

1 лек

OSI

Open Systems Interconnection model (OSI model)

[reference model](#) from the [International Organization for Standardization](#) (ISO) that "provides a common basis for the coordination of standards development for the purpose of systems interconnection." [source](#)

уровни протоколов?

HTTP

HyperText Transfer Protocol

протокол прикладного уровня, на котором упрощенно говоря общаются браузер и веб сервер. это протокол предметной области -- он знает что такое web, какие атрибуты(свойства) есть у запросов и у ответов (лекция)

HTTP — это [протокол](#), позволяющий получать различные ресурсы, например HTML-документы.

Можно передавать также изображения и т п!

факты:

https = http завернутый в tls чтобы шифровать

HTTP является протоколом прикладного уровня, который чаще всего использует возможности другого протокола - [TCP](#) (или [TLS](#) - защищённый TCP) - для пересылки своих сообщений, однако любой другой надёжный транспортный протокол теоретически может быть использован для доставки таких сообщений.

TCP Transmission Control Protocol

TCP (Протокол Управления Передачей) - важный [протокол](#) сети интернет, который позволяет двум хостам создать соединение и обмениваться потоками данных. TCP гарантирует доставку данных и пакетов в том же порядке, в котором они были отправлены.

лекция:

TCP -- протокол с помощью которого сетевые интерфейсы взаимодействуют между собой устанавливая надежное (reliable-)соединение

это делается трехэтапным рукопожатием во время установки соединения -- небыстрая тема

UDP

(User Datagram Protocol) is a long standing [protocol](#) used together with [IP](#) for sending data when transmission speed and efficiency matter more than security and reliability.

С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые [датаграммами](#)) другим [хостам](#) по [IP-сети](#) без необходимости предварительного сообщения для установки специальных [каналов передачи](#) или путей данных.

UDP использует простую модель передачи, без явных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных.

Датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа, но гарантируется, что если они придут, то в целостном состоянии.

TLS

transport layer security — Протокол защиты транспортного уровня как и его предшественник [SSL](#) ([англ.](#) secure sockets layer — слой защищённых сокетов), — [криптографические протоколы](#), обеспечивающие защищённую передачу данных между узлами в сети Интернет2(https://ru.wikipedia.org/wiki/TLS#cite_note-2).

Протоколы SSL и TLS

Протокол	Дата публикации	Состояние
SSL 1.0	не публиковался	не публиковался
SSL 2.0	1995	Признан устаревшим в 2011 году (RFC 6176)
SSL 3.0	1996	Признан устаревшим в 2015 году (RFC 7568)
TLS 1.0	1999	Признан устаревшим в 2020 году6(https://ru.wikipedia.org/wiki/TLS#cite_note-:2-6)
TLS 1.1	2006	Признан устаревшим в 2020 году6(https://ru.wikipedia.org/wiki/TLS#cite_note-:2-6)
TLS 1.2	2008	
TLS 1.3	2018	

user-agent

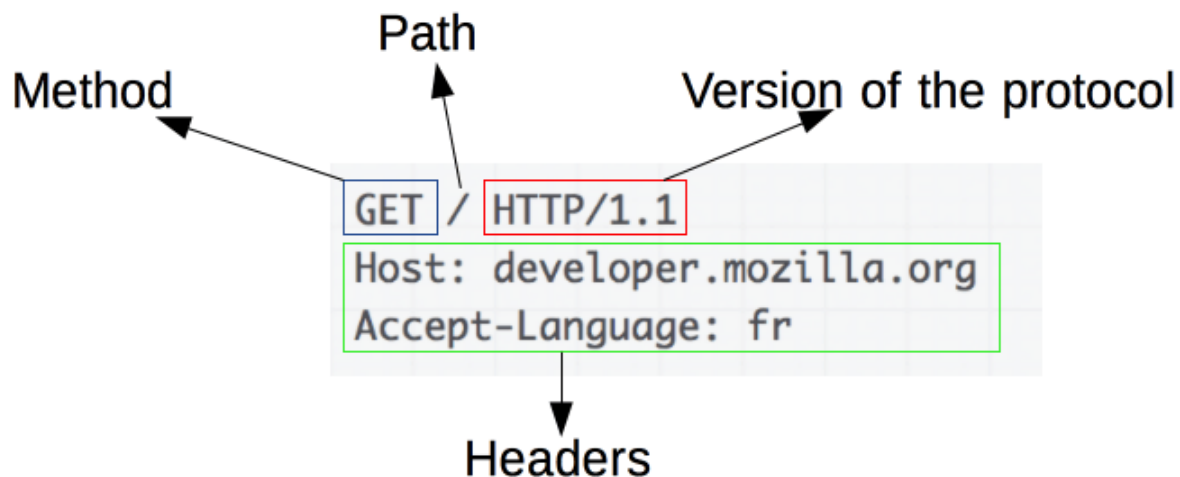
-- это любой инструмент или устройство, действующие от лица пользователя. Эту задачу преимущественно выполняет веб-браузер.

