Наряду с обычными регулярными и уменьшенными файлами CSS с помощью Materialize, как и с Bootstrap, вы можете использовать исходные файлы SCSS, написанные в Sass, которые облегчают вашу веб-поверхность.

Независимо от того, как выбор падает, у вас будет в вашем распоряжении 30 элементов:

- CSS : как и в Bootstrap и других структурах пользовательского интерфейса, элементарная функция CSS представлена гибкой сеткой, которая обеспечивает основу для типа веб-поверхности, которая работает на всех устройствах.
- Компоненты : это компоненты инфраструктуры интерфейса, которые вам понадобятся для создания элементов навигации и интерактивных полей. В дополнение к типичным компонентам, таким как коды для вставки страниц, формы контактов, навигационные панели, значки, вы также найдете модули, имеющие фундаментальное значение для реализации концепций Material Design. Например, к ним относятся «карточки», типичные объектные

ящики Google для презентации контента или символические «фишки», которые представляют теги или контакты.

- **JavaScript** : Что касается приложений JavaScript, то решение, несомненно, является одним из лучших альтернатив Bootstrap. Если вы хотите, чтобы изображение появлялось в режиме прокрутки, добавляйте интерактивные диалоги, вставляйте раскрывающиеся элементы или восстанавливайте поверхность веб-сайта с помощью так называемого эффекта «параллакса» (параллакс - это явление, когда объект, кажется, отходит от фонового режима, если вы меняете точку наблюдения), это не имеет значения, потому что у вас всегда есть все фрагменты JavaScript, которые вы хотите в своем распоряжении. Таким образом, вы будете лучше всего приспособлены для дизайна поверхности веб-сайта, которая характеризуется высокой удобством использования как на мобильных устройствах, так и на стационарных компьютерах.

Преимущества	Недостатки
Основан на дизайне Google	Доступно мало шаблонов и расширений
Большое разнообразие компонентов	Поддерживает только последние версии браузера
Доступна версия Sass	Очень строгие правила проектирования

Milligram

Миллиграмма - одна из самых легких фреймворков CSS, которая может помочь вам создавать быстрые и чистые веб-сайты. Вес решения - 2 КБ (в сжатом виде).

Но, несмотря на небольшой размер, Milligram поставляется с полным набором инструментов для веб-разработки. Разработчики также могут использовать все функции, предлагаемые спецификацией CSS3 с Milligram. Они могут добавить миллиграмму больше мощности и сделать его одним из трех лучших доступных фреймворков.

Дополнительные возможности Миллиграмма :

- На основе сетки Flexbox
- Темы супер дизайна

Skeleton

Skeleton - это минимальная адаптивная CSS-структура, содержащая всего 400 строк исходного кода. Несмотря на относительно небольшой размер, он предлагает множество опций (сетки, типографика, кнопки, формы, списки, таблицы, код и т. д.), Которые позволяют создавать сложные веб-сайты.

Когда лучше всего использовать каркас Skeleton?

Если вы занимаетесь небольшим проектом или вам не нужно использовать все возможности более крупных фреймворков, то Skeleton - ваш лучший выбор. Skeleton имеет ограниченное количество стандартных элементов HTML, но этого достаточно для начала.

Дополнительные возможности скелета:

- Создан для мобильных устройств
- Легко учить

Spectre

Spectre.css - отличный фреймворк, который может помочь вам ускорить и расширить возможности разработки с элегантно оформленными элементами, красивой типографикой и готовыми компонентами.

Кроме того, компоненты представляют собой чистый CSS, поэтому для их использования не требуется какой-либо язык JavaScript.

Дополнительные возможности Spectre:

- Макет на основе Flexbox
- Мобильный макет

Picnic CSS

Picnic - легкая библиотека CSS размером менее 10 КБ (в сжатом виде). Он предоставляет вам чистые CSS и интерактивные компоненты, включая сетку, формы, вкладки, всплывающие подсказки и предупреждения. Библиотека поможет вам создать отзывчивый веб-сайт и красивые веб-приложения.

