РАЗРАБОТКА ВЕБ-САЙТОВ*, API*

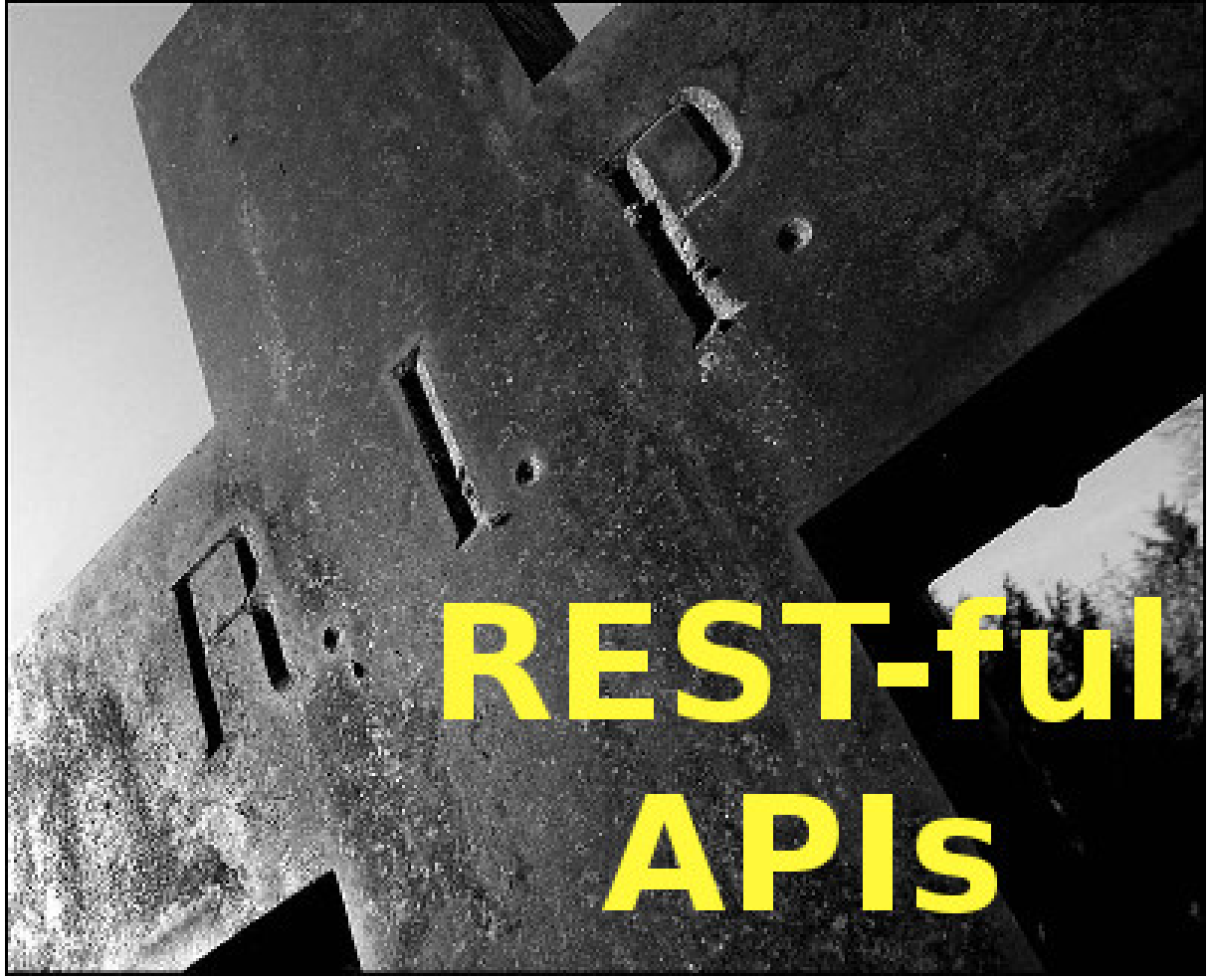
RESTful API — большая ложь

ИЗ ПЕСОЧНИЦЫ

Shakirok 1 сентября 2015 в 10:00 251k

От переводчика:
Я впервые попробовал перевести статью такого объёма и IT-тематики, с радостью прочту ваши комментарии и замечания. Что же касается самой статьи: я не согласен с автором как минимум потому, что, по сути, он заменяет REST на… REST (!!!), но немного в другом обрамлении. Однако, не смотря на то, что в статье преподносится много очевидных вещей, мне она показалась достойной обсуждения на Хабре.

