Оглавление

Протокол HTTP. Структура запросов и ответов, методы запросов, коды ответов сервера, заголовки запросов и ответов.

HTTP — это протокол, позволяющий получать различные ресурсы, например HTML-документы. Протокол HTTP лежит в основе обмена данными в Интернете. HTTP является протоколом клиент-серверного взаимодействия, что означает инициирование запросов к серверу самим получателем, обычно веб-браузером (web-browser).

Браузер (клиент) отправляет серверу HTTP запросы, а сервер отправляет клиенту HTTP ответы. Эти запросы и ответы оформляются по определенным правилам. Есть, что-то вроде синтаксиса, как и в какой последовательности, должно быть написано. Должна быть строго определенная структура.

HTTP запрос состоит из трех основных частей, которые идут в нем именно в том порядке, который указан ниже. Между заголовками и телом сообщения находится пустая строка (в качестве разделителя), она представляет собой символ перевода строки.

- строка запроса (Request Line) - указывает метод передачи, URL-адрес, к которому нужно обратиться и версию протокола HTTP.
- заголовки (Message Headers) - описывают тело сообщений, передают различные параметры и др. сведения и информацию.
- Пустая строка (разделитель)
- тело сообщения (Entity Body) – сами данные, которые передаются в запросе. необязательный параметр

Когда мы получаем ответный запрос от сервера, тело сообщения, чаще всего представляет собой содержимое веб-страницы. Но, при запросах к серверу, оно тоже может иногда присутствовать, например, когда мы передаем данные, которые заполнили в форме обратной связи на сервер.

Методы запросов:

- Метода GET в HTTP используется для получения информации от сервера по заданному URI (URI в HTTP). Запросы клиентов, использующие метод GET должны получать только данные и не должны никак влиять на эти данные. Сервер кэширует ответы
- HTTP метод HEAD работает точно так же, как GET, но в ответ сервер посылает только заголовки и статусную строку без тела HTTP сообщения. (используется для получения метаинформации об объекте без пересылки тела HTTP сообщения) (Метод HEAD часто используется для тестирования HTTP соединений и достижимости узлов и ресурсов, так как нет необходимости гонять по сети содержимое, тестирование HTTP методом HEAD производится гораздо быстрее.) Сервер может кэшировать ответы
- HTTP метод POST используется для отправки данных на сервер, например, из HTML форм, которые заполняет посетитель сайта. Сервер НЕ кэширует ответы
- HTTP метод PUT используется для загрузки содержимого запроса на указанный в этом же запросе URI. (метод POST и метод PUT выполняют совершенно разные операции. Метод POST обращается к ресурсу (странице или коду), которая содержит механизмы обработки сообщения метода POST, а вот метод PUT создает какой-то объект по URI, указанному в сообщение с HTTP методом PUT.)

- HTTP метод DELETE удаляет указанный в URI ресурс.
- HTTP метод CONNECT преобразует существующее соединение в тоннель. (HTTP метод CONNECT используется для преобразования HTTP соединения в прозрачный TCP/IP туннель. Пожалуй, это всё, что можно сказать про HTTP метод CONNECT в контексте рассматриваемого протокола, разве что стоит добавить, что данный метод используется в основном для шифрования соединения (не путайте с кодировкой сообщений).)
- HTTP метод OPTIONS используется для получения параметров текущего HTTP соединения.
- HTTP метод TRACE создает петлю, благодаря которой клиент может увидеть, что происходит с сообщением на всех узлах передачи. (HTTP метод TRACE применяется для диагностики, он позволяет видеть клиенту, что происходит в каждом звене цепочки между компьютером клиента и конечным получателем)

Коды ответов сервера:

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

- Информационные 100 - 199
- Успешные 200 - 299
- Перенаправления 300 - 399
- Клиентские ошибки 400 - 499
- Серверные ошибки 500 - 599

Заголовки запросов и ответов:

Поля общего(general-header) заголовка (он общий как для запросов так и для ответов):

Date:

Указывает дату запроса,например: Date: Sun, 20 Nov 1994 08:12:31 GMT

MIME-version:

Указывает версию MIME (по умолчанию 1.0) MIME-version: 1.0

Pragma:

Содержит указания для таких промежуточных агентов как прокси и шлюзы, Pragma: no-cache

Поля относящиеся к запросу(Request-Header):

Authorization:

Содержит информацию аутентификации Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

From:

Браузер может посылать адрес пользователя серверу From: quake@doom.ru

If-Modified-Since:

используется при методе GET ресурс возвращается ,если он был изменен с указаного момента, может использоваться при кешировании. If-Modified-Since:Mon 15 Jul 1997 00:15:24 GMT

Referer:

Содержит URL предшествующего ресурса.
Referer: http://www.uic.nnov.ru/~paaa/index.html

User-Agent:

Програмное обеспечение клиента. User-Agent: Mozilla/3.0

Заголовок информации сообщения (Entity-Header) применяется как в запросах так и в ответах (при этом некоторые поля только в ответах):

Allow: (в ответе сервера)

Список методов,поддерживаемых ресурсом. Allow: GET, HEAD

Content-Encoding:

идентифицирует метод кодировки,которым был закодирован ресурс Content-Encoding: x-gzip

Content-Length:

Длина тела сообщения Content-Length: 102

Content-Type: Содержит тип ресурса(MIME),для текстовых еще и кодировку символов(необязательно)

Content-Type: text/html; charset=windows-1251

Expires: (в ответе сервера)

Дата окончания действия ресурса,применяется в кешировании для запрета кеширования устаревших ресурсов (в ответе) Expires: Tue, 24 Sep 1998 23:00:15 GMT

Last-Modified: (в ответе сервера)

Время последнего обновления ресурса Last-Modified: Tue, 23 sep 1998 13:48:40 GMT

Другие поля:

Поля Accept: указывают серверу выдавать только указаные форматы данных,которые клиент может распознать.

