
indigo карма 59,9 рейтинг 0,0
79 голосов

24 января 2009 в 22:33

Типы HTTP-запросов и философия REST

Веб-разработка*

Этот пост — ответ на вопрос, заданный в комментарии к одной из моих статей.

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, 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 (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 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.

http, get, post, put, delete, rest

Комментарии (109)

Kane 24 января 2009 в 22:44 # +11

А что-нибудь предпринимается, для того, чтобы html таки стал поддерживать REST в полном объёме?

indigo 24 января 2009 в 23:08 # ↑ +4

Не знаю, за что вас заминусовали, но вопрос в любом случае поставлен неверно: проблема не в том, что HTML не поддерживает REST, а в том, что HTML не позволяет отправить иной запрос к серверу, кроме GET и POST. Поддержка других методов будет в HTML5 и XHTML2.

Kane 24 января 2009 в 23:14 # ↑ +4

Да, я понимаю в чём проблема, просто не верно сформулировал.
А почему люди здесь минусуют — одному богу известно…

etc 24 января 2009 в 22:49 # +13

Очень полезный для меня «метод» — HEAD.

Пример: «HEAD /index.php HTTP/1.0». Суть — это тот же GET, только он кроме заголовков ничего не возвращает. Удобно, когда нужно узнать размер файла, жива страница/нет или узнать Location, если мы точно знаем, что на странице редирект. Позволяет сэкономить немного времени и трафика.

etc 24 января 2009 в 22:50 # ↑ +5

Вот скопипастил кое-откуда, для тех, кто не совсем понял:

Метод HEAD.
Метод HEAD аналогичен методу GET, за исключением того, что сервер ничего не посылает в информационной части ответа. Метод HEAD запрашивает только информацию заголовка о файле и ресурсе. Информация заголовка запроса HEAD должна быть такой же, как в запросе GET.
Этот метод используется, когда клиент хочет найти информацию о документе, не получая его. Для метода HEAD существует множество приложений. Например, клиент может затребовать следующую информацию:
* время изменения документа (эти данные полезны для запросов, связанных с кэш-памятью);
* размер документа (необходим для компоновки страницы, оценки времени передачи, определения необходимости запроса более компактной версии документа);
* тип документа (позволяет клиенту изучать документы только определенного типа);
* тип сервера;
Следует отметить, что большая часть информации заголовка, которую посылает сервер, не является обязательной и может предоставляться не всеми срверами.
Ниже приведен пример HTTP-транакции с использованием запроса HEAD. Клиент посылает запрос:

HEAD /index.html HTTP/1.0
Connection: Kep-Alive
User-Agent: Mozilla/2.02Gold (WinNT; I)
Host: www.ora.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */* Сервер отвечает:

HTTP/1.0 200 Document follows
Date: Fri, 20 Sep 1996 08:17:58 GMT
Server: NCSA/1.5.2
Last-modified: Mon, 17 Jun 1996 21:53:08 GMT
Content-type: text/html
Content-lenght: 2482 (Тело содержимого в ответ на запрос HEAD не передается.)

theRavel 24 января 2009 в 23:03 # −2

А для чего например может применяться метод DELETE?
Ведь это грозит кучей проблем с безопасностью,

maxshopen 24 января 2009 в 23:10 # ↑ +8

Не грозит, точнее грозит ровно настолько же, насколько и запрос вида
./?action=delete&user_id=100.
Просто http-метод в данном случае выполняет туже функцию, что и action.

indigo 24 января 2009 в 23:12 # ↑ +1

Опередили :)

IntenT 24 января 2009 в 23:30 # ↑ −6

А нахрена это надо?
Если оно выполняет >туже функцию<?
Чем оно лучше?

НЛО прилетело и опубликовало эту надпись здесь

IntenT 25 января 2009 в 00:08 # ↑ **–35**

И чем-же этот инструмент специализированнее? Аргументы есть? Или в лужу пердим?

indigo 25 января 2009 в 02:37 # ↑ **+7**

Аргумент есть у W3C — называется спецификация протокола HTTP

TimTowdy 24 января 2009 в 23:47 # ↑ **+2**

А вот нахрена во всяких веб-фреймворках часто используют MVC-парадигму, если по сути спагетти-код выполняет ту же функцию (отдает хтмл)?

