Таск 2

REST-URL

REST – это Representational State Transfer, т. е. передача репрезентативного состояния. REST определяет набор функций, таких как GET, PUT, DELETE и т. д., которые клиенты могут использовать для доступа к данным сервера. Клиенты и серверы обмениваются данными по протоколу HTTP. REST API — это способ взаимодействия сайтов и веб-приложений с сервером. Его также называют RESTful.

Термин состоит из двух аббревиатур, которые расшифровываются следующим образом. API (Application Programming Interface) — это код, который позволяет двум приложениям обмениваться данными с сервера. На русском языке его принято называть программным интерфейсом приложения. На русском его называют «передачей состояния представления».

REST — архитектурный стиль взаимодействия, который определяет, как компоненты распределенной системы должны взаимодействовать друг с другом.

В общем случае взаимодействие происходит посредством запросов-ответов.

Компоненту, которая отправляет запрос, называют клиентом; компоненту, которая обрабатывает запрос и отправляет клиенту ответ, называют сервером. Запросы и ответы, чаще всего, отправляются по протоколу HTTP. Сервер получает информацию о результате своего запроса с помощью кодов состояния HTTP. Как правило, сервер — это некое веб-приложение. Клиентом может быть, например, мобильное приложение, которое запрашивает у сервера данные, либо браузер, который отправляет запросы с веб-страницы на сервер для загрузки данных.

Что такое REST API

Representational State Transfer (REST) в переводе — это передача состояния представления. Технология позволяет получать и модифицировать данные и состояния удаленных приложений, передавая HTTP-вызовы через интернет или любую другую сеть.

Если проще, то **REST API** — это когда серверное приложение дает доступ к своим данным клиентскому приложению по определенному URL. Далее разберем подробнее, начиная с базовых понятий.

Базовые понятия Rest API — HTTP-протокол и API

Каковы преимущества REST API?

REST API имеет четыре главных преимущества.

1. Интеграция

API используются для интеграции новых приложений с существующими программными системами. Это увеличивает скорость разработки, потому что каждую функцию не нужно писать с нуля. API можно использовать для усиления существующего кода.

2. Инновации

Целые отрасли могут измениться с появлением нового приложения. Компании должны быстро реагировать и поддерживать быстрое развертывание инновационных услуг. Они могут сделать это, внося изменения на уровне API без необходимости переписывать весь код.

3. Расширение

API-интерфейсы предоставляют компаниям уникальную возможность удовлетворять потребности своих клиентов на разных платформах. Например, карты API позволяет интегрировать информацию о картах через веб-сайты, Android, iOS и т. д. Любая компания может предоставить аналогичный доступ к своим внутренним базам данных, используя бесплатные или платные API.

4. Простота обслуживания

API действует как шлюз между двумя системами. Каждая система обязана вносить внутренние изменения, чтобы это не повлияло на API. Таким образом, любые будущие изменения кода одной стороной не повлияют на другую сторону.

Требования к REST

Каким образом REST может помочь нам достичь этих свойств и реализовать эти нефункциональные требования?

Чтобы это понять, давайте рассмотри **6 принципов** REST — ограничений, которые и помогают нам добиться этих нефункциональных требований.

6 принципов REST:

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.

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

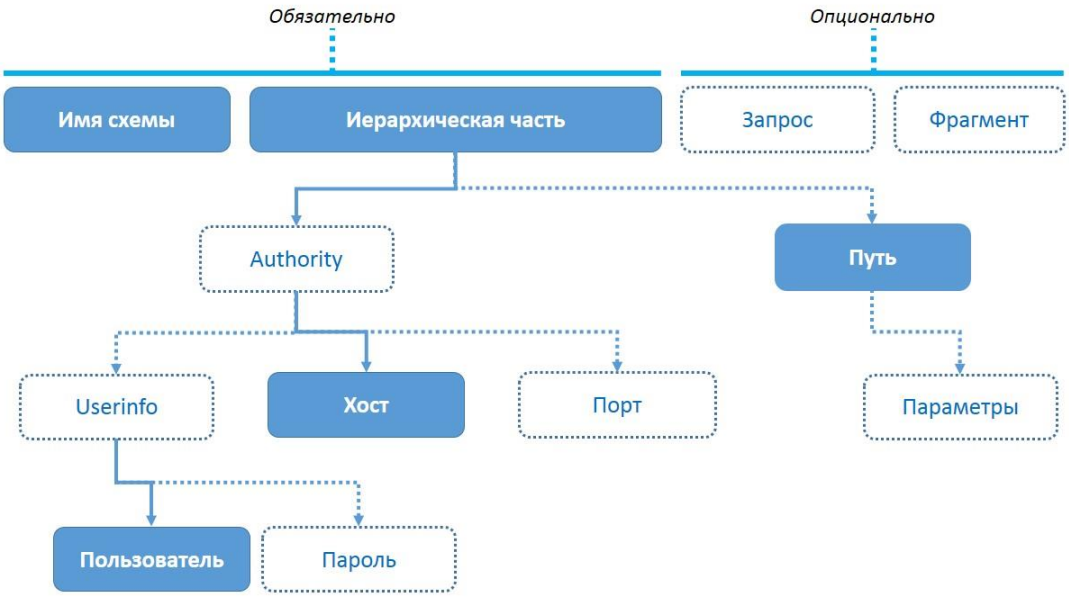
Идентификатором в REST является URI.

