URI, URL, URN

URI

Унифицированный Идентификатор Ресурса, в простонародье - **URI**

URI*— последовательность символов, идентифицирующая физический или абстрактный ресурс, который не обязательно должен быть доступен через сеть Интернет, причем, тип ресурса, к которому будет получен доступ, определяется контекстом и/или механизмом.*

Например:

- перейдя по `http://example.com` — мы попадем на http-сервер ресурса идентифицируемого как example.com
- в то же время `ftp://example.com` — приведет нас на ftp-сервер того же самого ресурса
- или например `http://localhost/` — URI идентифицирующий саму машину откуда производится доступ

В современном интернете, чаще всего используется две разновидности URI URL и URN.

Основное различие между ними — в задачах:

- **URL -** *Uniform Resource Locator*, помогает найти какой либо ресурс
- **URN -** *Uniform Resource Name*, помогает этот ресурс идентифицировать

Упрощая: URL - отвечает на вопрос: «Где и как найти что-то?», URN - отвечает на вопрос: «Как это что-то идентифицировать».

Парочка интересностей про URI

Многие из вас замечали, что на разных ресурсах ссылки называют то URL, то URI и, вероятно, становилось интересно — какой же из вариантов правильный?

Дело в том, что URL увидел свет и был документирован в 1990 году, в то время как URI был документирован лишь в 1994 году. И вплоть до 2002 года, до выхода RFC3305, уместными были оба варианта именования, что, порой вносило путаницу.

В п.2 RFC3305 сообщается об устаревании такого термина как URL, применимо к ссылкам, и что отныне верным будет именование URI, с того момента, во всех документах W3C использует термин URI. Исходя из этого, применяя термин URL к соответствующим ссылкам, вы не делаете смысловой ошибки, но делаете ее с точки зрения правильного именования.

Так же примечателен тот момент, что вплоть до выхода RFC2396, в 1997 году, URI расшифровывался как *Universal* Resource Identifier, что можно увидеть в RFC1630

Обобщая всевозможные варианты, URI имеет следующий вид:

Забегая вперед, стоит отметить, что не все три компоненты являются строго обязательными. Для того чтобы ссылка считалась URI необходимо наличие:

- либо scheme+authority+path,
- либо sheme+path,
- либо только path.

Синтаксис URI

Согласно п.2 RFC3986:

> URI составлен из ограниченного набора символов, состоящих из цифр, букв и нескольких графических символов, все эти символы вписываются в кодировку US-ASCII (ASCII). Зарезервированное подмножество символов может использоваться, чтобы разграничить компоненты синтаксиса в URI, в то время как остающиеся символы: не зарезервированный набор и включая те зарезервированные символы, которые не действуют как разделители в данной компоненте URI, определяют данные идентификации каждого компонента.

Зарезервированные символы

Зарезервированные символы делятся на два типа:

- gen-delims, они же «главные разделители», т.е. символы, разделяющие URI на крупные компоненты.

```
":", "/", "?", "#", "[", "]", "@"
```

- sub-delims, они же «под разделители» — символы, которые разделяют текущую крупную компоненту, на более мелкие составляющие, для каждой компоненты они свои, вот список самых распространенных:

```
"!", "$", "&", "'", "(", ")", "*", "+", ",", ";", "="
```

Не зарезервированные символы

Исходя из предыдущего пункта, не зарезервированные символы — символы, не входящие в gen-delims, а так же не значимые для данной компоненты sub-delims. Но в общем случае это:

```
ALPHA, DIGIT, "-", ".", "_", "~"
```

> Для данного случая, согласно ABNF:
> ALPHA— любая буква верхнего и нижнего регистров кодировки ASCII (в regExp [A-Za-z])
> DIGIT— любая цифра (в regExp [0-9])
> HEXDIG — шестнадцатиричная цифра (в regExp [0-9A-F])

Процентное кодирование

В случае, если используются символы выходящие за пределы кодировки ASCII используется механизм т.н. "Процентного кодирования". Так же он применяется для передачи зарезервированных символов в составе данных. Зарезервированные символы, по правилам, не участвуют в процентном

кодировании.
Процентно-кодированный символ представляет из себя символьный триплет, состоящий из знака "%" и следующих за ним двух шестнадцатиричных чисел:

```
pct-encoded = "%" HEXDIG HEXDIG
```

Т.о., %20, например, означает пробел.