Дополнительные возможности Panic CSS:

- Написано на Sass / SCSS
- Включает переменные и классы
- Красота HTML по умолчанию

Mustard UI

легкий CSS-фреймворк с открытым исходным кодом, специально разработанный для новичков. Вы можете начать с основных строительных блоков, потому что он разделен на модули.

Он также предоставляет компоненты для создания интерфейса. Эти компоненты представляют собой сетки, кнопки, таблицы, формы и карточки на основе flexbox.

Дополнительные возможности Mustard UI:

- Менее 6 КБ в сжатом виде
- Хорошо задокументированы
- Использует Open Sans в качестве шрифта по умолчанию

Dead Simple Grid

Dead Simple Grid - полезный инструмент, содержащий всего 250 байт кода CSS и всего два класса. Его нельзя рассматривать как законченный фреймворк CSS, но он удобен, когда веб-разработчики хотят использовать систему сеток.

Он также поддерживает все основные браузеры, начиная с IE 8, обеспечивая мобильную разметку с одним столбцом для старых браузеров.

Дополнительные возможности Dead Simple Grid:

- Поддерживает бесконечное вложение
- Построен с прогрессивным улучшением
- Ориентация на мобильные устройства

35. *AJAX. Принцип работы технологии AJAX. Форматы передаваемых данных. Достоинства и недостатки.*

AJAX (Асинхронный JavaScript и XML) представляет собой технологию гибкого взаимодействия между клиентом и сервером. Благодаря ее использованию мы можем осуществлять асинхронные запросы к серверу без перезагрузки всей страницы. Правда, в настоящее время все больше вместо формата XML используется формат JSON для взаимодействия между клиентом и сервером.

AJAX использует асинхронную передачу данных. Благодаря этому человек совершает разные действия при «фоновом» обмене информацией с сервером. При этом работает оповещение пользователя обо всех протекающих процессах. Это необходимо, чтобы пользователь не подумал, что на ресурсе возник какой-то сбой или он «завис».

Ответом сервера выступает обычный текст, XML/JSON. В первом случае результат сразу отображается на странице. При получении XML-документа он обычно конвертируется в HTML и выводится на монитор. При получении ответа в формате JSON, пользователь должен использовать полученный код. Потом будет выполняться формирование объекта JavaScript.

Форматы передаваемых данных

Существует несколько распространенных форматов общения с сервером.

Рассмотрим

- HTML
- XML
- JSON

HTML

В самом простом случае - ответом на AJAX-запрос является кусок HTML:

```
<h4>Готово</h4>
```

```
<p class="notification">Новость опубликована</p>
```

Этот кусок можно показать пользователю, записав внутрь тага:

```
domObject.innerHTML = data
```

XML

Сервер возвращает XML document типа:

```
1    <response>
```

```
2    <status>Готово</status>
```

```
3    <message>Новость опубликована</message>
```

```
4      </response>
```

На клиенте XML может либо анализироваться через javascript, либо преобразовываться XSL-шаблоном типа:

```
1      <xsl:template match="/reports/report">
2
3      <h4><xsl:value-of select="status"/></h4>
4
5      <p class="notification"><xsl:value-of select="message"/></p>
6
7      </xsl:template>
```

Код на клиенте:

```
// ... получить XML-ответ в xmlDoc и подготовить XSLT в xslDoc
domObject.innerHTML = xmlDoc.transformNode(xslDoc)
```

На практике, поддержка XSLT в браузерах очень ограничена и различается в деталях. Заявившись на обработку XML при помощи client-side XSLT, рано или поздно придется все же использовать javascript из-за ограничений XSLT. Также браузером не поддерживается EXSLT.

Плюсом XSLT является быстрота метода, по сравнению с javascript, но это актуально лишь на больших документах.