URI

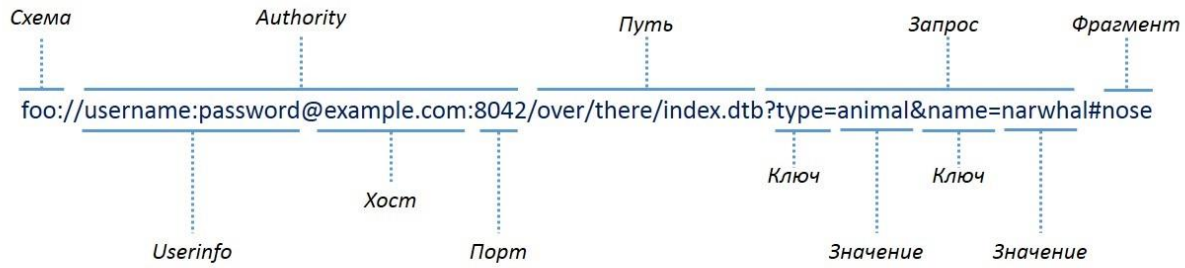
URI (англ. Uniform Resource Identifier) — унифицированный (единообразный) идентификатор ресурса. На английский манер произносится как [ю-ар-ай], по-русски чаще говорят [ури]. URI — это последовательность символов, идентифицирующая абстрактный или физический ресурс. Ранее назывался Universal Resource Identifier — универсальный идентификатор ресурса.

При этом URI может указывать как местоположение ресурса (URL), так и его имя (URN). А может содержать и то и другое. То есть URL и URN — это частные случаи URI.

URI строится по определенным правилам и состоит из обязательных схемы и иерархической части, а также опциональных запроса (ему предшествует знак "?") и фрагмента (ему предшествует знак "#"). Иерархическая часть в свою очередь состоит из необязательного Authority (думаю, перевод только усложнит понимание) и обязательного пути. Authority включает в себя Userinfo (логин и пароль), хост и порт. Кроме того, путь может содержать так называемые параметры. Параметры используются не часто, но нам повезло — в SIP URI они присутствуют. На схеме это выглядит вот так:



Выглядит довольно запутанно, поэтому приведу пример:



Отличия URI и URL.

URL – это локатор, тогда как URI является идентификатором

URL специфичен для веб-ресурсов и есть универсальный URI, который содержит URL

URL должен иметь протокол, такой как http, ftp, tel и т. Д., тогда как URI не должен иметь протокол

И URL, и URI могут использоваться для относительных и абсолютных ссылок

И URL, и URI могут принимать параметры запроса

И URL, и URI могут иметь фрагментированные идентификаторы

Согласно определению W3C, URL и URI – это одно и то же, и они могут быть взаимозаменяемыми

URL

URL (url адрес) — это аббревиатура, которая расшифровывается как Uniform Resource Locator, или «унифицированный указатель ресурса», т.е. адрес сайта размещенного в сети Интернет.