Прокси

Между веб-браузером и сервером находятся большое количество сетевых узлов, передающих HTTP сообщения. Из-за слоистой структуры большинство из них оперируют также на транспортном сетевом или физическом уровнях, становясь прозрачным на HTTP слое и потенциально снижая производительность. Эти операции на уровне приложений называются **прокси**. Они могут быть прозрачными или нет, (изменяющие запросы не пройдут через них), и способны исполнять множество функций:

- caching (кеш может быть публичным или приватными, как кеш браузера)
- фильтрация (как сканирование антивируса, родительский контроль, ...)
- выравнивание нагрузки (позволить нескольким серверам обслуживать разные запросы)
- аутентификация (контролировать доступом к разным ресурсам)
- протоколирование (разрешение на хранение истории операций)

http-запрос:



HTTP-запрос

GET / HTTP/1.1

Host: www.example.com

Accept: text/html,application/xhtml+xml

Accept-Encoding: gzip, deflate, br

Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

Cookie: k1=v1; k2=v2

Connection: keep-alive

Referer: https://polygon.codeforces.com/

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)

Общий вид

Первая строка: METHOD URL HTTP/Version

Потом заголовки: Name: Value

Потом пустая строка

Потом данные (если есть)

- HTTP-метод, обычно глагол подобно [GET](#), [POST](#) или существительное, как [OPTIONS](#) или [HEAD](#), определяющее операцию, которую клиент хочет выполнить
- Путь к ресурсу
- Версию HTTP-протокола.

заголовки (optional)

заголовки HTTP позволяют клиенту и серверу отправлять дополнительную информацию с HTTP запросом или ответом. В HTTP-заголовке содержится не чувствительное к регистру название, а затем после (:) непосредственно значение. [Пробелы](#) перед значением игнорируются.

Заголовки могут быть сгруппированы по следующим контекстам:

- [Основные заголовки](#) применяется как к запросам, так и к ответам, но не имеет отношения к данным, передаваемым в теле.

- Заголовки запроса содержит больше информации о ресурсе, который нужно получить, или о клиенте, запрашивающем ресурс.
- [Заголовки ответа \(en-US\)](#) содержат дополнительную информацию об ответе, например его местонахождение, или о сервере, предоставившем его.
- [Заголовки сущности](#) содержат информацию о теле ресурса, например его [длину содержимого](#) или тип [MIME \(en-US\)](#).

Заголовки также могут быть сгруппированы согласно тому, как [прокси \(proxies\)](#) обрабатывают их:

- [Connection](#)
- [Keep-Alive \(en-US\)](#)
- [Proxy-Authenticate \(en-US\)](#)
- [Proxy-Authorization \(en-US\)](#)
- [TE \(en-US\)](#)
- [Trailer \(en-US\)](#)
- [Transfer-Encoding \(en-US\)](#)
- [Upgrade \(en-US\)](#)