Accept: text/html Accept: text/plain Accept: image/gif

Поле Host: служит для того , чтобы указать, к какому хосту идет обращение. Данное поле не входит в число обязательных. Однако оно является необходимым в тех случаях, когда одному физическому серверу соответствует несколько виртуальных хостов. В этом поле тогда указывается какой из виртуальных хостов имеется в виду.

Host: www.nnov.city.ru

Заголовок ответа состоит из полей:

Location:

Содержит URI ресурса,может быть использован для переключения клиента в другое место, если например ресурс был перемещен в другое место или на другой сервер.
Location: http://www.uic.nnov.ru/newlocation/index.html

Server:

Информация о программном обеспечении сервера Server: Apache/1.1

WWW-Autenticate:

Параметры аутентификации. WWW-Autenticate: Basic realm="doomsday"

Язык разметки HTML. Особенности, основные теги и атрибуты тегов.

HTML (HyperText Markup Language) - язык разметки гипертекста - предназначен для создания Web-страниц. Под гипертекстом в этом случае понимается текст, связанный с другими текстами указателями-ссылками.

HTML представляет собой достаточно простой набор кодов, которые описывают структуру документа. HTML позволяет выделить в тексте отдельные логические части (заголовки, абзацы, списки и т.д.), поместить на Web-страницу подготовленную фотографию или картинку, организовать на странице ссылки для связи с другими документами.

Каковы особенности языка HTML?

HTML расшифровывается как язык маркировки гипертекста. С его помощью создаются страницы в интернете. Но он не относится к программированию, а является лишь элементом, необходимым для форматирования документов.

Составными частями языка HTML называют теги. От них зависит то, как текст будет выглядеть на странице. Тег задает команду браузеру, с помощью которого происходит просмотр сайтов, он анализирует полученную информацию, преобразует ее и выдает готовый результат в виде форматированных подзаголовков, списков. Теги позволяют также вставлять ссылки, картинки, создавать таблицы на странице.

Основные теги и атрибуты тегов.

- Тег (tag). Тег HTML это компонент, который командует Web- броузеру выполнить определенную задачу типа создания абзаца или вставки изображения.
- Атрибут (или аргумент). Атрибут HTML изменяет тег. Например, можно выровнять абзац или изображение внутри тега.
- Значение. Значения присваиваются атрибутам и определяют вносимые изменения. Например, если для тега используется атрибут выравнивания, то можно указать значение этого атрибута. Значения могут быть текстовыми, типа left или right, а также числовыми, как например ширина и высота изображения, где значения определяют размер изображения в пикселях.

Основные атрибуты HTML*

Существует четыре основных атрибута в HTML, которые могут использоваться для большинства html-элементов (хотя и не для всех):

-- id

Атрибут id html-тега может быть использован для однозначной идентификации любого элемента внутри html-страницы. Существуют две основные причины, по которым Вы можете использовать атрибут id для элемента:

- Если элемент содержит атрибут id как уникальный идентификатор, то можно идентифицировать только этот элемент и его содержимое.
- Если на веб-странице (или в таблице стилей) есть два элемента с одним и тем же именем, Вы можете использовать атрибут id для различения элементов, имеющих одно и то же имя.

-- title

Атрибут title — дает название элемента для элемента. Синтаксис для атрибута title аналогичен атрибуту id.

Поведение этого атрибута будет зависеть от элемента, который его несет, хотя он часто отображается как подсказка, когда курсор наводится на элемент или когда элемент загружается.

-- class

Атрибут class — используется для связывания элемента со списком стилей и задает класс элементу.

-- style

Атрибут style — позволяет указывать правила каскадной таблицы стилей (CSS) внутри элемента.

Атрибут	Опция	Функция
align	right, left, center	Горизонтальное выравнивание тегов.
valign	top, middle, bottom	Вертикально выравнивает тегов внутри html-элемента.
bgcolor	числовые, шестнадцатеричные, RGB значения	Помещает фоновый цвет за элемент.
background	URL	Помещает фоновое изображение за элемент.
id	определяется пользователем	Именование элемента для использования с каскадными таблицами стилей.
class	определяется пользователем	Классифицирует элемент для использования с каскадными таблицами стилей.
width	числовое или процентное значение	Задает ширину таблиц, изображений или ячеек таблицы.
height	числовое или процентное значение	Задает высоту таблиц, изображений или ячеек таблицы.
title	определяется пользователем	«Всплывающий» заголовок элементов.

