

Web exam

Билет 1

1. Технологии толстого клиента. Варианты организации персистентного хранилища данных на клиенте (localStorage, IndexedDB).

Толстый клиент, rich client архитектуре клиент-сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) полную функциональность и независимость от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства

- Толстый клиент обладает широкой функциональностью в отличие от тонкого.
- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Высокое быстродействие (зависит от аппаратных средств клиента)

Недостатки

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.
- Наличие бизнес-логики

localStorage: (<https://learn.javascript.ru/localstorage>)

Объекты веб-хранилища localStorage и sessionStorage позволяют хранить пары ключ/значение в браузере.

Что в них важно – данные, которые в них записаны, сохраняются после обновления страницы (в случае sessionStorage) и даже после перезапуска браузера (при использовании localStorage). Скоро мы это увидим.

Но ведь у нас уже есть куки. Зачем тогда эти объекты?

- В отличие от куки, объекты веб-хранилища не отправляются на сервер при каждом запросе. Поэтому мы можем хранить гораздо больше данных. Большинство браузеров могут сохранить как минимум 2 мегабайта данных (или больше), и этот размер можно поменять в настройках.
- Ещё одно отличие от куки – сервер не может манипулировать объектами хранилища через HTTP-заголовки. Всё делается при помощи JavaScript.
- Хранилище привязано к источнику (домен/протокол/порт). Это значит, что разные протоколы или поддомены определяют разные объекты хранилища, и они не могут получить доступ к данным друг друга.

Объекты хранилища localStorage и sessionStorage предоставляют одинаковые методы и свойства:

- setItem(key, value) – сохранить пару ключ/значение.
- getItem(key) – получить данные по ключу key.
- removeItem(key) – удалить данные с ключом key.
- clear() – удалить всё.
- key(index) – получить ключ на заданной позиции.
- length – количество элементов в хранилище.

Основные особенности localStorage:

- Этот объект один на все вкладки и окна в рамках источника (один и тот же домен/протокол/порт).
- Данные не имеют срока давности, по которому истекают и удаляются. Сохраняются после перезапуска браузера и даже ОС.

Объекты веб-хранилища localStorage и sessionStorage позволяют хранить пары ключ/значение в браузере.

- key и value должны быть строками.
- Лимит 2 Мб+, зависит от браузера.
- Данные не имеют «времени истечения».
- Данные привязаны к источнику (домен/протокол/порт).

IndexedDB: (<https://learn.javascript.ru/indexeddb>)

IndexedDB – это встроенная база данных, более мощная, чем localStorage.

- Хранилище ключей/значений: доступны несколько типов ключей, а значения могут быть (почти) любыми.
- Поддерживает транзакции для надёжности.
- Поддерживает запросы в диапазоне ключей и индексы.
- Позволяет хранить больше данных, чем localStorage.

Для традиционных клиент-серверных приложений эта мощность обычно чрезмерна. IndexedDB предназначена для оффлайн приложений, можно совмещать с ServiceWorkers и другими технологиями.

Интерфейс для IndexedDB, описанный в спецификации <https://www.w3.org/TR/IndexedDB>, основан на событиях.

У нас может быть множество баз данных с различными именами, но все они существуют в контексте текущего источника (домен/протокол/порт). Разные сайты не могут получить доступ к базам данных друг друга.

IndexedDB имеет встроенный механизм «версионирования схемы», который отсутствует в серверных базах данных.

В отличие от серверных баз данных, IndexedDB работает на стороне клиента, в браузере, и у нас нет прямого доступа к данным. Но когда мы публикуем новую версию нашего приложения, возможно, нам понадобится обновить базу данных.

Чтобы сохранить что-то в IndexedDB, нам нужно хранилище объектов.

Хранилище объектов – это основная концепция IndexedDB. В других базах данных это «таблицы» или «коллекции». Здесь хранятся данные. В базе данных может быть множество хранилищ: одно для пользователей, другое для товаров и так далее.

Несмотря на то, что название – «хранилище объектов», примитивы тоже могут там храниться.

Мы можем хранить почти любое значение, в том числе сложные объекты.

2. Базы данных. Понятие СУБД. Реляционные и нереляционные базы данных.

В вебе существует необходимость масштабирования БД. Масштабирование бывает вертикальным и горизонтальным. Для масштабирования есть репликация и

шардинг. Репликация повышает отказоустойчивость и ускоряет чтение. Шардингирование - раскидываем разные данные по группам и раскидываем по разным машинам.

Проблемы SQL:

- JOIN нельзя масштабировать;
- сложные проверки при выборке и вставке данных;
- семантический разрыв данных в БД и объектов в программе.

Проблемы NoSQL:

- обеспечение целостности - на приложении;
- приходится обходиться без JOIN;
- “после SQL кажется, что это НЕ ДОЛЖНО РАБОТАТЬ”.

База данных - совокупность хранимых постоянно данных, организованных по определённым правилам, предусматривающая общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. БД - самодокументированное собрание интегрированных записей.

Для любой БД определяется модель данных - совокупность структур данных и операций их обработки.

Виды БД: реляционные и нереляционные.

Реляционная модель использует организацию данных в двумерных таблицах с первичными ключами, с помощью которых устанавливаются связи. Любая такая таблица должна обладать следующими свойствами:

1. все элементы однородные (в каждом столбце хранятся данных одного и того же типа);
2. каждый столбец имеет уникальное имя;
3. отсутствуют полностью идентичные строки;
4. порядок следования столбцов и строк не имеет значения.

Реляционные модели строятся на отношениях. Для корректной работы реляционная БД должна характеризоваться:

1. целостностью - БД должна содержать полную и непротиворечивую информацию, необходимую для корректного формирования приложений;
2. независимостью - структура данных должна быть независима от программ, использующих эти данные, чтобы данные можно было изменять без

- корректировки программы;
3. восстановляемостью – возможность восстановления БД после сбоя системы или уничтожения данных;
 4. безопасностью – защита данных от постороннего доступа и модификации;
 5. эффективностью – оптимальное соотношение объёма данных и скорости доступа к ним.

Система управления базами данных (СУБД) – комплекс программных средств, предназначенных для структуризации, хранения и обработки больших объёмов информации в базе данных.

Требования к СУБД:

1. атомарность – СУБД должна гарантировать, что никакая операция в системе не будет зафиксирована частично, что регулируется механизмом отката транзакций;
2. согласованность – завершённая транзакция не должна нарушать согласованность БД (транзакция может быть выполнена только в том случае, если она фиксирует допустимые результаты);
3. изолированность – параллельные выполнения транзакции не должны влиять на ход друг друга;
4. устойчивость – независимо от проблем на нижних уровнях, изменения, зафиксированные успешной транзакцией, должны оставаться сохранными после восстановления работы системы.

NoSQL – ряд подходов, направленных на реализацию моделей БД, имеющих существенные отличия от используемых в традиционных реляционных СУБД. В отличие от реляционных, часто имеют более высокую скорость доступа к данным или распределённую архитектуру, которая позволяет хранить неограниченное количество данных или хранить неструктурированные объекты данных. Типы NoSQL:

1. документно-ориентированные;
2. графовые;
3. поколоночные;
4. “ключ-значение”;
5. иные.

"Ключ-значение" - ассоциативный массив с уникальными ключами. Характеризуется масштабируемостью, не требует никаких схем построения БД, нет никаких связей между значениями. Подобные хранилища используют в тех случаях, когда полностью отсутствуют связи между ячейками. Такие БД применяются в качестве кэшей для объектов. Примеры СУБД: Amazon DynamoDB, Berkeley DB, Redis, MemcacheDB, Riak.

Документо-ориентированная БД представляет собой систему хранения иерархических структур данных, имеющих структуру дерева. Структура дерева начинается с корневого узла и может иметь несколько внутренних и листовых узлов. Листовые узлы содержат конечные данные, которые при добавлении заносятся в индексы базы, благодаря которым можно осуществить быстрый поиск. Такие БД применяются в задачах, где требуется упорядоченное хранение информации, но нет множества связей между данными и не нужно постоянно собирать статистику по ним. Данные в таких системах не требуют определения схемы - документ может состоять из любого количества уникальных полей. Примеры СУБД: MongoDB, CouchDB, Couchbase, MarkLogic, eXist.

Графовые БД применимы в случаях, когда связи между данными имеют такую же ценность, как и сами данные. Графовые БД лучше всего подходят для реализации проектов, предполагающих естественную графовую структуру данных - социальных сетей и для создания семантических паутин. В таких задачах они сильно опережают реляционные БД по производительности, простоте внесения изменений и наглядности представления информации. Кроме того, для работы с достаточно большими графиками используются алгоритмы, предполагающие частичное помещение графа в оперативную память. Примеры СУБД - OrientDB, ArangoDB, Giraph, HyperGraphDB, FlockDB, Neo4j.

3. Варианты организации клиент-серверной архитектуры приложения в веб. Толстый, тонкий и изоморфный клиенты. Определение, принципы.

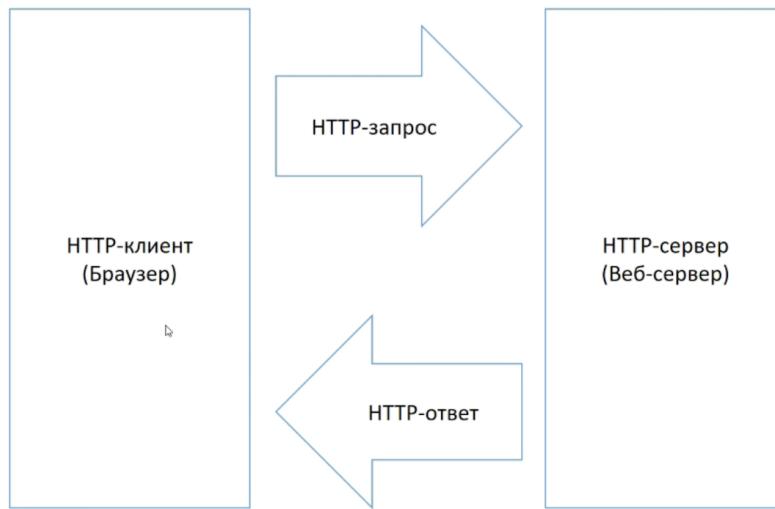
Клиент-сервер - сетевая архитектура, в которой сетевая нагрузка распределена между поставщиками услуг, называемыми серверами и заказчиками услуг, называемыми клиентами.

Классический "клиент-сервер" в WEB:

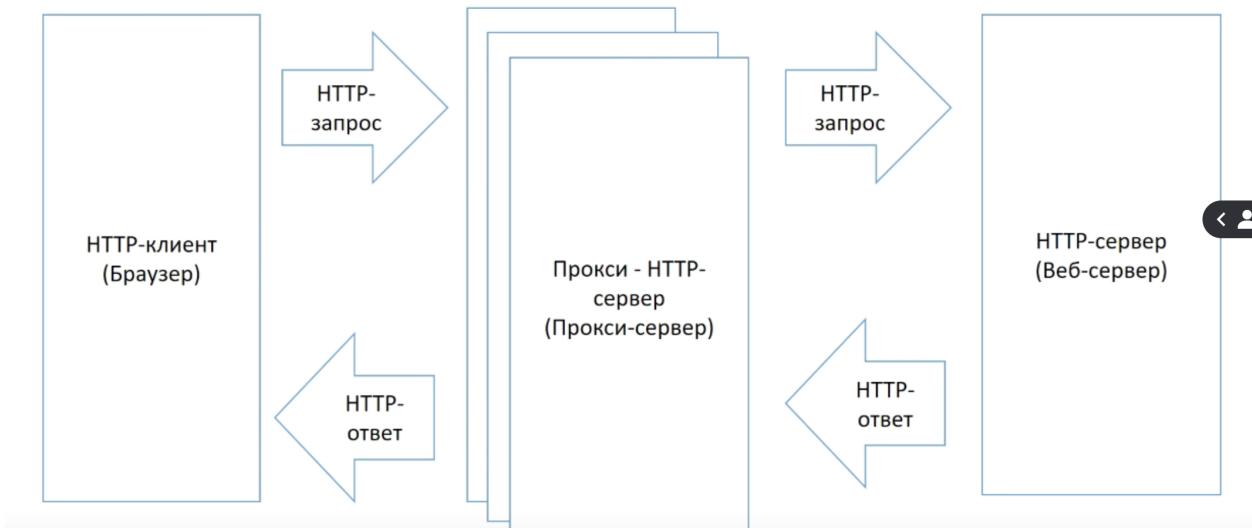
- Взаимодействие по протоколу HTTP/протоколам поверх HTTP
- HTTP работает поверх TCP

- Абстрагирование от физической реализации сети
- Адресация - сокетах (ip-фдрес + порт)
- Доменные имена преобразуются в ip-адрес
- Клиент - всегда инициатор сессии

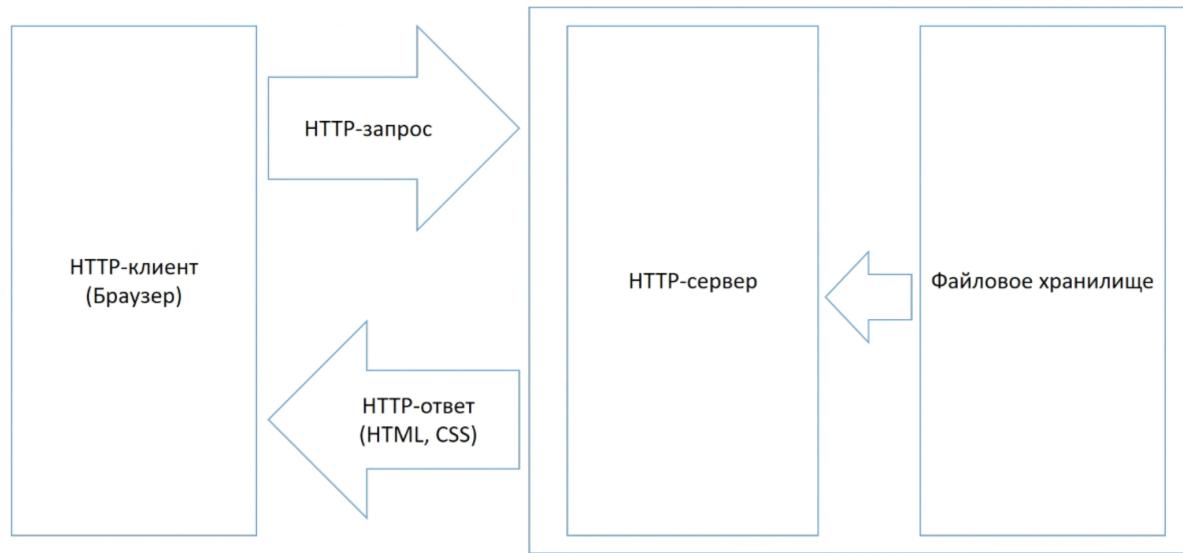
Клиент-серверное взаимодействие в HTTP



Клиент-серверное взаимодействие в HTTP



Клиент-серверная архитектура (статика)



Толстый клиент, rich client архитектуре клиент-сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) полную функциональность и независимость от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Тонкий клиент, thin client в компьютерных технологиях — компьютер или программа-клиент в сетях с клиент-серверной или терминальной архитектурой, где большая часть задач по обработке информации перенесена на сервер и права доступа клиента строго ограничены.

Изоморфный совмещает в себе элементы толстого и тонкого клиента.

Билет 2

1. HTML. Язык разметки гипертекста. История. Структура документа.

1991 Британец Тимоти Джон Бернерс-Ли в Женевском ЦЕРНе изобрел язык гипертекстовой разметки

1993 - HTML 1.2 В этой версии, из её сорока с копейками тегов уже появилось аж 3 тега, которые намекали на какое-то визуальное оформление документа (например,

полужирный курсив). Остальные же теги служили исключительно для логической разметки.

22 сентября 1995 – Версия 2.0 Процесс разработки и утверждения новой версии был очень неспешным, а единственным заметным улучшением новой версии стали: запросы (например, поиск по ключевым словам), формы для передачи данных с компьютера на сервер (например, ввести дату рождения или выбрать один из нескольких вариантов в опроснике)

Март 1995 – начало работы над HTML 3.0 Первый вариант стандарта включал в себя много интересностей: теги для создания таблиц, разметки математических формул, обтекание изображений текстом и др.

14 января 1997 – HTML 3.2 вышла спустя месяц после утверждения CSS, и была уже полностью приспособлена к взаимодействию с таблицами стилей.

18 декабря 1997 – HTML 4.0 версия включала поддержку фреймов, скриптов, общую процедуру внедрения разных объектов. Также в ней были усовершенствованы таблицы и формы, что кроме прочих плюсов обеспечивало большую доступность для людей с физическими недостатками.

24 декабря 1999 – HTML 4.01 В этой версии слегка подправили объекты, формы и изображения, пофиксили баги и в целом создали более стабильную версию, которой пользовались веб-разработчики более 10 лет.

28 октября 2014 – HTML5 Новая версия сделала синтаксис более строгим по сравнению с предыдущей. Улучшилась поддержка мультимедиа-технологий. Появились 28 новых структурных элементов, благодаря которым код стал более понятным. Исключена еще часть устаревших тегов Стало больше внимания уделяться поддержке скриптов, например javascript

HTML-документ заключается в теги <html> и </html>. Между этими тегами располагаются два раздела: раздел заголовка (элемент head) и раздел тела документа (элемент body для простого документа либо элемент frameset, задающий набор кадров). Секция заголовка содержит описание параметров, используемых при отображении документа, но не отражающихся непосредственно в окне браузера. Секция тела документа содержит текст, предназначенный для отображения браузером и элементы, указывающие на способ форматирования текста, определяющие графическое оформление документа, задающие параметры гиперссылок и так далее. Согласно спецификации HTML 4+, первым в документе должен указываться элемент doctype

2. MVC-фреймворки. Понятие фреймворка. Примеры. Типовая структура.

MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Компоненты MVC:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.
2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
3. Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC (Fat Stupid Ugly Controllers).

(MVC — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер. Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.)

ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.

Веб-фреймворки - это наборы функций, объектов, правил и других конструкций кода, предназначенных для решения общих проблем, ускорения разработки и упрощения различных типов задач, стоящих в конкретной области.

1. AngularJS является JavaScript-фреймворком с открытым исходным кодом, разрабатываемым Google. Предназначен для разработки одностраничных приложений....
2. Backbone.js придает структуру веб-приложениям с помощью моделей с биндингами по ключу и пользовательскими событиями, коллекций с богатым набором...

- EmberJS Главной особенностью EmberJS является привязка данных. Вы просто создаете переменную, и, когда значение этой переменной изменяется, обновляется любая...
- Knockout.js — это популярная JavaScript библиотека, позволяющая реализовать Model-View-View Model (MVVM) паттерн на клиенте.



3. Каскадные таблицы стилей. CSS. История. Структура описания стиля.

17 декабря 1996 – CSS

CSS (Cascading Style Sheets) таблицы стилей, которые присоединяются к документу HTML и служат для визуального оформления тех или иных частей документа.

Синтаксис CSS

Порядок написания свойств стилей довольно прост:

Селектор { свойство: значение; свойство2: значение; и. т. д.}

Селектор — определяет элемент HTML, к которому применяется форматирование.

Свойство — показывает параметр этого элемента HTML.

Значение — показывает какие изменения должен претерпеть данный параметр.

Пример:

```
Body { margin: 0; padding: 0 }
```

Билет 3

1. Классификация серверов по типу организации ввода-вывода. Примеры веб-серверов на каждый тип.

Процесс — экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы (например, процессорное время и память). Каждый процесс выполняется в отдельном адресном пространстве: один процесс не может получить доступ к переменным и структурам данных другого. Если процесс хочет получить доступ к чужим ресурсам, необходимо использовать межпроцессное взаимодействие. Это могут быть конвейеры, файлы, каналы связи между компьютерами и многое другое.

Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. У каждого потока есть собственные регистры и собственный стек, но другие потоки могут их использовать.

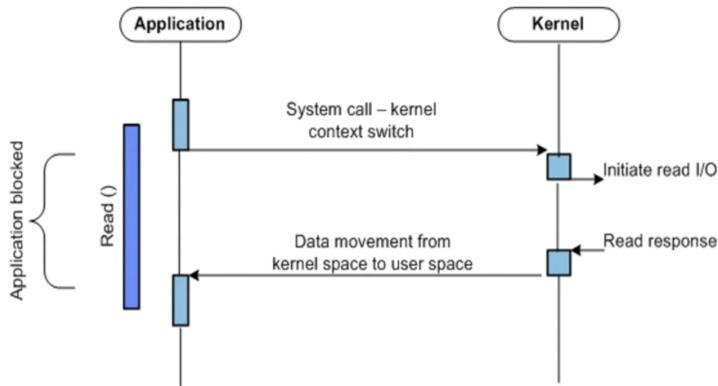
Поток — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса.

Любая пользовательская программа запускается внутри процесса, а код выполняется в контексте потока.

То есть блокирующий ввод-вывод называется так, потому что поток, который его использует, блокируется и переходит в режим ожидания, пока ввод-вывод не будет завершён.