IntenT 25 января 2009 в 00:01 # ↑ **0**

МВЦ — не единственная парадигма. И тем более не единствнно-верная. Не хочешь — не используй.
Просто, иногда она позволяет экономить время на разработке и сопровождении.

Если будет проект, написанный лапшой, на котором будет быстрее и эффективнее разрабатывать — перейду на него.

indigo 25 января 2009 в 02:39 # ↑ **0**

Если будет не только быстрее и эффективнее разрабатывать, но и *проще поддерживать*, то я присоединюсь к вам :) Только я ни разу не видел подобных проектов длиннее 500 строк. Хотя, возможно, они и имеют право на жизнь.

maxshopen 25 января 2009 в 00:10 # ↑ **+1**

Имхо, вопрос из разряда, зачем верстать дивами, если можно таблицами с тем же результатом :)

IntenT 25 января 2009 в 00:22 # ↑ **–8**

А действительно, зачем именно дивами? Можете объяснить?
Верстать семантически — это понятно зачем — чтобы поисковики больше зарабатывали.
Но какая семантика у ДИВа — непонятно. У table-cell и то семантики больше.

Optik 25 января 2009 в 00:43 # ↑ **+2**

pepelsbey.net/2008/04/semantic-coding-1/
В конце ссылка на продолжение. Там доступно все рассказано.

LevshinO 25 января 2009 в 13:14 # ↑ **+2**

Таблицы — на то и таблицы, чтобы представлять данные. Они в принципе не для верстки созданы были.

rumkin 25 января 2009 в 19:55 # ↑ **+1**

По удобству и простоте таблицы лучше дивов. Да и дивы создавались изначально не для вёрстки в современном понимании.
И вообще требую тег layout!

LevshinO 25 января 2009 в 22:08 # ↑ **0**

Тэг layout — это, конечно, да. :) А в HTML 5 вроде же отдельные теги для структурных частей страницы появятся, если не ошибаюсь…

rumkin 25 января 2009 в 23:37 # ↑ **+2**

Да, но как они будут реализованы на практике?
То что я вижу сейчас это подгонка страниц под формат хедер, (сайдбар,) контент, (сайдбар,) футер. что произойдёт к тому времени со структурой документа, куда переедет меню, сколько будет колонок и т.п. доподлинно неизвестно, поэтому мне кажется что создание более гибкого инструмента layout было бы более оправдано.

LevshinO 26 января 2009 в 07:22 # ↑ **0**

Да, поддерживаю вас. :)

alekciy 27 января 2009 в 19:19 # ↑ **0**

Ну хотя бы поэтому: alekciy.ru/articles/table_skeleton_page/index.php. Хотя как мне думается тролю этого не понять. Грубо, сударь, тролим, грубо.

Качайте скил для более тонкой работы ;)

НЛО прилетело и опубликовало эту надпись здесь

 **Covex** 25 января 2009 в 16:59 # ↑ +3

> не нужно читать тонны документации по реализации этого самого API
> …
> а точно известно что вам всего навсего надо сделать UPDATE по заранее определённому URI

Это, мягко говоря, не так!
REST нам всего-лишь предлагает использовать GET, POST и др. в запросе в качестве глаголов ДАЙ, ИЗМЕНИ и др., а формат запроса всё равно находится где-то в дебрях документации.

Для примера хотел дать ссылку на документацию по Google AJAX Search RESTful API, но нашёл только URI ресурса: ajax.googleapis.com/ajax/services/search/web. Попробуйте воспользоваться всеми приемуществами REST и вывести не 4, а 8 результатов поиска + завернуть результат в callBack-функцию, не пользуясь документацией!

 **indigo** 24 января 2009 в 23:11 # ↑ +5

Метод DELETE означает, намерение клиента удалить какой-либо ресурс, но сервер может ему это не позволить (например, если удаление невозможно в принципе или если у клиента нет прав). То есть DELETE /user/theRavel ничем не хуже с точки зрения безопасности, чем GET /user/theRavel/delete

 **theRavel** 24 января 2009 в 23:25 # ↑ 0

Просто получается для удаления при помощи этого метода требуется передача параметров (например пароля для идентификации админа), тогда в чем выигрыш между:

DELETE /index.html HTTP/1.0
pass=admin_pass

и

POST /index.html HTTP/1.0
pass=admin_pass&mode=del

?

