# Протокол HTTP. Особенности, типы запросов.

**HyperText Transfer Prоtocоl**

**прикладного уровня (на это уровне так же FTP и SMTP)**

**технология «клиент-сервер»**

**используется также в качестве «транспорта» для других протоколов прикладного уровня:**

**SOAP**

**(Simple Object Access Protocol)**

**протокол обмена структурированными сообщениями в распределённой вычислительной среде**

**XML-RPC**

**(Extensible Markup Language Remote Procedure Call)**

**XML-вызов удалённых процедур**

**WebDAV**

**(Web-based Distributed Authoring and Versioning)**

**защищённый сетевой протокол высокого уровня, работающий поверх HTTP для доступа к объектам и коллекциям.**

**URI**

**Uniform Resource Identifier**

**Не обязательно фаил**

**возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку**

**=> можно обмениваться двоичными данными, хотя данный протокол является текстовым**

**Преимущества**

**Простота**

**Протокол настолько прост в реализации, что позволяет с лёгкостью создавать клиентские приложения.**

**Расширяемость**

**Возможности протокола легко расширяются благодаря внедрению своих собственных заголовков,**

**с помощью которых можно получить необходимую функциональность при решении специфической задачи.**

**При этом сохраняется совместимость с другими клиентами и серверами: они будут просто игнорировать неизвестные им заголовки.**

**Распространённость**

**документации по протоколу на многих языках мира,**

**поддержка со стороны IDE,**

**поддержка протокола в качестве клиента многими программами**

**много хостинга с HTTP.**

**Недостатки и проблемы**

**Большой размер сообщений**

**из за**

**текстового формата**

**решается**

**кэширования на стороне клиента**

**компрессии**

**diff-кодирование - передача только измененной части документа**

**Отсутствие «навигации»**

**решения**

**site map**

**в расширяющем HTTP протоколе WebDAV**

**с помощью добавленного метода PROPFIND**

**Нет поддержки распределённости**

**Роли ПО**

**клиент**

**сервер**

**прокси**

**Структура протокола**

**Стартовая строка - определяет тип сообщения**

**запрос**

**GET URI — для версии протокола 0.9.**

**Метод URI HTTP/Версия — для остальных версий. (GET /wiki/HTTP HTTP/1.0)**

**ответ**

**HTTP/Версия КодСостояния Пояснение (HTTP/1.0 200 OK)**

**методы бывают:**

**OPTIONS**

**OPTIONS \* HTTP/1.1 - типа пинга + тест на поддержку 1.1**

**Результат выполнения этого метода не кэшируется**

**Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса**

**В ответ серверу следует включить заголовок Allow со списком поддерживаемых методов**

**GET**

**Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс.**

**GET /path/resource?param1=value1&param2=value2 HTTP/1.1**

**идемпотентны — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам.**

**(Это позволяет кэшировать ответы на запросы GET.)**

**HEAD**

**Как гет, только тела нет...**

**для - извлечения метаданных, проверки наличия ресурса**

**Заголовки ответа могут кэшироваться.**

**При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.**

**POST**

**Применяется для передачи пользовательских данных заданному ресурсу.**

**В отличие от метода GET, не считается идемпотентным**

**При результатах выполнения 200 (Ok) и 204 (No Content) в тело ответа следует включить сообщение об итоге выполнения запроса.**

**Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location.**

**Сообщение ответа сервера на выполнение метода POST не кэшируется.**

**PUT**

**для создания новых ресурсов**

**Применяется для загрузки содержимого запроса на указанный в запросе URI.**

**Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created).**

**Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content)**

**PATCH**

**Аналогично PUT, но применяется только к фрагменту ресурса.**

**DELETE**

**Удаляет указанный ресурс.**

**TRACE**

**Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе.**

**LINK**

**Устанавливает связь указанного ресурса с другими.**

**UNLINK**

**Убирает связь указанного ресурса с другими.**

**CONNECT**

**Преобразует соединение запроса в прозрачный TCP/IP туннель, обычно чтобы содействовать установлению защищенного SSL соединения через не шифрованный прокси.**

**Коды состояний:**

**1xx Informational**

**2xx Success**

**3xx Redirection**

**4xx Client Error**

**5xx Server Error**

**Заголовки HTTP — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение**

**Пример:**

**Server: Apache/2.2.11 (Win32) PHP/5.3.0**

**Last-Modified: Sat, 16 Jan 2010 21:16:42 GMT**

**Content-Type: text/plain; charset=windows-1251**

**Content-Language: ru**

**Группы:**

**General Headers (рус. Основные заголовки) — должны включаться в любое сообщение клиента и сервера.**

**Request Headers (рус. Заголовки запроса) — используются только в запросах клиента.**

**Response Headers (рус. Заголовки ответа) — только для ответов от сервера.**

**Entity Headers (рус. Заголовки сущности) — сопровождают каждую сущность сообщения.**

**Тело сообщения — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.**

**Тело HTTP сообщения (message-body), если оно присутствует, используется для передачи тела объекта, связанного с запросом или ответом.**