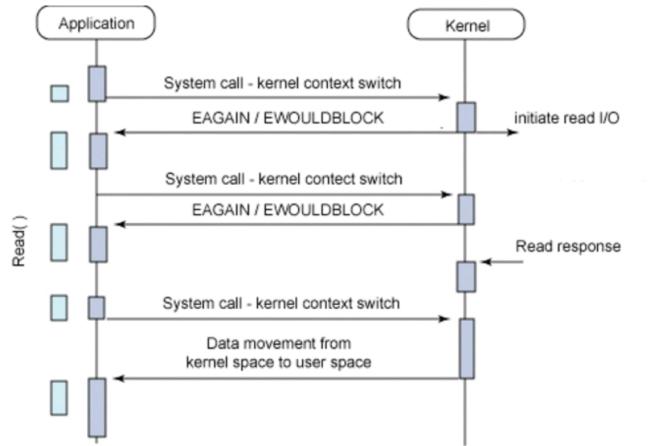
Проблема метода блокировки заключается в том, что поток будет спать, пока ввод-вывод не завершится. Поток не сможет выполнять никаких других задач, кроме ожидания завершения ввода-вывода.

Обработка запросов. Блокирующий ввод-вывод



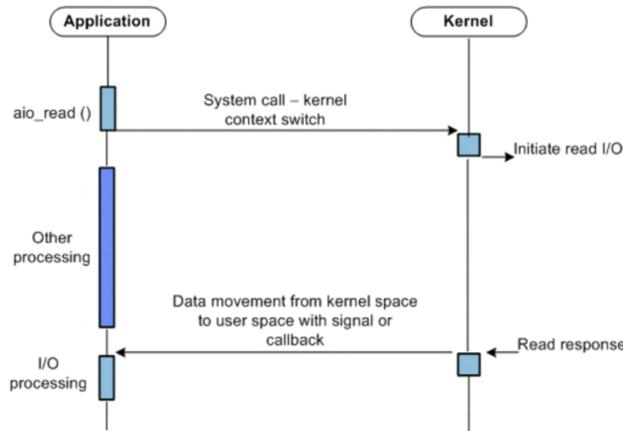
Его идея заключается в том, что когда программа делает вызов на чтение файла, ОС не будет блокировать поток, а просто вернёт ей либо готовые данные, либо информацию о том, что ввод-вывод ещё не закончен. Это не заблокирует поток, но программе придётся позже проверять, завершён ли ввод-вывод. Это означает, что ПО может по-разному реагировать в зависимости от того, завершён ли ввод-вывод и выполнять другие задачи. Когда же программе снова понадобится ввод-вывод, она сможет повторно попробовать прочесть содержимое файла, и если ввод-вывод завершён, то получит содержимое файла. В противном случае ПО снова получит сообщение о том, что операция ещё не завершена и сможет заняться другими задачами.

Обработка запросов. Неблокирующий ввод-вывод



Поток, выполняющий асинхронный ввод - вывод (asynchronous file I/O) файла, отправляет запрос на ввод-вывод данных ядру. Если запрос принят ядром, поток продолжает обрабатывать другое задание до тех пор, пока ядро не подаст сигналы потоку, что операция ввода/вывода (I/O) полностью завершилась. Тогда поток прерывает работу со своим текущим заданием и обрабатывает данные от операции ввода/вывода (I/O) по мере необходимости.

Обработка запросов. Асинхронный ввод-вывод



Популярные веб-сервера

Веб-сервера	Обработка запросов	Язык
Apache	prefork (процессы) worker (процессы+потоки)	C
nginx, lighttpd	Неблокирующие	C
IIS, Tomcat, Jetty	threads (потоки)	C#, Java
Node.js, Tornado, POE	Асинхронные, несколько рабочих процессов	языки высокого уровня
Starman, Hypnotoad	prefork (процессы)	языки высокого уровня
Erlang!	Легковесные потоки Erlang	Erlang
Для разработки (Django, webrick..)	Один поток	языки высокого уровня

2. CSS. Селекторы. Виды селекторов.

Пример:

```
Body { margin: 0; padding: 0 }
```

Селектором в данном примере служит тег body

Селектор выбирает элементы, для которых применяются пары свойство и значение.

Виды селекторов:

- Селектор тегов, в качестве селектора выступает имя тега, для которого необходимо изменить свойства.
- Универсальный селектор, который обозначается символом * и применяется для изменения необходимых свойств всех элементов на странице.
- Классы, применяются для элементов с атрибутом class и необходимым значением.
- Идентификаторы, применяются для элементов с атрибутом id и необходимым значением. Основное отличие классов от идентификаторов состоит в том, что имена вторых должны быть уникальными, не повторяться, что позволяет их использовать вместе со скриптами (JavaScript).
- Псевдо-классы предназначены для изменения стиля существующих элементов страницы в зависимости от их динамического состояния, например при работе со ссылками (:link, :visited, :hover, :active, :focus). Псевдо-элементы определяют стиль элементов, чётко не определённых в структуре документа (:first-letter, :first-line), а также позволяют генерировать и стилизовать несуществующее содержимое (:before, :after и свойство content). В CSS3 псевдо-элементы начинаются с двух двоеточий :: (::first-letter, ::first-line, ::before, ::after).
- Селекторы атрибутов. Позволяют стилизовать элемент не только по значению тега, но и по значению атрибута (a[attr]).
- Контекстные селекторы. Стилизация элементов, находящихся внутри другого элемента (a b).
- Дочерние селекторы. Стилизация элемента, расположенного сразу за другим элементом и являющегося его прямым потомком (a > b).
- Соседние селекторы. Предназначены для стилизации соседних элементов, у которых общий родитель. (a + b)
- Родственные селекторы. Похожи с соседними селекторами, но с тем различием, что стилизуются все соседние элементы, а не только первый соседний элемент. Впервые появились в CSS3. (a ~ b)

3. JS (ES5). Функции. This. Функция-конструктор.

JS - интерпретируемый, мультипарадигмальный, динамически типизируемый язык. Однопоточный. Событийный. Скриптовый. С управляемой памятью. С открытым стандартом ECMAScript.

ECMAScript — это стандарт на котором базируется язык JavaScript, и часто для сокращения его обозначают как ES.

ES5 был выпущен в декабре 2009 года, спустя 10 лет после выхода третьего издания. Среди изменений можно отметить:

- поддержку строгого режима (strict mode);
- аксессоры `getters` и `setters`;
- возможность использовать зарезервированные слова в качестве ключей свойств и ставить запятые в конце массива;
- многострочные строковые литералы;
- новую функциональность в стандартной библиотеке;
- поддержку JSON.

Примеры встроенных функций – это `alert(message)`, `prompt(message, default)` и `confirm(question)`. Но можно создавать и свои.

Объявление функции:

Вначале идёт ключевое слово `function`, после него имя функции, затем список параметров в круглых скобках через запятую и код функции, также называемый «телом функции», внутри фигурных скобок.

```
function имя(параметры) {  
    ...тело...  
}
```

Локальные переменные: Переменные, объявленные внутри функции, видны только внутри этой функции.

Пример:

```
function showMessage() {  
    let message = "Привет, я JavaScript!"; // локальная переменная
```

```
    alert( message );
}

showMessage(); // Привет, я JavaScript!

alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри функции

Внешние переменные:

У функции есть доступ к внешним переменным, например:

let userName = 'Вася';

function showMessage() {

let message = 'Привет, ' + userName;

alert(message);

}

showMessage(); // Привет, Вася
```

Функция обладает полным доступом к внешним переменным и может изменять их значение. Внешняя переменная используется, только если внутри функции нет такой локальной.

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю.

Мы можем передать внутрь функции любую информацию, используя параметры (также называемые аргументы функции). Если параметр не указан, то его значением становится `undefined`.

Если мы хотим задать параметру `text` значение по умолчанию, мы должны указать его после `=`:

```
function showMessage(from, text = "текст не добавлен") {
    alert( from + ": " + text );
}

showMessage("Аня"); // Аня: текст не добавлен
```

Функция может вернуть результат, который будет передан в вызвавший её код. Директива `return` может находиться в любом месте тела функции. Как только выполнение доходит до этого места, функция останавливается, и значение возвращается в вызвавший её код. Вызовов `return` может быть несколько, например:

```
function checkAge(age) {  
    if (age > 18) {  
        return true;  
    } else {  
        return confirm('А родители разрешили?');  
    }  
}  
  
let age = prompt('Сколько вам лет?', 18);  
  
if (checkAge(age)) {  
    alert('Доступ получен');  
} else {  
    alert('Доступ закрыт');  
}
```

Для доступа к информации внутри объекта метод может использовать ключевое слово `this`.

Значение `this` – это объект «перед точкой», который использовался для вызова метода.

Например:

```
let user = {  
    name: "Джон",  
    age: 30,  
    sayHi() {  
        // this - это "текущий объект"  
        alert(this.name);  
    }  
};  
  
user.sayHi(); // Джон
```

В JavaScript ключевое слово «`this`» ведёт себя иначе, чем в большинстве других языков программирования. Оно может использоваться в любой функции.

В большинстве случаев значение `this` определяется тем, каким образом вызвана функция. Значение `this` не может быть установлено путем присваивания во время

исполнения кода и может иметь разное значение при каждом вызове функции. В ES5 представлен метод `bind()`, который используется для привязки значения ключевого слова `this` независимо от того, как вызвана функция. Также в ES2015 представлены стрелочные функции, которые не создают собственные привязки к `this` (они сохраняют значение `this` лексического окружения, в котором были созданы).

В этом коде нет синтаксической ошибки:

```
function sayHi() {  
  alert( this.name );  
}
```

Обычный синтаксис `{...}` позволяет создать только один объект. Но зачастую нам нужно создать множество однотипных объектов, таких как пользователи, элементы меню и т.д.

Это можно сделать при помощи функции-конструктора и оператора "new".

Функция-конструктор:

Функции-конструкторы являются обычными функциями. Но есть два соглашения:

1. Имя функции-конструктора должно начинаться с большой буквы.
2. Функция-конструктор должна вызываться при помощи оператора "new".

Например:

```
function User(name) {  
  this.name = name;  
  this.isAdmin = false;  
}  
  
let user = new User("Вася");  
  
alert(user.name); // Вася  
alert(user.isAdmin); // false
```

Когда функция вызывается как `new User(...)`, происходит следующее:

1. Создаётся новый пустой объект, и он присваивается `this`.

2. Выполняется код функции. Обычно он модифицирует `this`, добавляет туда новые свойства.
3. Возвращается значение `this`.

Используя специальное свойство `new.target` внутри функции, мы можем проверить, вызвана ли функция при помощи оператора `new` или без него.

В случае, если функция вызвана при помощи `new`, то в `new.target` будет сама функция, в противном случае `undefined`.

Обычно конструкторы ничего не возвращают явно. Их задача – записать все необходимое в `this`, который в итоге станет результатом.

Но если `return` всё же есть, то применяется простое правило:

- При вызове `return` с объектом, будет возвращён объект, а не `this`.
- При вызове `return` с примитивным значением, примитивное значение будет отброшено.

Другими словами, `return` с объектом возвращает объект, в любом другом случае конструктор вернёт `this`.

Итого о функции-конструкторе:

- Функции-конструкторы или просто конструкторы являются обычными функциями, именовать которые следует с заглавной буквы.
- Конструкторы следует вызывать при помощи оператора `new`. Такой вызов создаёт пустой `this` в начале выполнения и возвращает заполненный в конце.

Билет 4

1. Основные идеи и принципы современного CI/CD. Инструменты. Докеризация.

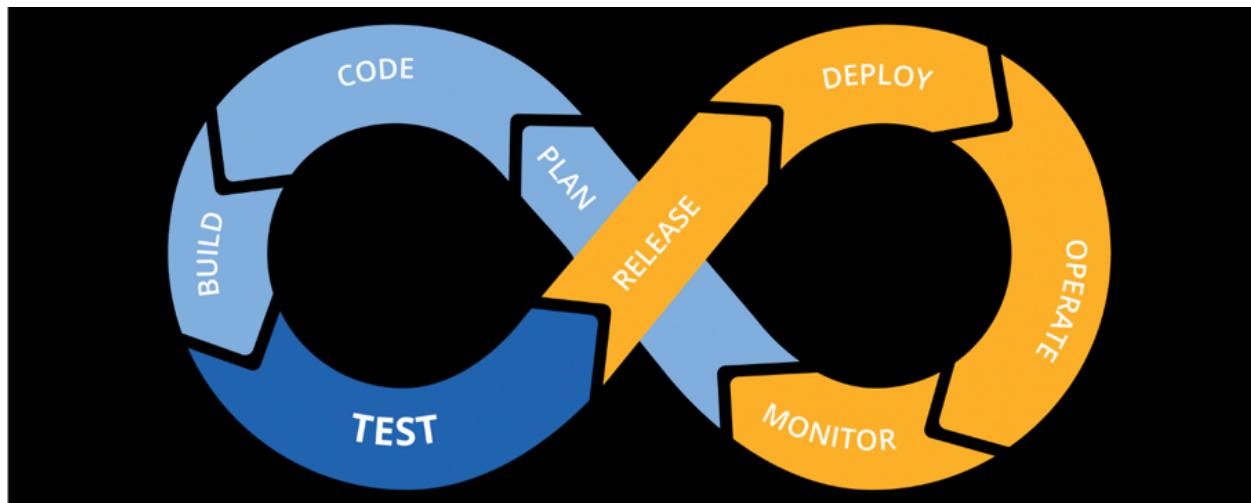
DevOps - интеграция того, что написал разработчик, в то, что есть в компании", - Ваня на лекции.

Существует понятие `Integration Hell` - чем дольше ветки не интегрируются с мастером, тем больше вероятность конфликтов и несовместимости.

CICD — это комбинация непрерывной интеграции (CI) и непрерывной доставки или непрерывного развертывания (CD).

CI/CD объединяет разработку, развёртывание и команду, ускоряя процесс сборки, тестирования и развёртывания приложения.

CI/CD - концепция непрерывной интеграции и доставки. Непрерывная интеграция (CI) заключается в постоянном слиянии рабочих копий (закоммиченного кода) в основную кодовую базу.



В идеале, схема на картинке должна выполняться каждый день. Главная идея состоит в поддержке "зелёного" мастера, то есть код оттуда может быть залит на боевой сервер. Важной составляющей хороших CI-процессов является Git Flow (подход по организации ветвления в проекте и по именованию всех основных веток).

Основные ветки:

- master - идеальный работоспособный код;
- develop - ветка разработки.

Вспомогательные ветки:

- ветки функциональностей;
- ветви релизов;
- ветки исправлений.

Способы достичь хороший CI-процесс:

1. система контроля версий;
2. автоматизация сборки (единий уникальный инструмент для сборки продукта);
3. автоматизация установки (централизованное управление);
4. самопостируемый билд;
5. сборка после каждого коммита (минус такого подхода - затраты по времени);
6. малое время сборки и тестирования;
7. production-like среда (простой выход - докер-контейнер, внутри которого можно инкапсулировать);
8. доступные и понятные отчёты;
9. централизованное хранилище билдов.

Инструмент - `gitlab.ci.yml` в Гите.

Вообще, инструменты делятся на локальные, облачные и правительственные. Гит, соответственно, локальный.

Концепция позволяет запускать различные типы тестов на каждом этапе (выполнение **интеграционного** аспекта) и завершать его запуском с развертыванием закомиченного кода в фактический продукт, который видят конечные пользователи (выполнение **доставки**).

CI/CD необходимы для разработки программного обеспечения с применением Agile-методологии, которая рекомендует использовать автоматическое тестирование для быстрой наладки рабочего программного обеспечения. Автоматическое тестирование дает заинтересованным лицам доступ к вновь созданным функциям и обеспечивает быструю обратную связь.

Принципы CI/CD:

- сегрегация ответственности заинтересованных сторон;
- снижение риска;
- короткий цикл обратной связи.

Сегрегация ответственности заинтересованных сторон: Одним из основных преимуществ CI/CD является своевременное участие различных заинтересованных сторон в любом проекте. **Разработчики и дизайнеры (Devs)** создают опыт и логику продукта, представленные в рассказах пользователей, и несут ответственность за создание работающих функций, доказывая это через модульные тесты. **Оперативный отдел (Ops)/ DevOps-инженеры** несут ответственность за доступность продукта пользователям. Работая в области CI/CD, они масштабируют

концепции по мере необходимости и разрабатывают логистику кода, чтобы код от разработчиков мог перейти в производственную среду и пользователи могли получать доступ. **Пользователи** несут ответственность за использование продукта. Подразумевается, что продукт должен быть полезен и оправдывать усилия. Каждый этап конвейерной обработки CI/CD создает среду, настроенную на то, чтобы **группы брали ответственность за соответствующую стадию тестирования**, обеспечивая целостность процесса.

Докер:

В общем смысле, докер - некоторое ПО с открытым исходным кодом, принцип работы которого схож с транспортными контейнерами (передача разных несовместимых типов вместе и обработка пакетов разного размера).

Докеризация отличается от виртуализации. Виртуализация обладает следующими недостатками:

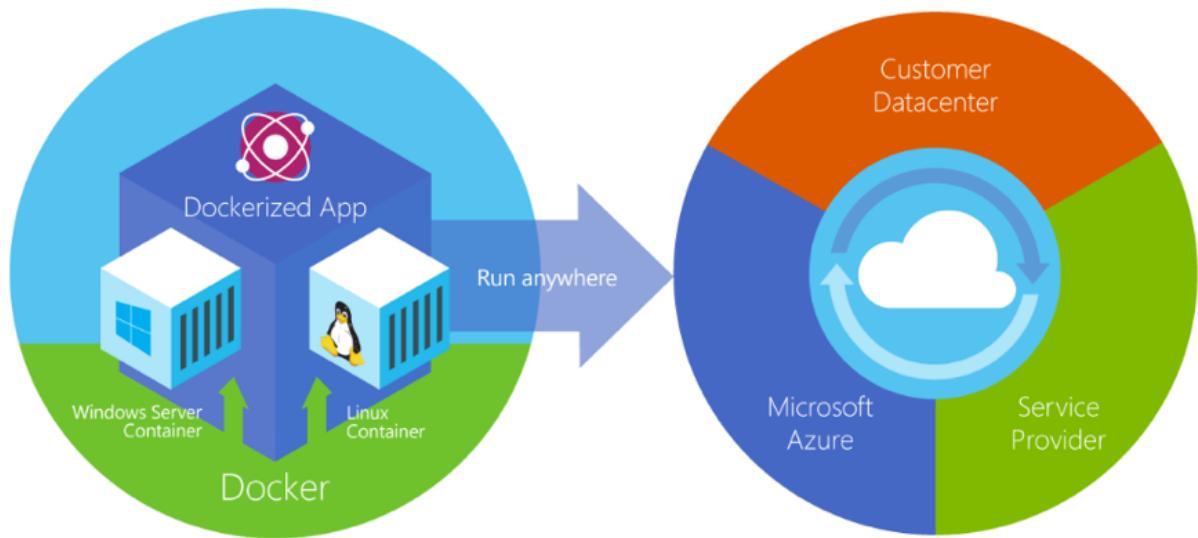
- медленная загрузка;
- возможная плата за предоставление дополнительного пространства;
- не все виртуальные машины поддерживают совместимое использование (или требуют сложной настройки);
- образ может быть слишком большим.

Докер же просто разделяет ядро ОС на все контейнеры (Docker container), работающие как отдельные процессы.

К его преимуществам относятся:

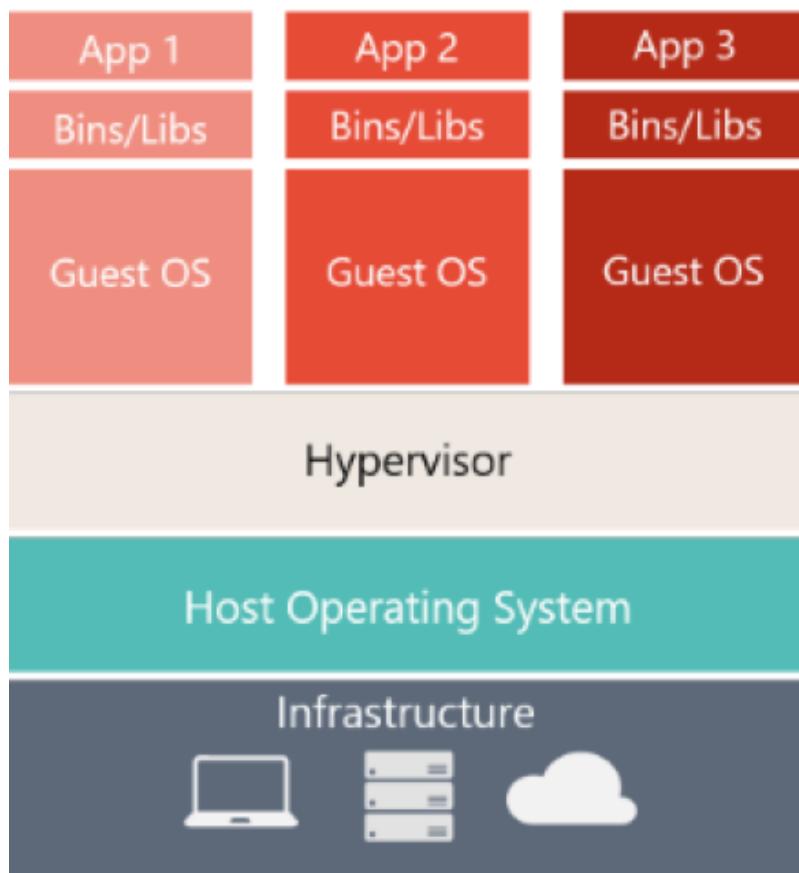
1. Ускоренный процесс разработки. Нет необходимости устанавливать вспомогательные инструменты вроде PostgreSQL, Redis, Elasticsearch: их можно запускать в контейнерах.
2. Удобная инкапсуляция приложений.
3. Понятный мониторинг.
4. Простое масштабирование.