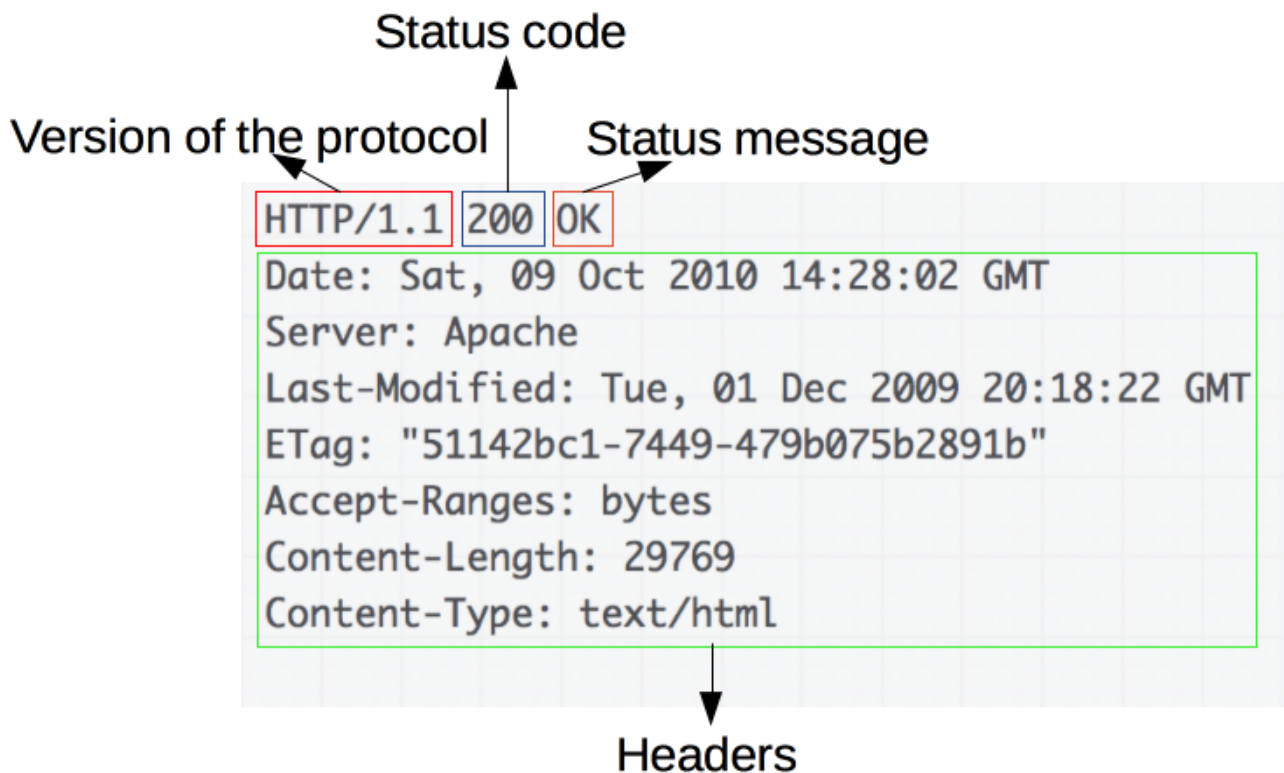
параметры запроса

параметры у запроса (своего рода) бывают 2х видов:

1. get-параметры -- которые содержатся в get строке -- используются в основном для гет запросов -- могут быть доступны и видны промежуточным прокси-серверам, попадают в логи
2. которые находятся в самом запросе лол

семантика методов

ОТВЕТ



статусный код

Код состояния ответа HTTP показывает, был ли успешно выполнен определённый [HTTP](#) запрос. Ответы сгруппированы в 5 классов:

1. [Информационные ответы](#) (100 – 199)
2. [Успешные ответы](#) (200 – 299)
3. [Сообщения о перенаправлении](#) (300 – 399)
4. [Ошибки клиента](#) (400 – 499)
5. [Ошибки сервера](#) (500 – 599)

Коды состояния определены в [RFC 9110](#). Но по сути это просто семантика

[200 OK](#)

[400 Bad Request](#)

[404 Not Found](#)

[408 Request Timeout](#)

[504 Gateway Timeout](#)

[500 Internal Server Error](#)

[502 Bad Gateway](#)

keep-alive

в 1.1 был придуман keep-alive концепция -- когда user-agent устанавливает соединение с сервером и может не разрывать его и уже в рамках этого соединения можно посылать много реквестов респонсов
кип-элайвы не бесконечны -- обычно 10ки-100ни секунд

stateless в контексте http

понятно что, сказать про куки и сессии

куки cookie

Получив HTTP-запрос, вместе с ответом сервер может отправить заголовок [Set-Cookie](#) . Куки обычно запоминаются браузером и посылаются в HTTP-заголовке [Cookie \(en-US\)](#) с каждым новым запросом к одному и тому же серверу. Можно задать срок действия кук, а также срок их жизни, после которого куки не будут отправляться. Также можно указать ограничения на путь и домен, то есть указать, в течении какого времени и к какому сайту они будут отправляться.

Заголовок [Set-Cookie](#) HTTP-ответа используется для отправки куки с сервера в клиентское приложение (браузер). Простой куки может задаваться так:

Set-Cookie: <имя-куки>=<заголовок-куки>

Этот заголовок с сервера даёт клиенту указание сохранить куки (это делают, например, [PHP](#), [Node.js](#), [Python](#) и [Ruby on Rails](#)). Ответ, отправляемый браузеру, содержит заголовок [Set-Cookie](#) , и куки запоминается браузером.

HTTP/1.0 200 OK

Content-type: text/html

Set-Cookie: yummy_cookie=choco

Set-Cookie: tasty_cookie=strawberry

Теперь с каждым новым запросом к серверу при помощи заголовка [Cookie \(en-US\)](#) браузер будет возвращать серверу все сохранённые ранее куки.