Структура HTML-страницы. Объектная модель документа (DOM).

DOM – это объектное представление исходного HTML-документа, попытка преобразовать его структуру и содержимое в объектную модель, с которой смогли бы работать различные программы.

В ближайшем приближении DOM – это "дерево узлов" (node tree). У него единый корень, который разветвляется на множество дочерних ветвей, каждая из которых может ветвиться сама и заканчивается "листьями". Корень – это элемент html, а ветви – вложенные элементы.

Модификация с помощью JavaScript DOM позволяет не только просматривать содержимое страницы, но и взаимодействовать с ним, изменять. Это не статичное отображение, а живой ресурс.

Например, с помощью JavaScript можно создать дополнительные элементы:

var newParagraph = document.createElement("p");

var paragraphContent = document.createTextNode("I'm new!");
newParagraph.appendChild(paragraphContent);

document.body.appendChild(newParagraph);

Document метод querySelector() возвращает первый элемент (Element) документа, который соответствует указанному селектору или группе селекторов. Если совпадений не найдено, возвращает значение null.

Основные характеристики объектной модели:

- основана на валидном HTML-коде;
- может быть модифицирована из JavaScript;
- не включает псевдоэлементы, созданные из CSS;
- включает скрытые элементы (display: none).

Объектная модель документа – очень полезная штука. Благодаря ей JavaScript может взаимодействовать со страницей, изменять ее содержимое, структуру и стили. Именно благодаря DOM вы можете отслеживать клиентские события.

DOM – понятие, специфичное для браузерной среды выполнения кода. Это не JavaScript, а лишь API, которым JavaScript может пользоваться.

HTML-формы. Задание метода HTTP-запроса. Правила размещения форм на страницах, виды полей ввода.

Форма предназначена для обмена данными между пользователем и сервером.

Задается с помощью тега

и могут содержать в себе атрибуты: action, содержищий URI обработчика формы (обязательный атрибут), method (по умолчанию, GET), enctype (тип кодирования), accept (MIME-типы для загрузки файлов), name, onsubmit/onreset (обработчик события submit/reset для скриптов), accept-character.

Виды полей:

- Кнопка . Типы кнопок
- Checkbox.
- Radio.
- Select.
- Text и многострочный textarea.
- Password.
- Hidden (скрытое поле).
- File.

Каскадные таблицы стилей (CSS). Структура - правила, селекторы. Виды селекторов, особенности их применения. Приоритеты правил. Преимущества CSS перед непосредственным заданием стилей через атрибуты тегов.

CSS (англ. Cascading Style Sheets — каскадные таблицы стилей) — технология описания внешнего вида документа, оформленного языком разметки.

Преимущественно используется как средство оформления веб-страниц в формате HTML и XHTML, но может применяться с любыми видами документов в формате XML, включая SVG и XUL.

Каскадные таблицы стилей используются создателями веб-страниц для задания цветов, шрифтов, расположения и других аспектов представления веб-документа. Основной целью разработки CSS являлось разделение содержимого (написанного на HTML или другом языке разметки) и оформления документа (написанного на CSS). Это разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печать, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля.

Каскадные таблицы стилей (Cascading Style Sheets, CSS) — это стандарт, определяющий представление данных в браузере. Если HTML предоставляет информацию о структуре документа, то таблицы стилей сообщают как он должен выглядеть.

Стиль — это совокупность правил, применяемых к элементу гипертекста и определяющих способ его отображения. Стиль включает все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице.

Таблица стилей — это совокупность стилей, применимых к гипертекстовому документу.

Каскадирование — это порядок применения различных стилей к веб-странице. Браузер, поддерживающий таблицы стилей, будет последовательно применять их в соответствии с приоритетом: сначала связанные, затем внедренные и, наконец, встроенные стили. Другой аспект каскадирования — наследование (inheritance), — означает, что если не указано иное, то конкретный стиль будет применен ко всем дочерним элементами

гипертекстового документа. Например, если вы примените определенный цвет текста в теге

, то все теги внутри этого блока будут отображаться этим же цветом.

Использование каскадных таблиц дает возможность разделить содержимое и его представление и гибко управлять отображением гипертекстовых документов путем изменения стилей.

Структура - правила, селекторы

Таблицы стилей строятся в соответствии с определенным порядком (синтаксисом), в противном случае они не могут нормально работать. Таблицы стилей составляются из определенных частей (рис. 1):

https://iit-web-lectures.readthedocs.io/ru/latest/_images/css-rule.png Рис. 1. Синтаксис описания стиля CSS

- Селектор (Selector). Селектор — это элемент, к которому будут применяться назначаемые стили. Это может быть тег, класс или идентификатор объекта гипертекстового документа.
- Свойство (Property). Свойство определяет одну или несколько характеристик селектора. Свойства задают формат отображения селектора: отступы, шрифты, выравнивание, размеры и т.д.
- Значение (Value). Значения — это фактические числовые или строковые константы, определяющие свойство селектора.
- Описание (Declaration). Совокупность свойств и их значений.
- Правило (Rule). Полное описание стиля (селектор + описание).

Таким образом, таблица стилей — это набор правил, задающих значения свойств селекторов, перечисленных в этой таблице.