JSON (JavaScript Object Notation)

Этот формат, как правило, самый оптимальный. Сервер возвращает JavaScript-объект:

```
{ "response":
{ "status": "Готово", "message": "Новость опубликована" }
}
```

Чтобы десериализовать объект, клиент просто пропускает текст через встроенный парсер, добавив скобки

```
var reports = eval( '('+data+')' )
```

Затем HTML может быть сформирован средствами JavaScript, собственной шаблонной системой и т.п.

```
domObject.innerHTML = jSmarty.fetch(reportsTemplate)
```

Достоинства:

- **Снижение нагрузки на сервер.** Благодаря грамотному использованию AJAX можно многократно снизить нагрузку на сервер. Например, можно использовать специальный шаблон и создавать постоянные элементы сайта: меню, логотип и пр. Обновление всей страницы для удовлетворения запросов не требуется. Например, пользователь решает проголосовать на сайте, он выбирает нужный вариант и кликает по этой кнопке. Сразу после этого информация отправляется на сервер, а затем приходит ответ. За все это время страница не обновляется.
- **Ускорение работы сервиса.** Так как подгружается только содержательная часть, пользователь видит результат действий значительно быстрее.
- **Снижение количества трафика.** Количество данных в процессе работы с web-приложениями существенно снижается. Это происходит в результате того, что загружать всю страницу целиком не нужно, достаточно получить набор данных или измененную часть. Потом JavaScript меняет содержимое страницы.
- **Широкий спектр возможностей.** Использование AJAX не ограничивается формами. Так, в процессе регистрации на некоторых сервисах пользователь должен ввести логин – и буквально через секунду на экране высвечивается информация о том, свободен он или занят. Аналогично при введении поискового запроса в браузере после каждого введенного слова или буквы пользователю предлагаются разные варианты, что существенно повышает комфорт работы.

Недостатки:

- **Снижение степени безопасности.** Значительный недостаток AJAX – пробелы в безопасности. Это связано с тем, что каждый пользователь может легко посмотреть исходный код в браузере.
- **Невозможность интеграции с инструментами браузера.** В процессе динамического формирования страниц браузер не может отображать их в истории посещения. Именно поэтому кнопка «Назад» не может переместиться на предыдущий этап работы. Данная проблема может решиться за счет использования специальных скриптов. Кроме всего вышеперечисленного, отсутствует возможность установки закладки на нужный материал.
- **Необходимость держать JavaScript подключенным.** Страницы web-сайтов, которые были созданы по технологии AJAX, не могут корректно работать при отключенном JavaScript. На них не получится разместить закладки.
- **Проблемы в индексации контента.** Нередко содержание, загружаемое динамическим способом, недоступно для поисковых роботов. Поэтому для некоторых частей контента рекомендуется использовать динамическую загрузку. В результате этого негативное влияние AJAX на поисковое продвижение можно уменьшить.
- **Невозможность установки числа обращений.** Механизм динамической загрузки контента существенно искажает статистические данные. Это связано с тем, что при перемещении пользователя по разным страницам их перезагрузка не выполняется, поэтому счетчик не регистрирует переходы. Из-за такого искусственного занижения количества просмотров крупные проекты теряют часть своего дохода.

36. CMS. Основные сведения. Описание распространённых функций CMS. Достоинства и недостатки.

Система управления содержимым (*Content management system, CMS, система управления контентом*) — информационная система или компьютерная программа, используемая для обеспечения и организации совместного процесса создания, редактирования и управления содержимым, иначе — контентом.

CMS обычно состоит из двух основных компонентов: приложения для управления контентом (CMA) в качестве внешнего пользовательского интерфейса, позволяющего пользователю добавлять, изменять и удалять контент с веб-сайта без вмешательства веб-мастера, и приложение доставки контента (CDA), которое компилирует контент и обновляет веб-сайт.

Основные функции CMS:

- предоставление инструментов для создания содержимого, организация совместной работы над содержимым;
- управление содержимым: хранение, контроль версий, соблюдение режима доступа, управление потоком документов;
- публикация содержимого;
- представление информации в виде, удобном для навигации, поиска.

В системе управления содержимым могут находиться самые различные данные: документы, фильмы, фотографии, номера телефонов, научные данные и так далее. Такая система часто используется для хранения, управления, пересмотра и публикации документации. Контроль версий является одной из важных возможностей, когда содержимое изменяется группой лиц.

В общем случае системы управления содержимым делятся на системы управления корпоративным контентом (англ. *Enterprise Content Management System*) — для работы с содержимым внутри какой-либо организации и системы управления веб-содержимым (англ. *Web Content Management System*) для поддержки работы веб-сайта.