 **LoneCat** 24 января 2009 в 23:29 # ↑ 0

Ну вот в DELETE вместо POST и выигрыш, сразу можно понять какое конкретно действие должен выполнить запрос, без изучения доп.параметров. Никаких скрытых фич, просто бОльшая наглядность :)

 **IntenT** 24 января 2009 в 23:35 # ↑ –9

Понятно. Тоесть никакой реальной пользы.
Очередной мертворожденный «принцип». Кто-то хочет стяжать славу «определителя стандартов»

 **LoneCat** 24 января 2009 в 23:46 # ↑ +4

Ну почему никакой пользы, польза есть, это-же все разрабатывается для унификации работы с веб-ресурсами, чтобы разработчик клиентской части не ломал себе голову изучением параметров каждой конкретной серверной части, а мог сразу написать запрос например DELETE /адрес/ресурса/ и получить при его выполнении гарантированный результат.

 **IntenT** 24 января 2009 в 23:56 # ↑ –4

Ага, а при передаче параметров в QWERY черт-знает что просиходит каждый раз, прям шайтан-машина а не сервер.

 **LoneCat** 25 января 2009 в 00:18 # ↑ 0

Что происходит здесь совершенно не причем, когда серверная часть построена с учетом REST и об этом известно — не обязательно углубленно изучать ее структуру чтобы составлять запросы для манипуляции информацией на оной, удаление всегда будет DELETE /адрес/ресурса, без REST оно может быть таким

> GET /адрес/ресурса/?delete

таким

> GET /?resource=адрес_ресурса&action=delete

или таким

> POST /адрес/ресурса
>
> action=delete

 **IntenT** 25 января 2009 в 00:27 # ↑ –4

Уродство строки запроса никак не зависит от того, REST это или нет.

А аргумент — проще увидеть что происходит — просто смешон. Сильно часто смотрите заголовки запросов? и сильно помогает?

LoneCat 25 января 2009 в 00:55 # ↑ +4

Да нет, не часто, только при разработке клиенской части, работающей с http-протоколом :) Я незнаю как вам еще объяснять что такое общий интерфейс, вы очевидно телепат, и вам мои проблемы, проблемы простого смертного, по определению непонятны, а мои проблемы при создании такого рода программ заключаются в том что я незнаю какой запрос нужно послать на сервер чтобы произвести на нем какое-либо действие, чтобы узнать — мне приходится изучать API, если оно есть, js-скрипты и верстку в случае веб-приложения, отлавливать сниффером запросы в случае если приложение является stand-alone, и работает не через браузер, и т.п., в случае повсеместного внедрения REST-архитектуры — мои проблемы не исчезли-бы, но сократились-бы на порядок.

НЛО прилетело и опубликовало эту надпись здесь

bolk 24 января 2009 в 23:49 # ↑ 0

У REST многолетняя история, куча поклонников и туева хуча случаев использования.

IntenT 25 января 2009 в 00:14 # ↑ –9

Угу, про которые никто никогда не слышал до недавнего времени.

DeTeam 25 января 2009 в 12:16 # ↑ +1

Те кто для работы используют ruby-фреймворки не могли не слышать.

bolk 25 января 2009 в 13:03 # ↑ +1

Если читать только «Хабр», то услышать раньше сложно. Читайте англоязычные источники, будете узнавать раньше.

indigo 25 января 2009 в 02:41 # ↑ +3

При том, что я люблю и уважаю REST, не могу с вами согласиться. Первое упоминание о REST датируется 2000 годом, а более-менее применяться он начал вообще два-три года назад.

bolk 25 января 2009 в 13:04 # ↑ +1

9 лет — это много, особенно в компьютерной сфере. Применять недавно начал кто? В англоязычной среде REST применяется давно, я же не замыкаю весь мир на аудиторию «Хабра» или исключительно на русскоговорящих разработчиков.

bolk 24 января 2009 в 23:50 # ↑ 0

У REST многолетняя история, куча поклонников и туева хуча случаев использования.

maxatwork 25 января 2009 в 06:47 # ↑ +1

Нет, польза очень даже есть. REST — это не сами запросы GET/POST/PUT/DELETE. Это, скажем, подход к реализации, и он шире, нежели HTTP, который является самым известным примером использования данного подхода.
Вот в SOAP, например, может быть куча методов (=глаголов в терминологии REST). Это некий бардак создает, т.к. один разработчик для получения данных пользователя сделает методы GetUserById, GetUserByLogin, SetUserEmail, SetUserName, а другой — GetUserInfoByUserId, GetUserInfoByUserName и SaveUser. И вариантов тут может быть куча. А если придерживаться REST, то методов тут два (GET и PUT), меняются параметры — URI объекта «пользователей» и непосредственно сами данные.
Теперь представим, что мы сделали софт, который работает с сервисом первого разработчика, а тут нам понадобилось подключить второй сервис.
В даже идеальном случае с SOAP'ом надо дописывать нужный класс-враппер, а с REST'ом в идеале надо только поменять URI объекта «пользователь». Бонусы видны?