GET /sample_page.html HTTP/1.1

Host: [www.example.org](#)

Cookie: yummy_cookie=choco; tasty_cookie=strawberry

cookie HTTPOnly

Куки HTTPOnly не доступны из JavaScript через свойства `Document.cookie` API, что помогает избежать межсайтового скриптинга ([XSS \(en-US\)](#)). Устанавливайте этот флаг для тех кук, к которым не требуется обращаться через JavaScript. В частности, если куки используются только для поддержки сеанса, то в JavaScript они не нужны, так что в этом случае следует устанавливать флаг `HttpOnly`.

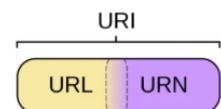
session

есть куки на сервере хранится мапа называем сессией

URI

URI это в принципе концепция , объединение URL и URN
но какая то организация это регулирует

URI и URL



Universal Resource Identifier (URI) — последовательность символов, идентифицирующая абстрактный или физический ресурс.

Uniform Resource Locator (URL) — стандарт записи ссылок на объекты в Интернете. Еще есть **URN** — идентифицирует имя (напр., `urn:isbn:5170224575`)



URL

-- "унифицированный локатор ресурсов", местоположение ресурсов в интернете
будем обращаться к базам данных и вместо foo будет mysql или jdbc
может быть ssh
для http дефолтный порт это 80
https -- 443

fragment: после решетки -- определенный кусок на странице куда проскроллить

URI и URL

Особенности (в HTTP):

- процентное кодирование для не-ASCII символов:
например, %20 означает пробел
- схема и хост — регистронезависимы, а все остальное нормализуется как регистрозависимое
- используются относительные, фрагмент обычно не передаётся на сервер
- в относительных записях части слева направо могут отсутствовать

сокет

конечная точка соединения
DNS

ip адрес

IP-адрес (от англ. [Internet Protocol](#)) — уникальный числовой [идентификатор](#) устройства в компьютерной [сети](#), работающей по протоколу [IP](#).

докинуть про

OSI и tcp/ip

ssh

smtp

telnet

cache-control (общие слова)

доп:

файлик hosts

реверс-прокси

nginx как пример

2 лек

HTML

HTML (Hypertext Markup Language) - это *язык разметки*, который используется для структурирования и отображения веб-страницы и её контента.

структура html

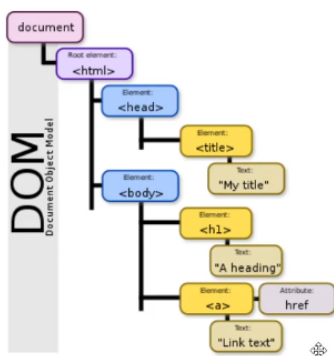
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Моя тестовая страница</title>
```

```
</head>
<body>
  
</body>
</html>
```

элемент `<title>`. Этот элемент устанавливает заголовок для вашей страницы, который является названием, появляющимся на вкладке браузера загружаемой страницы, и используется для описания страницы, когда вы добавляете её в закладки/избранное.

DOM

DOM



DOM (от англ. Document Object Model — «объектная модель документа») — это независимый от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект.

Например: стандарт DOM описывает, что метод `getElementsByTagName` в коде, указанном ниже, должен возвращать список всех элементов `p` в документе.

```
paragraphs = document.getElementsByTagName("P");
// paragraphs[0] это первый <p> элемент
// paragraphs[1] это второй <p> элемент и т.д.
alert(paragraphs[0].nodeName);
```

элементы

Элемент - это часть веб-страницы. В XML и HTML элемент может содержать данные, фрагмент текста или изображения, или не содержать ничего. Обычный элемент включает в себя открывающий тэг с некоторыми атрибутами, текст и закрывающий тэг.

Элементы и тэги это *не* одни и те же вещи. Тэги открывают или закрывают элементы в исходном коде, тогда как элементы являются частью [DOM](#), объектной моделью документа для отображения страницы в [браузере](#).

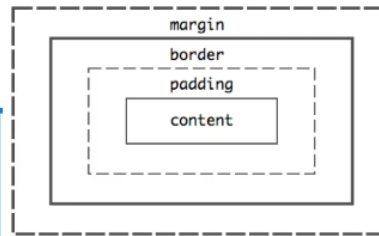
блоковые элементы, структура

(блочные) -- по умолчанию занимает в **ширину** сколько может, а **высоту** сколько минимально нужно.

HTML

- Блочные элементы (`display: block`)
Один в строке, можно управлять шириной и высотой. Примеры: `div`, `p`, `form`, `article`, `aside`, `main`, `nav`, `header`, `footer`, `section`, `ul`, `ol`, `li`.

По-умолчанию `width` и `height` относятся к `content`, но `box-sizing` может переопределить поведение.



margin -- внешний отступ

border -- граница

padding -- отступ контента от бордера

когда мы руками устанавливаем `width/height`, то по дефолту он устанавливается для контента чтобы это поведение менять юзаем **box-sizing**