Компоненты URI

Следующий список содержит описания крупных компонент, составляющих URI:

- Scheme (схема)

  Каждый URI начинается с имени схемы, которое относится к спецификации для присвоения идентификаторов в этой схеме. Также, синтаксис URI — объединенная и расширяемая система именования, причем, спецификация каждой схемы может далее ограничить синтаксис и семантику идентификаторов, использующих эту схему. Название схемы обязательно начинается с буквы и далее может быть продолжено любым количеством разрешенных символов. Разрешенные символы для схемы:

  ```
  ALPHA, DIGIT, "+", "-", "."
  ```

- Authority

  Компонента authority начинается с двойного слеша (//) и заканчивается следующим слешем (/), знаком вопроса(?) или октоторпом (#) или концом URI

  Authority состоит из:

  ```
  [ userinfo "@" ] host [ ":" port ]
  ```

  > где в квадратных скобках опциональные компоненты

Каждую из подкомпонент, отдельно, мы рассмотрим чуть позже, в разделе посвященном URL.

●Path (Путь)

Компонента пути содержит данные, обычно, организованные в иерархической форме, которые, вместе с данными в неиерархическом компоненте запроса (Query), служит, чтобы идентифицировать ресурс в рамках схемы URI и authority (если таковая компонента указана). Путь начинается со слеша(/) и заканчивается знаком вопроса(?), октоторпом(#) или концом URI

Разрешенные символы для пути:

> Не зарезервированные, процентно-кодированные, sub-delims, ":", "@"

●Query (Запрос)

Компонента запроса содержит данные, организованные в неиерархической форме, которые, вместе с данными в иерархическом компоненте пути (Path), служит, чтобы идентифицировать ресурс в рамках схемы URI и authority (если таковая компонента указана). Запрос начинается с первого знака вопроса(?) и заканчивается октоторпом(#) или концом URI

Разрешенные символы для запроса:

> Не зарезервированные, процентно-кодированные, sub-delims, ":", "@", "/", "?"

В запросе чаще всего передаются данные в формате key=value (ключ=значение), значение рекомендуется передавать в процентно-кодированном виде, обусловлено это тем, что в значении может встретиться символ "&", который используется для разделения пар ключ-

значение, в результате появления такого артефакта дальнейшая последовательность пар ключ-значение может быть нарушена.

●Fragment (Фрагмент)

Компонента фрагмент позволяет осуществить косвенную идентификацию вторичного ресурса по отношению к первому. Семантика фрагмента никак не ограничена, фрагмент начинается октоторпом(#) и заканчивается концом URI, при этом может состоять из абсолютно любого набора символов.

Пример применения фрагментов, наапример заголовок статьи, состоит из относительных ссылок

```
<a href="#someanchor"></a>
```

а по статье, в определенных местах, раскиданы т.н. «якоря» — теги

```
<anchor>someanchor</anchor>
```

Переходя по указанной в оглавлении ссылке, браузер производит переход ко вторичному ресурсу относительно данной страницы, т.е. скроллит вниз, до появления нужного

```
<anchor>
```

на экране.

На этом, пожалуй, знакомство с URI можно закончить и начать углубляться в отдельные подвиды URI, а именно

URL

Стандарт URL документирован в RFC1738.

Из п.2:

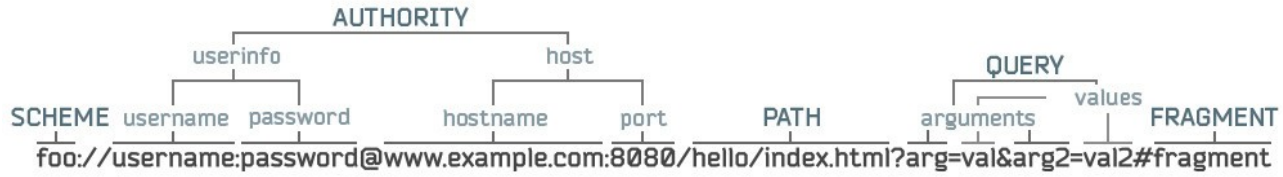
> URL используются, чтобы определить местоположение ресурсов, обеспечивая абстрактную идентификацию расположения ресурса. Определив местоположение ресурса, система может выполнить множество операций на ресурсе, которые могут быть характеризованы такими словами как 'доступ', 'обновление', 'замена', 'поиск атрибутов'. В целом только метод доступа должен быть определен для любой схемы URL.