Fortop 26 января 2009 в 16:17 # ↑ 0

Невидны абсолютно.

Точнее сложности с новым классом имеются далеко не во всех языках.

Никто не мешает мне точно так же как URI добавить новый метод обнаруженный в WSDL. Причем класс это сделает автоматически обнаружив, что он(метод) вызывается в моем коде.

LoneCat 24 января 2009 в 23:13 # ↑ 0

Ну так использование этого метода не отрицает проверку прав на удаление ресурса, а для чего применятся — ну собственно сабж :) для удаления чего-бы то ни было, собсно в моем понимании это и есть REST-подход, обращаясь различными

методами по одному и тому-же адресу будут инициированы различные действия, на примере какой-либо записи, например в блоге — GET выдаст запись, PUT изменит ее содержимое, DELETE удалит.

maxshopen 24 января 2009 в 23:13 # 0

Автор, было бы неплохо, для интересующихся, дополнить топик этой ссылкой.

indigo 25 января 2009 в 02:42 # ↑ 0

Спасибо, дополнил. На самом деле, долго сомневался, стоит ли давать ссылку на Википедию.

LoneCat 24 января 2009 в 23:22 # +1

Где-то на хабре уже была достаточно подробная статья о REST-подходе.
По теме: подход конечно правильный и хороший, но так как реализовать его в полной мере в веб-программировании сложно, собсно из-за описанного отсутствия поддержки в HTML-формах методов PUT и DELETE, то остается это все на уровне демагогий. Но некоторым принципам всеже следовать можно и нужно, как например разделение на GET — получение информации, POST — запись, изменение, удаление, хотя это по-моему воспринимается на интуитивном уровне и так.

TimTowdy 24 января 2009 в 23:32 # ↑ 0

Оно-то воспринимается, но тем не менее даже на хабре по-моему можно было запостить картинку с src="/logout" и все кто откроет страницу будут разлогинены.

LoneCat 24 января 2009 в 23:37 # ↑ 0

Ну значит не всеми воспринимается, впринципе многие сайты этим грешат, на том-же вконтакте по-мойму можно чуть-ли не в трусы разработчикам насрать, посредством GET-запроса :)

jeje 24 января 2009 в 23:49 # ↑ 0

>можно чуть-ли не в трусы разработчикам насрать
Поясните пожалуйста что вы имеете ввиду? :)

LoneCat 25 января 2009 в 00:07 # ↑ 0

Ну это на правах шутки конечно :) Но свои огрехи там тоже есть, из известных мне — например вступление в группу по ссылке.

hooey 25 января 2009 в 09:48 # ↑ +3

Была уязвимость с GET-запросом — можно было, скормив пользователю ссылку, без его ведома изменить, например, его имя в анкете. Вконтакте — это яркий пример того, как нарушаются классические правила: «не изменяй данных по GET-запросу», «не доверяй данным от пользователя» и т. д.

indigo 25 января 2009 в 02:43 # ↑ 0

Видимо, вы об этом

indigo 25 января 2009 в 02:46 # ↑ 0

В общем случае, действительно, весьма сложно реализовать PUT/DELETE запросы, тем не менее, решения есть (как я уже писал, есть XMLHttpRequest, есть решения на PHP, на Ruby on Rails (там это вообще нативно поддерживается) и на многих других платформах).

bat 25 января 2009 в 15:35 # ↑ 0

> подход конечно правильный и хороший, но так как реализовать его в полной мере в веб-программировании сложно, собсно из-за описанного отсутствия поддержки в HTML-формах методов PUT и DELETE, то остается это все на уровне демагогий.

А никто и не говорит, что это для браузера. Управлять ресурсами сервера может и много других программ, для которых такой унифицированный API очень даже.