Докер работает не только на его родной ОС, Linux, но также поддерживается Windows и macOS. Единственное отличие от взаимодействия с Linux в том, что на macOS и Windows платформа инкапсулируется в крошечную виртуальную машину. На данный момент Докер для macOS и Windows достиг значительного уровня удобства в использовании.



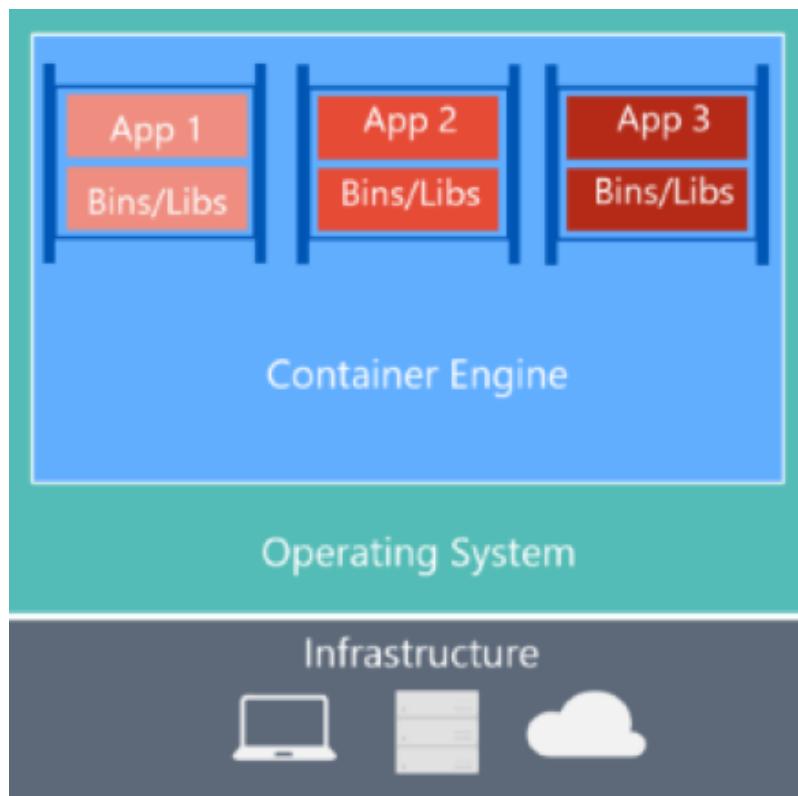
Виртуальная машина:

Виртуальные машины содержат приложение, необходимые библиотеки или двоичные файлы и всю операционную систему. Полная виртуализация требует больше ресурсов, чем создание контейнеров.



Докер:

Контейнеры включают в себя приложение и все его зависимости. Но они используют ядро ОС совместно с другими контейнерами, которые выполняются в изолированных процессах в пользовательском пространстве операционной системы узла. (Это не относится к контейнерам Hyper-V, где каждый контейнер запускается на отдельной виртуальной машине.)



2. Технологии толстого клиента. Организация дуплексной связи клиент-сервер (Poling, Long poling, WebSocket).

Толстый клиент: данное приложение обеспечивает полное функционирование вне зависимости от сервера. Часто он выступает в роли хранилища информации. Все расчеты, обработка совершаются на устройстве пользователя. Разработка позволяет работать сразу нескольким пользователям. Даже при отсутствии соединения с сервером (в некоторых случаях, зависит от программы), он может продолжать работать с локальными сохраненными копиями баз, а обмениваться информацией после появления связи.

Толстый клиент — в архитектуре клиент — сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства

- Толстый клиент обладает широкой функциональностью в отличие от тонкого.
- Режим многопользовательской работы.

- Предоставляет возможность работы даже при обрывах связи с сервером.
- Высокое быстродействие (зависит от аппаратных средств клиента)

Недостатки

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.
- Наличие бизнес-логики

Дуплексный контракт службы – это шаблон обмена сообщениями, в котором обе конечные точки могут отправлять сообщения друг другу независимо друг от друга.

Идея SSE проста — клиент подписывается на события сервера и как только происходит событие — клиент сразу же получает уведомление и некоторые данные, связанные с этим событием. Чтобы понять полезность протокола SSE необходимо сравнить его с привычными методами получения событий, вкратце объясню их суть:

Polling

Самый простой, но самый не эффективный, метод: клиент раз в несколько секунд опрашивает сервер на наличие событий. Даже если ничего нет, то клиент все равно делает запрос — а мало ли что придет.

Плюсы:

- Просто
- Данные могут быть пожаты

Минусы:

- Очень много лишних запросов
- События всегда приходят с опозданием
- Серверу приходится хранить события пока клиент не заберет их или пока они не устареют

Long Polling

Улучшенный вариант предыдущего метода. Клиент отправляет запрос на сервер, сервер держит открытым соединение пока не придут какие-нибудь данные или клиент не отключится самостоятельно. Как только данные пришли — отправляется ответ и соединение закрывается и открывается следующее и так далее.

Плюсы по сравнению с Polling:

- Минимальное количество запросов
- Высокая времененная точность событий
- Сервер хранит события только на время реконнекта

Минусы по сравнению с Polling:

- Более сложная схема

WebSockets

Это бинарный дуплексный протокол, позволяющий клиенту и серверу общаться на равных. Этот протокол можно применять для игр, чатов и всех тех приложений где вам нужны предельно точные события близкие к реальному времени.

Плюсы по сравнению с Long Polling:

- Поднимается одно соединение
- Предельно высокая времененная точность событий
- Управление сетевыми сбоями контролирует браузер

Минусы по сравнению с Long Polling:

- HTTP не совместимый протокол, нужен свой сервер, усложняется отладка
- Так почему же стоит применять SSE, раз у нас есть такой прекрасный протокол WebSockets?! Во-первых не каждому веб-приложению необходима двусторонняя связь — подойдет и SSE. Во-вторых SSE — HTTP совместимый протокол и вы можете реализовать рассылку событий на любом веб-сервере.

3. Что такое браузер? Какие функции он выполняет? Место в Web-стеке. Современное состояние рынка браузеров.

Браузер прикладное программное обеспечение для просмотра страниц, содержания веб-документов, компьютерных файлов и их каталогов; управления веб-приложениями; а также для решения других задач. (В глобальной сети браузеры используются для запроса, обработки, манипулирования и отображения содержания веб-сайтов. Многие современные браузеры также могут

использоваться для обмена файлами с серверами FTP, а также для непосредственного просмотра содержания файлов многих графических форматов (gif, jpeg, png, svg), аудио-видео форматов (mp3, mpeg), текстовых форматов (pdf, djvu) и других файлов.)

Какие основные функции:

1. Оперативный выход в сеть Всемирной паутины.
2. Предоставление всех необходимых инструментов для мониторинга интернет-среды.
3. Возможность скачивания улучшений как в ручном, так и в автоматическом режиме.
4. Периодические обновления для различных операционных систем.
5. Возможность задавать определенные настройки, отвечающие пользовательским требованиям.
6. Умение не только просматривать web-страницы, но и открывать электронные документы, мультимедийные файлы и прочее.
7. Предоставление средств для воспроизведения онлайн-видео со звуком, в том числе потоковых трансляций. В этом случае предполагается установка специализированных кодеков и подключение плагинов.
8. Тщательная обработка информации, вводимой владельцем. Способность корректировки текста в процессе поиска.

В настоящее время существует широкий выбор веб-обозревателей. Каждый браузер имеет свои достоинства, недостатки и индивидуальные возможности.

Практически все популярные браузеры распространяются бесплатно или «в комплекте» с другими приложениями, так что каждому пользователю не составит труда сравнить отдельные браузеры и выбрать для себя наиболее приемлемый вариант. При выборе браузера следует уделять внимание на поддержку им утвержденных веб-стандартов, простоту и удобство интерфейса, функциональность и безопасность.

Браузеры уже достигли своего максимального развития. Они все выполняют функции просмотра веб страниц, поддерживают мультимедиа. По моему мнению, в настоящее время развитие браузеров будет направлено на обеспечение конфиденциальности передаваемых данных между сайтом и пользователем.

В связи с разработкой новых вычислительных устройств, использующих явление квантовой суперпозиции (квантовый компьютер) и появлению угрозы расшифровки

информации, зашифрованной на основе системы открытого и закрытого ключа, требуется разработка новых видов шифрования. Соответственно новые программы браузеров должны поддерживать новые виды шифрования информации.

Возможно, следующим этапом в развитии программ браузеров станет внедрение искусственного интеллекта в их интерфейс для взаимодействия с пользователем. При этом появятся новые функции, например, управления голосом, взглядом, с помощью мысли, идентификации пользователя и автоматического применения его личных настроек.

Билет 5

1. Стек TCP/IP. Назначение, структура, применение.

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет. Название TCP/IP происходит из двух важнейших протоколов семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были первыми разработаны и описаны в данном стандарте.

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все связанные протоколы в соответствии с объемом задействованных сетей.

Стек протоколов TCP/IP включает в себя четыре уровня[6]:

- Прикладной уровень (Application Layer),
- Транспортный уровень (Transport Layer),
- Межсетевой уровень (Сетевой уровень) (Internet Layer),
- Канальный уровень (Network Access Layer).

Протоколы этих уровней полностью реализуют функциональные возможности модели OSI. На стеке протоколов TCP/IP построено всё взаимодействие пользователей в IP-сетях. Стек является независимым от физической среды передачи данных, благодаря чему, в частности, обеспечивается

полностью прозрачное взаимодействие между проводными и беспроводными сетями.

Прикладной (Application layer)	напр., HTTP , RTSP , FTP , DNS
Транспортный (Transport Layer)	напр., TCP , UDP , SCTP , DCCP (RIP , протоколы маршрутизации, подобные OSPF , что работают поверх IP , являются частью сетевого уровня)
Сетевой (Межсетевой) (Internet Layer)	Для TCP/IP это IP (вспомогательные протоколы, вроде ICMP и IGMP , работают поверх IP , но тоже относятся к сетевому уровню; протокол ARP является самостоятельным вспомогательным протоколом, работающим поверх канального уровня)
Уровень сетевого доступа (Канальный) (Network Access Layer)	Ethernet , IEEE 802.11 WLAN , SLIP , Token Ring , ATM и MPLS , физическая среда и принципы кодирования информации, T1 , E1

Прикладной уровень

На прикладном уровне (Application layer) работает большинство сетевых приложений.

Эти программы имеют свои собственные протоколы обмена информацией, например, [интернет браузер](#) для протокола [HTTP](#), [ftp-клиент](#) для протокола [FTP](#) (передача файлов), почтовая программа для протокола [SMTP](#) ([электронная почта](#)), [SSH](#) (безопасное соединение с удалённой машиной), [DNS](#) (преобразование символьных имён в [IP-адреса](#)) и многие другие.

Транспортный уровень

Протоколы [транспортного уровня](#) (Transport layer) могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных. В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные.

Протоколы автоматической маршрутизации, логически представленные на этом уровне (поскольку работают поверх IP), на самом деле являются частью протоколов сетевого уровня; например [OSPF](#) (IP идентификатор 89).

[TCP](#) (IP идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный [поток данных](#), дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от [UDP](#).

UDP (IP идентификатор 17) протокол передачи датаграмм без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол TCP.

UDP обычно используется в таких приложениях, как потоковое видео и компьютерные игры, где допускается потеря пакетов, а повторный запрос затруднён или не оправдан, либо в приложениях вида запрос-ответ (например, запросы к DNS), где создание соединения занимает больше ресурсов, чем повторная отправка.

И TCP, и UDP используют для определения протокола верхнего уровня число, называемое портом.

Сетевой (межсетевой) уровень [править | править код]

Межсетевой уровень (Internet layer) изначально разработан для передачи данных из одной сети в другую. На этом уровне работают маршрутизаторы, которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по маске сети. Примерами такого протокола является X.25 и IPC в сети ARPANET.

С развитием концепции глобальной сети в уровень были внесены дополнительные возможности по передаче из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также возможность запрашивать данные от удалённой стороны, например в протоколе ICMP (используется для передачи диагностической информации IP-соединения) и IGMP (используется для управления multicast-потоками).

ICMP и IGMP расположены над IP и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI.

Пакеты сетевого протокола IP могут содержать код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный IP-номер протокола. ICMP и IGMP имеют номера, соответственно, 1 и 2.

К этому уровню относятся: DVMRP, ICMP, IGMP, MARS, PIM, RIP, RIP2, RSVP

Канальный уровень

Канальный уровень (англ. Link layer) описывает способ кодирования данных для передачи пакета данных на физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также

обеспечивающие помехоустойчивость). Ethernet, например, в полях заголовка пакета содержит указание того, какой машине или машинам в сети предназначен этот пакет.

Примеры протоколов канального уровня — Ethernet, IEEE 802.11 WLAN, SLIP, Token Ring, ATM и MPLS.

PPP не совсем вписывается в такое определение, поэтому обычно описывается в виде пары протоколов HDLC/SDLC.

MPLS занимает промежуточное положение между канальным и сетевым уровнем и, строго говоря, его нельзя отнести ни к одному из них.

Канальный уровень иногда разделяют на 2 подуровня — LLC и MAC.

Кроме того, канальный уровень описывает среду передачи данных (будь то коаксиальный кабель, витая пара, оптическое волокно или радиоканал), физические характеристики такой среды и принцип передачи данных (разделение каналов, модуляцию, амплитуду сигналов, частоту сигналов, способ синхронизации передачи, время ожидания ответа и максимальное расстояние).

При проектировании стека протоколов на канальном уровне рассматривают помехоустойчивое кодирование — позволяющие обнаруживать и исправлять ошибки в данных вследствие воздействия шумов и помех на канал связи.

2. Протокол HTTP. Расшифровка, назначение, структура. Формат пакета.

HTTP - Hyper Text Transfer Protocol

2 особенности протокола:

- текстовый (вся коммуникация между данными – символьный формат) протокол;
- без сохранения состояния (отсутствует понятие состояния (только “костыль – cookie”: появились проблемы при появлении веб-приложений (например, чтобы не переавторизовываться в соц.сети))).

HTTP-сессия:

- клиент устанавливает TCP-соединение с сервером (обычно на 80 порт);
- сервер принимает запрос на соединение и ожидает текст HTTP-запроса;
- клиент отправляет запрос со всей необходимой информацией (URI, тип запроса, заголовки, тело запроса);

- сервер обрабатывает запрос и отдаёт ответ (код статуса, заголовки ответа и тело ответа).

HTTP-структура:

Структура запроса:

- строка запроса;
- заголовки;
- пустая строка;
- тело запроса.

Структура ответа:

- строка статуса ответа;
- заголовки ответа;
- пустая строка;
- тело ответа.

HTTP-методы:

- OPTIONS - запрос методов сервера;
- GET - запрос документа;
- HEAD - аналог GET, но без тела ответа (метаданные без данных);
- POST - передача данных от клиента;
- PUT - размещение файла по URI/изменение данных;
- DELETE - удаление файла по URI/удаление данных;
- TRACE, LINK, UNLINK, CONNECT - используются редко;
- PATCH - частичное изменение данных на сервере.

Коды ответа:

- 1xx - информационные;
- 2xx - успешное выполнение;
- 3xx - перенаправление;
- 4xx - ошибка клиента;
- 5xx - ошибка сервера.

3. CSS. Основные свойства отображения.

Свойства шрифта:

font-family- определяет используемый элементом шрифт. Если указать URL(file), то шрифт автоматически установится на компьютер пользователя;

font-style- стиль шрифта (normal, italic);

font-variant- варианты отображения шрифта (normal, small-caps);

font-weight- жирность шрифта (normal, bold, bolder, lighter, значение от 100 до 900);

font-size- размер шрифта (размер, xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger);

font - обобщает вышеперечисленные свойства (любая комбинация из вышеперечисленных значений);

Свойства фона и цвета:

color- цвет элемента (значение);

background-color- цвет фона элемента (значение);

background-image- изображение фон (none, URL);

background-repeat- варианты повторения фонового изображения (repeat, repeat-x, repeat-y, no-repeat);

background-attachment- возможность прокрутки фонового изображения (scroll, fixed);

background-position- положение фонового изображения (%ширины%высоты, top, middle, bottom, left, center, right);

background-обобщает вышеперечисленные свойства (любая комбинация из вышеперечисленных значений);

Свойства блока:

margin-top- определяет отступ сверху (значение, %, auto);

margin-right- определяет отступ справа (значение, %, auto);

margin-bottom- определяет отступ снизу (значение, %, auto);

margin-left- определяет отступ слева (значение, %, auto);

margin - обобщает все вышеперечисленные свойства;

padding-top- отступ от верхнего border'a (значение, %);

padding-right- отступ от правого border'a (значение, %);

padding-bottom - отступ от нижнего border'a (значение, %);
padding-left - отступ от левого border'a (значение, %);
padding - обобщает все вышеперечисленные свойства;

Билет 6

1. CSS3. Основные возможности стандарта (по сравнению с CSS2).

CSS (Cascading Style Sheets — каскадная таблица стилей) — язык описания внешнего вида документа, созданного с помощью языка разметки HTML.

Прежде всего, стоит отметить переработанную систему позиционирования элементов и множество новых доступных селекторов. Возможности CSS3 включают эффекты теней и скругленные углы у блоков, что позволяет избавиться от большого количества ненужной разметки и использования специальных изображений, как это было в предыдущей версии каскадных стилей. Сюда же относится и возможность устанавливать изображения в качестве фона и использовать их как границы. Существенно упростились и работа с анимацией — если раньше для ее создания нужно было работать с довольно сложным в освоении языком Java Script, то теперь возможностей самых каскадных стилей для этих целей вполне достаточно.

Связка CSS3 + HTML5 уже активно используется на сайтах ведущих технологических компаний, таких как Adobe, Mozilla и Apple. И хотя пока нельзя сказать о поддержке указанных спецификаций современными браузерами в полной мере, но их разработчики постепенно расширяют перечень поддерживаемых классов. Если вы только начинаете изучать языки верстки, рекомендую ознакомиться с советами как научиться верстке сайтов в одном из уроков на нашем сайте.

В качестве хорошего примера положительных нововведений можно назвать правило в CSS3, с помощью которого web-мастер может задать использование на сайте любого из шрифтов, установленных на сервере. Раньше можно было использовать только шрифты, установленные на компьютере пользователя. Таким образом, множество жестких правил, которые ранее ограничивали web-разработчиков, теперь уже становятся неактуальными, открывая новые просторы для развития и экспериментов.

2. Язык Javascript. История. Основные идеи. Область применения. Структура программы.

Появился в 1995

JS - интерпретируемый, мультипарадигмальный, динамически типизируемый язык. Однопоточный. Событийный. Скриптовый. С управляемой памятью. С открытым стандартом ECMAScript.

Синтаксис JS:

- чувствителен к регистру;
- пробелы, табуляции и переводы строки игнорируются;
- операторы разделяются точкой с запятой;
- комментарии аналогичны C;
- Идентификаторами являются имена переменных, функций, а также меток. Идентификаторы образуются из любого количества букв ASCII, подчеркивания (_) и символа доллара (\$). Первым символом не может быть цифра, а в версии JavaScript 1.0 не допускается использования символа \$.
- Ключевые слова не могут использоваться в качестве идентификаторов. Ключевыми словами являются: break, case, continue, default, delete, do, else, export, false, for, function, if, import, in, new, null, return, switch, this, true, typeof, with.

Язык программирования JavaScript был разработан Брэнданом Эйком (Brendan Eich) в Netscape Communications как язык сценариев для обозревателей Netscape Navigator, начиная с версии 2.0. В дальнейшем к развитию этого языка подключилась корпорация Microsoft, чьи обозреватели Internet Explorer поддерживают JavaScript, начиная с версии 3.0. Версия Microsoft получила название JScript, поскольку JavaScript является зарегистрированной маркой фирмы Netscape. В 1996 г. ECMA приняла решение о стандартизации этого языка, и в июне 1997 г. была принята первая версия стандарта под названием ECMAScript (ECMA-262). В апреле 1998 г. этот стандарт был принят ISO в качестве международного под номером ISO/IEC 16262. Используем название JavaScript, а не ECMAScript по двум причинам:

1. Это название является исторически первым, и под ним данный язык наиболее известен широкому кругу пользователей.
2. Соответствующий тип MIME, а именно "text/javascript", распознается всеми обозревателями, которые поддерживают сценарии на данном языке, в отличие

от JScript или ECMAScript.