Почему Вам стоит похоронить эту популярную технологию



RESTful api — это чудесно, ведь так?

Если за последние 10 лет Вы читали резюме веб-разработчиков, то Вам простительно думать, что RESTful API — это некое божественное дарование, сошедшее к нам с небес. REST API используется повсюду, даже маркетологи постоянно упоминают о нём в материалах, предназначенных сугубо для руководства или персонала.

Так на сколько всё же хороша идея REST API? Перед тем как мы разберемся с этим вопросом, давайте посмотрим откуда растут корни…

Откуда вообще взялся REST?

Данная технология стала популярной, когда она была подробно описана и представлена Роем Филдингом в его докторской диссертации под названием Architectural Styles and the Design of Network-based Software Architectures в 2000 году. Рой известен своими вкладами в развитие веба, в особенности HTTP.

Так что же такое RESTful API?

REST — это стиль архитектуры программного обеспечения для построения распределенных масштабируемых веб-сервисов. Рой выступал за использование стандартных HTTP методов так, чтобы придавать запросам определённый смысл.

Таким образом, данные HTTP-запросы будут иметь различный смысловую нагрузку в REST:

- GET /object/list
- POST /object/list
- PUT /object/list

Выше только некоторые виды запросов, а вот весь их список: *CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT, TRACE*. Если вы даже не слышали о некоторых из них — не беда, так как есть методы, которые практически никогда не поддерживаются ни клиентским, ни серверным приложением.

Рой также утверждал, что HTTP-коды ответов помогут в определении смысла самих ответов. Существует около 38 кодов ответа и ниже вы можете увидеть их список. Названия некоторых я

немного сократил для удобства:

Method	Method
200 OK	201 Created
202 Accepted	203 Not authorized
204 No content	205 Reset content
206 Partial content	
300 Multiple choice	301 Moved permanently
302 Found	303 See other
304 Not modified	306 (unused)
307 Temporary redirect	
400 Bad request	401 Unauthorized
402 Payment required	403 Forbidden
404 Not found	405 Method not allowed
406 Not acceptable	407 Proxy auth required
408 Timeout	409 Conflict
410 Gone	411 Length required
412 Preconditions failed	413 Request entity too large
414 Requested URI too long	415 Unsupported media
416 Bad request range	417 Expectation failed
500 Server error	501 Not implemented
502 Bad gateway	503 Service unavailable
504 Gateway timeout	505 Bad HTTP version

Итак, одна транзакция по такому API будет состоять, как минимум, из следующего:

- **Метод запроса**, например, GET
- **Путь запроса**, например, /object/list

- **Тело запроса**, например, форма
- **Код ответа**, например, 200 OK
- **Тело ответа**, например, данные в формате JSON

Многие положительно отнеслись к такой парадигме и стали использовать её в разработке веб-сервисов с использованием HTTP. Это и есть то, что мы называем **RESTful API**.

Кстати, транзакция может быть более сложной и причины этому мы еще обсудим. Однако мы опустим те усложняющие факторы, которые связаны с сетями и кешированием, так как эти вопросы актуальны и для других технологий.

На самом деле RESTful API довольно ужасно

REST является отличным механизмом для многих вещей, например, таких как получение контента, и он отслужил нам верой и правдой почти 20 лет. Однако, настало время раскрыть глаза и признать, что концепция RESTful API является одной из худших идей, когда-либо существовавших в веб-разработке. Нет, я не спорю, Рой — отличный парень и, конечно же, у него было множество классных идей… Тем не менее, я не уверен, что RESTful API попадает в их список.

Вскоре мы посмотрим на другое, более правильное решение для построения API, но, перед тем как сделать это, нам следует понять 5 главных проблем RESTful API, которые делают его дорогим, уязвимым к ошибкам и неудобным. Начнём!

Проблема №1: До сих пор нет общего согласования того, что такое RESTful API

Вряд ли кто-то задумывался над тем почему эта технология называется именно «RESTful», а не «RESTpure»? (*прим. переводчика: pure — чёткий, понятный*) А потому что никто не может определиться с тем, что из себя представляют все методы запроса, коды ответа, тела и т.д.