Реализации CMS

WordPress.org

WordPress долгое время была CMS номер один. Популярность WordPress связана с его невероятной универсальностью и простотой использования. Существуют буквально тысячи тем и плагинов, которые вы можете выбрать, чтобы настроить свой сайт в соответствии с вашими потребностями.

Drupal

Drupal — это профессиональная CMS, используемая главным образом в корпоративном мире. NASA, Tesla, Sony Music, Nokia и многие другие известные компании выбрали Drupal в качестве своей системы управления контентом. Сайты Drupal имеют отличное время загрузки страницы и надёжную защиту. Drupal поставляется со встроенными модулями, которые работают и

легко интегрируются со многими популярными инструментами аналитики, маркетинга и электронной коммерции.

PrestaShop

PrestaShop — это более популярная система управления контентом для онлайн-магазинов. В настоящее время его используют около 270 000 продавцов по всему миру. PrestaShop является отличным решением для малого и среднего бизнеса электронной коммерции. Он позволяет быстро настроить интернет-магазин, настроить его дизайн и загружать товары. PrestaShop уделяет большое внимание соблюдению закона. Вы можете настроить налоги в зависимости от местоположения, сделать процесс оформления заказа в соответствии с местным законодательством и многое другое.

Magento

Magento — популярная платформа электронной коммерции, используемая такими компаниями, как Samsung, Nike и Ford. Позволяет построить сложный интернет-магазин со всеми видами функций электронной коммерции. Вы можете создавать профессиональные страницы о продуктах и доставке, управлять остатками, заказами, доставкой и многим другим. С помощью Magento вы также можете добавлять купоны, целевые страницы, кампании по продаже и перекрёстной продаже в свой интернет-магазин.

Достоинства и недостатки.

Достоинства:

- можно создать сайт самостоятельно и за короткий промежуток времени;
- не нужно разбираться в дизайне и программировании;
- разработка сайта будет стоить дешевле;
- удобно управлять содержимым сайта.

Недостатки:

- нужно следить за обновлениями и совместимостью новых версий с дополнениями;
- производительность обычно снижается, если на сайте много дополнений;
- не весь функционал получится реализовать;
- не подходят для нетипичных задач.

37. Идентификация, авторизация, аутентификация в Web.

Главная задача – предоставление доступа тем, кто есть в базе данных и отказ в доступе тем, кого нет в ней нет.

Идентификация – процедура, в результате выполнения которой для субъекта выявляется его уникальный признак, однозначно определяющий его в информационной системе.

Аутентификация – процедура проверки подлинности, например, проверка подлинности пользователя путем сравнения введенного им пароля с паролем, сохраненным в базе данных.

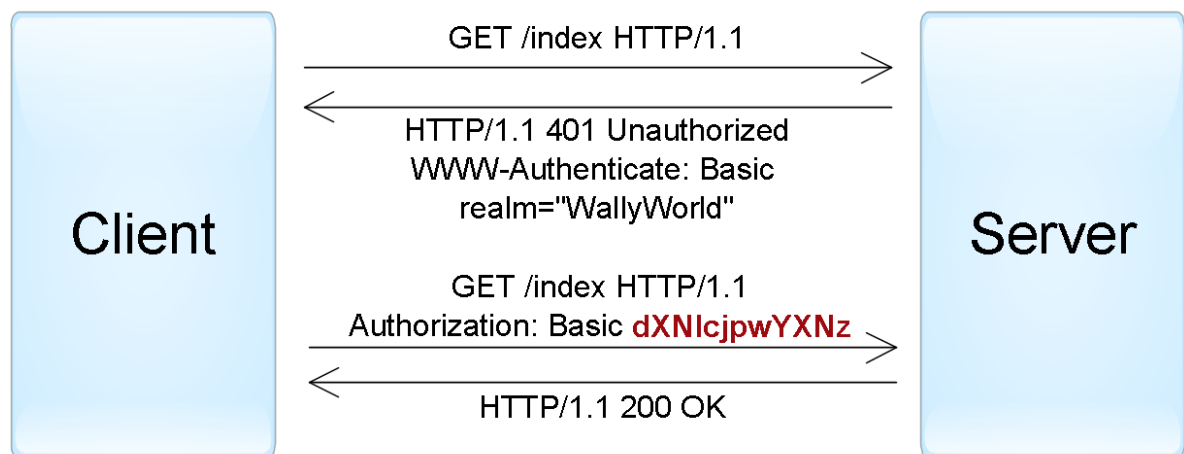
Авторизация – предоставление определенному лицу прав на выполнение определенных действий.