**Тело сообщения (message-body) отличается от тела объекта (entity-body) только в том случае,**

**когда применяется кодирование передачи, что указывается полем заголовка Transfer-Encoding.**

**Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа только тело сообщения.**

# Java-сервлеты. Особенности реализации, ключевые методы. Контейнеры сервлетов.

**Что делает сервлет?**

**Когда вы работаете с интерактивным Web-сайтом, все, что вы видите, отображается в браузере. За кулисами процесса Web-сервер принимает от вас запросы во время сессии, возможно, передает их в другой код (возможно, другим серверам) для обработки запроса и обращения к данным, а также генерирует результаты для отображения в браузере.**

**Сервлет - это диспетчер процесса. Он находится на Web-сервере и обрабатывает входящие запросы и исходящие ответы. Вообще говоря, он не имеет ничего общего с представлением и, в действительности, не должен иметь. Вы можете использовать сервлет для записи в поток, который добавляет содержимое к Web-странице, но это, обычно, не очень хорошая идея, поскольку происходит смешение логики представления и бизнес-логики.**

**Большинство Java-сервлетов предназначены для ответов на HTTP-запросы в контексте Web-приложения.**

**(javax.servlet и javax.servlet.http)**

**При создании Java-сервлета обычно создается подкласс HttpServlet.**

**Этот класс имеет методы, предоставляющие доступ к конвертам запроса и ответа для обработки запросов и создания ответов.**

**вход HttpServletRequest**

**выход HttpServletResponse**

**Контейнер, например Tomcat, управляет средой исполнения для сервлетов.**

**мост от URL (введенного пользователем в браузере) к серверным компонентам, обрабатывающим запрос, в который транслируется URL.**

**Во время работы вашего приложения контейнер загружает и инициализирует ваш сервлет (сервлеты) и управляет его жизненным циклом.**

**Вот обычный сценарий:**

**Пользователь вводит URL в браузере. Конфигурационный файл вашего Web-сервера указывает, что этот URL предназначен для сервлета, управляемого контейнером сервлетов на вашем сервере.**

**Если экземпляр сервлета еще не был создан (существует только один экземпляр сервлета для приложения), контейнер загружает класс и создает экземпляр объекта.**

**Контейнер вызывает метод init() сервлета.**

**Контейнер вызывает метод service() сервлета и передает HttpServletRequest и HttpServletResponse.**

**Сервлет обычно обращается к элементам запроса, передает запрос другим серверным классам для выполнения запрошенной службы и для доступа к таким ресурсам, как базы данных, а затем создает ответ, используя эту информацию.**

**При необходимости, когда сервлет выполнил полезную работу, контейнер вызывает метод destroy() сервлета для его финализации.**

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

**Язык HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1989—1991 годах в стенах Европейского совета по ядерным исследованиям в Женеве (Швейцария). HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. HTML успешно справлялся с проблемой сложности SGML путём определения небольшого набора структурных и семантических элементов — дескрипторов. Дескрипторы также часто называют «тегами». С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Мультимедийные возможности были добавлены позже.**

**Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). В идеале, текст с разметкой HTML должен был без стилистических и структурных искажений воспроизводиться на оборудовании с различной технической оснащённостью (цветной экран современного компьютера, монохромный экран органайзера, ограниченный по размерам экран мобильного телефона или устройства и программы голосового воспроизведения текстов). Однако современное применение HTML очень далеко от его изначальной задачи. Например, тег <TABLE>, несколько раз использованный для форматирования страницы, которую вы на даный момент читаете, предназначен для создания в документах самых обычных таблиц, но, как можно убедиться, здесь нет ни одной таблицы. С течением времени, основная идея платформонезависимости языка HTML была отдана в своеобразную жертву современным потребностям в мультимедийном и графическом оформлении.**

**Каждый HTML-документ, отвечающий спецификации HTML какой-либо версии, должен начинаться со строки объявления версии HTML <!DOCTYPE…>, которая обычно выглядит примерно так:**