JavaScript — это язык программирования, основанный на объектах: и языковые средства, и возможности среды представляются объектами, а сценарий (программа) на JavaScript — это набор взаимодействующих объектов.

Объект JavaScript — это неупорядоченный набор свойств, каждое из которых имеет нуль или более атрибутов, которые определяют, как это свойство может использоваться. Например, если атрибуту свойства ReadOnly (неизменяемый) присвоено значение true (истина), то все попытки программно изменить значение этого свойства будут безрезультатны. Свойства — это контейнеры, которые содержат другие объекты, примитивные значения и методы. Примитивное значение — это элемент любого из встроенных типов: Undefined, Null, Boolean, Number и String ; объект — это элемент еще одного встроенного типа Object ; метод — функция, ассоциированная с объектом через свойство.

JavaScript содержит несколько встроенных объектов, таких, как Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Кроме того, JavaScript содержит набор встроенных операций, которые, строго говоря, не обязательно являются функциями или методами, а также набор встроенных операторов, управляющих логикой выполнения программ. Синтаксис JavaScript в основном соответствует синтаксису языка Java, но упрощен в сравнении с ним, чтобы сделать язык сценариев легким для изучения. Так, к примеру, декларация переменной не содержит ее типа, свойства также не имеют типов, а декларация функции может стоять в тексте программы после ее вызова.

JavaScript находит применение в качестве скриптового языка доступа к объектам приложений. Платформа Mozilla (XUL/Gecko) использует JavaScript. Среди сторонних продуктов, например, Java, начиная с версии 6, содержит встроенный интерпретатор JavaScript на базе Rhino. Сценарии JavaScript поддерживаются в таких приложениях Adobe, как Adobe Photoshop, Adobe Dreamweaver или Adobe Illustrator.

3. Стек OSI/ISO. Структура. Применение. Протоколы.

Основная задача, решаемая при создании компьютерных сетей, — обеспечение совместимости оборудования по электрическим и механическим характеристикам и совместимости информационного обеспечения (программ и данных) по системам кодирования и формату данных. Решение этой задачи относится к области стандартизации. Методологической основой стандартизации в компьютерных сетях является многоуровневый подход к разработке средств сетевого взаимодействия. На основе этого подхода и технических предложений

Международной организации стандартов ISO (International Standards Organization) в начале 1980-х гг. была разработана стандартная модель взаимодействия открытых систем OSI (Open Systems Interconnection). Модель ISO/OSI сыграла важную роль в развитии компьютерных сетей.

Модель OSI определяет различные уровни взаимодействия систем и указывает, какие функции должен выполнять каждый уровень. В модели OSI средства взаимодействия делятся на семь уровней: прикладной (Application), представительный (Presentation), сеансовый (Session), транспортный (Transport), сетевой (Network), канальный (Data Link) и физический (Physical). Самый верхний уровень — прикладной. На этом уровне пользователь взаимодействует с приложениями. Самый нижний уровень — физический. Этот уровень обеспечивает обмен сигналами между устройствами.

Обмен данными через каналы связи происходит путем перемещения данных с верхнего уровня на нижний, затем транспортировки по линиям связи и, наконец, обратным воспроизведением данных в компьютере клиента в результате их перемещения с нижнего уровня на верхний.

Для обеспечения необходимой совместимости на каждом из уровней архитектуры компьютерной сети действуют специальные стандартные протоколы. Они представляют собой формализованные правила, определяющие последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне, но в разных узлах сети.

Иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети, называется стеком коммуникационных протоколов. Следует четко различать модель ISO/OSI и стек протоколов ISO/OSI. Модель ISO/OSI является концептуальной схемой взаимодействия открытых систем, а стек протоколов ISO/OSI представляет собой набор вполне конкретных спецификаций протоколов для семи уровней взаимодействия, которые определены в модели ISO/OSI.

Коммуникационные протоколы могут быть реализованы как программно, так и аппаратно. Протоколы нижних уровней часто реализуются комбинацией программных и аппаратных средств, а протоколы верхних уровней — как правило, чисто программными средствами.

Уровень 1, физический

Физический уровень получает пакеты данных от вышестоящего канального уровня и преобразует их в оптические или электрические сигналы, соответствующие 0 и 1 бинарного потока. Эти сигналы посылаются через среду передачи на приемный

узел. Механические и электрические/оптические свойства среды передачи определяются на физическом уровне и включают:

Тип кабелей и разъемов

Разводку контактов в разъемах

Схему кодирования сигналов для значений 0 и 1

USB, кабель («витая пара», коаксиальный, оптоволоконный), радиоканал

Уровень 2, канальный

Канальный уровень обеспечивает создание, передачу и прием кадров данных. Этот уровень обслуживает запросы сетевого уровня и использует сервис физического уровня для приема и передачи пакетов. Спецификации IEEE 802.x делят канальный уровень на два подуровня: управление логическим каналом (LLC) и управление доступом к среде (MAC). LLC обеспечивает обслуживание сетевого уровня, а подуровень MAC регулирует доступ к разделяемой физической среде.

PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.

Уровень 3, сетевой

Сетевой уровень отвечает за деление пользователей на группы. На этом уровне происходит маршрутизация пакетов на основе преобразования MAC-адресов в сетевые адреса. Сетевой уровень обеспечивает также прозрачную передачу пакетов на транспортный уровень.

IPv4, IPv6, IPsec, AppleTalk

Уровень 4, транспортный

Транспортный уровень делит потоки информации на достаточно малые фрагменты (пакеты) для передачи их на сетевой уровень.

TCP, UDP, SCTP, PORTS

Уровень 5, сеансовый

Сеансовый уровень отвечает за организацию сеансов обмена данными между оконечными машинами. Протоколы сеансового уровня обычно являются составной частью функций трех верхних уровней модели.

RPC, PAP, L2TP

Уровень 6, уровень представления

Уровень представления отвечает за возможность диалога между приложениями на разных машинах. Этот уровень обеспечивает преобразование данных (кодирование, компрессия и т.п.) прикладного уровня в поток информации для транспортного

уровня. Протоколы уровня представления обычно являются составной частью функций трех верхних уровней модели.

ASCII, EBCDIC

Уровень 7, прикладной

Прикладной уровень отвечает за доступ приложений в сеть. Задачами этого уровня является перенос файлов, обмен почтовыми сообщениями и управление сетью.

HTTP, FTP, POP3, WebSocket

Билет 7

1) Основные идеи и возможности HTML5. Область применения.

(<https://www.ibm.com/developerworks/ru/library/wa-html5fundamentals/index.html>)

в HTML5 было представлено введение таких элементов, как аудио и видео

- Основополагающими принципами новой версии языка стали:
- Меньшая зависимость от плагинов для раскрытия функциональности;
- Сценарии должны быть заменены разметкой, когда это возможно;
- Независимость от устройства (т. е. доступность на всех устройствах и обеспечение одинакового конечного опыта);
- Процесс общественного развития, чтобы люди могли видеть, что происходит.

В частности, HTML5 добавляет целую кучу новых тегов разметки

- семантическая вёрстка;
- толерантность к ошибкам;
- видео;
- аудио.

HTML5 — это инструмент для упорядочивания Web-контента. Он предназначен для упрощения Web-проектирования и Web-разработки за счет языка разметки, обеспечивающего стандартизованный и интуитивно понятный пользовательский интерфейс. HTML5 предоставляет разработчику средства для секционирования и структуризации Web-страниц, а также позволяет создавать обособленные компоненты, которые не только обеспечивают логическую организацию Web-сайта, но и предоставляют ему возможности синдикации. Язык HTML5 реализует подход к

проектированию Web-сайтов, основанный на отображении информации, поскольку он воплощает саму суть отображения информации — разделение и маркирование информации для упрощения ее использования и понимания. Именно в этом состоит огромная семантическая и эстетическая ценность HTML5. HTML5 предоставляет дизайнерам и разработчикам всех уровней возможности для предоставления в публичный доступ буквально любого контента – от простых текстов до мультимедийно насыщенных интерактивных материалов.

HTML5 предоставляет эффективные инструменты для управления данными, для рисования, для воспроизведения видео- и аудиоконтента. HTML5 облегчает разработку кросс-браузерных Web-приложений, а также приложений для мобильных устройств. HTML5 относится к числу технологий, которые стимулируют развитие мобильных сервисов на основе облачных вычислений. Кроме того, HTML5 способствует повышению гибкости – благодаря возможности создания впечатляющих и интерактивных Web-сайтов. И, наконец, HTML5 предлагает новые теги и усовершенствования, в числе которых следующие: элегантная структура, органы управления формами, API-интерфейсы, мультимедийные функции, поддержка баз данных, существенно увеличенная скорость обработки.

Новые теги HTML5 обладают «говорящими» названиями, которые раскрывают назначение и характер использования этих элементов. В предыдущих версиях HTML использовались весьма неопределенные названия тегов. В спецификации HTML5, напротив, используются весьма описательные, интуитивно понятные названия. HTML5 предоставляет информационно-насыщенные названия, которые однозначно идентифицируют соответствующий контент. Например, широко применявшийся до настоящего времени тег `<div>` был дополнен тегами `<section>` и `<article>`. Кроме того, были добавлены теги `<video>`, `<audio>`, `<canvas>` и `<figure>`, которые обеспечивают более точное описание определенных типов контента.

HTML5 предоставляет следующие возможности.

- Теги с описательными названиями, которые точно указывают, для содержания какого контента предназначены эти теги.
- Усовершенствованные сетевые коммуникации.
- Существенно улучшенное хранение данных
- Средства Web Worker для исполнения фоновых процессов.
- Интерфейс WebSocket для установки постоянного соединения между резидентным приложением и сервером.
- Улучшенное извлечение хранящихся данных.

- Повышенная скорость сохранения и загрузки страниц
- Поддержка CSS3 при управлении пользовательским интерфейсом, что обеспечивает контентную ориентированность HTML5.
- Улучшенная обработка форм в браузере.
- API-интерфейс баз данных на основе SQL, позволяющий применять локальное хранилище на стороне клиента.
- Теги canvas и video, позволяющие добавлять графические и видеоматериалы без установки сторонних подключаемых модулей.
- Спецификация API-интерфейса Geolocation, использующая геолокационные возможности смартфонов в интересах задействования облачных сервисов и приложений для мобильных устройств.
- Усовершенствованные формы, ослабляющие потребность в загрузке кода JavaScript, что обеспечивает более эффективную связь между мобильными устройствами и серверами cloud-среды.

HTML5 позволяет предоставить пользователю более впечатляющие возможности: страницы, спроектированные с использованием спецификации HTML5, способны предоставлять такие же возможности, как приложения для настольных систем. Кроме того, HTML5 существенно улучшает разработку для нескольких платформ благодаря сочетанию возможностей API-интерфейсов с совместностью браузера. HTML5 позволяет разработчикам предоставлять возможности современных приложений, беспрепятственно охватывающие несколько платформ.

Фактически HTML5 является синонимом непрерывных инноваций: новые теги, новые методики и общая инфраструктура разработки, базирующаяся на взаимодействии технологии HTML5 с родственными технологиями CSS3 и JavaScript. Это создает основу для функционирования приложений, ориентированных на клиентов. Помимо широкого распространения средств и методик технологии HTML5 на настольных системах, она может быть реализована в функционально насыщенных Web-браузерах для мобильных телефонов. Это растущий рынок, характерными представителями которого являются популярные и ведущие платформы Apple iPhone, Google Android и Palm webOS.

Важнейшим аспектом мощных возможностей HTML5 является препарирование информации — или разделение контента на блоки, которое делает процесс гораздо понятнее. Высокая эффективность этого инструмента при проектировании и разработке подтверждается его усиливающимся доминированием в сфере Web-обработки.

HTML5 знаменует приход более эффективного семантического процесса на текстовом уровне и преобладание контролируемости над конструированием и использованием форм. Все эти и многие другие инновационные аспекты HTML5 обуславливают усиливающееся доминирование этой новой парадигмы. Этот нарастающий эффект до той или иной степени повлиял на многие организации (причем не только на коммерческие), в том числе на многие организации, у которых обработка информации и коммуницирование лишь недавно вошли в число основных видов деятельности.

2. MVC-фреймворки. Понятие фреймворка. Примеры. Типовая структура.

MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Компоненты MVC:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.
2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
3. Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC (Fat Stupid Ugly Controllers).

(MVC — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер. Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.)

ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.

Веб-фреймворки - это наборы функций, объектов, правил и других конструкций кода, предназначенных для решения общих проблем, ускорения разработки и

упрощения различных типов задач, стоящих в конкретной области.

1. AngularJS является JavaScript-фреймворком с открытым исходным кодом, разрабатываемым Google. Предназначен для разработки одностраничных приложений....
2. Backbone.js придает структуру веб-приложениям с помощью моделей с биндингами по ключу и пользовательскими событиями, коллекций с богатым набором...
3. EmberJS Главной особенностью EmberJS является привязка данных. Вы просто создаете переменную, и, когда значение этой переменной изменяется, обновляется любая...
4. Knockout.js — это популярная JavaScript библиотека, позволяющая реализовать Model-View-View Model (MVVM) паттерн на клиенте.



3. Веб-сервер. Назначение, область применения, типы. Примеры.

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веббраузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Веб-сервером называют как программное обеспечение, выполняющее функции вебсервера, так и непосредственно компьютер, на котором это программное

обеспечение работает.

Клиент, которым обычно является веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URL-адресами. Ресурсы — это HTML-страницы, изображения, файлы, медиа-потоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Различают статические и активные серверы Web. Если страницы сервера содержат только статическую текстовую и мультимедийную информацию, а также гипертекстовые ссылки на другие страницы, то сервер называется статическим. Если страницы webсервера изменяют своё содержимое в зависимости от действий пользователя, то такие серверы называют активными. Статический сервер Web не может служить основой для создания интерактивных приложений с доступом через Интернет, так как он не предусматривает никаких средств ввода и обработки запросов.

Типы:

- статичные веб-сервера - отдают только готовые данные
- динамические - подготавливают контент и/или выполняют дополнительные задачи/скрипты

Самые популярные HTTP серверы:

Apache – наиболее популярный и распространенный HTTP сервер, используется для Unix систем, но есть версии и для ОС семейства Windows. Данный HTTP сервер является свободным;

IIS – веб-сервер от компании Microsoft, распространяется бесплатно с операционными системами семейства Windows;

nginx – свободный HTTP сервер, разрабатываемый российским программистом Игорем Сысоевым; стоит отметить, что многие крупные проекты используют NGNIX

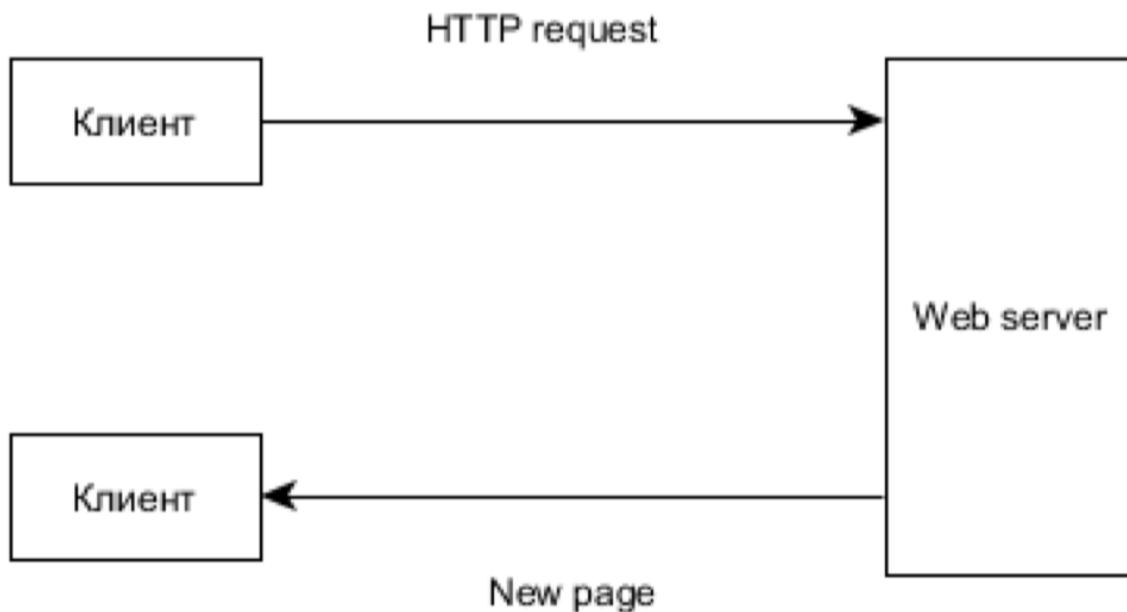
Google Web Server – этот веб-сервер распространяется и поддерживается компанией Гугл, за основу они взяли HTTP сервер Apache и доработали его.

Cherokee – свободный веб-сервер, особенность которого заключается в том, что управлять им можно только через веб-интерфейс.

Билет 8

1. Варианты организации клиент-серверной архитектуры приложения в веб. МРА, МРА+AJAX, SPA. Расшифровка, определение, принципы.

МРА (Multi Page Applications) - каждый раз, когда приложение запрашивает данные либо отправляет их на сервер, оно вынуждено получать новую страницу в полном объеме, а затем визуализировать ее в браузере. Это классическое веб-приложение.



Статический веб-сайт - гиперссылки между готовыми HTML-страницами.

Другой вариант - хранение на сервере шаблонов страниц. В настоящее время за счёт скорости Интернета не видно разницы при загрузке страниц.

МРА+AJAX (Multi Page Applications + Asynchronous Javascript and XML) - ajax позволяет обновлять только часть страницы, а не всю страницу целиком.

Пример такого сочетания - лайки ВК. Сам ВК - МРА, но внутри страниц JS.

Обработчик перехватывает лайк, происходит асинхронный запрос на сервер, обработка и возврат.

В таком варианте полностью ломается логика архитектуры, "как она размазана, никто не знает", полностью теряется контроль над ситуацией в таком приложении.

SPA (Single Page Applications) — это эволюция шаблона МРА + AJAX. При таком подходе только каркас веб-страницы строится на сервере, все остальное генерируется средствами JavaScript. SPA запрашивает разметку и данные раздельно, и визуализирует результаты непосредственно в браузере.

Пример SPA - gmail.

SPA не сохраняет состояние пользователя на сайте в большинстве случаев, но его преимущество как раз в возможности работать оффлайн.

SPA = MVC + AJAX/JSON + REST.

Недостаток SPA состоит в сложной разработке.

2. Структура типового MVC web-приложения на примере одного из фреймворков (Express, ASP, Django, Spring, Rails, etc.).

Django:

Вместе эти три части — логика доступа к данным, логика исполнения и логика представления данных — составляют концепт, который обычно называют концепцией Model-View-Controller (MVC) (модель-представление-контроллер) архитектуры приложений. В этой схеме «Модель» относится к структуре данным, «Представление» — относится к части, которая отвечает за выборку того, что и как отображать, а «Контроллер» — относится к части, которая определяет какое представление использовать, например — в зависимости от ввода данных пользователя.

Django следует концепции MVC достаточно строго, что бы его можно было называть «MVC фреймворк». Вот краткое описание того, как M, V и C разделены в среде Django:

- M — часть доступа к данным, обслуживается системой баз данных Django, которая описывается в этой главе.
- V — часть, которая занимается выборкой того, что именно и как именно отображать, в Django этим занимаются представления (views) и шаблоны (templates).
- C — часть, которая определяет представление в зависимости от указаний пользователя, этим занимается сам фреймворк, следя настройкам URLconf и вызывая соответствующую функцию Python для заданного URL-а.

Так как «C» занимается сам фреймворк и больше всего действий происходит в моделях, шаблонах и представлениях, Django может называться «MVC

фреймворком».

В концепции MTV:

- M означает «Model» — модель, уровень доступа к данным. Сюда относится всё, что связано с данными — как получить к ним доступ, как проверить их, какое у них поведение и отношение друг с другом.
- T означает «Template» — шаблоны. Этот уровень содержит в себе всё, что связано с отображением: как именно что-то должно быть отображено на вебстранице или любом другом документе.
- V означает «View» — представление. Этот уровень связан со всей логикой, связывающей модель и необходимые шаблоны. Вы можете себе представлять это уровень как мост между моделями и шаблонами.

Если вы знакомы с другими фреймворками MVC, такими как Ruby on Rails, вы можете рассматривать представления в Django как «контролеры», а шаблоны Django — как представления. Эта путаница вызвана различным толкованием концепции MVC. В интерпретации Django, представление (view) описывает данные, которые должны быть представлены пользователю; т.е. не столько как они будут переданы, сколько какие именно данные передать. В Ruby on Rails же, и других подобных фреймворках, предполагается что роль контроллера включает в себя выбор какие данные передать пользователю, тогда как представление определяет — как именно эти данные будут выглядеть, а не какие данные передавать.