Способы аутентификации:

- Аутентификация по паролю
- Аутентификация по сертификатам
- Аутентификация по одноразовым паролям
- Аутентификация по ключам доступа
- Аутентификация по токенам

Аутентификация по паролю. Этот метод основывается на том, что пользователь должен предоставить username и password для успешной идентификации и аутентификации в системе.

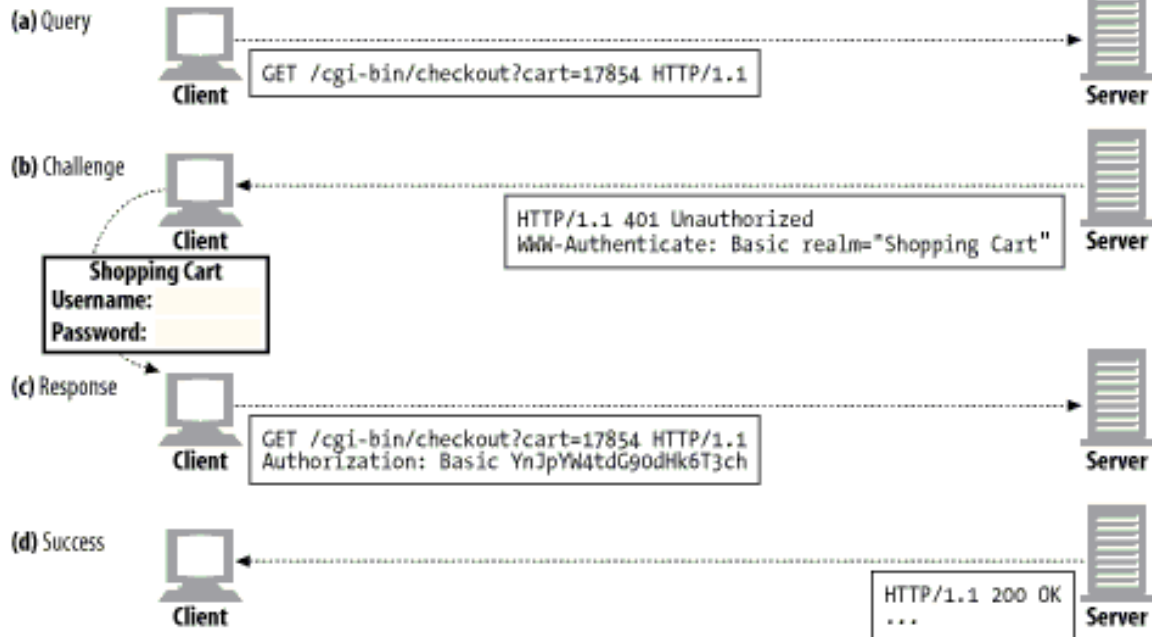
1.1 HTTP authentication



1.1.1 Basic

Наиболее простая схема, при которой username и password пользователя передаются в заголовке Authorization в незашифрованном виде (base64-encoded).

Basic authentication



1.1.2 Digest

Challenge-response-схема, при которой сервер посылает уникальное значение nonce, а браузер передает MD5 хэш пароля пользователя, вычисленный с использованием указанного nonce.