Виды селекторов, особенности их применения

- Селектор по элементу p {color:#666; font-size:12px;}
- Селекторы классов p {color:#666; font-size:12px;} p.redmal { color:#b50404; font-size:11px;}
- Селекторы по идентификатору p#sin { color:#548DD4; }
- Контекстный селектор strong {color:#C00000;}
- Псевдоэлементы p { color:#666; font-size:14px;} p:first-letter { color:#cc0000; font-size:36px;}
- Псевдоклассы a:link {color:#548DD4} a:visited {color:#666666} a:hover {color:#B1030D} a:active {color:#92D050}
- Группировка селекторов p, td, h1 {color:#cc0000; font-size:16px;}

Приоритеты правил

- Универсальный селектор — количество начисляемых баллов равно нулю (0).
- Селекторы тегов и псевдоэлементы — по одному (1) баллу за каждый.
- Селекторы атрибутов, классы и псевдоклассы — по десять (10) баллов за каждый.
- Идентификаторы — по сто (100) баллов за каждый идентификатор находящийся в селекторе.

- Атрибут style — встроенные стили не используют селекторов, а указываются непосредственно внутри тегов элементов, но при этом они имеют самый высокий приоритет исчисляемый тысячей (1000) баллов.

Правило !important используется в тех случаях, когда необходимо, чтобы конкретное свойство было обязательно применено к элементу вне зависимости от того, в каком селекторе и в каком месте CSS-кода оно находится, а также, невзирая на имеющиеся дубли. Общий синтаксис:

селектор { свойство: значение !important; }

- Самый низкий приоритет имеют стили установленные в браузере по умолчанию, как например вертикальные поля у параграфов.
- Более высокий приоритет получают обычные стили подключаемые пользователями.
- Еще более высокий — авторские стили, то есть ваши.
- Еще более высокий достается также авторским стилям, но у которых указан !important.
- И наконец самый высокий приоритет получают стили пользователей с правилом !important. Это сделано для того, чтобы пользователи с ограниченными возможностями в любом случае могли изменить внешний вид страниц сайта под свои нужды.

LESS, Sass, SCSS. Ключевые особенности, сравнительные характеристики. Совместимость с браузерами, трансляция в "обычный" CSS.

LESS - это динамический язык стилей, который является надстройкой над CSS (Поэтому любой CSS код будет валидный LESS).

Приемущества LESS:

Переменные (и области видимости переменных).

Операции (в том числе и для управления цветом, т.е можно смешивать цвета: #941f1f + #222222).

И другие функции для работы с цветом (осветление, затемнение и т.п.)

Вложенность (можно вложить одно правило в другое, article.post p {} <=> article.post { p{ }}).

Объединение аргуметов.

LESS-файл конвертируется в CSS при помощи js (для этого необходимо скачать less.js с сайта LESS).

<script src="less.js" type="text/javascript"></script>

Затем можно привязывать файлы с расширением .less.

<link rel="stylesheet/less" type="text/css" href="style.less">

Sass

Sass - это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей.

Приемущества Sass:

Вложенные правила.

Переменные.

Возможность создавать миксины, позволяющие создавать многоразовые CSS-правила - группы деклараций, для многократного использования. (LESS в это не может)

Расширения. Одиночный селектор может быть расширен больше, чес одним селектором с помошью @extend.

Есть логика. (if/then/for). (Этого в LESS тоже нет)

Не может компилироватся на сервере в CSS (LESS использует js).

Браузер не распознает файлы Sass, так что сначала их нужно скомпилировать в обычный CSS.

SCSS

SCSS — "диалект" языка SASS. Отличие SCSS от SASS заключается в том, что SCSS больше похож на обычный CSS код.

@import - @import "template" подключит template.scss.

Вложенность.

$переменные.

Математика чисел и цветов.

Строки (умеет складвать строки, поддерживает конструкцию #{$var})

Файлы, заканчивающиеся на .scss представляют собой стандартный синтаксис, поддерживаемый Sass. SCSS - это надмножество CSS. Файлы, заканчивающиеся на .sass представляют собой "старый" синтаксис, поддерживаемый Sass, происходящим в мире ruby.

Клиентские сценарии. Особенности, сферы применения. Язык JavaScript.

При разработке динамических web-сайтов в большинстве случаев целесообразно разделять бизнес-логику между клиентом и сервером, чтобы добиться оптимальной производительности в условиях низкоскоростных каналов Интернета и лимитированных ресурсов Web-серверов.

Например, предварительную обработку введенных данных, отправляемых серверу, имеет смысл выполнять на стороне клиента. Это позволит исключить повторные передачи неправильно заполненных форм. Графическое представление результатов запроса также стоит выполнять на стороне клиента, что существенно сократит объем данных, передаваемых по сети.