MVC в django = MTV + встроенное С

ASP и Visaul Studio:

Когда вы впервые создаете новый проект MVC Framework, у вас есть на выбор две базовых отправных точки: шаблон Empty (Пустой) и шаблон MVC. Эти имена несколько обманчивы, т.к. добавить базовые папки и сборки, требуемые для MVC Framework, можно в любой проект, отметив флажок MVC в разделе Add folders and core references for (Добавить папки и основные ссылки для) диалогового окна New ASP.NET Project (Новый проект ASP.NET), как показано на рисунке ниже. В случае выбора шаблона MVC этот флажок отмечается автоматически.

Реальное отличие связано с дополнительным содержимым, которое шаблон MVC добавляет в новые проекты. Оно предоставляет готовую отправную точку, включающую стандартные контроллеры и представления, конфигурацию безопасности, ряд популярных пакетов JavaScript и CSS (таких как jQuery и Bootstrap), а также компоновку - которая применяет библиотеку Bootstrap для

построения темы, оформляющей пользовательский интерфейс приложения. Вариант Empty проекта содержит только базовые ссылки, требуемые для MVC Framework, и базовую структуру папок.

В проекте MVC применяются два вида соглашений. Соглашения первого вида - это просто предположения о том, как может выглядеть структура проекта.

Например, общепринято размещать файлы JavaScript в папке Scripts. Здесь их рассчитывают обнаружить другие разработчики, использующие MVC, и сюда будут устанавливаться пакеты NuGet. Однако вы вольны переименовать папку Scripts или вообще удалить ее - разместив файлы сценариев где-то в другом месте. Это не помешает инфраструктуре MVC Framework запустить ваше приложение при условии, что элементы <script> внутри представлений ссылаются на местоположение, в котором находятся файлы сценариев.

Соглашения второго вида произрастают из принципа соглашения по конфигурации (convention over configuration) или соглашения над конфигурацией, если делать акцент на преимуществе соглашения перед конфигурацией, который был одним из главных аспектов, обеспечивших популярность платформе Ruby on Rails. Соглашение по конфигурации означает, что вы не должны явно конфигурировать, к примеру, ассоциации между контроллерами и их представлениями. Нужно просто следовать определенному соглашению об именовании для файлов - и все будет работать. При соглашении такого рода снижается гибкость в изменении структуры проекта. В последующих разделах объясняются соглашения, которые используются вместо конфигурации.

Классы контроллеров должны иметь имена, заканчивающиеся на Controller, например, ProductController, ShopController. При ссылке на контроллер где-либо в проекте, скажем, при вызове вспомогательного метода HTML, указывается первая часть имени (такая как Product), а инфраструктура MVC Framework автоматически добавляет к этому имени слово Controller и начинает поиск класса контроллера.

Представления и частичные представления располагаются в папке:
/Views/ИмяКонтроллера

3. JS и ООП. Реализация ООП в ES5 и ES2015+.

Пространство имён — это контейнер, который позволяет разработчикам собрать функциональность под уникальным именем приложения. Пространство имён в JavaScript — это объект, содержащий методы, свойства и другие объекты.

Важно отметить, что на уровне языка в JavaScript нет разницы между пространством имён и любым другим объектом. Это отличает JS от множества других объектноориентированных языков и может стать причиной путаницы у начинающих JS программистов.

Принцип работы пространства имён в JS прост: создать один глобальный объект и все переменные, методы и функции объявлять как свойства этого объекта. Также использование пространств имён снижает вероятность возникновения конфликтов имён в приложении так как каждый объект приложения является свойством глобального объекта.

В JavaScript есть несколько объектов, встроенных в ядро, например Math, Object, Array и String.

Объекты, создаваемые пользователем

Класс

JavaScript — это прототипно-ориентированный язык, и в нём нет оператора `class`, который имеет место в C++ или Java. Вместо этого JavaScript использует функции как конструкторы классов. Объявить класс так же просто как объявить функцию. В примере ниже мы объявляем новый класс Person с пустым конструктором:

```
var Person = function () {};
```

Объект (экземпляр класса)

Для создания нового экземпляра объекта `obj` мы используем оператор `new obj`, присваивая результат (который имеет тип `obj`) в переменную.

В примере выше мы определили класс `Person`. В примере ниже мы создаём два его экземпляра (`person1` и `person2`).

```
var person1 = new Person();
var person2 = new Person();
```

Ознакомьтесь с `Object.create()`, новым, дополнительным методом инстанцирования, который создаёт неинициализированный экземпляр.

Конструктор

Конструктор вызывается в момент создания экземпляра класса (в тот самый момент, когда создается объект). Конструктор является методом класса. В JavaScript функция служит конструктором объекта, поэтому нет необходимости явно определять метод конструктор. Любое действие определенное в конструкторе будет выполнено в момент создания экземпляра класса.

Конструктор используется для задания свойств объекта или для вызова методов, которые подготовят объект к использованию. Добавление методов и их описаний производится с использованием другого синтаксиса, описанного далее в этой статье.

В примере ниже, конструктор класса `Person` выводит в консоль сообщение в момент создания нового экземпляра `Person`.

```
var Person = function () {
    console.log('instance created');
};

var person1 = new Person();
var person2 = new Person();
```

Свойство (аттрибут объекта)

Свойства — это переменные, содержащиеся в классе; каждый экземпляр объекта имеет эти свойства. Свойства устанавливаются в конструкторе (функции) класса, таким образом они создаются для каждого экземпляра.

Ключевое слово `this`, которое ссылается на текущий объект, позволяет вам работать со свойствами класса. Доступ (чтение и запись) к свойствам снаружи класса осуществляется синтаксисом `InstanceName.Property`, так же как в C++, Java и некоторых других языках. (Внутри класса для получения и изменения значений свойств используется синтаксис `this.Property`)

В примере ниже, мы определяем свойство `firstName` для класса `Person` при создании экземпляра:

```
var Person = function (firstName) {
    this.firstName = firstName;
    console.log('Person instantiated');
};

var person1 = new Person('Alice');
var person2 = new Person('Bob');

// Выводит свойство firstName в консоль
console.log('person1 is ' + person1.firstName); // выведет "person1 is Alice"
console.log('person2 is ' + person2.firstName); // выведет "person2 is Bob"
```

Методы

Методы — это функции (и определяются как функции), но с другой стороны следуют той же логике, что и свойства. Вызов метода похож на доступ к свойству, но вы добавляете () на конце имени метода, возможно, с аргументами. Чтобы объявить метод, присвойте функцию в именованное свойство свойства `prototype` класса. Потом вы сможете вызвать метод объекта под тем именем, которое вы присвоили функции.

В примере ниже мы определяем и используем метод `sayHello()` для класса `Person`.

```
var Person = function (firstName) {
    this.firstName = firstName;
};

Person.prototype.sayHello = function() {
    console.log("Hello, I'm " + this.firstName);
};

var person1 = new Person("Alice");
var person2 = new Person("Bob");

// вызываем метод sayHello() класса Person
person1.sayHello(); // выведет "Hello, I'm Alice"
person2.sayHello(); // выведет "Hello, I'm Bob"
```

В JavaScript методы это — обычные объекты функций, связанные с объектом как свойства: это означает, что вы можете вызывать методы "вне контекста".

Рассмотрим следующий пример:

```
var Person = function (firstName) {
    this.firstName = firstName;
};

Person.prototype.sayHello = function() {
    console.log("Hello, I'm " + this.firstName);
};

var person1 = new Person("Alice");
var person2 = new Person("Bob");
var helloFunction = person1.sayHello;

// выведет "Hello, I'm Alice"
person1.sayHello();

// выведет "Hello, I'm Bob"
person2.sayHello();
```

```
// выведет "Hello, I'm undefined" (or fails
// with a TypeError in strict mode)
helloFunction();

// выведет true
console.log(helloFunction === person1.sayHello);

// выведет true
console.log(helloFunction === Person.prototype.sayHello);

// выведет "Hello, I'm Alice"
helloFunction.call(person1);
```

Как показывает пример, все ссылки, которые мы имеем на функцию `sayHello` — `person1`, `Person.prototype`, переменная `helloFunction` и т.д. — ссылаются на одну и ту же функцию. Значение `this` в момент вызова функции зависит от того, как мы её вызываем. Наиболее часто мы обращаемся к `this` в выражениях, где мы получаем функцию из свойства объекта — `person1.sayHello()` — `this` устанавливается на объект, из которого мы получили функцию (`person1`), вот почему `person1.sayHello()` использует имя "Alice", а `person2.sayHello()` использует имя "Bob". Но если вызов будет совершён иначе, то `this` будет иным: вызов `this` из переменной — `helloFunction()` — установит `this` на глобальный объект (`window` в браузерах). Так как этот объект (вероятно) не имеет свойства `firstName`, функция выведет "Hello, I'm undefined" (так произойдёт в нестрогом режиме; в `strict mode` всё будет иначе (ошибка), не будем сейчас вдаваться в подробности, чтобы избежать путаницы). Или мы можем указать `this` явно с помощью `Function#call` (или `Function#apply`) как показано в конце примера.

Наследование

Наследование — это способ создать класс как специализированную версию одного или нескольких классов (JavaScript поддерживает только одиночное наследование). Специализированный класс, как правило, называют потомком, а другой класс родителем. В JavaScript наследование осуществляется присвоением экземпляра класса родителя классу потомку. В современных браузерах вы можете реализовать наследование с помощью `Object.create`.

В примере ниже мы определяем класс `Student` как потомка класса `Person`. Потом мы переопределяем метод `sayHello()` и добавляем метод `addGoodBye()`.

```
// Определяем конструктор Person
var Person = function(firstName) {
```

```

        this.firstName = firstName;
    };

    // Добавляем пару методов в Person.prototype
    Person.prototype.walk = function(){
        console.log("I am walking!");
    };

    Person.prototype.sayHello = function(){
        console.log("Hello, I'm " + this.firstName);
    };

    // Определяем конструктор Student
    function Student(firstName, subject) {
        // Вызываем конструктор родителя, убедившись (используя Function#call)
        // что "this" в момент вызова установлен корректно
        Person.call(this, firstName);

        // Инициируем свойства класса Student
        this.subject = subject;
    };

    // Создаём объект Student.prototype, который наследуется от Person.prototype.
    // Примечание: Расспространённая ошибка здесь, это использование "new Person()", чтобы создать
    // Student.prototype. Это неверно по нескольким причинам, не в последнюю очередь
    // потому, что нам нечего передать в Person в качестве аргумента "firstName"
    // Правильное место для вызова Person показано выше, где мы вызываем
    // его в конструкторе Student.
    Student.prototype = Object.create(Person.prototype); // Смотрите примечание выше

    // Устанавливаем свойство "constructor" для ссылки на класс Student
    Student.prototype.constructor = Student;

    // Заменяем метод "sayHello"
    Student.prototype.sayHello = function(){
        console.log("Hello, I'm " + this.firstName + ". I'm studying "
            + this.subject + ".");
    };

    // Добавляем метод "sayGoodBye"
    Student.prototype.sayGoodBye = function(){
        console.log("Goodbye!");
    };

    // Пример использования:
    var student1 = new Student("Janet", "Applied Physics");
    student1.sayHello(); // "Hello, I'm Janet. I'm studying Applied Physics."
    student1.walk(); // "I am walking!"
    student1.sayGoodBye(); // "Goodbye!"

    // Проверяем, что instanceof работает корректно
    console.log(student1 instanceof Person); // true
    console.log(student1 instanceof Student); // true

```

Относительно строки `Student.prototype = Object.create(Person.prototype);`: В старых движках JavaScript, в которых нет `Object.create` можно использовать полифилл

(ещё известный как "shim") или функцию которая достигает тех же результатов, такую как:

```
function createObject(proto) {  
    function ctor() {}  
    ctor.prototype = proto;  
    return new ctor();  
}  
  
// Пример использования:  
Student.prototype = createObject(Person.prototype);
```

Инкапсуляция

В примере выше классу `Student` нет необходимости знать о реализации метода `walk()` класса `Person`, но он может его использовать; Класс `Student` не должен явно определять этот метод, пока мы не хотим его изменить. Это называется **инкапсуляция**, благодаря чему каждый класс собирает данные и методы в одном блоке.

Скрытие информации распространённая особенность, часто реализуемая в других языках программирования как приватные и защищённые методы/ свойства. Однако в JavaScript можно лишь имитировать нечто подобное, это не является необходимым требованием объектно-ориентированного программирования.²

Абстракция

Абстракция это механизм который позволяет смоделировать текущий фрагмент рабочей проблемы, с помощью наследования (специализации) или композиции. JavaScript достигает специализации наследованием, а композиции возможностью экземплярам класса быть значениями атрибутов других объектов.

В JavaScript класс `Function` наследуется от класса `Object` (это демонстрирует специализацию), а свойство `Function.prototype` это экземпляр класса `Object` (это демонстрирует композицию).

```
var foo = function () {};  
  
// выведет "foo is a Function: true"  
console.log('foo is a Function: ' + (foo instanceof Function));  
  
// выведет "foo.prototype is an Object: true"  
console.log('foo.prototype is an Object: ' + (foo.prototype instanceof Object));
```

Полиморфизм

Так как все методы и свойства определяются внутри свойства `prototype`, различные классы могут определять методы с одинаковыми именами; методы находятся в области видимости класса в котором они определены, пока два класса не имеют связи родитель-потомок (например, один наследуется от другого в цепочке наследований).

Билет 9

1. JS. Область определения. Объявление переменных (ES5 и ES2015). Замыкание. Примеры.

В JavaScript есть два типа области видимости:

- Локальная сфера
- Глобальная сфера

JavaScript имеет область действия функции: каждая функция создает новую область действия. Область действия определяет доступность (видимость) этих переменных. Переменные, определенные внутри функции, не доступны (не видны) снаружи функции.

Локальные переменные JavaScript

Переменные, объявленные внутри функции JavaScript, становятся ЛОКАЛЬНЫМИ для этой функции.

Доступ к локальным переменным можно получить только из функции.

Поскольку локальные переменные распознаются только внутри функций то переменные с одинаковыми именами могут использоваться в разных функциях.

Локальные переменные создаются при запуске функции и удаляются при ее завершении.

Переменная, объявленная вне функции, становится GLOBAL.

Глобальные переменные JavaScript

Глобальная переменная имеет глобальную область действия: все скрипты и функции на веб-странице могут получить к ней доступ.

Переменные javaScript

В JavaScript объекты и функции также являются переменными.

Область действия определяет доступность переменных, объектов и функций из разных частей кода.

Автоматически Глобальная

Если вы присвоите значение переменной, которая не была объявлена, она автоматически станет глобальной переменной.

Глобальные переменные в HTML

В JavaScript глобальная область-это полная Среда JavaScript.

В HTML глобальной областью является объект window. Все глобальные переменные принадлежат объекту window.

Время жизни переменной JavaScript начинается, когда она объявлена.

Локальные переменные удаляются после завершения функции.

В веб-браузере глобальные переменные удаляются при закрытии окна браузера (или вкладки), но остаются доступными для новых страниц, загруженных в то же окно. Аргументы функций (параметры) работают как локальные переменные внутри функций.

Лексическое окружение

Все переменные внутри функции – это свойства специального внутреннего объекта LexicalEnvironment, который создаётся при её запуске.

Мы будем называть этот объект «лексическое окружение» или просто «объект переменных».

При запуске функция создаёт объект LexicalEnvironment, записывает туда аргументы, функции и переменные. Процесс инициализации выполняется в том же порядке, что и для глобального объекта, который, вообще говоря, является частным случаем лексического окружения.

В отличие от window, объект LexicalEnvironment является внутренним, он скрыт от прямого доступа.

Доступ ко внешним переменным

Из функции мы можем обратиться не только к локальной переменной, но и к внешней.

Интерпретатор, при доступе к переменной, сначала пытается найти переменную в текущем LexicalEnvironment, а затем, если её нет – ищет во внешнем объекте переменных. В данном случае им является window.

Такой порядок поиска возможен благодаря тому, что ссылка на внешний объект переменных хранится в специальном внутреннем свойстве функции, которое называется `[[Scope]]`. Это свойство закрыто от прямого доступа, но знание о нём очень важно для понимания того, как работает JavaScript.

При создании функция получает скрытое свойство `[[Scope]]`, которое ссылается на лексическое окружение, в котором она была создана.

Если обобщить:

Каждая функция при создании получает ссылку `[[Scope]]` на объект с переменными, в контексте которого была создана.

При запуске функции создаётся новый объект с переменными `LexicalEnvironment`. Он получает ссылку на внешний объект переменных из `[[Scope]]`.

При поиске переменных он осуществляется сначала в текущем объекте переменных, а потом – по этой ссылке.

Замыкание – это функция вместе со всеми внешними переменными, которые ей доступны.

Тем не менее, в JavaScript есть небольшая терминологическая особенность.

Обычно, говоря «замыкание функции», подразумевают не саму эту функцию, а именно внешние переменные.

Иногда говорят «переменная берётся из замыкания». Это означает – из внешнего объекта переменных.

«Понимать замыкания» в JavaScript означает понимать следующие вещи: Все переменные и параметры функций являются свойствами объекта переменных `LexicalEnvironment`. Каждый запуск функции создаёт новый такой объект. На верхнем уровне им является «глобальный объект», в браузере – `window`.

При создании функция получает системное свойство `[[Scope]]`, которое ссылается на `LexicalEnvironment`, в котором она была создана.

При вызове функции, куда бы её ни передали в коде – она будет искать переменные сначала у себя, а затем во внешних `LexicalEnvironment` с места своего «рождения».

Область видимости (`scope`) определяет видимость или доступность переменной или другого ресурса в коде.

В JavaScript есть только одна **глобальная область**. Область за пределами всех функций считается глобальной областью, и переменные, определенные в

глобальной области, могут быть доступны и изменены в любых других областях.

```
var num = 5;  
console.log(num);  
function getNum(){  
    console.log(num);  
}  
getNum();
```

Переменные, объявленные внутри функций, становятся **локальными** для функции и рассматриваются в соответствующей **локальной области**. Каждая функция имеет свою область видимости. Одна и та же переменная может использоваться в разных функциях, поскольку они связаны с соответствующими функциями и не являются взаимно видимыми.

```
// Глобальная область  
function foo1(){  
    // Локальная область 1  
    function foo2(){  
        // Локальная область 2  
    }  
}  
  
// Глобальная область  
function foo3(){  
    // Локальная область 3  
}  
  
// Глобальная область
```

Локальная область видимости может быть разделена на **область видимости функции** и **область видимости блока**. Концепция **область видимости блока** или **block scope** была представлена в ECMAScript6 (ES6) вместе с новыми способами объявления переменных - const и let.

Всякий раз, когда мы объявляем переменную в функции, переменная видна только внутри функции. Мы не можем получить к ней доступ вне функции. var - это ключевое слово, определяющее переменную для доступности области функций.

```
function foo(){  
    var num = 10;  
    console.log('inside function: ', num);  
}  
foo();           // inside function: 10  
console.log(num); // ReferenceError: num is not defined
```

[[Scope]] - это скрытое внутреннее свойство функции, которое она получает во время вызова. Данное свойство содержит ссылку на ту область видимости, в которой данная функция была объявлена.

JavaScript всегда начинает поиск переменной или функции с текущей области видимости. Если она в ней не будет найдена, то интерпретатор переместится к следующей области, указанной в [[Scope]], и попробует отыскать её там. Последовательность областей видимости, которые интерпретатор использует при разрешении имени идентификатора, называется в JavaScript цепочкой областей видимости (scope chain).

Функции вида function declaration можно использовать в JavaScript до их объявления. Это происходит из-за того что они поднимаются (**hoisting**) или другими словами «перемещаются» в начало текущего контекста. Но поднятие в JavaScript выполняется не только функций вида function declaration, но и переменных, объявленных с помощью ключевого слова var. При этом поднятие осуществляется только самого объявления переменной.

Область видимости блока - это область в условиях if и switch или циклов for, и while. Вообще говоря, всякий раз, когда мы видим фигурные скобки {} - это блок. В ES6 ключевые слова const и let позволяют разработчикам объявлять переменные в **области видимости блока**, что означает, что эти переменные существуют только в соответствующем блоке.

```
function foo(){  
    if (true) {  
        var num1 = 5;      // существуют в области видимости функции  
        const num2 = 10;   // существуют в области видимости блока  
        let num3 = 23;    // существуют в области видимости блока  
    }  
    console.log(num1);
```

```
console.log(num2);
console.log(num3);
}
foo();
// 5
// ReferenceError: num2 is not defined
// ReferenceError: num3 is not defined
```

Ещё один момент, о котором стоит упомянуть - это **лексическая область**. **Лексическая область** означает, что дочерняя область имеет доступ к переменным, определенным в родительской области. Дочерние функции лексически связаны с контекстом исполнения их родителей.

```
function foo1(){
var num1 = 5;
const num2 = 10;
let num3 = 23;
function foo2(){
console.log(num1);
console.log(num2);
console.log(num3);
}
foo2();
}
foo1();
// 5
// 10
// 23
```

Лексическая область видимости - это набор правил о том, как и где движок JavaScript может найти переменную. Ключевой характеристикой лексического контекста является то, что он определяется во время написания кода (при условии, что мы не используем eval () или with).