Digest authentication

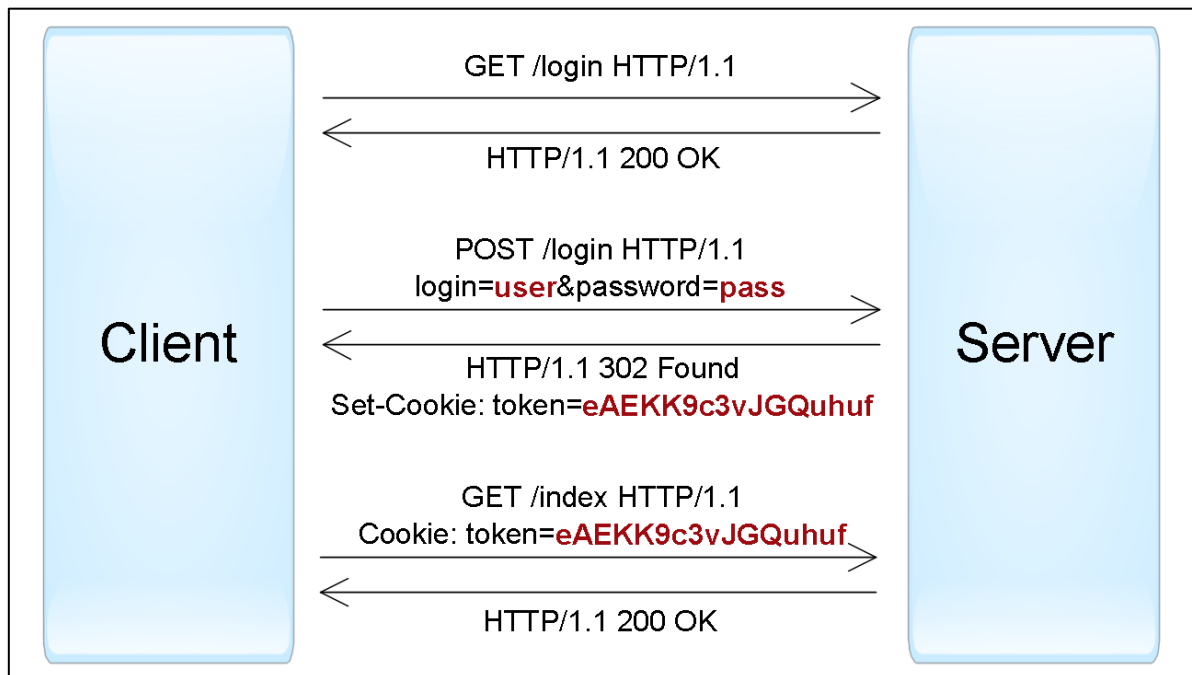


1.1.3 NTLM (известная как Windows authentication) — также основана на challenge-response подходе, при котором пароль не передается в чистом виде. Эта схема не является стандартом HTTP, но поддерживается большинством браузеров и веб-серверов.

1.1.4 Negotiate — еще одна схема из семейства Windows authentication, которая позволяет клиенту выбрать между NTLM и Kerberos аутентификацией. Kerberos — более безопасный протокол, основанный на принципе Single Sign-On.

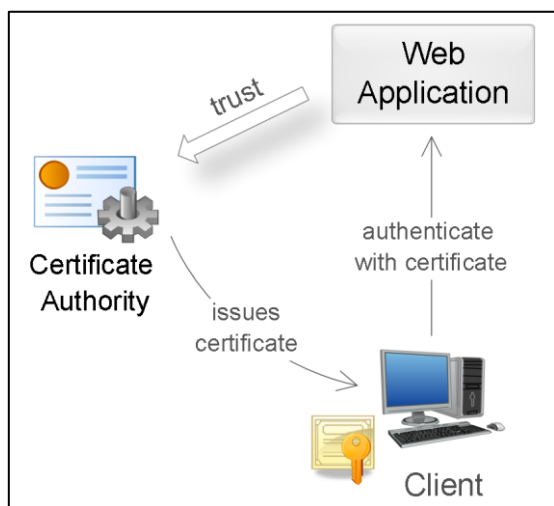
1.2 Аутентификация с помощью форм

В веб-приложение включается HTML-форма, в которую пользователь должен ввести свои username/password и отправить их на сервер через HTTP POST для аутентификации.



2. Аутентификация по сертификатам

Сертификат представляет собой набор атрибутов, идентифицирующих владельца, подписанный certificate authority (CA). CA выступает в роли посредника, который гарантирует подлинность сертификатов.



