REST-URL

URL

> ⚙️ **URL - Uniform Resource Locator** (унифицированный **определитель местонахождения** ресурса)

Система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса.

Используется как стандарт записи ссылок на объекты в интернете.

Для обозначения электронного адреса используют аббревиатуру «URL» по ГОСТ Р 7.0.5-2008.

URL был изобретён Тимом Бернерсом-Ли в 1990 году в стенах Европейского совета по ядерным исследованиям. URL стал фундаментальной инновацией в Интернете. Изначально URL предназначался для обозначения мест расположения ресурсов (чаще всего файлов) в интернете. Сейчас URL применяется для обозначения адресов почти всех ресурсов Интернета. Стандарт URL закреплён в документе RFC 3986. URL позиционируется как часть более общей системы идентификации ресурсов URI, сам термин URL постепенно уступает место более широкому термину URI. Стандарт URL регулируется организацией IETF и её подразделениями.

- *В 2009 году Тим Бернерс-Ли высказал мнение об избыточности двойного слеша `//` в начале URL, после указания сетевого протокола.*

Структура URL:

URL был разработан как система для максимально естественного указания на местонахождения ресурсов в сети. Локатор должен был быть легко расширяемым и использовать лишь ограниченный набор ASCII*-символов (к

примеру, в URL никогда не применяется пробел). В связи с этим возникла следующая традиционная форма записи URL:

`<схема>://[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL-путь>][?<параметры>][#<якорь>]`

В этой записи:

Схема - схема обращения к ресурсу; в большинстве случаев имеется в виду сетевой протакол.

Логин - имя пользователя, используемое для доступа к ресурсу.

Пароль - пароль указанного пользователя

Хост - полностью прописанное доменное имя хоста в системе DNS или IP-адрес хоста в форме четырёх групп десятичных чисел, разделённых точками; числа — целые в интервале от 0 до 255.

Порт - порт хоста для подключения

URL-путь - уточняющая информация о месте нахождения ресурса; зависит от протокола.

Параметры - строка запроса с передаваемыми на сервер (методом GET)* параметрами. Начинается с символа `?`, разделитель параметров — знак `&`.

Пример: `?параметр_1=значение_1&параметр_2=значение_2&параметр3=значение_3`

Якорь - идентификатор якоря с предшествующим символом `#`. Якорем может быть указан заголовок внутри документа или атрибут id элемента. По такой ссылке браузер откроет страницу и переместит окно к указанному элементу.

- *ASCII - название таблицы (кодировки, набора), в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды.*
- *GET - Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.*

⚙️ **URI - Uniform Resource Identifier** (унифицированный **идентификатор** ресурса)

URI означает Uniform Resource Identifier и по сути является последовательностью символов, которая идентифицирует какой-то ресурс. URI может содержать URL и URN.

- ***URN - Unifrorm Resource Name*** *(унифицированное* ***имя*** *ресурса)*

Отличия URL и URI

URL	URI
Определитель местонахождения	Идентификатор ресурса
Синтаксис: [protocol]://www.[domain_name]:[port 80]/[path or exaction resource location]?[query]#[fragment]	Синтаксис: scheme:[//authority]path[?query][#fragment], где authority = [userinfo@]host[:port]
Схема всегда является протоколом, таким как http, https, ftp, LDAP и так далее	Схема может быть любой - протокол, имя или спецификация и так далее
Основная цель - получить адрес или местоположение ресурса.	Основная цель URI - идентифицировать ресурс и отличить его от других ресурсов, используя местоположение или имя.
Пример: https://www.ikea.com/ru/ru/campaigns/heroisk-talrika-recall-pub84a7d6b0	Пример: contact: +1 883-345-1111, urn:isbn:1234567890
URL используется для поиска только веб-страниц	Используется в файлах HTML, XML и библиотек тегов, таких как XSLT и jstl, для идентификации ресурсов и двоичных файлов.

REST

> ⚙️ **REST -** ***Re****presentational* ***S****tate* ***T****ransfer (*«передача репрезентативного состояния» или «передача „самоописываемого“ состояния»)

Стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами. В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то

ополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным
идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

REST - способ создания API с помощью протокола HTTP. Технология REST позволяет получать данные и сосояния удалённых приложений, работать с этими данными и модифицировать их.