Т. о.: URL призван решить широкий ряд задач, начиная с получения и заканчивая изменением данных на ресурсе, а обязательным параметром для получения доступа — является метод, т. е. любой полноценный (абсолютный) URL можно свести к виду:

```
<scheme>:<часть свойственная этой схеме>
```

Структура URL

В целом, URL имеет схожую структуру, для всех схем, хотя для каждой отдельно взятой схемы, структура может отличаться от общего шаблона. Графически ее можно выразить в следующем виде:



И вот, примерно на этом моменте, можно рассмотреть различия между абсолютными (absolute) и относительными (relative) URL, данные определения распространяются не только на URL, но и на URI в целом.

- **Относительная ссылка** использует иерархический синтаксис, чтобы выразить ссылку URI относительно пространства имен другого иерархического URI.

  Относительные ссылки так же делятся на несколько подвидов:

  - Ссылка сетевого пути

    Имеет вид:

```
//<authority> <path> [<query>] [<fragment>]
```

Такой тип ссылок применяется не часто, смысл заключается в переходе по указанной ссылке с применением текущей схемы. Т. е.: находясь, например на `http://example.com` и перейдя по ссылке `//domain.com` — мы попадем на `http://domain.com`. А в случае если перейдем по той же ссылке с `ftp://example.com`, то попадем на `ftp://domain.com`

●Ссылка абсолютного пути

Имеет вид:

```
/<path> [<query>] [<fragment>]
```

На этот раз мы останемся в пределах текущего хоста, но попадем по пути path в любом случае, по какому бы пути мы сейчас не находились.

Т. е.: даже находясь на `http://example.com/just/some/long/path` и перейдя по ссылке `/path`, мы попадем на `http://example.com/path`

●Ссылка относительного пути

Имеет вид:

```
<path> [<query>] [<fragment>]
```

Теперь же, мы будем перемещаться в пределах текущего положения. Т.е.: находясь на `http://example.com/just/some/long/path` и перейдя по ссылке `path`, мы попадем на `http://example.com/just/some/long/path/path`

●Ссылка того же документа

Фактически это ссылки, состоящие только из фрагментарной части URI, либо же ссылки, у которых все компоненты за исключением

фрагментарной совпадают с исходной.

T.e. #fragment и `http://habrahabr.ru/topic/232385/#fragment` являются ссылками того же документа.

●**Абсолютная ссылка** — ссылка вида

```
<scheme> <authority> [<path>] [<query>] [<fragment>]
```

Применяя абсолютные ссылки, мы будем попадать на нужный нам ресурс вне зависимости от того, откуда мы переходим. Т. е.: находись мы на `http://example.com/just/some/long/path` или же на ftp://example.com, перейдя по http://domain.com/path, мы в любом случае попадем на http://domain.com/path

Мы уже рассмотрели крупные компоненты, а теперь давайте углубимся в детали построения URL.

●Scheme — как говорилось ранее: схема определяет метод доступа к ресурсу. Список актуальных схем можно посмотреть тут.

●Userinfo — под-компонент authority, использующийся для авторизации пользователя на ресурсе. Состоит из username и необязательного password, от остальной части authority отделяется символом "@". Не смотря на то, что параметр password указан в спецификации, его использование крайне не рекомендуется, т. к. фактически производится передача пароля к учетной записи username, в незашифрованном виде. Разрешенные символы:

```
Не зарезервированные, процентно-кодированные, sub-delims, ":"
```

●Host — компонент authority, использующийся для определения целевого узла (или ресурса, если угодно, но понятие «узел» будет более точным), который может находиться как в сети интернет, так и вне её, в зависимости от указанной схемы. Данная компонента не чувствительна к регистру.