**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"**

**"http://www.w3.org/TR/html4/strict.dtd">**

**Если эта строка не указана, то добиться корректного отображения документа в браузере становится труднее.**

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

**DOM (от англ. Document Object Model — «объектная модель документа») — это не зависящий 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.**

**Модель DOM не налагает ограничений на структуру документа. 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. Узлы связаны между собой отношениями "родительский-дочерний".**

**Изначально различные браузеры имели собственные модели документов (DOM), несовместимые с остальными. Для того чтобы обеспечить взаимную и обратную совместимость, специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.**

**Еще один интерфейс прикладного программирования, называемый JDOM, обеспечивает более высокий, чем W3C DOM, уровень для работы с XML-документами на Java.**

**HTML-формы. Виды полей ввода. Задание метода HTTP-запроса.**

**<form action="action.php" name="myform" method="post">**

**<input type="text" name="mytext" size="50">**

**<textarea name="msg" cols="20" rows=”10” ></textarea>**

**<input name="Submit" type=submit value="Отправить данные">**

**</form>**

**Клиентские сценарии. Особенности, сферы применения. Язык JavaScript.**

**Синхронная и асинхронная обработка HTTP-запросов. AJAX.**

**Каскадные таблицы стилей (CSS). Особенности применения и преимущества перед непосредственным заданием стилей через атрибуты тегов.**

# Клиентские сценарии. Особенности, сферы применения. Язык JavaScript.

**Сценариями JavaScript называются программы, работающие с объектами HTML-документа. Параметры элеменов документа, заданные с помощью атрибутов соответствующих тегов и таблиц стилей, можно изменить или даже заменить весь загруженный HTML-документ на другой. Сделать это можно с помощью сценариев JavaScript и представляющими их объектами.**

**JavaScript — объектно-ориентированный скриптовый язык программирования. Является диалектом языка ECMAScript[~ 1].**

**JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.**

**Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.**

**JavaScript обладает рядом свойств объектно-ориентированного языка, но реализованное в языке прототипирование обусловливает отличия в работе с объектами по сравнению с традиционными объектно-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.**

**клиентские скрипты - специальные веб-сценарии, которые дают возможность изменять содержимое html-страницы без перезагрузки самой страницы с сервера. Часто клиентские скрипты встраиваются прямо в html‑код страницы и для их выполнения не требуется установка какого-либо дополнительного программного обеспечения. Все что нужно - это браузер с поддержкой клиентских скриптов. Все современные интернет-браузеры (кроме некоторых старых версий) поддерживают выполнение распространенных клиентских скриптов.**

**К языкам веб-программирования, предназначенным для создания клиентских скриптов относят Javascript, VBScript, ActionScript, использующийся в технологии flash и SilverLight. Остановимся подробнее на языках Javascript и VBScript, как на наиболее ярких представителях семейства клиентских языков веб-программирования.**

**Клиентские скрипты**

**Javascript**

**Среди клиентских языков веб-программирования почетное место заслуженно занимает Javascript. Язык Javascript был разработан в начале 90-х годов прошлого века фирмой Netscape. Изначально разрабатывался для взаимодействия исключительно с браузером Netscape Navigator - продуктом компании Netscape. Однако на сегодняшний день это язык веб-программирования получил широчайшее распространение и практически все браузеры его поддерживают. Кроме того, сценарии Javascript поддерживаются в таких приложениях как Adobe Photoshop, Adobe Dreamweaver, Adobe Illustrator или Adobe InDesign, которые активно используются профессионалами для создания веб-дизайна.**

**Клиентские скрипты**

**Что же делает Javascript таким популярным и востребованным?**

**Самый распространенный пример - заполнение регистрационных форм. Клиентский скрипт проверяет данные в форме еще до отправки на сервер и в случае ошибки указывает на нее. Остальные данные при этом сохраняются в динамической памяти, и нет необходимости при ошибке в одном поле ввода еще раз полностью проходить процесс заполнения. Других подобных случаев, в которых применение Javascript реализует задачи, недоступные для статических страниц, еще множество. Среди них: изменение содержимого страницы в ответ на действие пользователя; создание всплывающих подсказок; реагирование на клик мыши, движение курсора.**

**Синхронная и асинхронная обработка HTTP-запросов. AJAX.**

**Каскадные таблицы стилей (CSS). Особенности применения и преимущества перед непосредственным заданием стилей через атрибуты тегов.**

# Синхронная и асинхронная обработка HTTP-запросов. AJAX.

**В синхронной модели браузер отправляет запрос на сервер и висит, ждет, пока тот совершит всю необходимую работу. Сервер выполняет запросы к базе данных, заворачивает ответ в необходимый формат и выводит его. Браузер. получив ответ, вызывает функцию показа.**