Динамическая область видимости, по понятным причинам, подразумевает, что существует модель, в которой область видимости может определяться динамически во время выполнения, а не статически во время создания. **Динамическая область видимости** не связана с тем, как и где объявляются функции и области, а связана с тем, откуда они вызываются. Другими словами, цепочка областей видимости основана на стеке вызовов, а не на вложении областей видимости в коде.

Но **JavaScript, на самом деле, не имеет динамической области видимости**. Он имеет только **лексическую область**. А вот механизм this подобен **динамической области видимости**.

Автоматически Глобальная

Если вы присвоите значение переменной, которая не была объявлена, она автоматически станет глобальной переменной.

В HTML глобальной областью является объект window. Все глобальные переменные принадлежат объекту window.

Время жизни переменной JavaScript начинается, когда она объявлена. Локальные переменные удаляются после завершения функции. В веб-браузере глобальные переменные удаляются при закрытии окна браузера (или вкладки), но остаются доступными для новых страниц, загруженных в то же окно.

- -----

Объявление переменной является нужным, так как вам нужно это сделать перед тем как использовать ее. Объявленные переменные инициализируются до выполнения любого кода. Необъявленные переменные не существуют до тех пор, пока к ним не выполнено присваивание.

Синтаксис **ES5** использует ключевое слово **var** для объявления переменной. Оператор **var** объявляет переменную, инициализируя ее, при необходимости. Область видимости переменной, объявленной через var, это её текущий контекст выполнения. Который может ограничиваться функцией или быть глобальным, для переменных, объявленных за пределами функции.

```
// Declare variables:  
  
var a1;  
  
console.log(a1); // undefined  
  
var a2 = 100;  
  
console.log(a2); // 100
```

ES6(ES-2015) добавляет 2 новых ключевых слова это **let & const** чтобы объявить переменную.

Директива **let** объявляет переменную с блочной областью видимости с возможностью инициализировать её значением. отличие от ключевого слова var, которое объявляет переменную глобально или локально во всей функции, независимо от области блока.

```
let b1;  
console.log(b1); // undefined  
let b2 = "Hello";  
console.log(b2); // Hello
```

Ключевое слово **const** используется для объявления константы (constant). Значение констант не может быть изменено новым присваиванием, а также не может быть переопределено. Константы (const) подчиняются области видимости уровня блока так же, как переменные, объявленные с использованием ключевого слова let.

Это объявление создаёт константу, чья область действия может быть как глобальной, так и локальной внутри блока, в котором она объявлена. Глобальные константы не становятся свойствами объекта window, в отличие от var-переменных. Инициализация константы обязательна; необходимо указать значение одновременно с объявлением (смысл в том, что потом это значение изменить уже нельзя).

```
// Declare a constant with a value  
const greeting = "Hello";  
// Assign new value to 'greeting'  
greeting = "Hi"; // => Error!!  
  
// Declare a constant without a value  
const i ; // => Error!!
```

- -----

Замыкание – это функция вместе со всеми внешними переменными, которые ей доступны. То есть, замыкание – это функция + внешние переменные. Замыкание – это функция, которая запоминает свои внешние

переменные и может получить к ним доступ. Все функции в JavaScript являются замыканиями.

Тем не менее, в JavaScript есть небольшая терминологическая особенность.

Обычно, говоря «замыкание функции», подразумевают не саму эту функцию, а именно внешние переменные.

```
function makeCounter() {  
    let count = 0;  
    return function() {  
        return count++; // есть доступ к внешней переменной "count"  
    };  
}  
  
let counter = makeCounter();  
alert( counter() ); // 0  
alert( counter() ); // 1  
alert( counter() ); // 2
```

Переменной присваивается функция, которая берет внешнюю count, при этом после присваивания внешняя функция уже не существует, т.к. она просто вернула внутреннюю функцию при первом ее вызове. Соответственно внутрення хранит в своем замыкании ту самую переменную.

2. Унифицированный идентификатор ресурса (URI). URL. URN. Структура, применение.

URI — символьная строка, позволяющая идентифицировать какой-либо ресурс: документ, изображение, файл, службу, ящик электронной почты и т.д. Прежде всего, речь идёт о ресурсах сети Интернет и Всемирной паутины. URI предоставляет простой и расширяемый способ идентификации ресурсов. Расширяемость URI означает, что уже существуют несколько схем идентификации внутри URI, и ещё больше будет создано в будущем.

URI является либо URL, либо URN, либо одновременно обоими.

URL — это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса. А URN — это URI, который только идентифицирует ресурс в определённом пространстве имён (и, соответственно, в определённом контексте), но не указывает его

местонахождение. Например, URN urn:ISBN:0-395-36341-1 — это URI, который указывает на ресурс (книгу) 0-395-36341-1 в пространстве имён ISBN, но, в отличие от URL, URN не указывает на местонахождение этого ресурса: в нём не сказано, в каком магазине её можно купить или на каком сайте скачать. Впрочем, в последнее время появилась тенденция говорить просто URI о любой строке-идентификаторе, без дальнейших уточнений. Так что, возможно, термины URL и URN скоро уйдут в прошлое.

URI = [схема ":"] иерархическая-часть ["?" запрос] ["#" фрагмент]

(например: <https://tools.ietf.org/html/rfc3986#section-3>)

В этой записи:

схема

схема обращения к ресурсу (часто указывает на сетевой протокол), например http, ftp, file, ldap, mailto, urn

иерархическая-часть

содержит данные, обычно организованные в иерархической форме, которые, совместно с данными в неиерархическом компоненте запрос, служат для идентификации ресурса в пределах видимости URI-схемы. Обычно иер-часть содержит путь к ресурсу (и, возможно, перед ним, адрес сервера, на котором тот располагается) или идентификатор ресурса (в случае URN).

запрос

этот необязательный компонент URI описан выше.

фрагмент

(тоже необязательный компонент)

RFC 3986:

позволяет косвенно идентифицировать вторичный ресурс посредством ссылки на первичный и указанием дополнительной информации. Вторичный идентифицируемый ресурс может быть некоторой частью или подмножеством первичного, некоторым его представлением или другим ресурсом, определённым или описанным таким ресурсом. Часть идентификатора URI без схемы обращения к ресурсу часто называется «ссылкой URI» (англ. URI reference). Прецеденты применения ссылок URI имеются в HTML, XHTML, XML и XSLT. Процесс превращения ссылки URI в абсолютную форму URI называют «разрешением URI» (англ. URI resolution).

URI == URL || URN || URL + URN

scheme://host:[port]/path/.../[;url-params][?query-string][#anchor]

- scheme – протокол (http,https,ftp)
- host – ip-адрес или доменное имя
- port – порт, необязательно если стандартный
- path – путь в файловой системе корневого каталога сервера (или псевдоним, обрабатываемый сервером)
- url-params – необязательные пары ключ-значение. Используется для идентификаторов сессии пользователя.
- query-string – пары ключ-значение – параметры запроса
- anchor – ссылка на позиционный маркер в пределах документа



3. Верстка. Типы Верстки. Дизайн. Виды дизайна. CSS-фреймворки.

Вёрстка веб-страниц — создание структуры гипертекстового документа на основе HTML разметки, как правило, при использовании таблиц стилей и клиентских сценариев, таким образом, чтобы элементы дизайна выглядели аналогично макету.

Фиксированная.

Является одним из наиболее распространённых на сегодняшний день типов.

Основная особенность: сайт имеет строго фиксированную ширину и высоту всех блоков и страницы в целом, и не зависит от диагонали монитора.

Достоинства:

- простота изготовления (а следовательно и меньшая цена)
- корректное и одинаковое отображение сайта на экранах с разной диагональю.

Недостатки:

- на маленьких экранах такой сайт будет просто масштабироваться. То есть вы получите уменьшенную точную копию вашей страницы для прочтения которой её нужно будет сначала увеличить. В эпоху мобильного интернета это не самый лучший вариант.

- на широкоформатных мониторах остаётся много пустого места по бокам (особенно если ширина страницы достаточно узкая (900 пикселей, к примеру.)
- при масштабировании страницы она просто становится больше и вылезает за пределы экрана как по высоте, так и по ширине.

Обычно данный тип вёрстки используют для создания Landing Page.

Резиновая (тянущаяся).

Основная особенность: смысл этой вёрстки заключается в том что размеры всех элементов задаются не в пикселях, а в процентах. И таким образом одни и те же блоки на мониторах с разной диагональю будут иметь разный размер и возможно немного по другому отображаться.

Достоинства:

- Хорошо смотрится на экранах разной величины.
- Изменяется при масштабировании страницы.

Недостатки:

- Более сложный в реализации (а следовательно и более дорогой)
- Так как блоки сайта растягиваются и сжимаются под ширину экрана – некоторые элементы выглядят не очень красиво на планшетах и телефонах.
- На мобильных устройствах некоторые элементы за счёт сжатия становятся очень неудобными в использовании. Особенно это касается меню и форм обратной связи.

Такая вёрстка подходит для макетов сайтов, элементы которых размещены в одну колонку. Или в сочетании с фиксированной вёрсткой.

Адаптивная.

Этот тип вёрстки наиболее популярен на сегодняшний день так как позволяет создавать «живые» сайты, способные перестраивать свои элементы и изменять их размер в зависимости от того на каком устройстве вы его просматриваете.

Основная особенность:

Адаптивная вёрстка базируется на резиновой вёрстке + использует медиазапросы или специальные скрипты, которые позволяют определять размер экрана пользователя и в зависимости от этого приписывать конкретным элементам те или иные стили (ширина элементов и положение блоков, размер шрифтов и картинок и даже цвета).

Достоинства:

- Адаптируется по размеры экрана, за счёт чего ваш сайт хорошо отображается на всех видах устройств.
- Такой сайт удобно просматривать на мобильных устройствах, так как не нужно ничего увеличивать (всё само подстраивается).
- При масштабировании страницы в перестраиваются и страница не теряет свой вид.

Недостатки:

- Очень трудоёмкий вид вёрстки, что отражается на его цене.
- Контроль качества выполнения такой вёрстки тоже требует не мало времени (особенно в том случае если сайт большой) так как нужно протестировать вёрстку на разных размерах экранов через онлайн сервисы либо через масштабирование и изменение размеров окна в браузере.

Также можно выделить табличную и блочную вёрстку, одна использует html таблицы для размещения элементов, другая - div блоки (более гибкий вариант).

UI – пользовательский интерфейс, включающей в себя различные элементы: информационную архитектуру, проектирование взаимодействия, графический дизайн и контент. Это довольно узкое понятие, которое помогает пользователю взаимодействовать с определенной программой или сайтом.

UX дизайн более обобщенное понятие, частью которого является и сам UI дизайн. UX предполагает поведение человека, его отношение, эмоции, которые вызванные и связанные с использованием продукта. UX обеспечивает взаимодействие пользователей с сайтом. В переводе с оригинала термина User Experience Design расшифровывают как проектирование опыта взаимодействия.

CSS-фреймворк – фреймворк, который, прежде всего, упрощает работу верстальщика, ускоряет процесс разработки и исключает возможные ошибки. Как и библиотеки скриптовых языков программирования, CSS-библиотеки, обычно имеющие вид внешнего css-файла, «подключаются» к проекту (добавляются в заголовок вебстраницы).

Преимущества

- Позволяет неискусенному в тонкостях вёрстки программисту или дизайнеру правильно создать HTML-макет.
- Вёрстка на базе слоёв, а не таблиц.
- Более быстрая разработка.

- Кросбраузерность.
- Возможность использования генераторов кода и визуальных редакторов.
- Единообразие кода при работе в команде позволяет снизить число разногласий при разработке.

Недостатки

- Библиотеки бывают сильно «раздуты» — может быть много лишнего кода, который не будет использоваться в проекте.
- Дизайн будет зависеть от css-библиотеки.
- Из-за необходимости добавления множества классов к одному элементу нарушается принцип, ради которого и был создан CSS: разделение описаний структуры и внешнего вида.

Типы CSS-библиотек

Из-за упомянутых недостатков — использование CSS-библиотек вызывает споры в профессиональном сообществе. Также это привело к появлению различных типов CSS-библиотек. Условно можно выделить два типа: Всеобъемлющие и Ограниченнные. Третьим вариантом может быть разработка собственной библиотеки. Этот вариант предпочитает большинство разработчиков, так как это дает выгоды персонального решения и уменьшает негативные моменты зависимости от использования сторонних библиотек.

Всеобъемлющие CSS-библиотеки

Этот тип библиотеки пытается охватить большинство вещей, которые могут понадобиться разработчику. К этому типу относятся библиотеки, которые включают CSS для вёрстки и сброса (или какую-то основу).

Ограниченнные CSS-библиотеки

Как следует из названия, библиотеки этого типа охватывают лишь ограниченный объём потребностей или имеют конкретную цель.

Примеры CSS-библиотек

Всеобъемлющие

Bootstrap — библиотека, созданная разработчиками Twitter.

W3.css — CSS-фреймворк от w3schools.

Twitter Flight

Blueprint 960 Grid System — библиотека для быстрой разработки макетов.

Yet Another Multicolumn Layout (YAML) — имеет инструмент генерации кода, возможность создания адаптивных интерфейсов.

css-framework — российская библиотека для разработки веб-интерфейсов.

Foundation — продвинутая библиотека для разработки адаптивных интерфейсов.

Golden Grid System — библиотека для разработки адаптивных интерфейсов на основе золотого сечения.

Gumby Framework

Topcoat — небольшая библиотека от Adobe распространяется по лицензии Apache 2.0

Compass — CSS-фреймворк независимый от Rails

Ограниченные

jQuery UI CSS Framework

CSS-based Slideshow System — библиотека для создания презентаций.

Билет 10

1. Эволюция HTTP. Протоколы HTTP2 и HTTP3. Основные идеи, решаемые проблемы. Тенденции дальнейшего развития.

HTTP/2 (2016):

1. бинарный;
2. Server-Push (решает проблему скорости первоначальной загрузки страницы);
3. мультиплексирование.

HTTP/3 — трансляция транспортного протокола QUIC для прикладного уровня.

Название HTTP/3 официально утвердили лишь недавно, в 17-й версии черновика (draft-ietf-quic-http-17). Его предложили в конце октября 2018 года, а консенсус был достигнут на встрече IETF 103 в Бангкоке в ноябре.

Раньше HTTP/3 был известен как HTTP по QUIC, а до этого — как HTTP/2 по gQUIC, а ещё раньше — SPDY по gQUIC. Но суть в том, что HTTP/3 — просто новый синтаксис HTTP, который работает на протоколе IETF QUIC, мультиплексированном и безопасном транспорте на основе UDP.

Основная проблема - передача мультимедийного контента. HTTP/2 вводится в первую очередь для мультимедийного бинарного контента.

2. Клиент в Web. Браузер. Алгоритм исполнения типового GET-запроса на web-ресурс.

Web-клиент (англ. Web client) как программа — браузер.

Браузер, или веб-обозреватель (web browser) — прикладное программное обеспечение для просмотра веб страниц, содержания веб-документов, компьютерных файлов и их каталогов; управления веб-приложениями; а также для решения других задач. В глобальной сети браузеры используются для запроса, обработки, манипулирования и отображения содержания веб-сайтов. Многие современные браузеры также могут использоваться для обмена файлами с серверами FTP, а также для непосредственного просмотра содержания файлов многих графических форматов, аудио-видео форматов, текстовых форматов и других файлов.

Одним из способов, как можно отправить запрос по протоколу HTTP к серверу, является запрос методом GET. Этот метод является самым распространенным и запросы к серверу чаще всего происходят с его использованием.

Самый простой способ, как можно создать запрос методом GET - это набрать URL адрес в адресную строку браузера.

Браузер сформирует и передаст веб-серверу примерно следующую информацию:

GET / HTTP/1.1

Host: site.ru

User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:18.0) Gecko/20100101 Firefox/18.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/q=0.8

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

Cookie: wp-settings

Connection: keep-alive

Запрос состоит из двух частей:

1. строка запроса (Request Line)
2. заголовки (Message Headers)

Обратите внимание, что GET запрос не имеет тела сообщения. Но, это не означает, что с его помощью мы не можем передать серверу никакую информацию. Это можно делать с помощью специальных GET параметров.

Чтобы добавить GET параметры к запросу, нужно в конце URL-адреса поставить знак «?» и после него начинать задавать их по следующему правилу:

имя_параметра1=значение_параметра1& имя_параметра2=значение_параметра2&...

Разделителем между параметрами служит знак «&».

К примеру, если мы хотим передать серверу два значения, имя пользователя и его возраст, то это можно сделать следующей строкой:

<http://site.ru/page.php?name=dima&age=27>

Когда выполнен данный запрос, данные попадают в так называемую переменную окружения QUERY_STRING, из которой их можно получить на сервере с помощью серверного языка веб-программирования.

Сервер обработает запрос, после чего вернёт код ответа и содержимое.

3. История создания WWW. Основные технологии и принципы.

Тим Бернерс-Ли в 80-х годах разработал для личного пользования программу называвшуюся Enquire, дословно переводимую как «Дознаватель». Она использовала случайные ассоциации для хранения данных. Именно эта программа послужила фундаментом для Всемирной паутины . Это было почти за 9 лет до появления проекта WWW, который был представлен в 1989 году Тимом Бернерсом.

В то время Тим работал в CERN (Европейском совете по ядерным исследованиям) и разрабатывал для них внутреннюю сеть. В 1989 году он представил проект, предназначенный для публикации гипертекстовых документов, связанных между собой гиперссылками, что облегчило бы поиск и консолидацию информации для учёных CERN.

Для этой технологии были разработаны такие стандарты: протокол HTTP (Hyper Text Transfer Protocol), идентификаторы URL и язык текстовой разметки HTML.

Официальный год рождения сети – 1989 год, хотя данные стандарты Бернерс совершенствовал вплоть до 1993 года.

В 1994 году был образован «Консорциум Всемирной паутины (World Wide Web Consortium, W3C)», который действует до сих пор.

Технология WWW позволяет создавать ссылки, которые реализуют переходы не только внутри исходного документа, но и на любой другой документ, находящийся

на данном компьютере и на любой документ любого компьютера, подключенного к Интернету.

В 1989 году Тим Бернерс-Ли предложил свой проект гипертекстовой системы, согласно которой нажатие на ссылку вызывает переход на требуемый документ или фрагмент документа. В качестве указателей ссылок, то есть объектов, активизация которых вызывает переход на другой документ, могут использоваться не только фрагменты текста, но и графические изображения.

Серверы Интернета, реализующие WWW-технологию, называются Web-серверами, а документы, реализованные по технологии WWW, называются Web-страницами.

Создание Web-страниц осуществляется с помощью языка разметки гипертекста (Hyper Text Markup Language - HTML). Основа используемой в HTML технологии состоит в том, что в обычный текстовый документ вставляются управляющие символы (тэги). В результате текстовый документ в браузере выглядит как Web-страница.

Базовым кирпичиком для WWW является компьютер с установленным на нём ВЕБсервером подключённый к сети, то есть к другим компьютерам. ВЕБ-сервер программа, запускаемая на подключённом к сети компьютере, и использующий протокол HTTP для передачи данных. В простейшем виде такая программа получает по сети HTTP-запрос на определённый ресурс, находит соответствующий файл на локальном жёстком диске и отправляет его по сети запросившему компьютеру. Более сложные веб-серверы способны динамически формировать ресурсы в ответ на HTTPзапрос.

Для идентификации ресурсов во Всемирной паутине используются единообразные идентификаторы ресурсов URI (Uniform Resource Identifier). Для определения местонахождения ресурсов в сети используются единообразные локаторы ресурсов URL (Uniform Resource Locator). Такие URL-локаторы сочетают в себе технологию идентификации URL и систему доменных имён DNS (Domain Name System) — доменное имя (или непосредственно IP-адрес в числовой записи) входит в состав URL для обозначения компьютера (точнее — одного из его сетевых интерфейсов), который исполняет код нужного веб-сервера.

Для просмотра информации, полученной от веб-сервера, на клиентском компьютере применяется специальная программа — веб-браузер. Основная функция веб-браузера — отображение гипертекста. Гипертекст — это текст, размеченный языком гипертекстовой разметки HTML, после HTML-разметки получившийся гипертекст помещается в файл, такой HTML-файл является самым распространённым ресурсом Всемирной паутины.

После того, как HTML-файл становится доступен веб-серверу, его начинают называть «веб-страницей». Набор веб-страниц образует веб-сайт. В гипертекст веб-страниц добавляются гиперссылки.

Гиперссылки, основанные на технологии URL, помогают пользователям Всемирной паутины легко перемещаться между ресурсами (файлами) вне зависимости от того, находятся ресурсы на локальном компьютере или на удалённом сервере.

Билет 11

1. JS (ES5). Базовые типы данных.

В JS используются следующие типы данных:

1. Number;
2. Boolean;
3. String;
4. Undefined;
5. Null;
6. Object (Array, Set, Map, RegExp,...);
7. Function;
8. Symbol;
9. BigInt.