LoneCat 25 января 2009 в 15:56 # ↑ +1

Так в этом собственно и заключается суть проблемы, современные тенденции наоборот способствуют тому чтобы все, даже прикладные, программы перекочевали на сторону сервера и управлялись через браузер, а-ля всякие там web os, google docs тот-же, и т.п., а чтобы поддержать REST нужны наоборот отдельные приложения, хотя реальной пользы от REST-подхода можно было-бы добиться в случае когда он поддерживается и на стороне браузера тоже, как например есть возможность работать с ресурсом через веб-интерфейс, и есть равноценная возможность работать с ресурсом используя отдельное приложение, уже оптимизированное под какую-либо конкретную среду, иже компьютер, ось и т.п., а так получается и не туда, и не сюда, если реализовывать REST нужно пожертвовать веб-интерфейсом, если не реализовывать REST нужно пожертвовать простотой API для разработки отдельного приложения.

bat 25 января 2009 в 16:23 # 0

Сервера могут общаться между собой (об этом уже кто-то здесь писал). Еще REST противопоставили SOAP. Взгляните на проблему с этой стороны.

MaxElc 24 января 2009 в 23:31 # 0

Ой, эт значит, что мне можно не писать о REST? :) Буду ссылаться на вашу статью

indigo 25 января 2009 в 02:46 # 0

С учетом того, что вы пишете о Rails, то пару слов про :method => :put и map.resource :sessions сказать вам всё равно придется ;)

НЛО прилетело и опубликовало эту надпись здесь

TimTowdy 24 января 2009 в 23:53 # +1

Все-таки обычно лучше GET-запрос всегда расценивать как GET, а остальные эмулировать через POST.

НЛО прилетело и опубликовало эту надпись здесь

indigo 25 января 2009 в 02:51 # 0

Ничего не копировал и не знаю, что именно я не понял.

Про ваш п.1 я написал следующее:

> 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

Про п. 2 я тоже писал: HTML 4.01 не позволяет писать в форме иные методы, кроме GET и POST, но во всех основных браузерах вы можете отправить XMLHttpRequest любым методом из как минимум GET/POST/DELETE/PUT. Так что этого вашего возражения я тоже не понял.

НЛО прилетело и опубликовало эту надпись здесь

enartemy 25 января 2009 в 08:38 # +4

Я делаю так:
Просмотр: GET
Создание: POST без указания ID ресурса и с данными в теле запроса
Изменение: POST с указанием ID ресурса и с данными в теле запроса
Удаление: POST с указанием ID ресурса и с пустым телом запроса

Methos 25 января 2009 в 11:46 # +1

по моему, лучше применять hidden поле в форме или дополнительный параметр при запросе.
типа _method

enartemy 25 января 2009 в 19:57 # 0

Да, это пропрет в ROR. Но в других фреймфорках редко встречается такая возможность.

VolCh 26 января 2009 в 06:03 # 0

В последнее время все чаще, например в PHP-фреймворках поддержка REST есть в Zend Framework, CakePHP, symfony.

TimTowdy 24 января 2009 в 23:41 # +1

По RFC POST и PUT немного отличаются от того что вы пишете.
PUT — создает новый или обновляет ресурс если он уже существует.
POST — грубо говоря, добавляет что-то к ресурсу или отсылает ему какие-то данные.

indigo 25 января 2009 в 03:01 # 0

Почитал RFC и готов с вами согласиться (хотя POST тоже может создавать ресурс, но это не основное его применение). До этого, честно говоря, ориентировался на менее официальную литературу (блоги и книги). Большое спасибо за замечание!

Covex 25 января 2009 в 00:53 # 0

Принципы RESTful совсем не предполагают использование DELETE и PUT, а в Википедии про них сказано только в качестве якобы «понятного» всем примера.

indigo 25 января 2009 в 02:56 # 0

Принципы RESTful вообще ничего не говорят об HTTP, если уж на то пошло, как, впрочем, и оригинальная диссертация Роя Филдинга (даже в части, касающейся HTTP).

Применение DELETE и PUT является скорее стандартом де-факто (по крайней мере, я не знаю официальной документации ни на REST, ни на применение REST over HTTP).

Covex 25 января 2009 в 15:36 # +1

В таком случае DELETE и PUT — это стандарт только для HTTP, но не для REST. А если учесть, что использование DELETE и PUT затруднено, то отождествление «REST» и «применения этих двух методов» вносит одну только путаницу.