Сферы применения

- Оперативная проверка достоверности заполняемых пользователем полей форм HTML до передачи их на сервер. Сценарии JavaScript способны обрабатывать данные, введенные пользователями в полях форм, а также события, возникающие в процессе манипуляций пользователя с мышью, копировать в окно браузера другие страницы HTML или изменять содержимое уже загруженных страниц.
- Создание динамических HTML-страниц совместно с каскадными таблицами стилей и объектной моделью документа.
- Взаимодействие с пользователем при решении "локальных" задач, решаемых приложением JavaScript, встроенном в HTML-страницу. В частности, сценарии JavaScript широко применяются для создания различных визуальных эффектов. Например, изменение внешнего вида элементов управления, над которыми установлен курсор мыши, анимация графических изображений, создание звуковых эффектов и т. д. Механизм локальной памяти Cookie позволяет сценариям JavaScript сохранять на компьютере локальную информацию, введенную пользователем. Например, в Cookie может храниться список товаров из Интернет-магазина, отобранных для покупки.

Основные архитектурные черты:

- динамическая типизация;
- слабая типизация;
- автоматическое управление памятью;
- прототипное программирование;
- функции как объекты первого класса.

Версии ECMAScript, новые возможности ES6 и ES7.

ECMAScript — это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков. Стандартизирован международной организацией ECMA в спецификации. (ECMAScript это стандарт, а JavaScript его реализация).

ES6

Это обновление добавило новый синтаксис для написания классов и модулей, добавились итераторы и циклы for/of, Python-style генераторы, двоичные данные, лямбда-выражения, типизированные массивы, коллекции, обещания (promises), рефлексию и прокси, усовершенствовали числа и математику. Добавлено ключевое слово let (которое помогает объявить переменной область видимости - блок) и const.

ES7

- Метод includes — это простой метод объектов типа Array, который позволяет выяснить, имеется ли в массиве некий элемент (он, в отличие от indexOf, подходит и для работы со значениями NaN).

- инфиксный оператор, который используется для возведения в степень. Такой оператор, выглядящий как «**», был представлен в ECMAScript 2016, он может служить заменой Math.pow().

Синхронная и асинхронная обработка HTTP-запросов. AJAX.

Синхронный запрос - запрос с ожидением ответа. (скрипт послал запрос (объект) на сервер и ждет ответ).

Асинхронный запрос - запрос без ожидания ответа от сервера. (скрипт послал запрос (объект) на сервер, но продолжает выполняться, когда данные вернутся вступает в дело событие onreadystatechenge. Сам объект меняет это событие, когда у него меняется свойство readyState. Для события создается собственная функция, в которой проверяется свойство readyState. И как только оно становится равным "4" - это значит, что данные с сервера пришли. Теперь можно полученные данные обрабатывать).

```
   var request = getXmlHttpRequest(); // создание объекта
   request.onreadystatechenge = function(){ // установка обработчика
onreadystatechenge и проверка свойства readyState

       if(request.readyState == 4)
       ...
   }
   request.open('GET', url, true); // готовим запрос
   request.send(null); // посылаем запрос
```

Asynchronous Javascript and XML - подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.

Пользователь что-то делает -> скрипт определяет с чем там надо работать -> браузер отправляет запрос на сервер -> сервер возвращает только то, от чего ожидаются изменения -> скрипт вносит изменения обратно (без перезагрузки страницы).

Библиотека jQuery. Назначение, основные API. Использование для реализации AJAX и работы с DOM.

jQuery – это библиотека JavaScript (т.е., она написана на JavaScript), предназначенная для абстрагирования, выравнивания, исправления и упрощения скриптинга при работе с узлами HTML-элементов в браузере или для работы в браузере без графического интерфейса.

jQuery включается в страницу как внешний файл.

```
<script src="jquery-2.2.2.min.js">
```

Вся работа с jQuery ведется с помощью функции $. Работу с jQuery можно разделить на 2 типа:

- Получение jQuery-объекта с помощью функции $().
- Вызов глобальных методов у объекта $.

Типичный пример манипуляции сразу несколькими узлами DOM заключается в вызове $ функции со строкой селектора CSS, что возвращает объект jQuery, содержащий некоторое количество элементов HTML-страницы. Эти элементы затем обрабатываются методами jQuery.

```
$("div.test").add("p.quote").addClass("blue").slideDown("slow");
//находит элементы div с классом test, все элементы p с классом quote, добавляет им
класс blue, ...
```

Итак:

- Абстрагируется интерфейс объектной модели документа (он же DOM API).
- Выравниваются все различия реализаций DOM, существующие между браузерами.
- Исправляются известные браузерные баги, связанные с CSS и DOM.

Все это оборачивается в более простой и менее корявый API, нежели нативный API DOM – вот вам и библиотека jQuery.

Теперь позвольте объяснить, что я понимаю под “скриптингом HTML-элементов”. При помощи jQuery совершенно тривиально решаются задачи вроде «визуально скрыть второй элемент h2 в документе .html. Код jQuery для такой задачи выглядел бы так:

```
jQuery('h2:eq(1)').hide();
```

Теперь сравним его с нативным DOM-кодом, который потребовалось бы написать, если бы мы не работали с jQuery.

```
document.querySelectorAll('h2')[1].style.setProperty('display','none');
```

Две базовые концепции, на которых основана jQuery, таковы: “найди и сделай” и “пиши меньше, делай больше.” Две этих концепции можно развернуть и переформулировать в виде следующего утверждения: первоочередная задача jQuery заключается в организации выбора (то есть, нахождении) или в создании HTML-элементов для решения практических задач. Без jQuery для этого потребовалось бы больше кода и больше сноровки в обращении с DOM. Достаточно вспомнить рассмотренный выше пример с сокрытием элемента h2.