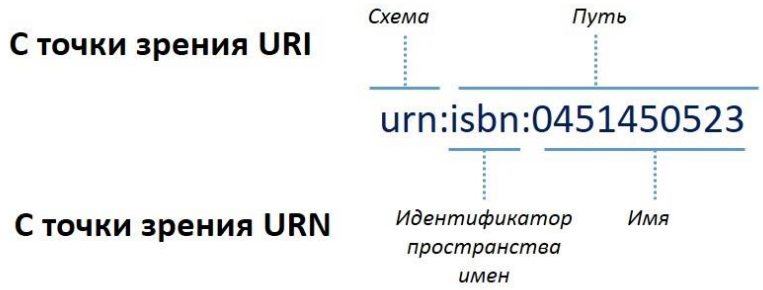
URL (Uniform Resource Locator) указывает путь (локацию) объекта и метод получения доступа к нему. Например, en.wikipedia.org/wiki/Main_Page указывает на главную страницу английской Википедии и в качестве метода доступа предлагает использовать протокол http.
URL — та самая ссылка, которая показывается в адресной строке браузера. Его можно получить, если выделить ссылку (она автоматически выделяется при установке курсора в адресную строку), нажать правую кнопку мыши и выбрать из выпадающего меню пункт «Копировать».

Поскольку URL — это частный случай URI, схема в общем случае выглядит точно так же, однако для разных протоколов актуальны те или иные ее части. Например, для протокола telnet, схема URL выглядит следующим образом:

URN

URN не используется в рамках SIP, однако без него рассказ был бы неполным.

URN (Uniform Resource Name) является уникальным именем объекта. URN включает в себя название пространства имен и идентификатора в этом пространстве. Типичный пример URN — это ISDN-Имя книги. URN состоит из NID (namespace identifier или идентификатор пространства имен) и NSS (namespace-specific string или уникального для данного пространства имен имени). Схематично это выглядит следующим образом:

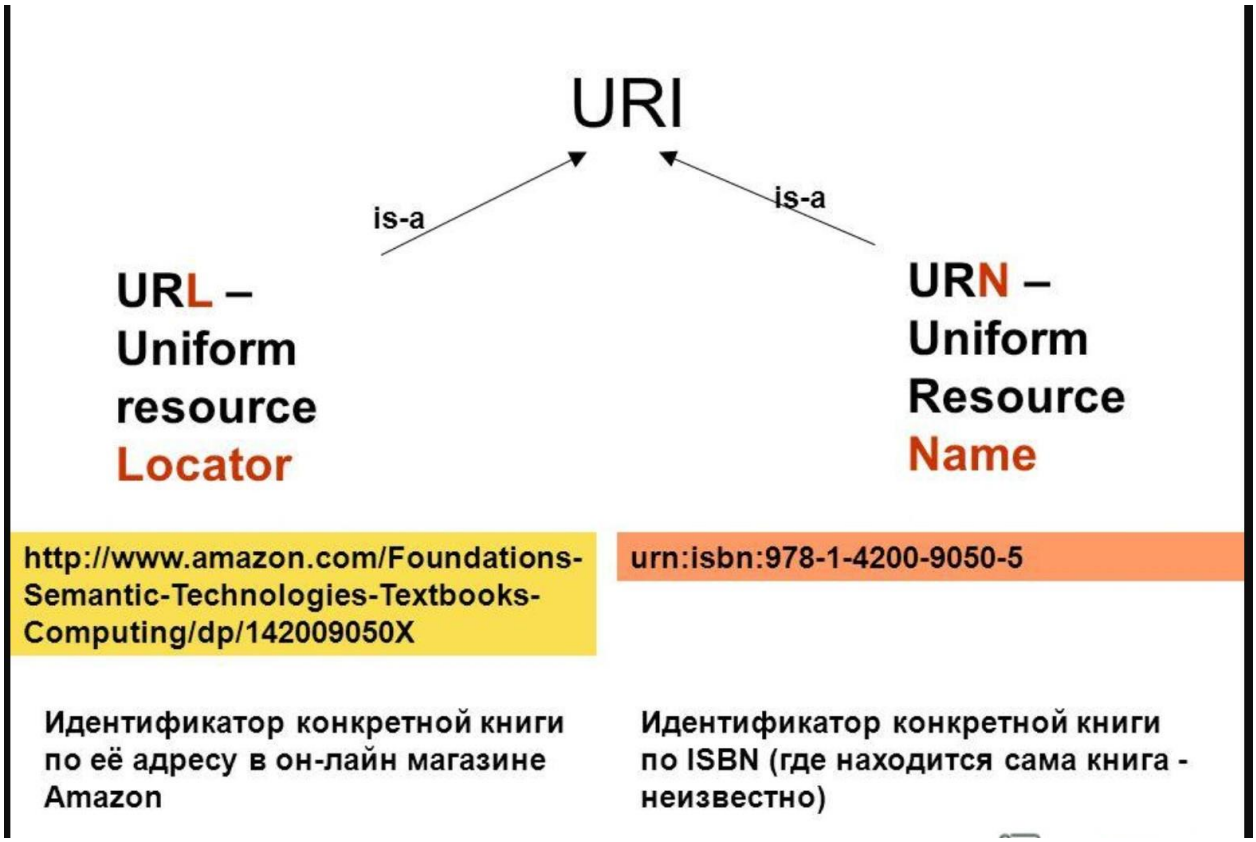


Чтобы стало совсем понятно, приведу следующий пример. Допустим, мы хотим описать некого Ивана.

URN в данном случае будет выглядеть следующим образом: **паспорт РФ: Иванов Иван Иванович, паспорт серия 1234 номер 123456**. Где «паспорт РФ» — это название идентификатора пространства имен, а «Иванов Иван Иванович, паспорт серия 1234 номер 123456» — это уникальное имя в этом пространстве.

С помощью этого URN мы однозначно идентифицируем Ивана, но не сможем определить его местоположение. Здесь нам поможет URL. Выглядеть это может примерно так: **машина: город N/улица M/квартира L**. Где «машина» — это метод получения доступа, а «город N...» — путь.

URN отвечает идентифицирует ресурс по имени и отвечает на вопрос «Что?». URL — указывает путь и метод доступа к ресурсу и отвечает на вопросы «Где?» и «Как?». При этом URN и URL — это частные случаи URI.



Архитектура REST API — самое популярное решение для организации взаимодействия между различными программами. Так произошло, поскольку HTTP-протокол реализован во всех языках программирования и всех операционных системах, в отличие от проприетарных протоколов.

Чаще всего REST API применяют:

1. Для связи мобильных приложений с серверными.
2. Для построения микросервисных серверных приложений. Это архитектурный подход, при котором большие приложения разбиваются на много маленьких частей.

3. Для предоставления доступа к программам сторонних разработчиков. Например, Stripe API позволяет программистам встраивать обработку платежей в свои приложения.

4. Что еще важно знать при работе с REST API
5. Каждый REST API запрос сообщает о результатах работы числовыми кодами — HTTP-статусами.
6. Например, редактирование записи на сервере может отработать успешно (код 200), может быть заблокировано по соображениям безопасности (код 401 или 403), а то и вообще сломаться в процессе из-за ошибки сервера (код 500). Цифровые статусы выполнения ошибок — аналог пользовательских сообщений с результатами работы программы.
7. Также REST API позволяет обмениваться не только текстовой информацией. С помощью этого инструмента можно передавать файлы и данные в специальных форматах: XML, JSON, Protobuf.

HTTP Request-Response Structure, структура HTTP запроса и ответа

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как *HyperText Transfer Protocol*, «протокол передачи гипертекста».

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

Также HTTP часто используется как протокол передачи информации для других протоколов прикладного уровня, таких как SOAP, XML-RPC и WebDAV. В таком случае

говорят, что протокол HTTP используется как «транспорт».

API многих программных продуктов также подразумевает использование HTTP для передачи данных — сами данные при этом могут иметь любой формат, например, XML или JSON.

Методы HTTP: основа работы REST API

Чтобы ресурс, который вы запрашиваете, выполнял нужные действия, используют разные способы обращения к нему. Например, если вы работаете со счетами с помощью ресурса /invoices, который мы придумали выше, то можете их просматривать, редактировать или удалять.

В API-системе четыре классических метода:

1. **GET** — метод чтения информации. GET-запросы всегда только возвращают данные с сервера, и никогда их не меняют и не удаляют. В бухгалтерском приложении GET /invoices вы открываете список всех счетов.

2. **POST** — создание новых записей. В нашем приложении POST /invoices используется, когда вы создаете новый счет на оплату.

3. **PUT** — редактирование записей. Например, PUT /invoices вы исправляете номер счета, сумму или корректируете реквизиты.

4. **DELETE** — удаление записей. В нашем приложении DELETE /invoices удаляет старые счета, которые контрагенты уже оплатили.

