HyperText Transfer Protocol

Web-программирование

HTTP

HTTP — протокол прикладного уровня передачи данных.

Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

- Потребителей (клиентов), которые инициируют соединение и посылают запрос;
- Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI в запросе клиента.

Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное.

Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д.

Именно благодаря возможности указания способа кодирования сообщения, клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

HTTP

HTTP — протокол прикладного уровня; аналогичными ему являются FTP и SMTP. Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI.

В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ».

HTTP

Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера). Браузер, посылающий запросы, может отслеживать задержки ответов.

Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

HTTP

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей, которые передаются в следующем порядке:

- Стартовая строка (англ. Starting line) — определяет тип сообщения;
- Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
- Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

HTTP

Пример запроса:

GET /web-programming/index.html HTTP/1.1

Стартовая **строка ответа** сервера имеет следующий формат:

HTTP/Версия КодСостояния [Пояснение]

Например, на предыдущий наш запрос клиентом данной страницы сервер ответил строкой:

HTTP/1.1 200 Ok

Методы HTTP

Метод HTTP — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Названия метода чувствительны к регистру.

Методы HTTP

- OPTIONS
- GET
- HEAD
- POST
- PUT
- PATCH
- DELETE
- TRACE
- CONNECT

Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented).

Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

GET

Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:

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

Согласно стандарту HTTP, запросы типа GET считаются идемпотентными[4]

Кроме обычного метода GET, различают ещё:

Условный GET — содержит заголовки If-Modified-Since, If-Match, If-Range и подобные;

Частичный GET — содержит в запросе Range.

Порядок выполнения подобных запросов определён стандартами отдельно.

POST

Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер.

В отличие от метода GET, метод POST не считается идемпотентным[4], то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться очередная копия этого комментария).

При результате выполнения 200 (Ok) 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.

Коды состояний HTTP

Код состояния информирует клиента о результатах выполнения запроса и определяет его дальнейшее поведение. Набор кодов состояния является стандартом, и все они описаны в соответствующих документах RFC.

Каждый код представляется целым трехзначным числом. Первая цифра указывает на класс состояния, последующие - порядковый номер состояния. За кодом ответа обычно следует краткое описание на английском языке.

Коды состояний HTTP

1xx Informational (Информационный)

В этот класс выделены коды, информирующие о процессе передачи. В HTTP/1.0 сообщения с такими кодами должны игнорироваться. В HTTP/1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-сервера подобные сообщения должны отправлять дальше от сервера к клиенту.

Коды состояний HTTP

1xx Informational (Информационный)

Примеры ответов сервера:

- 100 Continue (Продолжать)
- 101 Switching Protocols (Переключение протоколов)
- 102 Processing (Идёт обработка)

Коды состояний HTTP

2xx Success (Успешно)

Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.

Коды состояний HTTP

2xx Success (Успешно)

Примеры ответов сервера:

- 200 OK (Успешно).
- 201 Created (Создано)
- 202 Accepted (Принято)
- 204 No Content (Нет содержимого)
- 206 Partial Content (Частичное содержимое)

Коды состояний HTTP

3xx Redirection (Перенаправление)

Коды статуса класса 3xx сообщают клиенту, что для успешного выполнения операции нужно произвести следующий запрос к другому URI. В большинстве случаев новый адрес указывается в поле Location заголовка. Клиент в этом случае должен, как правило, произвести автоматический переход (жарг. «редирект»).

Коды состояний HTTP

3xx Redirection (Перенаправление)

Примеры ответов сервера:

- 300 Multiple Choices (Множественный выбор)
- 301 Moved Permanently (Перемещено навсегда)
- 304 Not Modified (Не изменялось)

Коды состояний HTTP

4xx Client Error (Ошибка клиента)

Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.

Коды состояний HTTP

4xx Client Error (Ошибка клиента)

Примеры ответов сервера:

- 401 Unauthorized (Неавторизован)
- 402 Payment Required (Требуется оплата)
- 403 Forbidden (Запрещено)
- 404 Not Found (Не найдено)
- 405 Method Not Allowed (Метод не поддерживается)
- 406 Not Acceptable (Не приемлемо)
- 407 Proxy Authentication Required (Требуется аутентификация прокси)

Коды состояний HTTP

5xx Server Error (Ошибка сервера)

Коды 5xx выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

Коды состояний HTTP

5xx Server Error (Ошибка сервера)

Примеры ответов сервера:

- 500 Internal Server Error (Внутренняя ошибка сервера)
- 502 Bad Gateway (Плохой шлюз)
- 503 Service Unavailable (Сервис недоступен)
- 504 Gateway Timeout (Шлюз не отвечает)

REST

REST (сокр. от англ. ***Representational State Transfer*** — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле[*уточнить*] компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. REST является альтернативой RPC.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют *«REST-запрос»*), а необходимые данные передаются в качестве параметров запроса.

Для веб-служб, построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «**RESTful**».

Javascript

Динамический HTML

HTML является языком разметки и не имеет каких-либо средств, которые могли бы использоваться для изменения содержимого страницы. Эту проблему решает использование языка DHTML (Dynamic HTML), поддерживающего средства программирования на клиентской стороне.Для этого в DHTML встроена поддержка скриптового языка JavaScript (должен поддерживаться браузером).

Javascript

Чтобы понимать, как соотносятся Javascript и ECMAscript, нужно окунуться немного в историю. Javascript был создан как скриптовый язык для Netscape.

Netscape Navigator — браузер, производившийся компанией Netscape Communications с 1994 по 2007 год. Версии Netscape до 4 были основными конкурентами Internet Explorer, версии 6—7.2 были основаны на Mozilla Application Suite.

Майкрософт создал свой скриптовый язык для IE, который назывался JScript. Естественно использовать 2 языка для разных браузеров было не кошерно и Netscape инициировало стандартизацию, в результате чего родился стандарт языка ECMAscript. ECMAscript не привязан к браузерам, сам по себе не имеет средств ввода/вывода. Последующие версии языков Javascript и JScript были приведены в соответствие стандарту ECMAscript. На основе этого стандарта также был создан ActionScript.

Javascript

В итоге сегодня Javascript состоит из 3-х практически отдельных частей:

- Ядро (полностью соответствует стандарту ECMAscript);
- Document Object model (DOM);
- Browser Object Model (BOM).

Javascript

DOM — это API для достука к HTML. Его спецификацию вы можете найти на сайте W3C. BOM в каждом браузере реализован по-своему. Соответственно вы найдете спецификацию Javascript от Mozilla (как наследницы Netscape) на их сайте. Отдельно можете почитать спецификации ECMAscript и DOM.

В итоге получается, что изначально стандарт ECMAscript был основан на Javascript, а потом Javascript основан на ECMAscript.

Ну а приставка Java — это маркетинговый ход. Java была очень на слуху в 90-х, поэтому и выбрали такое название. Точнее даже переименовали язык из LiveScript в Javascript.

Практическая часть

Как выглядит современная веб-разработка?