Следует отметить, что круг возможностей jQuery этим не ограничивается. Она не просто абстрагирует нативные DOM-взаимодействия, но и абстрагирует асинхронные HTTP-запросы (т.н. AJAX) при помощи объекта XMLHttpRequest. Кроме того, в ней есть еще ряд вспомогательных решений на JavaScript и мелких инструментов. Но основная польза jQuery заключается именно в упрощении HTML-скриптинга и просто в том, что с ней приятно работать.

$.ajax и соответствующие функции позволяют использовать методы AJAX

```
$.ajax({
  type: "POST",
  url: "some.php",      // обращение к some.php
  data: {name: 'John', location: 'Boston'},   //с какими-то параметрами
  success: function(msg){
    alert( "Data Saved: " + msg );  // получный результат выводится в alert
  }
});
```

Реализация AJAX с помощью SuperAgent.

Библиотека Superagent, как и Axios, подходит для Node.js и для современных браузеров. Она предоставляет разработчику простое и понятное API, с которым удобно работать.

Что такое Node.js?

Node.js (или просто Node) — это серверная платформа для работы с JavaScript через движок V8. JavaScript выполняет действие на стороне клиента, а Node — на сервере. С помощью Node можно писать полноценные приложения. Node умеет работать с внешними библиотеками, вызывать команды из кода на JavaScript и выполнять роль веб-сервера.

В чём преимущество Node?

С Node проще масштабироваться. При одновременном подключении к серверу тысяч пользователей Node работает асинхронно, то есть ставит приоритеты и распределяет ресурсы грамотнее. Java же, например, выделяет на каждое подключение отдельный поток.

Откуда Node вообще взялся?

Node появился в 2009 году благодаря Райану Далу. До этого в серверах царил подход «один поток на каждое соединение», а Дал придумал использовать систему, которая ориентирована на события. То есть реагирует на действие или бездействие и выделяет под это ресурс. Главная цель Node — построение масштабируемых сетевых серверов.

SuperAgent позиционируется разработчиками как легковесная и расширяемая библиотека для работы с AJAX. Для того чтобы выполнить HTTP-запрос средствами Superagent, достаточно вызвать подходящий метод объекта request:

request .post('/api/pet') .send({ name: 'Manny', species: 'cat' }) .set('X-API-Key', 'foobar') .set('Accept', 'application/json') .then(res =>{ console.log('I got: ' + JSON.stringify(res.body)); }).catch(error => { console.log("Something wrong", errror); });

Сильные стороны

Поддерживает плагины. Поддаётся конфигурированию. Имеет приятный интерфейс для выполнения HTTP-запросов. Поддерживает объединение в цепочку нескольких вызовов для выполнения запросов. Работает в среде Node.js и в браузерах. Поддерживает индикацию прогресса для выгрузки и загрузки данных. Поддерживает механизм chunked-transfer encoding. Поддерживает коллбэки. Для этой библиотеки разработано множество плагинов.

Слабые стороны

Имеет своеобразное API, не придерживающееся каких-либо стандартов.

Серверные сценарии. CGI - определение, назначение, ключевые особенности.

CGI (Common Gateway Interface — общий интерфейс шлюза) — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято называть шлюзом (оно же скрипт или CGI-программа). По сути позволяет использовать консоль ввода и вывода для взаимодействия с клиентом.

Поскольку гипертекст статичен по своей природе, веб-страница не может непосредственно взаимодействовать с пользователем. До появления JavaScript, не было иной возможности отреагировать на действия пользователя, кроме как передать введенные им данные на веб-сервер для дальнейшей обработки. В случае CGI эта обработка осуществляется с помощью внешних программ и скриптов, обращение к которым выполняется через стандартизованный (см. RFC 3875: CGI Version 1.1) интерфейс — общий шлюз.

Обобщенный алгоритм работы через CGI можно представить в следующем виде:

- Клиент запрашивает CGI-приложение по его URI.
- Веб-сервер принимает запрос и устанавливает переменные окружения, через них приложению передаются данные и служебная информация.
- Веб-сервер перенаправляет запросы через стандартный поток ввода (stdin) на вход вызываемой программы.
- CGI-приложение выполняет все необходимые операции и формирует результаты в виде HTML.
- Сформированный гипертекст возвращается веб-серверу через стандартный поток вывода (stdout). Сообщения об ошибках передаются через stderr.
- Веб-сервер передает результаты запроса клиенту.

Наиболее частая задача, для решения которой применяется CGI — создание интерактивных страниц, содержание которых зависит от действий пользователя. Типичными примерами таких веб-страниц является форма регистрации на сайте или форма для отправки комментария. Другая область применения CGI, остающаяся за кулисами взаимодействия с пользователем, связана со сбором и обработкой информации о клиенте: установка и чтение «печенюшек»-cookies; получение данных о браузере и операционной системе; подсчет количества посещений веб-страницы; мониторинг веб-трафика и т.п.