```
box-sizing: border-box;
```

значения **margin-box** кажется не существует

строковые элементы

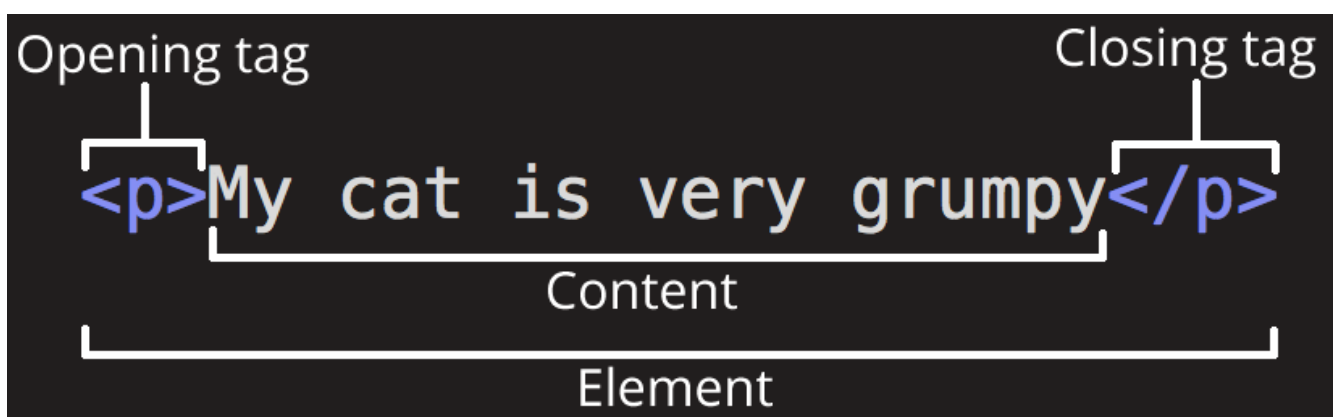
HTML

- Строковые элементы (`display: inline`)
Являются частью строки. `width-height` автоматически. Верхние-нижние отступы не применимы. Примеры: `span`, `a`, `strong`, `button`, `input`, `label`, `select`, `textarea`, `b`, `i`.
- Не отображаемые элементы (`display: none`)
- Строково-блочные (`display: inline-block`)
Упрощенно: похожи на блочные, но не разрывают строку

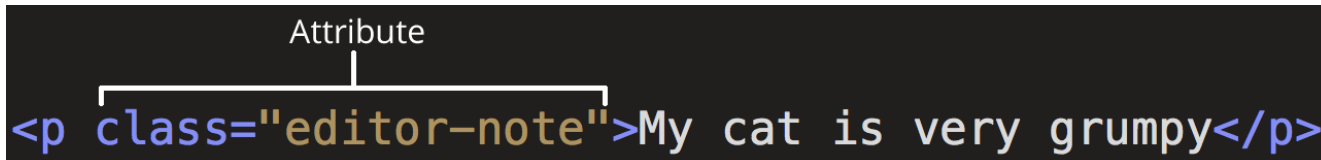
к `inline` неприменимы верхние/нижние отступы и ширина

`inline-block` -- не разрывает строку, но можно указывать левые/правые отступы и т.п.

строение элемента



Элементы также могут иметь **атрибуты**, которые выглядят так:



атрибуты

Атрибуты содержат дополнительную информацию об элементе, которую вы не хотите показывать в фактическом контенте.

Атрибут всегда должен иметь:

1. Пробел между ним и именем элемента (или предыдущим атрибутом, если элемент уже имеет один или несколько атрибутов).
2. Имя атрибута, за которым следует знак равенства.
3. Значение атрибута, заключённое с двух сторон в кавычки.

- ВИДЫ ЭЛЕМЕНТОВ

Элементы можно разделить на:

- **inline**-элементы -- типа кусочек строки, могут даже начинаться на одной строке и перенестись на другую строку.

Примеры (соответствующих тегов): p, div, main, header, section, article

- **блоковые** (блочные) -- по умолчанию занимает в **ширину** сколько может, а **высоту** сколько минимально нужно.

Эти режимы отображения можно насильственно менять (см [2 -- HTML + CSS > ^displaySlide](#)) по сути в html теги одного типа делают одно и то же, разные названия передают семантику ну только **div** и **span** не обладают семантикой, это простой блочковый и inline элементы определения элементов соответственно

- теги

В [HTML](#) теги используются для создания элементов. **Имя** HTML-элемента - это **имя** заключённое в угловые скобки, как например

для "абзаца". Обратите внимание, что концу **имени** предшествует символ косой черты (слеша), "

", и что в пустых элементах закрывающий тег не требуется и не допускается. Если атрибуты не указаны, то для них применяются значения по умолчанию.

REM

"root-em"

размер шрифта в документе

величина размера

CSS

структура css-файла

CSS

CSS (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Простые селекторы:

