Типы HTTP-запросов и философия REST

В статье я хочу рассказать, что же из себя представляют HTTP-методы GET/POST/PUT/DELETE и другие, для чего они были придуманы и как их использовать в соответствии с REST.

HTTP

Итак, что же представляет из себя один из основных протоколов интернета? Педантов отправлю к [RFC2616](http://tools.ietf.org/html/rfc2616), а остальным расскажу по-человечески :)

Этот протокол описывает взаимодействие между двумя компьютерами (клиентом и сервером), построенное на базе сообщений, называемых запрос (Request) и ответ (Response). Каждое сообщение состоит из трех частей: стартовая строка, заголовки и тело. При этом обязательной является только стартовая строка.

Стартовые строки для запроса и ответа имеют различный формат — нам интересна только стартовая строка запроса, которая выглядит так:  
*METHOD URI* HTTP/*VERSION*,  
где METHOD — это как раз метод HTTP-запроса,

URI — идентификатор ресурса,

VERSION — версия протокола (на данный момент актуальна версия 1.1).

Заголовки — это набор пар имя-значение, разделенных двоеточием. В заголовках передается различная служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришел клиент (Referrer) и так далее.

Тело сообщения — это, собственно, передаваемые данные. В ответе передаваемыми данными, как правило, является html-страница, которую запросил браузер, а в запросе, например, в теле сообщения передается содержимое файлов, загружаемых на сервер. Но как правило, тело сообщения в запросе вообще отсутствует.

Пример HTTP-взаимодействия

Рассмотрим пример.  
Запрос:  
GET /index.php HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5

Accept: text/html

Connection: close

Первая строка — это строка запроса, остальные — заголовки; тело сообщения отсутствует  
  
Ответ:

HTTP/1.0 200 OK

Server: nginx/0.6.31

Content-Language: ru

Content-Type: text/html; charset=utf-8

Content-Length: 1234

Connection: close

... САМА HTML-СТРАНИЦА ...

Ресурсы и методы  
Вернемся к стартовой строке запроса и вспомним, что в ней присутствует такой параметр, как URI. Это расшифровывается, как Uniform Resource Identifier — единообразный идентификатор ресурса. Ресурс — это, как правило, файл на сервере (пример URI в данном случае '/styles.css'), но вообще ресурсом может являться и какой-либо абстрактный объект ('/blogs/webdev/' — указывает на блок «Веб-разработка», а не на конкретный файл).  
  
Тип HTTP-запроса (также называемый HTTP-метод) указывает серверу на то, какое действие мы хотим произвести с ресурсом. Изначально (в начале 90-х) предполагалось, что клиент может хотеть от ресурса только одно — получить его, однако сейчас по протоколу HTTP можно создавать посты, редактировать профиль, удалять сообщения и многое другое. И эти действия сложно объединить термином «получение».

Для разграничения действий с ресурсами на уровне HTTP-методов и были придуманы следующие варианты:

* GET — получение ресурса
* POST — создание ресурса
* PUT — обновление ресурса
* DELETE — удаление ресурса

Обратите внимание на тот факт, что спецификация HTTP не обязывает сервер понимать все методы (которых на самом деле гораздо больше, чем 4) — обязателен только GET, а также не указывает серверу, что он должен делать при получении запроса с тем или иным методом. А это значит, что сервер в ответ на запрос DELETE /index.php HTTP/1.1 **не обязан** удалять страницу index.php на сервере, так же как на запрос GET /index.php HTTP/1.1 **не обязан** возвращать вам страницу index.php, он может ее удалять, например :)

В игру вступает REST  
[REST](http://en.wikipedia.org/wiki/Representational_State_Transfer) (REpresentational State Transfer) — это термин был введен в 2000-м году Роем Филдингом (Roy Fielding) — одним из разработчиков протокола HTTP — в качестве названия группы принципов построения веб-приложений. Вообще REST охватывает более широкую область, нежели HTTP — его можно применять и в других сетях с другими протоколами. REST описывает принципы взаимодействия клиента и сервера, основанные на понятиях «ресурса» и «глагола» (можно понимать их как подлежащее и сказуемое). В случае HTTP ресурс определяется своим URI, а глагол — это HTTP-метод.

REST предлагает отказаться от использования одинаковых URI для разных ресурсов (то есть адреса двух разных статей вроде /index.php?article\_id=10 и /index.php?article\_id=20 — это не REST-way) и использовать разные HTTP-методы для разных действий. То есть веб-приложение, написанное с использованием REST подхода будет удалять ресурс при обращении к нему с HTTP-методом DELETE (разумеется, это не значит, что надо давать возможность удалить всё и вся, но *любой* запрос на удаление в приложении должен использовать HTTP-метод DELETE).

REST дает программистам возможность писать стандартизованные и чуть более красивые веб-приложения, чем раньше. Используя REST, URI для добавления нового юзера будет не /user.php?action=create (метод GET/POST), а просто /user.php (метод строго POST).

В итоге, совместив имеющуюся спецификацию HTTP и REST-подход наконец-то обретают смысл различные HTTP-методы. GET — возвращает ресурс, POST — создает новый, PUT — обновляет существующий, DELETE — удаляет.

Проблемы?

Да, есть небольшая проблема с применением REST на практике. Проблема эта называется HTML.

PUT/DELETE запросы можно отправлять посредством XMLHttpRequest, посредством обращения к серверу «вручную» (скажем, через curl или даже через telnet), но нельзя сделать HTML-форму, отправляющую полноценный PUT/DELETE-запрос.

Дело в том, [спецификация HTML](http://www.w3.org/TR/html401/interact/forms.html#h-17.3) не позволяет создавать формы, отправляющие данные иначе, чем через GET или POST. Поэтому для нормальной работы с другими методами приходится имитировать их искусственно. Например, в Rack (механизм, на базе которого Ruby взаимодействует с веб-сервером; с применением Rack сделаны Rails, Merb и другие Ruby-фреймворки) в форму можно добавить hidden-поле с именем "\_method", а в качестве значения указать название метода (например, «PUT») — в этом случае будет отправлен POST-запрос, но Rack сможет сделать вид, что получил PUT, а не POST.