Хост может представлять из себя либо IP-адрес, либо регистрационное имя (reg-name) и, опционально, следующий за ними порт(port). Предусматривается как поддержка существующих форматов IP-адресов (IPv4, IPv6), так и всевозможных будущих, которые будут описаны впоследствии.

Регистрационное имя — привычные нам, т. н. доменные имена — последовательность символов, обычно предназначенных для поиска в локально определенном узле или реестре имени службы, хотя специфичная для схемы семантика URI может потребовать, чтобы вместо этого использовался определенный реестр (или фиксированная таблица имен).

Наиболее распространенный механизм реестра имен — Система Доменных Имен (DNS). Доменное имя, используемое для поиска в DNS, состоит из доменных меток, разделенных при помощи ".", каждая доменная метка может содержать следующие символы:

Не зарезервированные, процентно-кодированные, sub-delims

Синтаксис регистрационного имени позволяет использование процентно-кодированных символов, для представления не-ASCII символов, в едином порядке, не зависящем от технологии разрешения имен. Символы, не входящие в ASCII, должны быть сначала закодированы в UTF-8, а затем каждый октет UTF-8 последовательности должен быть процентно закодирован.

В случае, если регистрационное имя с не-ASCII символами представляет собой многоязычное доменное имя, разрешаемое через DNS, оно должно быть преобразовано в кодировку IDNA (RFC3490) до поиска имени и, как следствие, регистраторами доменных имен такие регистрационные имена должны предоставляться в кодировке IDNA.

Port (Порт) — десятичный номер порта, отделяется от hostname одним двоеточием ":", может состоять только из цифр. Схема может определять порт по-умолчанию, который будет использоваться в случае если порт не указан. Например, для схемы HTTP порт по-умолчанию — 80, что соответствует зарезервированному под неё порту 80/TCP. Тип порта, так же как и назначенный номер порта, определяется схемой.

●Компоненты Запрос и Фрагмент полностью описаны ранее.

URN

Стандарт URN документирован в RFC2141.
Из п.1:

> Унифицированные имена ресурсов (URN) предназначены, чтобы служить постоянными, независимыми от расположения, идентификаторами ресурсов и разработаны для упрощения отображения других пространств имен (которые совместно используют свойства URN) в URN-пространство. Таким образом, синтаксис URN обеспечивает средство закодировать символьные данные в форме, которая может быть отправлена посредством существующих протоколов, записана при помощи большинства клавиатур, и т.д.

Т. е., в отличие от URL, который ссылается на како-то место, где хранится документ, URN ссылается на сам документ, и при перемещении документа в другое место ссылка не изменится. В силу того, что URN концептуально отличается от URL, то и система разрешения имен у него другая — DDDS, которая преобразует URN в URL, по которым можно найти ресурс/объект или что бы то ни было, на что ссылается URN.

Структура

URN имеет следующий вид:

```
"urn:" <NID> ":" <NSS>
```

- «urn:» — обязательная, регистронезависимая часть URN
- NID — Namespace Identifier, данная компонента определяет синтаксическую интерпретацию компоненты NSS. Минимальная длина — 2 символа, максимальная — 32, разрешенные символы:

```
латинские буквы, цифры, "-"
```

NID должен начинаться только с буквы или цифры. Так же, слово «urn» для NID является зарезервированным, дабы избежать неоднозначности при определении URN в целом.

- NSS — Namespace Specific String, эта компонента служит непосредственно для передачи каких-либо данных. Разрешенные символы:

```
латинские буквы, цифры, процентно-кодированные, "(", ")", "+", ",", "-", ".", ":", "=", "@", ";", "$", "_", "!", "*", ""
```

Зарезервированные символы:

```
"%", "/", "?", "#"
```

Запрещенные символы должны быть процентно-кодированы. Если указанный символ встретится в явном виде, его позиция будет считаться концом URN:

> октеты 0-32 (0-20 hex), "\", """, "&", "<", ">", "[", "]", "^", "`", "{", "|", "}", "~", октеты 127-255 (7F-FF hex)

Самоидентифицирующийся URN

Такие URN содержат в NID название хэш-функции, а в NSS значение хэша, вычисленного для идентифицируемого объекта. Такие ссылки используются в magnet-ссылках и заголовках p2p-сети Gnutela2.