```
селектор {  
  свойство: значение;  
  свойство: значение;  
  свойство: значение;  
}  
  
* - все элементы  
#id - выбор элемента по атрибуту id  
.cls - выбор всех элементов по классу cls  
tag - выбор всех элементов по имени тега  
[attr=value] - выбор всех элементов по атрибуту и значению  
  
#user {  
  margin: 1rem;  
}  
.postcard {  
  border: 1px solid black;  
}
```



селектор

декларативный запрос к **DOM** (Document Object Model = дереву документа) "найди набор элементов из этого дерева"

составные селекторы (механики)

- пробел = "предок->потомок"
note: потомок, а не сын
- больше = "родитель->ребенок"

```
div > span {  
  color: red;  
}
```

- запятая=объединение путей, "или"
- двоеточие=псевдоклассы -- всякие штуки))
- псевдоклассы:

CSS



Псевдоклассы

- :nth-child(even) - отфильтрует только такие, которые являются четным ребенком. Другие примеры :nth-child(odd), :nth-child(1)
- :visited - посещенная ссылка
- :hover - мышь сверху

```
li:nth-child(odd) {  
  color: #aae;  
}  
  
a:visited {  
  color: gray;  
}  
  
.comment:hover {  
  cursor: pointer;  
}
```

```
li:nth-child(3n+1) {  
  color: red;  
}  
li:nth-child(odd) {  
  color: red;  
}  
  
:root {  
  font-size: 16px;  
  font-family: Verdana, serif;  
}
```

Не путать!

"пэшка" с id="mike":

```
p#mike {
```

id="mike" у которого есть предок "пэшка":

```
p #mike {
```

```
p : { } -- все теги <p>
* : { } -- BCE элементы
#[идентификатор] { } -- для
<[тег] id="[идентификатор]">...</>
```

предполагается что id уникальны

```
.[класс] {
} -- для всех
<[тег] class="[класс1 класс ...]">...</>
```

position

Позиционирование

- position: static - по-умолчанию
- position: relative - отобразить по-умолчанию, потом сдвинуть относительно
- position: absolute - отобразить как задали, точка отчета - это ближайший относительно позиционированный предок или корень
- position: fixed - отобразить как задали относительно окна браузера (viewport-a)

речь идет про блочные элементы

relative -- нарисовали как статик, а потом вырезали и подвинули (все размеры не меняются)

absolute -- забываем про контекст, отображаем от какой-то "точки отсчета", другие элементы про него не знают (не должны знать, забывают)

точка отсчета: дефолтно -- левый верхний угол, иначе -- идем вверх по дереву до первого элемента с

position: relative

fixed -- абсолютно относительно левого верхнего угла браузера а не документа, например для уведомлений

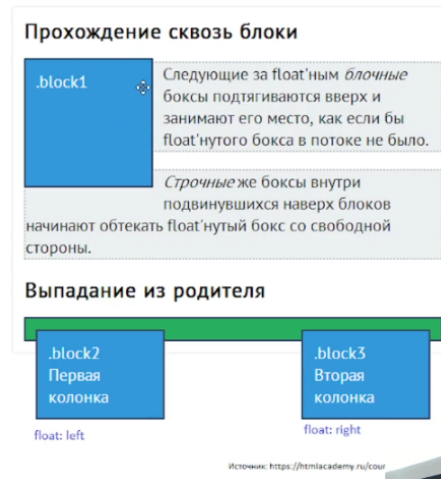
поплавки

Поплавки