Ещё я хотел бы отметить, что без DELETE и PUT на первое место выходят принципы «отсутствие состояния клиента на сервере» и «кэшируемость всего» — а это всего лишь «сложные» и уже «не невозможные» задачи.

indigo 25 января 2009 в 15:45 # 0

Да, согласен с вами.

Честно говоря, не смотря на громкий заголовок, я не ставил своей целью рассказать про REST «в общем случае», я лишь пытался ответить на вопрос «какие плюшки дает применение PUT/DELETE», о чем честно заявил в самом начале статьи.

egorinsk 25 января 2009 в 04:46 # +2

Муть какая-то. PUT предназначался для загрузки файла (ресурса) на сервер, а DELETE — для удаления (или для отметки для удаления) файла. POST — для отправки данных, например комментария к файлу (ресурсу). Почитайте указанный вами RFC.

Кстати, если уж реализовывать использование методов как действий. было бы логичнее дать им названия вроде CREATE/UPDATE или INSERT.

merlin-vrn 25 января 2009 в 11:12 # +1

Любитель баз данных? :)

Да нет, на самом деле, никто не запрещает расширять RFC, более того, в нём даже средства для расширения есть: метод OPTION, который возвращает список допустимых методов.
И они используются, такие расширения, например, в серверах icecast и shoutcast.

enartemy 25 января 2009 в 08:22 # 0

Я писал похожую статью не так давно: habrahabr.ru/blogs/webdev/38730/

negrienko 25 января 2009 в 11:23 # +2

Некоторые комментаторы статьи узко смотрят на REST. Просто иногда на одной стороне пользователь а на другой сервер, а иногда сервера общаются между собой.
Хотел дополнить, что есть не только REST (Иван Сагалаев об отличиях REST и WS*), но REST это действительно просто и наглядно.
Для меня достоинства REST неоспоримы, и даже ничего и никому доказывать не хочется.

jeje 25 января 2009 в 13:26 # 0

Все хорошо только никто так и не дал четкого ответа, как идентифицировать пользователя без cookie и session, или нужно просто генерировать временный хеш и работать с ним передавая в теле запроса?

dmx 25 января 2009 в 15:21 # 0

Не совсем уловил этот вопрос в топике или комментариях.
Чем существующие методы не подходят?

jeje 25 января 2009 в 15:25 # 0

По обсуждениям выходит, что rest не терпит cookie и session, какой-то тогда логичный способ понимать какой пользователь отослал запрос :)

indigo 25 января 2009 в 15:42 # +1

Не совсем так, но вопрос действительно интересный и сложный.

REST постулирует, что сервер должен отвечать за состояние ресурса, в то время как клиент — за состояние «приложения» (=контекст). Если cookies используются в качестве «ссылки» на состояние приложения, хранимое на сервере (например, содержат session_id), то это не RESTful подход (вернее, не совсем RESTful), если же Cookies содержат всю информацию, необходимую для определения контекста, то это вполне RESTful. Потому что запрос клиента должен полностью и однозначно определять контекст, а cookies — это часть запроса.

VolCh 26 января 2009 в 06:22 # ↑ 0

Проще говоря идентифицирующее данные, например, логин/пароль (хеш пароля) должны быть в каждом запросе, предполагающем аутенфикацию/авторизацию?

P.S. И, кстати, а собственно запрос аутенфикации в идеологии REST должен передаваться каким HTTP методом? По идее этот запрос только изменяет контекст, а не создает, не изменяет и уж, конечно, не удаляет ресурсы на сервере, и методом исключения приходим, что это должен быть GET запрос :-/

indigo 26 января 2009 в 13:01 # ↑ 0

> Проще говоря идентифицирующее данные, например, логин/пароль (хеш пароля) должны быть в каждом запросе, предполагающем аутенфикацию/авторизацию?

Совершенно верно.

> P.S. И, кстати, а собственно запрос аутенфикации в идеологии REST должен передаваться каким HTTP методом? По идее этот запрос только изменяет контекст, а не создает, не изменяет и уж, конечно, не удаляет ресурсы на сервере, и методом исключения приходим, что это должен быть GET запрос :-/

Вы же сами только что написали, что логин-пароль должны присутствовать в каждом запросе — это и есть чистый REST. В реальной жизни он неудобен, поэтому обычно всё-таки используют сессии и авторизацию 1 раз за сеанс. В этом случае, авторизация — это создание сессии — значит POST-запрос. А выход из системы — удаление сессии — DELETE запрос.