**Все процессы выполняются последовательно, один за другим.**

**Сетевые задержки включены во время ожидания, обозначенное на схеме серым цветом.**

**Пользователь не может заниматься чем-то другим на этой же странице, пока происходит синхронный обмен данными.**

**В асинхронной модели запрос отсылается ("удочка поставлена"), и можно заняться чем-то другим. Когда запрос выполнился ("клюнуло") - запускается заранее**

**подготовленная программистом функция ("подтянуть спиннинг") показа сообщения сервера.**

**Из-за такого разрыва между действием и реальным результатом приложение становится гораздо более чувствительно к ошибкам.**

**Особенно в случае нескольких одновременных асинхронных запросов, нужно заботиться об очередности выполнения и ответа (race-conditions) и, в случае ошибки, оставлять приложение в целостном (consistent) состоянии.**

**AJAX, 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 более быстрыми и удобными.**

**AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:**

**использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например:**

**с использованием XMLHttpRequest (основной объект);**

**через динамическое создание дочерних фреймов[1];**

**через динамическое создание тега <script>[2].**

**использование DHTML для динамического изменения содержания страницы;**

**В качестве формата передачи данных обычно используются JSON или XML.**

**Преимущества**

**Экономия трафика**

**Уменьшение нагрузки на сервер**

**Ускорение реакции интерфейса**

**Недостатки**

**Отсутствие интеграции со стандартными инструментами браузера**

**Динамически создаваемые страницы не регистрируются браузером в истории посещения страниц, поэтому не работает кнопка «Назад», предоставляющая пользователям возможность вернуться к просмотренным ранее страницам, но существуют скрипты, которые могут решить эту проблему.**

**Другой недостаток изменения содержимого страницы при постоянном URL заключается в невозможности сохранения закладки на желаемый материал. Частично решить эти проблемы можно с помощью динамического изменения идентификатора фрагмента (части URL после #), что позволяют многие браузеры.[4]**

**Динамически загружаемое содержимое недоступно поисковикам (если не проверять запрос, обычный он или XMLHttpRequest)**

**Старые методы учёта статистики сайтов становятся неактуальными**

**Многие сервисы статистики ведут учёт просмотров новых страниц сайта. Для сайтов, страницы которых широко используют AJAX, такая статистика теряет актуальность.**

**Усложнение проекта**

**Перераспределяется логика обработки данных — происходит выделение и частичный перенос на сторону клиента процессов первичного форматирования данных. Это усложняет контроль целостности форматов и типов. Конечный эффект технологии может быть нивелирован необоснованным ростом затрат на кодирование и управление проектом, а также риском снижения доступности сервиса для конечных пользователей.**

**Требуется включенный JavaScript в браузере**

**Альтернативы**

**В хронологическом порядке:**

**Java-апплеты, позднее технология JavaFX;**

**Стек технологий Flash в виде ActionScript 3, Adobe Flex и Flash Remoting составляет технологическую основу RIA (Rich Internet Applications) активно продвигаемых Macromedia (теперь часть Adobe);**

**Технология Silverlight фирмы Microsoft;**

**Протокол WebSocket.**

# Каскадные таблицы стилей (CSS). Особенности применения и преимущества перед непосредственным заданием стилей через атрибуты тегов.

**Цель создания CSS**

**CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое 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, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля.**

**CSS-вёрстка**

**До появления CSS оформление веб-страниц осуществлялось исключительно средствами HTML, непосредственно внутри содержимого документа. Однако с появлением CSS стало возможным принципиальное разделение содержания и представления документа. За счёт этого нововведения стало возможным лёгкое применение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.**

**Преимущества:**

**Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.**

**Уменьшение времени загрузки страниц сайта за счет переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и могут быть закешированы.**

**Простота последующего изменения дизайна. Не нужно править каждую страницу, а лишь изменить CSS-файл.**

**Дополнительные возможности оформления. Например, с помощью CSS-вёрстки можно сделать блок текста, который остальной текст будет обтекать (например для меню) или сделать так, чтобы меню было всегда видно при прокрутке страницы.**

**Недостатки:**

**Различное отображение вёрстки в различных браузерах (особенно устаревших), которые по разному интерпретируют одни и те же данные CSS.**

**Часто встречающаяся необходимость на практике исправлять не только один CSS-файл, но и теги HTML, которые сложным и ненаглядным способом связаны с селекторами CSS, что иногда сводит на нет простоту применения единых файлов стилей и значительно удлиняет время редактирования и тестирования.**