```
.block1 {  
  float: left;  
}
```

Отображение в стиле картинок в тексте, которые прижаты влево-вправо. Их обтекает content родителя, они не влияют на высоту родителя. Они всегда блочные.

Для “не выпадания” из родителя можно использовать `display: flow-root;`



normalize.css

у браузера есть встроенные типа-css-ки которые говорят какие отступы делать на элементах по умолчанию

хотим ? верстку которая будет одинаково смотреться во всех браузерах в мире

есть два пути:

1. ресетнуть нахуй все
2. сделать нормалайз -- сбросить стили до нормального одинакового четкого значения во всех браузерах

#непонел

допилить:

нормалайз

@media

3 лек

API

API (application programming interface дословно *интерфейс программирования приложения*) — программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими.

war

специального вида архив специального вида приложения каким-то образом общается с контейнером
.war файл -- zip архив специального вида, в который соберется наше приложение

maven3

Apache Maven — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM, являющемся подмножеством XML.

утилита mvn

она есть, пусть страна знает свои утилиты

архетип

в многих системах сборки (в т ч мэвэн) есть шаблоны сборки -- **архетипы** -- заготовки для определенного вида проектов

`mvn archetype:generate` -- в интерактивном режиме выберем и сделаем архетип

артефакты

Артефакт - это “что угодно” (любой файл), к которому можно обратиться, используя его координаты, и Maven загружает, устанавливает или развертывает за вас. (с офиц сайта)

Координаты **артефакта** чаще всего представляются как `groupId:artifactId:version`

groupId и **artifactId** - как фамилия имя

хорошая идея использовать в качестве `groupId` название пакета

В то время как Maven внутренне тщательно использует понятие “артефакт” (просто посмотрите на исходники!), конечные пользователи, возможно, никогда не столкнутся с этим термином. Это связано с тем фактом, что в то время как для Maven “все является артефактом” (внутренне), конечные пользователи Maven на самом деле говорят о “проектах”, “родительских проектах”, “зависимостях”, “плагилах сборки”, “плагилах отчетов”, “расширениях сборки” и так далее.

pom.xml

дескриптор модели нашего проекта

который описывает где исходники расположены, где зависимости на какие библиотеки что нужно делать на какую фазу какие плагины подключить и т п

в `pom.xml` прописываем зависимости в т ч `javax.servlet`

`scope:provided` -- зависимость нужна только на этапе компиляции, потому что далее ее будет предоставлять сам контейнер

Servlet API, servlet

Servlet API -- программный интерфейс который стоит между нашим веб приложением и контейнером где мы его размещаем -- с лекции

A **servlet** is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

сессия

`request.getSession` есть и он работает через куки `cookie`

doXXX

гарантируется что `init()` и `destroy()` запустятся ровно один раз за жизнь сервлета

но функции `doXXX()` будут вызываться типа параллельно, поэтому нужно думать об общих ресурсах

tomcat

-- `servlet`-контейнер, очень популярный

Catalina

Tomcat на самом деле состоит из ряда компонентов, включая движок Tomcat JSP и множество различных разъемов, но его основной компонент называется Catalina. Catalina предоставляет фактическую реализацию спецификации сервлета Tomcat; когда вы запускаете свой сервер Tomcat, вы фактически запускаете Catalina

4 лек

freemarker

Apache FreeMarker is a *template engine*: a Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data.

Templates are written in the FreeMarker Template Language (FTL), which is a simple, specialized language (not a full-blown programming language like PHP). Usually, a general-purpose programming language (like Java) is used to prepare the data (issue database queries, do business calculations). Then, Apache FreeMarker displays that prepared data using templates. In the template you are focusing on how to present the data, and outside the template you are focusing on what data to present.

шаблонизатор

СВОИМИ СЛОВАМИ ПОНЯТНО

как работает (пример из нашего проекта)

чтобы .ftlh-ки рендерились и т п создаем сервлет в котором прописываем путь к фтлш-кам , кодировку, то как шаблонный движок должен обрабатывать исключения (рекомендуется отдавать `TemplateExceptionHandler.HTML_DEBUG_HANDLER`)

```
TemplateExceptionHandler.HTML_DEBUG_HANDLER

Template t = cfg.getTemplate(File file)