Преимущества CGI

- Процесс CGI скрипта не зависит от Веб-сервера и, в случае падения, никак не отразится на работе последнего
- Может быть написан на любом языке программирования
- Поддерживается большинством Веб-серверов

Недостатки

Самым большим недостатком этой технологии являются повышенные требования к производительности веб-сервера. Дело в том, что каждое обращение к CGI-приложению вызывает порождение нового процесса, со всеми вытекающими отсюда накладными расходами. Если же приложение написано с ошибками, то возможна ситуация, когда оно,

например, зациклится. Браузер прервет соединение по истечении тайм-аута, но на серверной стороне процесс будет продолжаться, пока администратор не снимет его принудительно.

FastCGI - особенности технологии, преимущества и недостатки относительно CGI.

Дальнейшее развитие технологии CGI, является более производительным и безопасным, снимает множество ограничений CGI-программ.

CGI — устаревшая технология, позволяющая взаимодействовать веб-серверу с сервером приложений. Для каждого запроса запускается процесс с интерпретатором PHP, после возвращения ответа он завершается. Поскольку это очень неэффективно, был создан FastCGI, в котором процесс интерпретатора не завершается, а используется для последующих запросов.

Особенности технлогии

FastCGI программа работает следующим образом: программа единожды загружается в память в качестве демона (независимо от HTTP-сервера), а затем входит в цикл обработки запросов от HTTP-сервера. Один и тот же процесс обрабатывает несколько различных запросов один за другим, что отличается от работы в CGI-режиме, когда на каждый запрос создается отдельный процесс, "умирающий" после окончания обработки.

Недостатки

Написание FastCGI программ-демонов сложнее чем CGI, нужны дополнительные библиотеки, зависящие от языка.

Язык PHP - синтаксис, типы данных, встраивание в веб-страницы, правила обработки HTTP-запросов. Особенности реализации принципов ООП в PHP.

PHP - скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений.

Типы данных

PHP - яп с динамической типизацией. Преобразования меджду скалярными типами происходят неявно. Скалярные типы: integer, float, double, boolean, string. Нескалярные: array, object, resource, null Псевдотипы: mixed (любой тип), number, callback (string или анонимная фунуция), void.

ООП в PHP

Следует учитывать, что в PHP несколько упрощенная реализация ООП (объектно-ориентированное программирование). Поэтому, когда речь идёт об ООП как абстрактной парадигме, то следует использовать какой-то более серьёзный язык, вроде Java, C++ или Object Pascal. Потому что на этих языкам можно посмотреть практическую реализацию принципов ООП. В PHP программисты пытаются подражать другим ЯП, что в итоге приводит к излишней сложности и путанице, поскольку язык сам по себе не позволяет сделать «как в теории».

Главная проблема такого (спагетти) кода в том, что у него низкая читабельность и слишком большая запутанность. Там где нужно выполнить какой-то один метод, подтягивается еще десяток классов. При этом каждый класс в отдельном файле, что может окончательно свести с ума даже опытных программистов.

```
class ClassName {
  public $publicName;
  private $privateName;
  protected $protectedName;

  const CONST_VAL = 'val';

  public function getPrivateName() {
    return $this->$privateName; // $this -- ссылка на сам объект, $parent - на
родительский.
  }
};

echo ClassName::CONST_VAL; // для обращения к константам

$classname = new ClassName();

$className->publicName; // доступ к переменной
```

PHP поддерживает все три основных механизма ООП — инкапсуляцию, полиморфизм подтипов и наследование (с помощью extend). Поддерживаются интерфейсы (с помощью implements). Есть абстрактные и final методы и классы. Множественное наследование не поддерживается, но класс может реализовывть несколько интерфейсов или с помощью механизма особенностей (trait), который имеет средства для разрешения конфликтов.

Методы: __construct() -- конструктор __destruct() -- для деинициализации объекта __get(), __set() __sleep(), __wakeup() __clone()

Polling

Первое решение, которое приходит в голову для получения событий с сервера - это "поллинг" (polling), т.е периодический опрос сервера стандартными пакетами: "я тут, изменилось ли что-нибудь?”

В ответ сервер во-первых помечает у себя, что клиент онлайн, а во-вторых посылает датаграмму, в которой в специальном формате содержится весь пакет событий, накопившихся к данному моменту

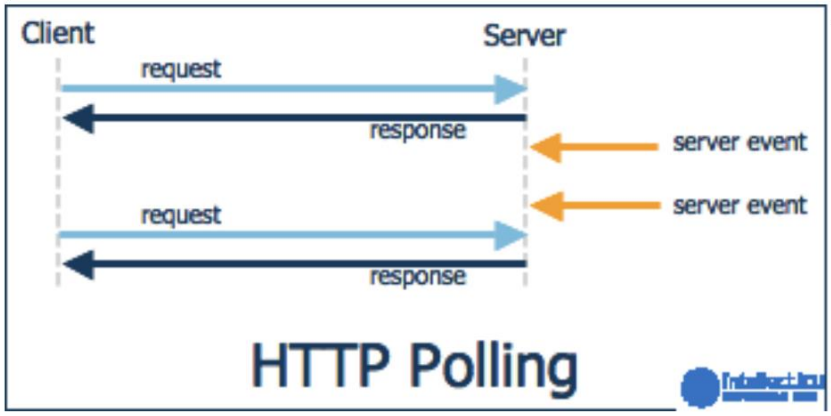
У этого способа есть одна большая проблема, а именно - большие задержки между созданием и получением данных. Сервер отсылает их не тогда, когда они появились, а когда настанет время очередного запроса.