Таким образом, мы получаем четыре функции, которые одна программа может использовать при обращении к данным ресурса, в примере — это ресурс для работы со счетами /invoices

HTTP запросы и ответы имеют близкую структуру. Они состоят из:

1. Стартовой строки, описывающей запрос, или статус (успех или сбой). Это всегда одна строка.
2. Произвольного набора HTTP заголовков, определяющих запрос или описывающих тело сообщения.
3. Пустой строки, указывающей, что вся мета информация отправлена.
4. Произвольного тела, содержащего пересылаемые с запросом данные (например, содержимое HTML-формы) или отправляемый в ответ документ. Наличие тела и его размер определяется стартовой строкой и заголовками HTTP.

Как отправить HTTP-запрос?

Самый простой способ разобраться с протоколом HTTP — это попробовать обратиться к какому-нибудь веб-ресурсу вручную. Представьте, что вы браузер, и у вас есть пользователь, который очень хочет прочитать статьи Анатолия Ализара.

Предположим, что он ввёл в адресной строке следующее:

```
http://alizar.habrahabr.ru/
```

Соответственно вам, как веб-браузеру, теперь необходимо подключиться к веб-серверу по адресу alizar.habrahabr.ru.

Для этого вы можете воспользоваться любой подходящей утилитой командной строки. Например, telnet:

```
telnet alizar.habrahabr.ru 80
```

Сразу уточню, что если вы вдруг передумаете, то нажмите Ctrl + «]», и затем ввод — это позволит вам закрыть HTTP-соединение. Помимо telnet можете попробовать nc (или ncat) — по вкусу.

После того, как вы подключитесь к серверу, нужно отправить HTTP-запрос. Это, кстати, очень легко — HTTP-запросы могут состоять всего из двух строчек.

Для того, чтобы сформировать HTTP-запрос, необходимо составить стартовую строку, а также задать по крайней мере один заголовок — это заголовок Host, который является обязательным, и должен присутствовать в каждом запросе. Дело в том, что преобразование доменного имени в IP-адрес осуществляется на стороне клиента, и, соответственно, когда вы открываете TCP-соединение, то удалённый сервер не обладает никакой информацией о том, какой именно адрес использовался для соединения: это мог быть, например, адрес alizar.habrahabr.ru, habrahabr.ru или m.habrahabr.ru — и во всех этих случаях ответ может отличаться. Однако фактически сетевое соединение во всех случаях открывается с узлом 212.24.43.44, и даже если первоначально при открытии соединения был задан не этот IP-адрес, а какое-либо доменное имя, то сервер об этом никак не информируется — и именно поэтому этот адрес необходимо передать в заголовке Host.

Стартовая (начальная) строка запроса для HTTP 1.1 составляется по следующей схеме:

Метод URI HTTP/Версия

Например (такая стартовая строка может указывать на то, что запрашивается главная страница сайта):

```
GET / HTTP/1.1
```

Метод (в англоязычной тематической литературе используется слово *method*, а также иногда слово *verb* — «глагол») представляет собой последовательность из любых символов, кроме управляющих и разделителей, и определяет операцию, которую нужно осуществить с указанным ресурсом. Спецификация HTTP 1.1 не ограничивает количество разных методов, которые могут быть использованы, однако в целях соответствия общим стандартам и сохранения совместимости с максимально широким спектром программного обеспечения как правило используются лишь некоторые, наиболее стандартные методы, смысл которых однозначно раскрыт в спецификации протокола.

URI (*Uniform Resource Identifier*, унифицированный идентификатор ресурса) — путь до конкретного ресурса (например, документа), над которым необходимо осуществить

операцию (например, в случае использования метода GET подразумевается получение ресурса). Некоторые запросы могут не относиться к какому-либо ресурсу, в этом случае вместо URI в стартовую строку может быть добавлена звёздочка (астериск, символ «*»). Например, это может быть запрос, который относится к самому веб-серверу, а не какому-либо конкретному ресурсу. В этом случае стартовая строка может выглядеть так:

```
OPTIONS * HTTP/1.1
```

Версия определяет, в соответствии с какой версией стандарта HTTP составлен запрос. Указывается как два числа, разделённых точкой (например **1.1**).

Для того, чтобы обратиться к веб-странице по определённому адресу (в данном случае путь к ресурсу — это «/»), нам следует отправить следующий запрос:

```
GET / HTTP/1.1
Host: alizar.habrahabr.ru
```

При этом учитывайте, что для переноса строки следует использовать символ возврата каретки (Carriage Return), за которым следует символ перевода строки (Line Feed). После объявления последнего заголовка последовательность символов для переноса строки добавляется дважды.