Например, когда мы должны использовать код *200 OK*? Можем ли мы использовать его для подтверждения успешного апдейта записи, или нам стоит использовать код *201 Created*? Судя по всему, нужно использовать код *250 Updated*, однако его не существует. И еще, кто-нибудь может объяснить что означает код *417 Expectation failed*?! Кто-нибудь кроме Роя, конечно.

Словарь HTTP методов и кодов слишком расплывчатый и неполный, чтобы прийти наконец к единым определениям. Нет никого, если я не ошибаюсь, кто нашел единый, общий порядок и призвал остальных его соблюдать. То, что подразумевается под *200 OK* в одной компании может обозначать вовсе иную информацию в другой, что делает обсуждаемую технологию ***непредсказуемой***.

Если бы это было единственной проблемой, то я, наверное, смирился бы и продолжал писать RESTful API по сей день. Однако, наш список только раскрывается…

Проблема №2: Словарь REST поддерживается не полностью

Даже если бы мы решили первую проблему, то столкнулись бы со следующей, практической: большинство клиентских и серверных приложений поддерживают не все коды ответа и, собственно, глаголы, означающие HTTP-методы. Например, большинство браузеров имеют ограниченную поддержку PUT и DELETE.

Как же мы с этим справляемся? Одним из способов является вставка *глагола*, обозначающего нужный метод, в отправляемую форму. Это значит, что в данном случае запрос включает в себя:

- **Метод HTTP запроса**, например, POST
- **Адрес запроса**, например, /object/list
- ***Метод, который мы на самом деле подразумеваем***, например, DELETE
- **Тело запроса**, например, данные из формы

Ситуация с кодами ответа не лучше. Разные браузеры (и серверные приложения тоже) часто понимают эти коды по-разному. Например, получив код *307 Temporary redirect*, один браузер может позволить пользовательскому скрипту рассмотреть этот ответ и *отменить действие* до его выполнения. Другой браузер может просто напросто запретить скрипту делать что-либо. На самом деле, единственными кодами, обработки которых можно не бояться, являются *200 OK* и *500 Internal server error*. В остальных же случаях поддержка ответов варьируется от «довольно хорошей» до «просто ужасной». Именно по-этому нам часто приходится дополнять тело ответа ***кодом, который мы на самом деле подразумевали***.

Даже если мы всё же смогли бы согласовать всё вышеописанное, а еще магическим образом пофиксили всё подключённое к интернету, но не приспособленное к REST программное обеспечение — мы всё равно столкнёмся с очередной проблемой.

Проблема №3: Словарь REST недостаточно насыщен

Словарь, состоящий только из HTTP методов и кодов ответа, является слишком ограниченным для эффективной передачи и приёма разнообразной информации, необходимой всем приложениям. Представьте, что мы создали приложение, из которого мы хотим отправить клиенту ответ «render complete». К сожалению, мы не можем сделать это с помощью HTTP кодов, так как, во-первых, *такого кода не существует*, а во-вторых *мы не можем его создать, так как HTTP — не расширяемый*. Минутка разочарования. Думаю нам снова придётся вставлять ***то, что мы подразумеваем*** в тело ответа.

Также проблема в том, что у нас не один словарь, у нас их три! *Коды ответов* — это числовые значения (200, 201, 500), которые отличаются от представления *методов запроса* (GET, POST, PUT и т.д.), а *тело ответа* и вовсе в формате JSON. Выполнение REST транзакций — это как отправка письма на английском языке в Китай и получение оттуда ответа морзянкой. Все эти сложности являются крупным источником путаницы и ошибок. Вот мы и перешли к следующей глобальной проблеме: дебаггинг.