REST

(REpresentational State Transfer) — это архитектура, т.е. принципы построения распределенных гипермедиа систем, того что другими словами называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP

Автор идеи и термина Рой Филдинг 2000г.

REST на сегодняшний день практически вытеснил все остальные подходы, в том числе дизайн основанный на SOAP и WSDL

Что нам дает REST подход:

- Масштабируемости взаимодействия компонентов системы (приложения)
- Общность интерфейсов
- Независимое внедрение компонентов
- Промежуточные компоненты, снижающие задержку, усиливающие безопасность

Когда использовать REST?

- Когда есть ограничение пропускной способности соединения
- Если необходимо кэшировать запросы
- Если система предполагает значительное масштабирование
- В сервисах, использующих AJAX

Преимущества REST:

- Отсутствие дополнительных внутренних прослоек, что означает передачу данных в том же виде, что и сами данные. Т.е. данные не оборачиваются в XML, как это делает **SOAP** и XML-RPC, не используется AMF, как это делает Flash и т.д. Просто отдаются сами данные.
- Каждая единица информации (ресурс) однозначно определяется URL — это значит, что URL по сути является первичным ключом для единицы данных. Причем совершенно не имеет значения, в каком формате находятся данные по адресу — это может быть и HTML, и jpeg, и документ Microsoft Word.
- Как происходит управление информацией ресурса — это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Для HTTP действие над данными задается с помощью методов : GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия **CRUD** (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Что такое RESTful:

Чтобы распределенная система считалась сконструированной по REST архитектуре (Restful), необходимо, чтобы она удовлетворяла следующим критериям:

1.Client-Server. Система должна быть разделена на клиентов и на серверов. Разделение интерфейсов означает, что, например, клиенты не связаны с хранением данных, которое остается внутри каждого сервера, так что мобильность кода клиента улучшается. Серверы не связаны с интерфейсом пользователя или состоянием, так что серверы могут быть проще и

масштабируемы. Серверы и клиенты могут быть заменяемы и разрабатываться независимо, пока интерфейс не изменяется.

2.Stateless. Сервер не должен хранить какой-либо информации о клиентах. В запросе должна храниться вся необходимая информация для обработки запроса и если необходимо, идентификации клиента.

3.Cache. Каждый ответ должен быть отмечен является ли он кэшируемым или нет, для предотвращения повторного использования клиентами устаревших или некорректных данных в ответ на дальнейшие запросы.

4.Uniform Interface. Единый интерфейс определяет интерфейс между клиентами и серверами. Это упрощает и отделяет архитектуру, которая позволяет каждой части развиваться самостоятельно.

Четыре принципа единого интерфейса:

•Identification of resources (основан на ресурсах). В REST ресурсом является все то, чему можно дать имя. Например,пользователь, изображение, предмет (майка, голодная собака, текущая погода) и т.д. Каждый ресурс в REST должен быть идентифицирован посредством стабильного идентификатора, который не меняется при изменении состояния ресурса. Идентификатором в REST является URI.

•Manipulation of resources through representations. (Манипуляции над ресурсами через представления). Представление в REST используется для выполнения действий над ресурсами. Представление ресурса представляет собой текущее или желаемое состояние ресурса. Например, если ресурсом является пользователь, то представлением может являться XML или HTML описание этого пользователя.

•Self-descriptive messages (само-документируемые сообщения). Под само-описательностью имеется ввиду, что запрос и ответ должны хранить в себе всю необходимую информацию для их обработки. Не должны быть дополнительные сообщения или кэши для обработки одного запроса. Другими словами отсутствие состояния, сохраняемого между запросами к ресурсам. Это очень важно для масштабирования системы.