Присвоение значений осуществляется с помощью команды let. Например: let a = 5

Раньше вместо let использовалось var.

Примитив

Это – значение «примитивного» типа.

Есть 6 примитивных типов: string, number, boolean, symbol, null и undefined.

Объект

Может хранить множество значений как свойства.

Объявляется при помощи фигурных скобок {}, например: {name: "Рома", age:

30}. В JavaScript есть и другие виды объектов: например, функции тоже являются объектами.

В JavaScript есть разные типы кавычек.

Строчку можно создать с помощью одинарных, двойных либо обратных кавычек:

```
let single = 'single-quoted';
let double = "double-quoted";
let backticks = `backticks`;
```

Одинарные и двойные кавычки работают, по сути, одинаково, а если использовать обратные кавычки, то в такую строку мы сможем вставлять произвольные выражения, обернув их в \${...}:

```
function sum(a, b) {
  return a + b;
}

alert(`1 + 2 = ${sum(1, 2)} // 1 + 2 = 3.
```

Ещё одно преимущество обратных кавычек — они могут занимать более одной строки, вот так:

```
let guestList = `Guests:
John
Pete
Mary
`;

alert(guestList); // список гостей, состоящий из нескольких строк
```

Существует два варианта синтаксиса для создания пустого массива:

```
let arr = new Array();
let arr = []; Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:
```

```
let fruits = ["Яблоко", "Апельсин", "Слива"];
```

Элементы массива нумеруются, начиная с нуля.

Мы можем получить элемент, указав его номер в квадратных скобках:

```
let fruits = ["Яблоко", "Апельсин", "Слива"];
alert(fruits[0]); // Яблоко
```

```
alert( fruits[1] ); // Апельсин  
alert( fruits[2] ); // Слива
```

«Символ» представляет собой уникальный идентификатор.

Создаются новые символы с помощью функции `Symbol()`:

```
// Создаём новый символ - id
```

```
let id = Symbol();
```

При создании символу можно дать описание (также называемое имя), в основном использующееся для отладки кода:

```
// Создаём символ id с описанием (именем) "id"
```

```
let id = Symbol("id");
```

Символы гарантированно уникальны. Даже если мы создадим множество символов с одинаковым описанием, это всё равно будут разные символы. Описание – это просто метка, которая ни на что не влияет.

2. HTML. Атрибуты.

(Глобальные атрибуты)

<https://html5book.ru/html-attributes/>

Атрибут — используется для определения характеристик html-элемента и помещается внутри открытого тега элемента.

Все атрибуты состоят из двух частей — это имя и значение:

`<p align="left"></p>` - align(имя) "left"(значение)

3. Передача данных по HTTP. СерIALIZАЦИЯ. ФОРМАТЫ СЕРИАЛИЗАЦИИ (XML, JSON, Protobuf).

HTTP — текстовый протокол, из-за этого бинарные данные нужно привести к текстовому представлению — сериализовать. Наиболее понятный способ — использование кодировки Base64.

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализации (структуризации) — восстановление начального состояния структуры данных из битовой последовательности.

Сериализация используется для передачи объектов по сети и для сохранения их в файлы.

При XML-сериализации в поток XML сериализуются только открытые поля и значения свойств объекта. XML-сериализация не учитывает информацию о типе. Например, если имеется объект

Book, который существует в пространстве имен **Library**, нет никакой гарантии, что он десериализуется в объект аналогичного типа.

XML-сериализация не выполняет преобразование методов, индексаторов, закрытых полей или свойств только для чтения (кроме коллекций только для чтения).

JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми.

За счёт своей лаконичности по сравнению с XML формат JSON может быть более подходящим для сериализации сложных структур. Применяется в веб-приложениях как для обмена данными между браузером и сервером (AJAX), так и между серверами (программные HTTP-сопряжения).

<https://ru.wikipedia.org/wiki/JSON#Использование>

Protocol Buffers — протокол сериализации (передачи) структурированных данных, предложенный Google как эффективная бинарная альтернатива текстовому формату XML. Разработчики сообщают, что Protocol Buffers проще, компактнее и быстрее, чем XML, поскольку осуществляется передача бинарных данных, оптимизированных под минимальный размер сообщения.

https://ru.wikipedia.org/wiki/Protocol_Buffers

Билет 12

1. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.

Наиболее распространенные виды MVC-паттерна, это:

Model-View-Controller

Model-View-Presenter

Model-View-View Model

(Model View Intent)(какой-то таинственный про который мало пишут)

Model-Template-View - Django-взгляд на MVC (на самом деле, классический MVC, только

V == T, а C == V)

Семейство MVC-паттернов (по лекциям):

1. MVC;
2. MTV;
3. MPV;
4. MVVM;
5. MVPVM.

MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Компоненты MVC:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.
2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
3. Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC (Fat Stupid Ugly Controllers).

MVC — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер.

Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.



2. CSS. Правила каскадирования и наследования.

Стили, присвоенные некоторому элементу, наследуются всеми потомками (вложенными элементами), если они не переопределены явно.

Правила каскадирования позволяют разрешать ситуации, когда для одного элемента прописано несколько стилей. Каскадирование основано на присвоении некоторого приоритета каждому правилу.

Правила каскадирования определяют следующие приоритеты:

- пользовательские стили, отмеченные !important
- авторские стили, отмеченные !important
- авторские стили
- пользовательские стили
- стили по умолчанию

После каскадирования правила упорядочиваются на основе специфичности селекторов.

Специфичность — это некоторое число в системе с неопределенным основанием, которое является отражением приоритета какого-либо правила.

Наследуемые и не наследуемые:

<https://developer.mozilla.org/ru/docs/Web/CSS/inheritance>

3. HTML. Элементы. Виды элементов.

<https://zametkinapolyah.ru/verstka-sajtov/vidy-html-elementov-blochnye-element-i-strochnye-html-elementy.html>

HTML элементы – это то, что пользователь видит на странице в браузере

HTML тэги – это то, что разработчик пишет, когда создает HTML документ. HTML элементы делятся на два вида: блочные HTML элементы и строчные HTML элементы. Первые чаще всего используются для создания структуры HTML страницы (не стоит путать со структурой HTML документа), вторые чаще всего применяются для оформления и логического выделения контента на странице.

Блочные HTML элементы чаще всего используются для создания структуры HTML страниц или для логического разбиения HTML документа на части.

Типичным примером блочных HTML элементов являются:

HTML тэг `<div>`, который создает блоки на странице;

HTML тэг `<p>`, который делит HTML документ на параграфы или абзацы;

тэги HTML списка: ``, ``, `` и другие;

HTML заголовки.

Строчные HTML элементы

Строчные HTML элементы – это такие HTML элементы, ширина которых, равна ширине их содержимого.

Строчные HTML элементы идут друг за другом, это означает, что если закончился строчный HTML элемент, то в этой же строке может начаться следующий строчный HTML элемент. Опять же, если HTML страница это стена, то строчные HTML элемент – это кирпичи, из которых сделана стена.

В качестве примера строчных HTML элементов можно привести:

HTML тэг `<a>`, с помощью которого мы можем создавать ссылки;

HTML тэг ``, позволяющий оформлять участки текста внутри параграфа;

HTML тэг ``, который говорит браузеру о том, что текст является важным и браузер выделяет этот текст курсивом;

HTML тэг ``, так же говорит, что текст важен, но браузер выделяет такой текст жирным.

Билет 13

1. История развития сети Интернет.

Тим Бернерс-Ли в 80-х годах разработал для личного пользования программку называвшуюся Enquire, дословно переводимую как «Дознаватель». Она использовала случайные ассоциации для хранения данных. Именно эта программка послужила фундаментом для Всемирной паутины . Это было почти за 9 лет до появления проекта WWW, который был представлен в 1989 году Тимом Бернерсом.

В то время Тим работал в CERN (Европейском совете по ядерным исследованиям) и разрабатывал для них внутреннюю сеть. В 1989 году он представил проект, предназначенный для публикации гипертекстовых документов, связанных между собой гиперссылками, что облегчило бы поиск и консолидацию информации для учёных CERN.

Для этой технологии были разработаны такие стандарты: протокол HTTP (Hyper Text Transfer Protocol), идентификаторы URL и язык текстовой разметки HTML.

Официальный год рождения сети – 1989 год, хотя данные стандарты Бернерс совершенствовал вплоть до 1993 года.

В 1994 году был образован «Консорциум Всемирной паутины (World Wide Web Consortium, W3C)», который действует до сих пор.

Технология WWW позволяет создавать ссылки, которые реализуют переходы не только внутри исходного документа, но и на любой другой документ, находящийся на данном компьютере и на любой документ любого компьютера, подключенного к Интернету.

В 1989 году Тим Бернерс-Ли предложил свой проект гипертекстовой системы, согласно которой нажатие на ссылку вызывает переход на требуемый документ или фрагмент документа. В качестве указателей ссылок, то есть объектов, активизация которых вызывает переход на другой документ, могут использоваться на только фрагменты текста, но и графические изображения.

Серверы Интернета, реализующие WWW-технологию, называются Web-серверами, а документы, реализованные по технологии WWW, называются Web-страницами.

Создание Web-страниц осуществляется с помощью языка разметки гипертекста (Hyper Text Markup Language - HTML). Основа используемой в HTML технологии состоит в том, что в обычный текстовый документ вставляются управляющие символы (тэги). В результате текстовый документ в браузере выглядит как Web-страница.

Базовым кирпичиком для WWW является компьютер с установленным на нём ВЕБсервером подключённый к сети, то есть к другим компьютерам. ВЕБ-сервер программа, запускаемая на подключённом к сети компьютере, и использующей протокол HTTP для передачи данных. В простейшем виде такая программа получает по сети HTTP-запрос на определённый ресурс, находит соответствующий файл на локальном жёстком диске и отправляет его по сети запросившему компьютеру. Более сложные веб-серверы способны динамически формировать ресурсы в ответ на HTTPзапрос.

Для идентификации ресурсов во Всемирной паутине используются единообразные идентификаторы ресурсов URI (Uniform Resource Identifier). Для определения местонахождения ресурсов в сети используются единообразные локаторы ресурсов URL (Uniform Resource Locator). Такие URL-локаторы сочетают в себе технологию идентификации URL и систему доменных имён DNS (Domain Name System) — доменное имя (или непосредственно IP-адрес в числовой записи) входит в состав URL для обозначения компьютера (точнее — одного из его сетевых интерфейсов), который исполняет код нужного веб-сервера.

Для просмотра информации, полученной от веб-сервера, на клиентском компьютере применяется специальная программа — веб-браузер. Основная функция веб-браузера — отображение гипертекста. Гипертекст — это текст, размеченный языком гипертекстовой разметки HTML, после HTML-разметки получившийся гипертекст помещается в файл, такой HTML-файл является самым распространённым ресурсом Всемирной паутины.

После того, как HTML-файл становится доступен веб-серверу, его начинают называть «веб-страницей». Набор веб-страниц образует веб-сайт. В гипертекст веб-страниц добавляются гиперссылки.

Гиперссылки, основанные на технологии URL, помогают пользователям Всемирной паутины легко перемещаться между ресурсами (файлами) вне зависимости от того, находятся ресурсы на локальном компьютере или на удалённом сервере.

2. Технологии толстого клиента. Пример с отображением списка пользователей на JS/JQuery

Толстый клиент, rich client архитектуре клиент-сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) полную функциональность и независимость от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства

- Толстый клиент обладает широкой функциональностью в отличие от тонкого.
- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Высокое быстродействие (зависит от аппаратных средств клиента)

Недостатки

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.
- Наличие бизнес-логики

jQuery — javascript библиотека, состоящая из кросбраузерных функций — оплеток для манипулирования элементами DOM (Document Object Model — Объектная модель документа).

Обращение к функции `$()` равносильно `jQuery()`

получить любой элемент документа html в виде объекта, можно выражением `$('#objID')`. Где `objID` - ID объекта.

<https://coderoad.ru/41806039/Как-мне-извлечь-данные-из-API-в-HTML-CSS-и-JS>

3. Понятие "статического" и "динамического" веб-сервера.

Статический сайт - это тот, который возвращает одно и то же жестко закодированное содержимое с сервера всякий раз, когда запрашивается конкретный ресурс

Динамический веб-сайт - это тот, где часть содержимого ответа генерируется динамически только при необходимости. На динамическом веб-сайте HTML-страницы обычно создаются путем вставки данных из базы данных в заполнители в HTML-шаблонах

Динамический веб-сервер. CGI:

1. новый процесс на каждый запрос;
2. передача данных через stdin, stdout, переменные окружения.

Динамический веб-сервер. FastCGI:

1. пулл процессов;
2. передача данных через сокеты.

Динамический веб-сервер. SSI:

Первый язык шаблонизации на сервере с помощью макрокоманд.

1. выполнение скриптов CGI;
2. условные операторы;
3. включение других файлов.

Билет 14

1. Методы и заголовки протокола HTTP 1.1

HTTP-методы:

- OPTIONS - запрос методов сервера;
- GET - запрос документа;
- HEAD - аналог GET, но без тела ответа (метаданные без данных);
- POST - передача данных от клиента;

- PUT - размещение файла по URI/изменение данных;
- DELETE - удаление файла по URI/удаление данных;
- TRACE, LINK, UNLINK, CONNECT - используются редко;
- PATCH - частичное изменение данных на сервере.

Заголовки HTTP (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA (см. RFC 822). Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Все заголовки разделяются на четыре основных группы:

General Headers («Основные заголовки») — могут включаться в любое сообщение клиента и сервера;

Request Headers («Заголовки запроса») — используются только в запросах клиента;

Response Headers («Заголовки ответа») — только для ответов от сервера;

Entity Headers («Заголовки сущности») — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посыпать заголовки получателю.

Все необходимые для функционирования HTTP заголовки описаны в основных RFC. Если не хватает существующих, то можно вводить свои. Традиционно к именам таких дополнительных заголовков добавляют префикс «X-» для избежания конфликта имён с возможными существующими.

Заголовок	Назначение
Общие заголовки	
Connection	Указывает серверу на завершение (close) или продолжение (keep-alive) сеанса
Date	Дата и время формирования сообщения
Pragma	Специальные, зависящие от реализации команды, касающиеся передаваемого содержимого (например, no-cache)
Transfer-Encoding	Способ кодирования сообщения при передаче (например, win1251, koi-8r)
Заголовки запроса	
Accept	Типы содержимого, которое клиент способен разработать и может воспроизвести

Accept-Charset	Кодировки символов, в которых клиент может принимать текстовое содержимое
Accept-Encoding	Способ, которым сервер может закодировать сообщение
Host	Хост и номер порта, с которого запрашивается документ
If-Modified-Since If-Match If-None-Match If-Range If-Unmodified-Since	Заголовки запроса для условного обращения к ресурсу
Range	Запрос части документа
User-Agent	Название программного обеспечения клиента
Заголовки ответа	
Age	Число секунд, через которое нужно повторить запрос для получения нового содержимого
Location	URI ресурса, к которому нужно обратиться для получения содержимого
Retry-After	Дата и время или число секунд, через которое нужно повторить запрос, чтобы получить успешный ответ
Server	Название программного обеспечения сервера, приславшего ответ
Заголовки объекта	
Allow	Перечисляет поддерживаемые сервером методы
Content-Encoding	Способ, которым закодировано тело сообщения, например, с целью уменьшения размера
Content-Length	Длина сообщения в байтах

Content-Type	Тип содержимого и, возможно, некоторые параметры
ETag	Уникальный тэг ресурса на сервере, позволяющий сравнивать ресурсы
Expires	Дата и время, когда ресурс на сервере будет изменен, и его нужно получать заново
Last-Modified	Дата и время последней модификации содержимого

2. Клиент-серверная архитектура. Принципы построения. Понятие клиента и сервера в Web.

Архитектура «Клиент-Сервер» представляет собой взаимодействие структурных компонентов в сети на основе определенных принципов организации данной сети, где структурными компонентами являются сервер и узлы-поставщики определенных специализированных функций (сервисов), а также клиенты, которые пользуются данным сервисом. Специфические функции принято делить на три группы на основе решения определенных задач:

- функции ввода и представления данных предназначены для взаимодействия пользователя с системой;
- прикладные функции – для каждой предметной области имеется собственный набор;
- функции управления ресурсами предназначены для управления файловой системой, различными базами данных и прочими компонентами.

Для разработки клиент / серверных систем имеется два подхода:

- построение систем на основе двухзвенной архитектуры;
- построение систем на основе трехзвенной архитектуры.

Двухзвенная архитектура состоит из клиентской и серверной части. Как правило, серверная часть представляет собой сервер БД, на котором расположены общие данные. А клиентская часть представляет приложение, которое связывается с сервером БД, осуществляет к нему запросы и получает ответы. Такие системы

используются в локальных сетях, т. к. нет затруднений с установкой клиентской части.

При разработке информационных систем, рассчитанных на широкую аудиторию, возникают проблемы с использованием двухзвенной архитектуры. Во-первых, пользователю необходимо иметь в наличии клиентскую часть, а, во-вторых, у неопытного пользователя, могут возникнуть проблемы с конфигурированием такой системы. Поэтому в последнее время, более часто разрабатывают приложения на базе трехзвенной архитектуры.

Трехвенная архитектура также состоит из двух частей: клиента и сервера. Но серверная часть в этой архитектуре представляет собой сервер приложений и сервер БД. А в качестве клиента выступает web браузер. Такая система очень проста для пользователя. Ему необходимо знать только адрес сервера приложения и иметь web браузер. Все данные представляются в виде html разметки с использование графики (jpeg, gif, flash), каскадных слоев CSS и JavaScript. Передача запросов от клиента к серверу приложений происходит по CGI интерфейсу. Сервер приложений общается с сервером БД, используя другой интерфейс, зависящий от того, на основе каких средств строится конкретная информационная система. Сервером мы считаем абстрактную машину в сети, способную получить HTTP-запрос, обработать его и вернуть корректный ответ. Не важны его физическая суть и внутренняя архитектура.

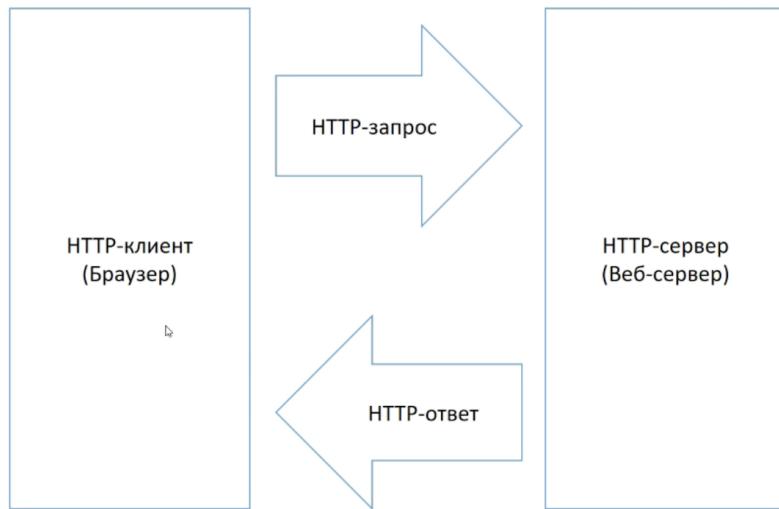
Клиентом может быть все, что угодно, что способно сформировать и отправить HTTPзапрос. Клиентом может быть JavaScript-сценарий, работающий в браузере, мобильное приложение, демон, запущенный на сервере и т.д.

Клиент-сервер - сетевая архитектура, в которой сетевая нагрузка распределена между поставщиками услуг, называемыми серверами и заказчиками услуг, называемыми клиентами.