Использование сертификатов для аутентификации — куда более надежный способ, чем аутентификация посредством паролей.

3. Аутентификация по одноразовым паролям

Аутентификация по одноразовым паролям обычно применяется дополнительно к аутентификации по паролям для реализации *two-factor authentication* (2FA). В этой концепции пользователю необходимо предоставить данные двух типов для входа в систему: что-то, что он знает (например, пароль), и что-то, чем он.

Другой популярный сценарий использования одноразовых паролей — дополнительная аутентификация пользователя во время выполнения важных действий: перевод денег, изменение настроек и т. п.

4. Аутентификация по ключам доступа

Этот способ чаще всего используется для аутентификации устройств, сервисов или других приложений при обращении к веб-сервисам. Здесь в качестве секрета применяются ключи доступа (*access key*, *API key*) — длинные уникальные строки, содержащие произвольный набор символов, по сути заменяющие собой комбинацию username/password.



5. Аутентификация по токенам

Такой способ аутентификации чаще всего применяется при построении распределенных систем Single Sign-On (SSO), где одно приложение (service provider) делегирует функцию аутентификации пользователей другому приложению (identity provider).

5.1 Simple Web Token (SWT)

Наиболее простой формат, представляющий собой набор произвольных пар имя/значение в формате кодирования HTML form. Токен подписывается с помощью симметричного ключа, таким образом оба IP- и SP-приложения должны иметь этот ключ для возможности создания/проверки токена.

Пример SWT токена (после декодирования).

Issuer=http://auth.myservice.com&

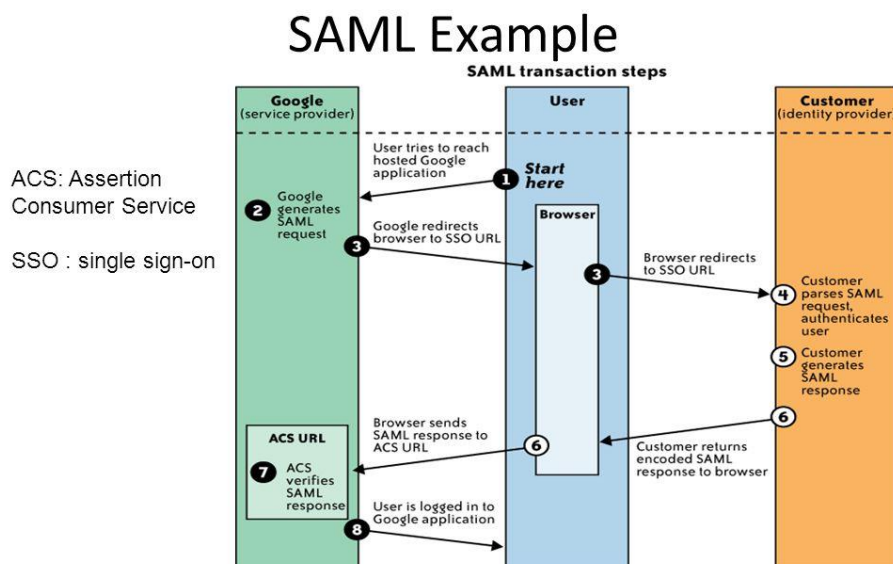
Audience=http://myservice.com& ExpiresOn=1435937883&

UserName=John Smith& UserRole=Admin&

HMACSHA256=KOUQRPSpy64rvT2KnYyQKtFFXUIggnesSpE7ADA4o9w

5.2 Security Assertion Markup Language (SAML)

Определяет токены (SAML assertions) в XML-формате, включающем информацию об эмитенте, о субъекте, необходимые условия для проверки токена, набор дополнительных утверждений (statements) о пользователе. Подпись SAML-токенов осуществляется при помощи ассиметричной криптографии.



5.3 JSON Web Token (JWT)

Содержит три блока, разделенных точками: заголовок, набор полей и подпись. Первые два блока представлены в JSON-формате и дополнительно закодированы в формат base64. Подпись может генерироваться при помощи и симметричных алгоритмов шифрования, и ассиметричных.