Проблема №4: RESTful API очень трудно дебажить

Если Вы когда-то работали с REST API, то Вы наверняка в курсе, что его почти невозможно дебажить. Для того, чтобы понять то, что происходит во время транзакции, нам приходится просматривать сразу 7 мест:

- **Метод HTTP запроса**, например, POST
- **Адрес запроса**, например, /object/list
- ***Метод, который мы на самом деле подразумеваем (в теле запроса)***, например, DELETE
- **Собственно, тело запроса**, например, данные из формы
- **Код ответа**, например, 200 OK
- ***Код, который мы подразумевали (в теле ответа)***, например, 206 Partial Content
- **Собственно, тело ответа**

Так вот теперь у нас не только два сильно ограниченных словаря, так еще и 7 разных точек в которых может крыться ошибка. Единственное, что могло бы еще более усугубить ситуацию — это если бы технология REST была полностью привязана к одному протоколу и было бы невозможно использовать какой-либо другой канал связи. Собственно, так и есть, и это — наша следующая большая проблема!

Проблема №5: Как правило, RESTful API привязаны к протоколу HTTP

RESTful API в дребезги разбивает один из фундаментальных законов о хорошей связи: *содержимое сообщения должно быть*

абсолютно независимо от канала передачи. Их смешивание — это путь к сплошной путанице.

Постоянное переплетение HTTP протокола и передаваемой информации полностью лишает нас возможности переноса RESTful API на другие каналы связи. Портирование RESTfulAPI с HTTP на какой-либо другой протокол передачи данных требует полного распутывания и реструктуризации информации из **семи** разных точек, о которых мы говорили ранее.

К счастью, есть хорошее решение, которое позволяет избежать либо минимизировать все проблемы RESTful API. Встречайте!

Шаг вперёд: JSON-pure API

JSON-pure API справляется с большинством проблем, которые мы только что рассмотрели.

- Использует только один метод для передачи данных — обычно *POST* для HTTP и *SEND* в случае использования Web Sockets
- Механизм передачи и содержимое запроса полностью независимы. Все ошибки, предупреждения и данные передаются в теле запроса, в формате JSON
- Используется лишь один код ответа, чтобы подтвердить успешную передачу, обычно это *200 OK*
- Механизм передачи и содержимое ответа полностью независимы. Все ошибки, предупреждения и данные передаются в теле ответа, в формате JSON

- Гораздо проще дебажить, ведь все данные находятся в одном месте в легко-читаемом формате JSON
- Легко перенести на любой канал связи, например, *HTTP/S, WebSockets, XMPP, telnet, SFTP, SCP, or SSH*

JSON-pure API появилось в следствии осознания разработчиками того факта, что RESTful API не особо дружелюбно к браузерам и самим разработчикам. Разделение сообщения и способа передачи делает JSON-pure API быстрым, надежным, простым в использовании, портировании и поиске ошибок. Сегодня, если нам понадобится, например, использовать API Твиттера, то мазохисты выберут RESTful API. Остальные же обратятся к JSON-pure API, или, как его еще называют, «Web API».

За последние десять лет меня не раз просили использовать RESTful вместо JSON-pure. Крайний раз, когда мне чуть было не пришлось поддерживать RESTful API, был в 2011 году. К моему счастью, бэк-енд команда согласилась параллельно с RESTful запустить JSON-pure API, просто перенеся все свои методы и коды в JSON.
Спустя несколько месяцев все мои знакомые, ранее использовавшие RESTful, перешли на JSON-pure, осознав, что это гораздо удобнее.

Оригинал статьи: mmikowski.github.io/
Автор: Michael S. Mikowski

Поделиться:

Сохранить:

Комментарии (148)

Похожие публикации

VIM + screen. Организация удаленной среды web-разработки

31

BoogerWooger • 20 декабря 2013 в 19:43

Web-разработка на node.js и express. Изучаем node.js на практике

64

f0rk • 2 июля 2012 в 23:58

Фото-отчет с .тостер{web-разработка}

47

Boomburum • 1 ноября 2011 в 16:33

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета)

69

BarakAdama • вчера в 10:52

Почему следует игнорировать истории основателей успешных стартапов 20

ПЕРЕВОД

m1rko • вчера в 10:44

Как получить телефон (почти) любой красотки в Москве, или интересная особенность MT_FREE 24

ИЗ ПЕСОЧНИЦЫ

cab404 • вчера в 20:27

Java и Project Reactor 10

zealot_and_frenzy • вчера в 10:56

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle 6

erogov • вчера в 12:46

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке 31

NAGru • вчера в 10:10

Энтузиаст сделал новую материнскую плату для ThinkPad X200s 49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить

85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе

139

marks • вчера в 14:19

Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16