VolCh 26 января 2009 в 13:21 # ↑ 0

Я имел в виду авторизацию один раз за сеанс, но без использования сессий. В тех же куки сохраняем логин/пароль (понятно, что не в чистом виде, а с использованием элементов криптографии). Но вообще да, и в этом случае можно назвать запрос запросом на созданием ресурса (куки), а значит действительно POST.

indigo 26 января 2009 в 13:28 # ↑ 0

А зачем авторизация без сессий, если мы храним логин-пароль в cookies? Мы просто при всех запросах отсылаем такой cookie и нас либо пускает, либо не пускает (например, выдает 403).

VolCh 26 января 2009 в 13:42 # ↑ 0

Ну откуда-то куки же должны взяться? :) Первым запросом посылаем логин и пароль в открытом виде, сервер устанавливает куки шифруя (обратимо-необратимо, не важно, главное чтобы можно было проверить права) логин и пароль и уж дальше с ними работаем. Небезопасно, конечно, зато идеологически безупречно, по-моему — на стороне сервера не хранится информация о состоянии приложения, в частности залогинился ли пользователь

Fortop 26 января 2009 в 16:05 # ↑ 0

А смысл?

Чем зашифрованные права отличаются от SESSID? И то и другое будет сравниваться с данными на сервере.
Более того в случае чистого REST вы будете вынуждены гонять кучу ~~ненужной~~ информации о контексте между клиентом и сервером.
Вместо использования короткого и более безопасного хеша-идентификатора для контекста хранящегося на сервере.

Да вы хоть кол на голове тешите — нет возможности реализовать чистый REST в вебе исходя из вопросов безопасности. Просто потому, что клиент и сервер это разные приложения фактически.

А частичная реализация положений REST и так достаточно давно многими применяется.
Половина агрументации тут (в топике) — это вопрос единообразия API и не более того.

VolCh 26 января 2009 в 16:50 # ↑ 0

Я вовсе не предлагал всем кинуться переписывать свой код :) Для меня это проблема скорее академическая, чем практическая.

>Да вы хоть кол на голове тешите — нет возможности реализовать чистый REST в вебе исходя из вопросов безопасности. Просто потому, что клиент и сервер это разные приложения фактически.

Ну почему нет? Отказаться от архаичной, в принципе, аутенфикации по паре логин/пароль и перейти к использованию нормальных криптографических алгоритмов, например https за глаза хватит очень многим приложениям, а можно и свой механизм реализовать, например, как это сделано в WebMoney или некоторых системах интернет-банкинга. Ведь говоря «веб» мы подразумеваем не только браузер в качестве клиента? :)

Fortop 26 января 2009 в 17:07 # 0

:) нелюблю сферических коней в вакууме :)

Что касается любой попытки уйти от HTTP, то этим вы просто часть функций веб-приложения перекладываете на дополнительное ПО сервера и клиента. (ту же авторизацию)

Все удобство веб-приложений, как раз и заключается в том, что клиент получает части приложения дистанционно и может работать с ними, при этом ему не надо заботиться об актуальности приложения. Используя единую среду (браузер) он может выполнять сколь угодно много приложений.
Да, мы можем написать свою программу-клиент, но она будет заведомо тяжелее для разовой загрузки и будет иметь больше проблем с обновлением и более того для каждого приложения потребуется свой клиент :)

А это как раз то, от чего хотели уйти используя веб, а не просто сеть. Внутри же большинства не-веб приложений REST успешно заменяется CRUD. Поскольку имея отдельный клиент нам нет необходимости беспокоиться о едином API для множества приложений.

VolCh 26 января 2009 в 17:40 # ↑ 0

Никто от HTTP не уходит, где я это предлагал? :) На самом деле я думаю о создании единого API и для непосредственно браузера, и для AJAX-запросов, и для запросов от других серверов, а возможно и дектопных программ-клиентов. Концепция REST мне кажется вполне подходящей, не ясен только вопрос аутенфикации пользователя.Но в принципе Вы правы, если контекст приложения все равно будет завязан на куки (не заставлять же пользователя каждый раз вводить пароль), то безопасней хранить на стороне пользователя хешерованный id сессии, чем пускай и шифрованный, но пароль.

Fortop 26 января 2009 в 17:52 # ↑ 0

Таких API, наверное, уже десятки, просто нужно чтобы использовалось что-то одно :)

Занимаясь разработкой задач чуть более сложных чем просто 3-5 страниц вебсайта понимаешь — всеравно придется допиливать напильником :) поэтому KISS важнее REST (для меня).