Задержка = время между опросами + установление соединения + передача данных.

Другой минус - лишний входящий трафик на сервер. При каждом запросе браузер передает множество заголовков, причем заголовки всегда идут в несжатом виде. Для некоторых приложений входящий трафик заголовков может в 10 и более раз превосходить исходящий трафик реальных данных.

- Задержки между событием и уведомлением
- Входящий трафик на сервер
- Простота реализации

Polling



Самый простой, но самый не эффективный, метод: клиент раз в несколько секунд опрашивает сервер на наличие событий. Даже если ничего нет, то клиент всеравно 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

Чтобы сэкономить на ресурсах, можно использовать long polling. Устроен он так же, как и polling, с одним отличием: сервер дольше отвечает. Вообще, при лонг-поллинге сервер отвечает в двух случаях: или потому, что пришло новое сообщение, или потому, что соединение пора разрывать.

У каждого запроса есть timeout — время, в течении которого нужно ответить. Если на запрос не ответили за это время, считается, что сервер не ответит вообще.

Поэтому сервер смотрит на timeout и решает так:

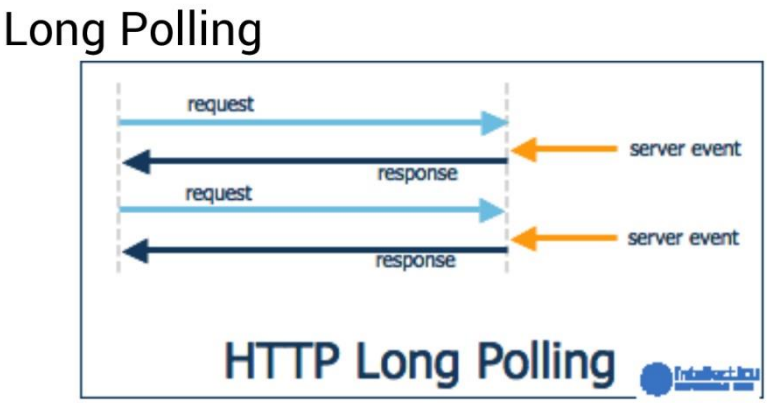
- Если за это время у меня не появится обновлений для клиента, я отвечу ему, что их нет.
- Если появятся, я отправлю ему обновления сразу, не дожидаясь таймаута.

__Чтобы реализовать long polling на стороне клиента, нужно выставить большой timeout: 30 или 60 секунд.

Итак, что же из себя представляет Long Polling?

Выглядит это примерно следующим образом:

- Клиент отсылает на сервер обычный ajax-запрос
- Сервер, вместо того, чтобы быстро обработать этот запрос и отправить ответ клиенту, запускает цикл, в каждой итерации которого следит за возникновением событий (другой клиент добавил запись или удалил).
- При возникновении события сервер генерирует ответ и отсылает его клиенту, таким образом завершая запрос.
- Клиент, получив ответ от сервера, запускает обработчик события и параллельно отправляет очередной «длинный» запрос серверу.



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 временная точность событий

— Сервер хранит события только на время реконнекта

Минусы по сравнению с Polling:

— Более сложная схема

Ку́ки (англ. cookie, буквально — печенье) — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса.

Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- сведения статистики о пользователях.

Cookie легко перехватить и подменить (например, для получения доступа к учётной записи), если пользователь использует нешифрованное соединение с сервером. В группе риска пользователи, выходящие в интернет при помощи публичных точек доступа Wi-Fi и не использующие при этом таких механизмов, как SSL и TLS. Шифрование позволяет также решить и другие проблемы, связанные с безопасностью передаваемых данных.

Большинство современных браузеров позволяет пользователям выбрать — принимать cookie или нет, но их отключение делает невозможной работу с некоторыми сайтами. Кроме того, по законам некоторых стран (например, по закону Евросоюза от 2016 года, см. общий регламент по защите данных) сайты должны в обязательном порядке запрашивать согласие перед установкой cookie.

Назначение

- Cookie используются веб-серверами для идентификации пользователей и хранения данных о них. К примеру, если вход на сайт осуществляется при помощи cookie, то, после ввода пользователем своих данных на странице входа, cookie позволяют серверу запомнить, что пользователь уже идентифицирован и ему разрешён доступ к соответствующим услугам и операциям.
- Многие сайты также используют cookie для сохранения настроек пользователя. Эти настройки могут использоваться для персонализации, которая включает в себя выбор оформления и функциональности. Например, Википедия позволяет авторизованным пользователям выбрать дизайн сайта. Поисковая система Google позволяет пользователям (в том числе и не зарегистрированным в ней) выбрать количество результатов поиска, отображаемых на одной странице.
- Cookie также используются для отслеживания действий пользователей на сайте. Как правило, это делается с целью сбора статистики, а рекламные компании на основе такой статистики формируют анонимные профили пользователей для более точного нацеливания рекламы.