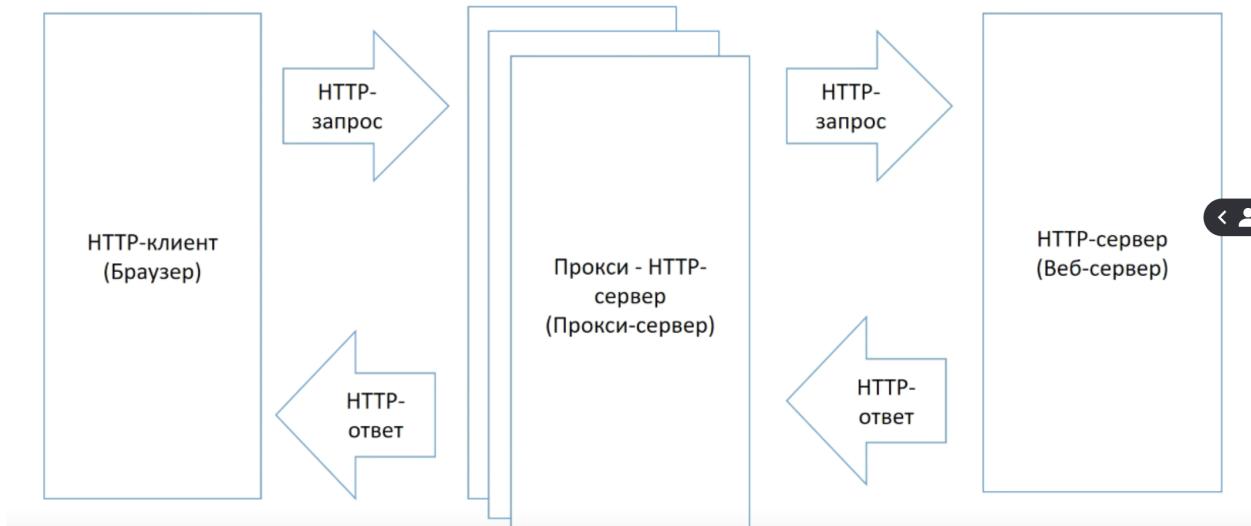
Классический "клиент-сервер" в WEB:

- Взаимодействие по протоколу HTTP/протоколам поверх HTTP
- HTTP работает поверх TCP
- Абстрагирование от физической реализации сети
- Адресация - сокетах (ip-фдрес + порт)
- Доменные имена преобразуются в ip-адрес
- Клиент - всегда инициатор сессии

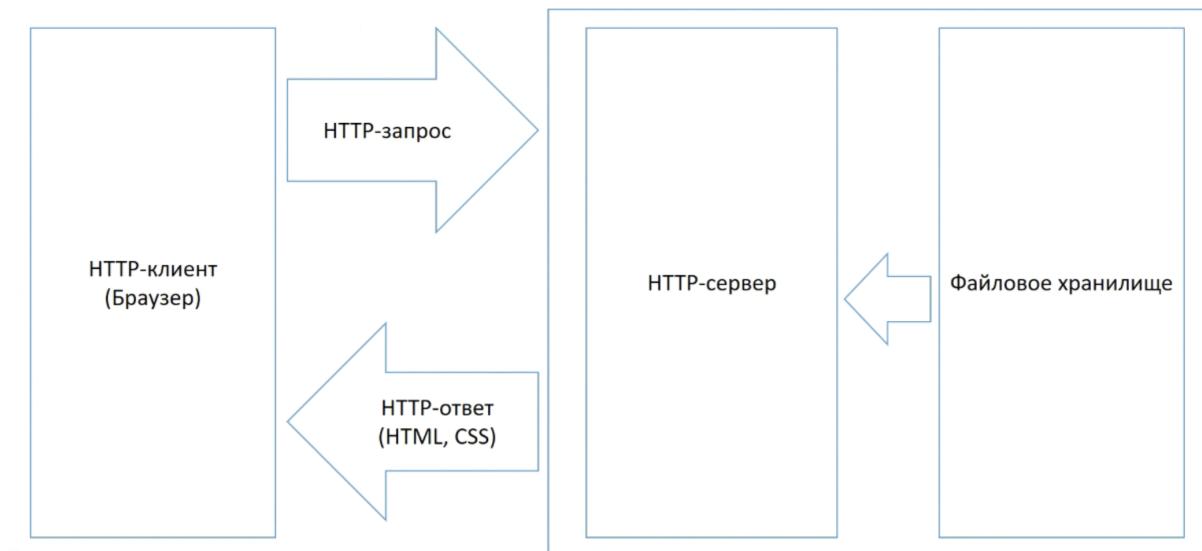
Клиент-серверное взаимодействие в HTTP



Клиент-серверное взаимодействие в HTTP



Клиент-серверная архитектура (статика)



Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веббраузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Веб-сервером называют как программное обеспечение, выполняющее функции вебсервера, так и непосредственно компьютер, на котором это программное обеспечение работает.

3. JS. Выражения. Операторы.

https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators

Операторы

В JavaScript есть следующие типы операторов:

- Операторы присваивания
- Операторы сравнения
- Арифметические операторы
- Бинарные операторы
- Логические операторы
- Строковые операторы

- Условный (тернарный) оператор
- Оператор запятая
- Унарные операторы

Операторы отношения JavaScript поддерживает бинарные и унарные операторы, а также ещё один специальный тернарный оператор - условный оператор. Бинарная операция использует два операнда, один перед оператором и другой за ним:

operand1 operator operand2

Например: $3+4$ или $x*y$.

В свою очередь унарная операция использует один операнд, перед или после оператора:

operator operand

или

operand operator

Например: $x++$ или $++x$.

Выражения

Выражением является любой корректный блок кода, который возвращает значение.

Концептуально, существуют два типа выражений: те которые присваивают переменной значение, и те, которые вычисляют значение без его присваивания.

Выражение $x = 7$ является примером выражения первого типа. Данное выражение использует оператор $=$ для присваивания переменной x значения 7. Само выражение также равняется 7.

Код $3 + 4$ является примером выражения второго типа. Данное выражение использует оператор $+$ для сложения чисел 3 и 4 без присваивания переменной полученного результата 7.

Все выражения в JavaScript делятся на следующие категории:

- Арифметические: вычисляются в число, например: 3.14 (Используют арифметические операторы).
- Строковые: вычисляются в текстовую строку, например: "Fred" или "234" (Используют строковые операторы).
- Логические: вычисляются в true или false (Используют логические операторы).
- Основные выражения: Базовые ключевые слова и основные выражения в JavaScript.

- Левосторонние выражения: Значениям слева назначаются значения справа.

Билет 15

1. Понятие шаблонизатора и маршрутизатора в MVC-фреймворке на сервере и клиенте.

Система маршрутизации определяет, какое приложение должно запуститься и какую операцию это приложение должно выполнить.

Для формирования HTML-кода Шаблонизатор на основе соответствующих шаблонов формирует код веб-страницы с использованием полученных данных.

Шаблонизатор (в web) — это программное обеспечение, позволяющее использовать html-шаблоны для генерации конечных html-страниц. Основная цель использования шаблонизаторов — это отделение представления данных от исполняемого кода. Часто это необходимо для обеспечения возможности параллельной работы программиста и дизайнера-верстальщика.

Класс MVC `Router` (который является частью более широкого **фронтального контроллера**) разбивает URL запроса HTTP--в частности, компонент пути (и потенциально строку запроса).

`Router` пытается сопоставить первую или две части компонента path соответствующей комбинации маршрутов (`Controller` / Action [`method`], или просто `Controller` , который выполняет действие по умолчанию (`method`)).

Действие или команда-это просто **метод** от конкретного `Controller` .

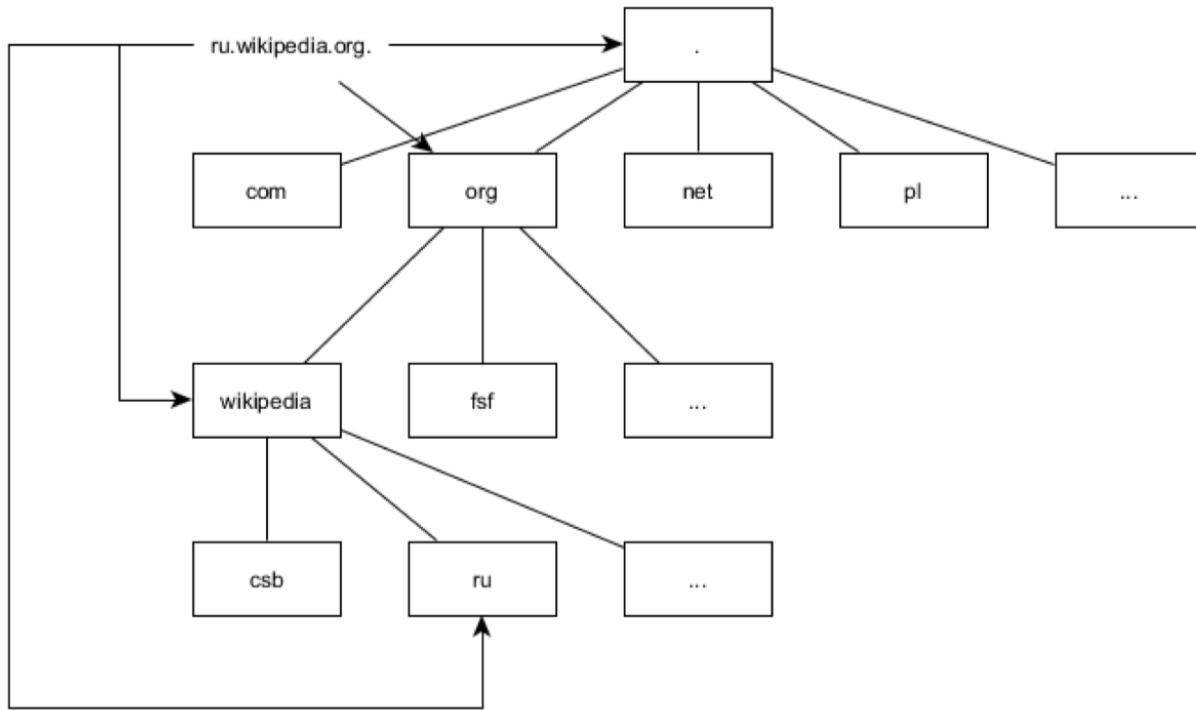
Обычно существует `abstract Controller` и много потомков `Controller` , по одному для каждой веб-страницы (вообще говоря).

2. Система доменных имен (DNS). Назначение, расшифровка, алгоритм работы. Пример работы.

DNS (Domain Name System - система доменных имён) – система для получения информации о доменах.

Изначально адресация узлов сети шла исключительно по IP-адресам. Запоминать их неудобно, что повлекло появление DNS (большой самообновляемый реестр записи, распределённый по всему миру; делегирует записи и ответственность).

Пример делегирования полномочий:



URI == URL || URN || URL + URN

Scheme - протокол (http, https, ftp)

Host - IP-адрес или доменное имя

Port - порт (необязательно, если стандартный)

Path - путь в файловой системе корневого каталога сервера (или псевдоним, обрабатываемый сервером); может быть виртуальный путь

Url-params - необязательные пары ключ-значение, используются для идентификаторов сессии пользователя;

Query-string - пары ключ-значение (параметры запроса)

Anchor - ссылка на позиционный маркер в пределах документа

scheme://host:[port]/path/.../[;url-params][?query-string][#anchor]

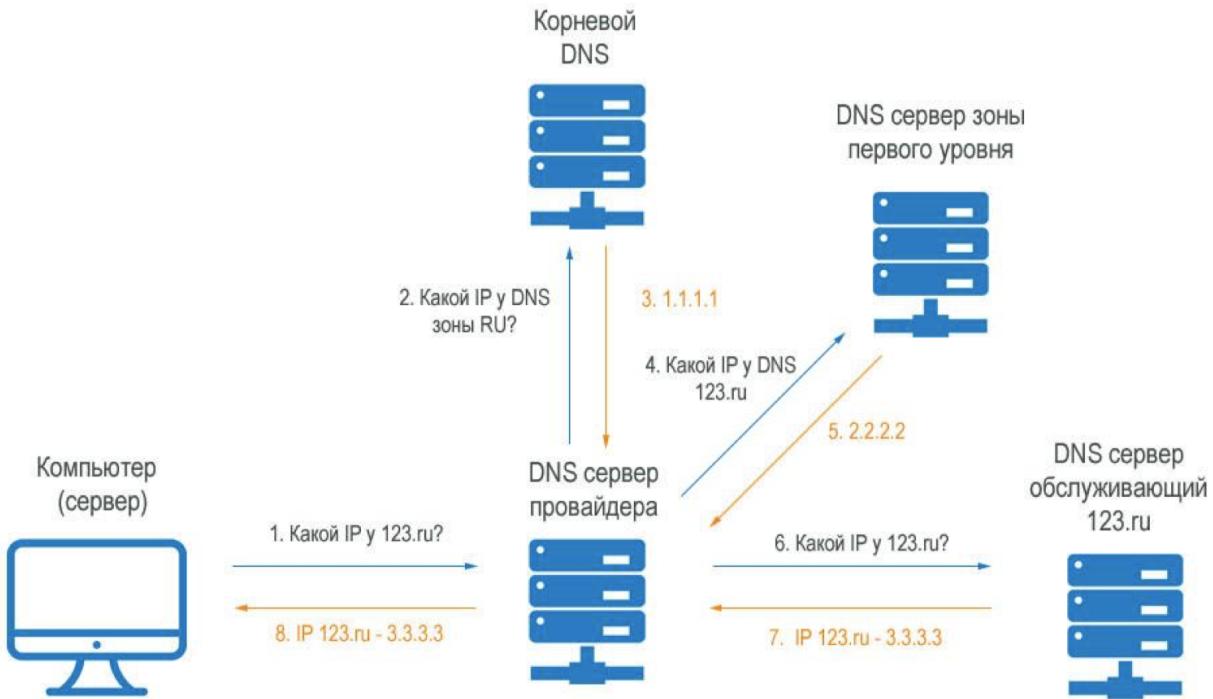
Как работает DNS

Система доменных имен состоит из следующих компонентов:

Иерархическая структура доменных имен:

- Доменные зоны верхнего уровня (первого уровня) – например: "ru", "com", или "org". Они включают в себя все доменные имена, входящие в эту зону. В любую доменную зону может входить неограниченное количество доменов.
- Доменные имена (доменные зоны второго уровня) – например: "[google.com](#)" или "[yandex.ru](#)". Т.к. система доменных имен является иерархичной, то "[yandex.ru](#)" можно также назвать поддоменом вышестоящей зоны "ru". Поэтому, правильнее указывать именно уровень домена. Однако, на практике, доменную зону любого уровня называют просто «доменом».
- Поддомены (доменные зоны третьего уровня) – например: "[api.google.com](#)" или "[mail.yandex.ru](#)". Могут быть доменные зоны 4, 5 уровней и так далее. "[www.google.com](#)" и "[google.com](#)" – это, фактически, разные домены. Надо не забывать указывать А-записи для каждого из них.

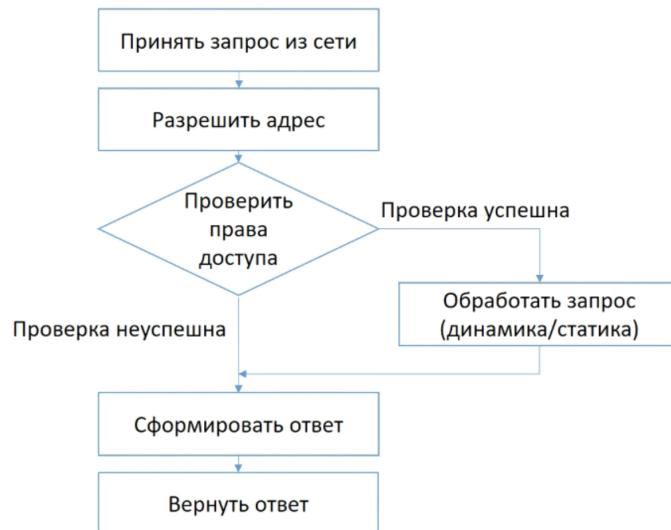
Для того, чтобы узнать IP адрес, домена компьютер / сервер обращается к DNSсерверу, который указан у него в сетевых настройках. Обычно, это DNS сервер Интернет провайдера. DNS сервер проверяет делегирован домен ему или нет. Если да, то сразу отвечает на запрос. Если нет, то запрашивает информацию о DNS сервере, обслуживающем этот домен, у корневого сервера, и затем у сервера доменных зон верхнего уровня. После этого, непосредственно делает запрос на NS (name server) сервер, обслуживающий этот домен, и транслирует ответ вашему компьютеру / серверу.



Кэширование данных используется на всех устройствах (компьютерах, серверах, DNS серверах). То есть, они запоминают ответы на последние пришедшие к ним запросы. И когда приходит аналогичный запрос, они просто отвечают то же самое, что и в предыдущий раз. Например, если вы в браузере открыли сайт google.com первый раз после включения, то компьютер сделает DNS запрос, а при последующих запросах будет брать данные, которые ему были присланы DNS сервером в первый раз. Таким образом, для популярных запросов не надо каждый раз проходить всю цепочку и генерировать запросы к NS серверам. Это значительно снижает нагрузку на них, и увеличивает скорость работы. Однако, как результат, обновление данных в системе DNS происходит не сразу. При изменении IP адреса домена, информацию об этом будет расходиться по сети Интернет от 1 до 24 часов.

3. Алгоритм запуска и работы веб-сервера.

Алгоритм веб-сервера



Типовой запуск веб-сервера:

1. чтение конфигурации;
2. получение порта (80);
3. открытие логов;
4. понижение привилегий;
5. запуск дочерних процессов/потоков;
6. ожидание запросов.

Web-серверы и браузеры обмениваются между собой HTTP-сообщениями. Серверы получают и обрабатывают HTTP-запросы, определяют местоположение запрашиваемых ресурсов и выполняют к ним доступ, формируют ответы, которые они отправляют назад браузерам, сделавшим эти запросы. На рис. 2.1 показано, как webсервер обрабатывает поступающие запросы, формирует на них ответы и передает их запрашивающему браузеру.

Модуль поддержки работы с сетью отвечает за получение запросов и передачу сформированных ответов по сети. При получении очередного HTTP-запроса сервер

прежде всего передает его модулю разрешении запроса, который отвечает за анализ и предварительную обработку поступившего запроса.

Модуль предварительной обработки запроса включает:

1. Виртуальный хостинг: если web-сервер предоставляет доступ к нескольким web-сайтам, имеющим разные доменные имена, то для поступившего запроса требуется определить целевой web-сайт и использовать его для выбора параметров конфигурации.
2. Разрешение адреса: определение типа запрашиваемого контента статический или динамический; на основе заданного URL-пути и выбранных параметров конфигурации сервера выполняется разрешение URL-адреса (т. е. его преобразование) в реальный адрес в файловой системе сервера.
3. Аутентификация: если запрашиваемый ресурс является защищенным, то требуется проверить данные авторизации (имя и пароль), чтобы определить, имеет ли право пользователь использовать данный ресурс.

После завершения предварительной обработки запрос передается модулю обработки запроса, который вызывает подмодули для выполнения соответствующей обработки статического и динамического контента. Если запрос обращается к динамическому контенту, то сервер передает данные на выполнение некоторой среде: контейнеру сервлетов или серверу приложений, которые выполняются в других процессах. Данная среда управляет выполнением web-приложений:

- организует взаимодействие web-сервера с приложением (передает данные сообщения и параметры сервера приложению и возвращает сформированный код серверу);
- запускает требуемое приложение на выполнение;
- решает дополнительные задачи (управление состоянием сеанса работы, поддержку очереди сообщений, управление кэшем и т. п.).

Когда выбранный подмодуль или сервер приложений закончит обработку запроса, он передаст результаты выполнения модулю формирования HTTP-ответа, который формирует заголовки ответа, объединяет их с полученным результатом обработки и передает модулю поддержки работы с сетью для передачи сформированного ответа тому клиенту, который прислал данный запрос. Следует отметить, что в связи с тем, что HTTP не поддерживает состояние сеанса работы, то единственная информация, которая доступна серверу о поступившем запросе, содержится в самом запросе (заголовках и теле).

Билет 16

1. Макетирование. Основные подходы и идеи. Инструменты (Figma).

Figma — это графический онлайн-редактор для дизайнеров интерфейсов и веб-разработчиков. Сейчас это удобная, бесплатная альтернатива Photoshop. Большое преимущество платформы — возможность работать прямо в браузере. При этом есть и десктопная версия. Расскажем, что надо знать верстальщику при работе с макетом в Figma.

2. Паттерны проектирования MVC. Примеры паттернов MVC-семейства. Расшифровка аббревиатуры. Описание архитектуры.

Наиболее распространенные виды MVC-паттерна, это:

Model-View-Controller

Model-View-Presenter

Model-View-View Model

(Model View Intent)(какой-то таинственный про который мало пишут)

Model-Template-View - Django-взгляд на MVC (на самом деле, классический MVC, только

V == T, а C == V)

Семейство MVC-паттернов (по лекциям):

1. MVC;
2. MTV;
3. MPV;
4. MVVM;
5. MVPVM.

MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение

блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Компоненты MVC:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента `model` будет определять список задач и отдельные задачи.
2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента `view` определяет внешний вид приложения и способы его использования.
3. Контроллер — этот компонент отвечает за связь между `model` и `view`. Код компонента `controller` определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения. Контроллер решает задачи, он не должен включать в себя бизнес-логику приложения. Не стоит заниматься существенным в контроллере, иначе возникает FSUC (Fat Stupid Ugly Controllers).

MVC — подход к проектированию приложения, который предполагает выделение кода в блоки модель, представление и контроллер.

Контроллер обрабатывает входящие запросы. Модель достаёт из базы данных информацию, нужную для выполнения конкретных запросов. Представление определяет результат запроса, который получает пользователь.

Типовая структура MVC



3. Технологии толстого клиента. Пример с отображением списка пользователей на Angular/React/Vue.

Толстый клиент, rich client архитектуре клиент-сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) полную функциональность и независимость от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства

- Толстый клиент обладает широкой функциональностью в отличие от тонкого.
- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Высокое быстродействие (зависит от аппаратных средств клиента)

Недостатки

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.

- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.
- Наличие бизнес-логики

<https://angular-doc.ru/tutorial/toh-pt2>

<https://habr.com/ru/post/418463/>