Для меня гораздо более интересно мета-програмиирование с самогенерирующимся и самомодифицирующимся кодом.

Covex 26 января 2009 в 18:20 # ↑ 0

А не обязательно хранить пароль в куках =) Закриптованный UserID в куках, который можно расшифровать на сервере, уже можно считать подтверждённой аутентификацией!

Fortop 26 января 2009 в 18:35 # ↑ 0

т.е. возвращаемся к SESSID

Covex 26 января 2009 в 18:40 # ↑ 0

Это труЪ, если шифровать только UserID.

indigo 26 января 2009 в 19:44 # ↑ 0

Почему же SESSID? Это как раз некое подобие SSL выходит — криптография с открытым ключом. Клиент шифрует свой логин+IP закрытым ключом (=паролем) и отсылает серверу — сервер проводит аналогичную операцию и сравнивает результат. При этом кража зашифрованного значения подойдет только при попытке зайти под тем же юзером с того же IP.

Covex 26 января 2009 в 16:54 # ↑ 0

> А смысл?
Смысл в «кэшировании всего» даже для авторизированных пользователей!
Если получится сделать такое кэширование, то серверное приложение требовало бы намного меньше ресурсов. Сервер даже мог бы отдавать 304 Not Modified — а это очень контрастирует с вашим «гонять кучу информации».

Fortop 26 января 2009 в 17:11 # ↑ 0

Пока далеко не все браузеры это поддерживают, да и те, что поддерживают, дают не такие уж большие объемы. А нынешние cookie слишком малы для этого :) вот хранить хеш — самое то.

Во флеше хранилище есть.

Но в любом случае имея закешированные данные (исключая справочики и прочую статику) получите геморрой с репликацией на 10-1000 клиентов. Оно вам надо?

Covex 26 января 2009 в 17:39 # ↑ 0

Ни о каких справочниках и прочей статике речь не идёт. Речь об обычных динамических html-страницах. Фактически в куках нужно будет хранить только информацию об авторизации + какие-то другие некритичные данные.

ЗЫЖ Геморрой — не аргумент! =)

Fortop 26 января 2009 в 17:46 # ↑ 0

«Ну здрасте» :) как это не аргумент? :)

Вы видели неленивого программиста? :) Так что очень даже аргумент.

А если вернутся к динамическим страницам…
Представьте себе биржевые котировки и их историю включая различные условия и фильтры. Фактически для хранения подобной информации потребуется места на порядки больше чем имеется самой информации. Страница с такими фильтрами, страница с другими фильтрами…
Нужно ли такое кеширование? :)

Более того оно реализовано в текущих браузерах в виде истории посещенных страниц. Подходит ли это для веб-приложений?

Covex 26 января 2009 в 18:05 # ↑ 0

«Нужно ли» и «подходит ли» — зависит задачи и от её реализации. Но я всё же считаю, что экономия серверных ресурсов должна перевесить все остальные аргументы и домыслы.

> как это не аргумент? :)
Скоро будет готов «прототип» подобной системы (правда, пока без кэширования =). И как только придумаю «описание для людей» — можно будет обсудить =)

Fortop 26 января 2009 в 18:36 # ↑ 0

Вот на этом и сойдемся, что все зависит от задачи и имеющихся инструментов под рукой. :)

Covex 26 января 2009 в 13:18 # ↑ +1

На highload++ Марко Кевац в своём докладе привёл очень хороший (и главное — работающий) пример RESTful авторизации: highload.ru/papers2008/7166.html

VolCh 26 января 2009 в 13:44 # ↑ 0

Спасибо, и не только за авторизацию :)

Fortop 26 января 2009 в 16:07 # ↑ 0

Постоянно передавать логин и пароль в каждом URI или HEADER, по-большому счету, это единственный способ для чистого REST

все остальное будет вольной интерпретацией текущих сессий.

ZEGO 26 января 2009 в 14:43 # +1

Вот уже третья ваша статья на животрепещущие дял меня темы. Большое спасибо! Был бы очень рад почитать еще на темы Rack, Sinatra или Ruby для веба (без фреймворков вообще) и прочее.

Lesanol 26 января 2009 в 20:02 # 0

Большое спасибо, как — раз искал что-то похожее на эту статью!!! Очень поможет.

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.