•HATEOAS (hypermedia as the engine of application state). Статус ресурса передается через содержимое body, параметры строки запроса, заголовки запросов и запрашиваемый URI (имя ресурса). Это называется гипермедиа (или гиперссылки с гипертекстом). HATEOAS также означает, что, в случае необходимости ссылки могут содержатся в теле ответа (или заголовках) для поддержки URI , извлечения самого объекта или запрошенных объектов.

5 .Layered System. В REST допускается разделить систему на иерархию слоев но с условием, что каждый компонент может видеть компоненты только непосредственно следующего слоя. Например, если вы вызывайте службу PayPal а он в свою очередь вызывает службу Visa, вы о вызове службы Visa ничего не должны знать.

6. Code-On-Demand (опционально). В REST позволяется загрузка и выполнение кода или программы на стороне клиента.

Серверы могут временно расширять или кастомизировать функционал клиента, передавая ему логику, которую он может исполнять. Например, это могут быть скомпилированные Java-апплеты или клиентские скрипты на Javascript.

Важно ! Сама архитектура REST не привязана к конкретным технологиям и протоколам, но в реалиях современного Веб, построение RESTful API почти всегда подразумевает использование HTTP и каких-либо распространенных форматов представления ресурсов, например JSON, или, менее популярного сегодня, XML.

Унифицированные идентификаторы ресурсов (URI) — основа концепции REST. Они позволяют определять ресурсы и действия с ними. Иногда конечную точку, или URI называют путем (path) — путем до ресурса. У каждого конкретного ресурса должен быть уникальный URI. Ответственность за то, чтобы у каждого ресурса всегда был свой URI лежит на плечах разработчика сервера. Подобно тому, как в реляционной базе данных часто принято первичным ключом задавать некоторый числовой ID, в REST у каждого ресурса есть свой ID. Часто бывает так, что ID ресурса в REST совпадает с ID записи в базе данных, в которой хранится информация о данном ресурсе. URI в REST

принято начинать с множественной формы существительного, описывающего некоторый ресурс. Например, со слова clients. Далее через слэш указывают ID — идентификатор некоторого конкретного клиента. Примеры:

- /clients — URI всех имеющихся клиентов;
- /clients/23 — URI конкретного клиента, а именно клиента с ID=23;
- /clients/4 — URI конкретного клиента, а именно клиента с ID=4.

Но и это еще не все. Мы можем продолжить URI, добавив к нему заказы:

- /clients/4/orders — URI всех заказов клиента №4;
- /clients/1/orders/12 — URI заказа №12 клиента №1.

Если мы продолжим эту цепочку и добавим еще и товары, получим:

- /clients/1/orders/12/items — URI списка всех товаров в заказе №12 сделанного клиентом №1.

С уровнями вложенности главное — делать URI интуитивно понятными.

HTTP Request-Response Structure

Структура HTTP-запросов и ответов

В HTTP и запрос, и ответ имеют похожую структуру:

1. URL
2. Метод
3. Версия HTTP
4. Заголовки
5. Статус-код (обязательно только для HTTP-ответов)
6. Тело (необязательно)

Пример HTTP-запроса:

```
POST /cgi-bin/process.cgi HTTP/1.1
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: <a class="vglnk" href="http://www.tutorialspoint.com"
rel="nofollow"><span>www</span><span>.</span><span>tutorialspoint</
span><span>.</span><span>com</span></a>
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

, где

- `POST` – метод,
- `/cgi-bin/process.cgi` – URL,
- `HTTP/1.1` – версия протокола HTTP,
- `User-Agent`, `Host` и другие — заголовки
- `licenseID=string&content=string&/paramsXML=string` — тело запроса.

Пример HTTP-ответа:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

В данном ответе отсутствует URL и метод, зато появился статус-код 200, означающий «успех».

HTTP методы для создания RESTful сервисов

HTTP метод GET используется для получения (или чтения) представления ресурса. В случае “удачного” (или не содержащего ошибок) адреса, GET возвращается представление ресурса в формате XML или JSON в сочетании с кодом состояния HTTP 200 (OK). В случае наличия ошибок обычно возвращается код 404 (NOT FOUND) или 400 (BAD REQUEST).

- GET http://www.example.com/api/v1.0/users (вернуть список пользователей)
- GET http://www.example.com/api/v1.0/users/12345 (вернуть данные о пользователе с id 12345)
- GET http://www.example.com/api/v1.0/users/12345/orders

HTTP метод PUT обычно используется для предоставления возможности обновления ресурса. Тело запроса при отправке PUT-запроса к существующему ресурсу URI должно содержать обновленные данные оригинального ресурса (полностью, или только обновляемую часть).

Для создания новых экземпляров ресурса предпочтительнее использование POST запроса. В данном случае, при создании экземпляра будет предоставлен корректный идентификатор экземпляра ресурса в возвращенных данных об экземпляре.

При успешном обновлении посредством выполнения PUT запроса возвращается код 200 (или 204 если не был передан какой либо контент в теле ответа). PUT не безопасная операция, так как вследствии ее выполнения происходит модификация (или создание) экземпляров ресурса на стороне сервера, но этот метод идемпотентен. Другими словами, создание или обновление ресурса посредством отправки PUT запроса — ресурс не исчезнет, будет располагаться там же, где и был при первом обращении, а также, многократное выполнение одного и того же PUT запроса не изменит общего состояния системы

- PUT http://www.example.com/api/v1.0/users/12345 (обновить данные пользователя с id 12345)
- PUT http://www.example.com/api/v1.0/users/12345/orders/98765 (обновить данные заказа с id 98765 для пользователя с id 12345)

НTTP метод POST запрос наиболее часто используется для создания новых ресурсов. На практике он используется для создания вложенных ресурсов. Другими словами, при создании нового ресурса, POST запрос отправляется к родительскому ресурсу и, таким образом, сервис берет на себя ответственность на установление связи создаваемого ресурса с родительским ресурсом, назначение новому ресурсу ID и т.п.

При успешном создании ресурса возвращается HTTP код 201, а также в заголовке `Location` передается адрес созданного ресурса.

POST не является безопасным или идемпотентным запросом. Потому рекомендуется его использование для не идемпотентных запросов. В результате выполнения идентичных POST запросов предоставляются сильно похожие, но не идентичные данные.

- POST http://www.example.com/api/v1.0/customers (создать новый ресурс в разделе customers)
- POST http://www.example.com/api/v1.0/customers/12345/orders (создать заказ для ресурса с id 12345)

HTTP метод DELETE используется для удаления ресурса, идентифицированного конкретным URI (ID).

При успешном удалении возвращается 200 (OK) код HTTP, совместно с телом ответа, содержащим данные удаленного ресурса. Также возможно использование HTTP кода 204 (NO CONTENT) без тела ответа. Согласно спецификации HTTP, DELETE запрос идемпотентен. Если вы выполняете DELETE запрос к ресурсу, он удаляется. Повторный DELETE запрос к ресурсу закончится также: ресурс удален. Если DELETE запрос используется для декремента счетчика, DELETE запрос не является идемпотентным. Используйте POST для не идемпотентных операций.

Тем не менее, существует предостережение об идемпотентности DELETE. Повторный DELETE запрос к ресурсу часто сопровождается 404 (NOT FOUND) кодом HTTP по причине того, что ресурс уже удален (например из базы данных) и более не доступен. Это делает DELETE операцию не идемпотентной, но это общепринятый компромисс на тот случай, если ресурс был удален из базы данных, а не помечен, как удаленный.

- DELETE http://www.example.com/api/v1.0/customers/12345 (удалить из customers ресурс с id 12345)
- DELETE http://www.example.com/api/v1.0/customers/12345/orders/21 (удалить у ресурса с id 12345 заказ с id 21)

Коды состояний HTTP

Основные группы кодов состояний: (link)

1xx: Information

100: Continue

2xx: Success

200: OK

201: Created

202: Accepted

204: No Content

3xx: Redirect

301: Moved Permanently

307: Temporary Redirect

4xx: Client Error

400: Bad Request

401: Unauthorized

403: Forbidden

404: Not Found

5xx: Server Error

500: Internal Server Error

501: Not Implemented

502: Bad Gateway

503: Service Unavailable

504: Gateway Timeout