Впрочем, в спецификации HTTP рекомендуется программировать HTTP-сервер таким образом, чтобы при обработке запросов в качестве межстрочного разделителя воспринимался символ LF, а предшествующий символ CR, при наличии такового, игнорировался. Соответственно, на практике бóльшая часть серверов корректно обработает и такой запрос, где заголовки отделены символом LF, и он же дважды добавлен после объявления последнего заголовка.

Если вы хотите отправить запрос в точном соответствии со спецификацией, можете воспользоваться управляющими последовательностями \r и \n:

```
echo -en "GET / HTTP/1.1\r\nHost: alizar.habrahabr.ru\r\n\r\n" | ncat
alizar.habrahabr.ru 80
```

Как прочитать ответ?

Стартовая строка ответа имеет следующую структуру:

HTTP/Версия Код состояния Пояснение

Версия протокола здесь задаётся так же, как в запросе.

Код состояния (*Status Code*) — три цифры (первая из которых указывает на класс состояния), которые определяют результат совершения запроса. Например, в случае, если был использован метод GET, и сервер предоставляет ресурс с указанным идентификатором, то такое состояние задаётся с помощью кода 200. Если сервер сообщает о том, что такого ресурса не существует — 404. Если сервер сообщает о том, что не может предоставить доступ к данному ресурсу по причине отсутствия необходимых привилегий у клиента, то используется код 403. Спецификация HTTP 1.1 определяет 40 различных кодов HTTP, а также допускается расширение протокола и

использование дополнительных кодов состояний.

Пояснение к коду состояния (*Reason Phrase*) — текстовое (но не включающее символы *CR* и *LF*) пояснение к коду ответа, предназначено для упрощения чтения ответа человеком. Пояснение может не учитываться клиентским программным обеспечением, а также может отличаться от стандартного в некоторых реализациях серверного ПО.

После стартовой строки следуют заголовки, а также тело ответа. Например:

```
HTTP/1.1 200 OK
Server: nginx/1.2.1
Date: Sat, 08 Mar 2014 22:53:46 GMT
Content-Type: application/octet-stream
Content-Length: 7
Last-Modified: Sat, 08 Mar 2014 22:53:30 GMT
Connection: keep-alive
Accept-Ranges: bytes

Wisdom
```

Тело ответа следует через два переноса строки после последнего заголовка. Для определения окончания тела ответа используется значение заголовка **Content-Length** (в данном случае ответ содержит 7 восьмеричных байтов: слово «Wisdom» и символ переноса строки).

Но вот по тому запросу, который мы составили ранее, веб-сервер вернёт ответ не с кодом 200, а с кодом 302. Таким образом он сообщает клиенту о том, что обращаться к данному ресурсу на данный момент нужно по другому адресу.

Смотрите сами:

```
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Sat, 08 Mar 2014 22:29:53 GMT
Content-Type: text/html
Content-Length: 154
Connection: keep-alive
Keep-Alive: timeout=25
Location: http://habrahabr.ru/users/alizar/

<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

В заголовке Location передан новый адрес. Теперь URI (идентификатор ресурса)

изменился на /users/alizar/, а обращаться нужно на этот раз к серверу по адресу habrahabr.ru (впрочем, в данном случае это тот же самый сервер), и его же указывать в заголовке Host.

То есть:

```
GET /users/alizar/ HTTP/1.1
Host: habrahabr.ru
```

В ответ на этот запрос веб-сервер Хабрахабра уже выдаст ответ с кодом 200 и достаточно большой документ в формате HTML.

HTTP запросы - это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера. Их стартовая строка состоит из трёх элементов:

1. *Метод HTTP*, глагол (например, GET, PUT или POST) или существительное (например, HEAD или OPTIONS), описывающие требуемое действие. Например, GET указывает, что нужно доставить некоторый ресурс, а POST означает отправку данных на сервер (для создания или модификации ресурса, или генерации возвращаемого документа).
2. *Цель запроса*, обычно URL, или абсолютный путь протокола, порт и домен обычно характеризуются контекстом запроса. Формат цели запроса зависит от используемого HTTP-метода. Это может быть
    - Абсолютный путь, за которым следует '?' и строка запроса. Это самая распространённая форма, называемая *исходной формой* (*origin form*) . Используется с методами GET, POST, HEAD, и OPTIONS.
      POST / HTTP 1.1
      GET /background.png HTTP/1.0
      HEAD /test.html?query=alibaba HTTP/1.1
      OPTIONS /anypage.html HTTP/1.0
    - Полный URL - *абсолютная форма* (*absolute form*) , обычно используется с GET при подключении к прокси.
      GET http://developer.mozilla.org/ru/docs/Web/HTTP/Messages HTTP/1.1
    - Компонента URL "authority", состоящая из имени домена и (необязательно) порта (предваряемого символом ':'), *называется authority form*. Используется только с методом CONNECT при установке туннеля HTTP.
      CONNECT developer.mozilla.org:80 HTTP/1.1
    - Форма звёздочки (*asterisk form)*, просто "звёздочка" ('*') используется с методом OPTIONS и представляет сервер.
      OPTIONS * HTTP/1.1
3. *Версия HTTP*, определяющая структуру оставшегося сообщения, указывая, какую версию предполагается использовать для ответа.

Заголовки

Заголовки запроса HTTP имеют стандартную для заголовка HTTP структуру: не зависящая от регистра строка, завершаемая (':') и значение, структура которого определяется заголовком. Весь заголовок, включая значение, представляет собой одну строку, которая может быть довольно длинной.

Существует множество заголовков запроса. Их можно разделить на несколько групп:

- *Основные заголовки (General headers*), например, Via (en-US), относящиеся к сообщению в целом
- *Заголовки запроса (Request headers*), например, User-Agent, Accept-Type, уточняющие запрос (как, например, Accept-Language), придающие контекст (как Referer), или накладывающие ограничения на условия (like If-None).
- *Заголовки сущности*, например Content-Length, относящиеся к телу сообщения. Как легко понять, они отсутствуют, если у запроса нет тела.

Тело

Последней частью запроса является его тело. Оно бывает не у всех запросов: запросы, собирающие (fetching) ресурсы, такие как GET, HEAD, DELETE, или OPTIONS, в нем обычно не нуждаются. Но некоторые запросы отправляют на сервер данные для обновления, как это часто бывает с запросами POST (содержащими данные HTML-форм).

Тела можно грубо разделить на две категории:

- Одноресурсные тела (Single-resource bodies), состоящие из одного отдельного файла, определяемого двумя заголовками: Content-Type и Content-Length.
- Многоресурсные тела (Multiple-resource bodies), состоящие из множества частей, каждая из которых содержит свой бит информации. Они обычно связаны с HTML-формами .

Ответы HTTP

Строка статуса (Status line)

Стартовая строка ответа HTTP, называемая строкой статуса, содержит следующую информацию:

1. *Версию протокола*, обычно HTTP/1.1.
2. *Код состояния (status code)*, показывающая, был ли запрос успешным. Примеры: 200, 404 или 302
3. *Пояснение (status text*). Краткое текстовое описание кода состояния, помогающее пользователю понять сообщение HTTP..

Пример строки статуса: HTTP/1.1 404 Not Found.

Заголовки

Заголовки ответов HTTP имеют ту же структуру, что и все остальные заголовки: не зависящая от регистра строка, завершаемая двоеточием (':') и значение, структура которого определяется типом заголовка. Весь заголовок, включая значение, представляет собой одну строку.

Существует множество заголовков ответов. Их можно разделить на несколько групп:

- *Основные заголовки (General headers*), например, Via (en-US), относящиеся к сообщению в целом.
- *Заголовки ответа (Response headers*), например, Vary и Accept-Ranges, сообщающие дополнительную информацию о сервере, которая не уместилась в строку состояния.
- *Заголовки сущности (Entity headers*), например, `Content-Length`, относящиеся к телу ответа. Отсутствуют, если у запроса нет тела.

Тело

Последней частью ответа является его тело. Оно есть не у всех ответов: у ответов с кодом состояния, например, `201` или `204`, оно обычно отсутствует.

Тела можно разделить на три категории:

- Одноресурсные тела (Single-resource bodies), состоящие из отдельного файла известной длины, определяемые двумя заголовками: `Content-Type` и `Content-Length`.
- Одноресурсные тела (Single-resource bodies), состоящие из отдельного файла неизвестной длины, разбитого на небольшие части (chunks) с заголовком Transfer-Encoding (en-US), значением которого является `chunked`.
- Многоресурсные тела (Multiple-resource bodies), состоящие из многокомпонентного тела, каждая часть которого содержит свой сегмент информации. Они относительно редки.