template.process(Map data, response.getWriter()); -- рендер
```

и еще кладем всякие ключи-значения из реквеста в data

макросы

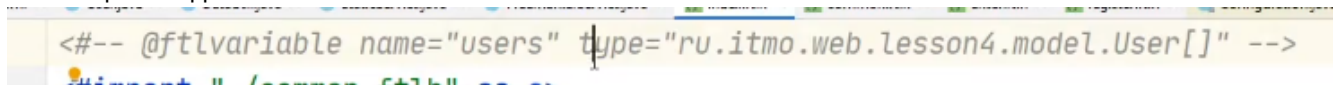
это макросы =)

функции

это функции

аннотации

аннотации? вида



-- подсказка для среды

5+6 лек

MVC

MVC это концепция о разделении веб приложения на 3 слоя: Model, View, Controller

1. Model -- предметная область: сущности (User, Event), их взаимодействия между собой и т.п.
2. Controller -- диспетчеризация, связующее звено
3. View -- все что связано с отображением контента
более высокий слой не зависит от более низкого, но более низкий зависит от более высоких
Model не должен меняться, если меняются CV
View скорее всего будет меняться если будут меняться M или C

схемки, примеры, описание реализации с 5 лекции

#todo

раутинг

нам приходит запрос, и мы по каким-то штукам решаем в какой контроллер? его направить

доменные сущности

Доменные объекты — это объекты в [объектно-ориентированных компьютерных программах](#), выражающие сущности из модели предметной области, относящейся к программе, и реализующие [бизнес-логику](#) программы. Например, программа, управляющая заказами, может содержать такие доменные объекты, как «заказ», «позиция заказа», «счёт-фактура».

сервис

сервисы -- находятся в слое модели

Сервис — это набор методов, который реализует логику приложения. Условия, которые были определены сервису:

- сервис не должен самостоятельно обращаться к контроллерам и представлениям;
- сервис может обращаться к базе данных, моделям и сущностям.

контроллер

-- диспетчеризация, раутинг, по сути -- связующее звено между View и Model, использует сервисы (которые находятся в model)

java reflection

#todo

схемки схемки схемки

соль в паролях

просто хэшить пароли не достаточно потому что

1. облегчается реверсинжиниринг (см радужные таблицы)
2. если кто-то вставляет пароль 12345 мы сразу увидим этот хэш
соль это что-то разное для разных паролей, например имя юзера
всегда стоит проектировать систему с расчетом на то что данные могут утечь

упражнение:

Почему не стоит использовать время регистрации как соль? (2 причины)

TODO:

#todo

JDBC, что использовали
prepared statement
primary-ключ
индекс дерево поиска

7 лек

jquery

полезная js библиотека

CDN

Content Delivery Network

нам нужно для сайта чтобы у браузера юзера была какая то библиотека (например jquery)
мы можем

1. просто эту библиотеку скачать на сервер и присылать на запросы
2. брать ее с CDN

CDN это сервера по всему миру на которых лежат всякие популярные библиотеки
их предоставляют всякие компании
из плюсов

- то что все может работать быстрее (если пользователь далеко от нас но не от cdn)
- когда юзер впервые зайдет на сайт вполне возможно что с cdn-а уже будет скачан jquery (т к он популярный), и значит браузеру не будет его снова скачивать и наш сайт отобразится быстрее
- минусы
- сайт не работает без интернета
- проблемы с cdn (их дудосят, инженеры проебались и т п)

что используем из jquery

#todo

ajax

Asynchronous Javascript and XML

-- подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. асинхронный неблокирующий запрос на сервер

window

window это типа супер глобальная переменная
определили у нее метод notify и теперь сможем его вызывать

#todo

почему .prop("content")

#todo

так а зачем нужен ajax?

8 лек

SPRING

фреймворк (система фреймворков, настолько он большой)

мы использовали его часть с :

принцип convention over configuration

вместо того чтобы в конфигурационных файлах много чего настраивать мы просто называем файлы определенным образом и определенным образом кладем их в папки и все работает

фреймворк

— программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

convention over configuration

spring initialiser

еще более высокий уровень (чем boot) -- spring initialiser -- сайт на котором покликаешь какие нужны зависимости, он отдаст архив где будет готовое для spring boot приложение? со всеми зависимостями

spring boot

фреймворк второго порядка, нужен для управления ("оркестрации") spring приложением, облегчает его настройку

используя spring boot можно легко построить веб приложение

spring web

классическая часть спринга, позволяет создавать классические mvc приложения

JPA

раньше мы использовали jdbc, он был довольно низкоуровневым (результаты, соединения и тп). JPA это более высокоуровневый **API** для взаимодействия нашего объектно-ориентированного кода с persistent-слоем, в нашем случае -- реляционной базой данных

Hibernate

Hibernate -- одна из реализаций (стандарта) JPA

ORM

#todo

аннотации

@Repository

@Query

@Controller

@Component

@GetMapping

@ModelAttribute

@Valid

аннотации для валидации

BindingResult

@InitBinder

Spring bind-механика

@Autowired

IoC, DI

лек 9

spring core

иногда можем и через properties аннотацией autowired ???

виды взаимосвязей объектов

практика

interceptor

про ленивость

реально не знал

что в html означает ter template