Требования к архитектуре REST:

1. **Модель клиент-сервер:**

Первым ограничением, применимым к гибридной модели, является приведение архитектуры к модели клиент-сервер. Разграничение потребностей является принципом, лежащим в основе данного накладываемого ограничения. Отделение потребности интерфейса кликнта от потребностей сервера, хранящего данные, повышает переносимость кода клиентского интерфейса на другие платформы, а упрощение серверной част улучшает масштабируемость. Наибольшее же влияние на всемирную паутину, пожалуй, имеет само разграничение, которое позволяет отдельным частям развиваться независимо друг от друга, поддерживая потребности в развитии интернета со стороны различных организаций.

2. **Отсутствие состояния:**

Протокол взаимодействия между клиентом и сервером требует соблюдения следующего
условия: в период между запросами клиента никакая информация о состоянии клиента на сервере не хранится (Stateless protocol или «протокол без сохранения состояния»). Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Состояние сессии при этом сохраняется на стороне клиента. Информация о состоянии сессии может быть передана сервером какому-либо
другому сервису (например, в службу базы данных) для поддержания устойчивого состояния, например, на период установления аутентификации. Клиент инициирует отправку запросов, когда он готов (возникает необходимость) перейти в новое состояние. Во время обработки клиентских запросов считается, что клиент находится в переходном состоянии. Каждое отдельное состояние приложения представлено связями, которые могут быть задействованы при следующем обращении клиента.

3. **Кэширование:**

Как и во всемирной паутине, клиенты, а также промежуточные узлы, могут выполнять кэштрование ответов сервера. Ответы сервера, в свою очередь, должны иметь явное или
неявное обозначение как кэшируемые или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных в ответ на последующие запросы. Правильное использование кэширования способно частично или полностью устранить некоторые клиент-серверные взаимодействия, ещё больше повышая производительность и масштабируемость системы.

4. **Единообразие интерфейса:**

Наличие унифицированного интерфейса является фундаментальным требованием дизайна REST-сервисов. Унифицированные интерфейсы позволяют каждому из сервисов развиваться независимо. К унифицированным интерфейсам предъявляются следующие четыре ограничительных условия:

- **Идентификация ресурсов**
  Все ресурсы идентифицируются в запросах, например, с использованием URI в интернет-системах. Ресурсы концептуально отделены от представлений, которые возвращаются клиентам. Например, сервер может отсылать данные из базы данных в виде HTML, XML или JSON, ни один из которых не является типом хранения внутри сервера.
- **Манипуляция ресурсами через представление**
  Если клиент хранит представление ресурса, включая метаданные — он обладает достаточной информацией для модификации или удаления ресурса.
- **«Самоописываемые» сообщения**
  Каждое сообщение содержит достаточно информации, чтобы понять, каким образом его обрабатывать. К примеру, обработчик сообщения (parser), необходимый для извлечения данных, может быть указан в списке MIME-типов.
- **Гипермедиа как средство изменения состояния приложения (HATEOAS)**
  Клиенты изменяют состояние системы только через действия, которые динамически определены в гипермедиа на сервере (к примеру, гиперссылки в гипертексте).
  Исключая простые точки входа в приложение, клиент не может предположить, что доступна какая-то операция над каким-то ресурсом, если

не получил информацию об этом в предыдущих запросах к серверу. Не существует универсального формата для предоставления ссылок между ресурсами, Web Linking (RFC 5988 -> RFC 8288) и JSON Hypermedia API Language на Wayback Machine являются двумя популярными форматами предоставления ссылок в REST HYPERMEDIA сервисах.

5. **Слои:**

Клиент обычно не способен точно определить, взаимодействует он напрямую с сервером или же с промежуточным узлом, в связи с иерархической структурой сетей (подразумевая, что такая структура образует слои). Применение промежуточных серверов способно повысить масштабируемость за счёт балансировки нагрузок и распределённого кэширования. Промежуточные узлы также могут подчиняться политике безопасности с целью обеспечения конфиденциальности информации.

6. **Код по требованию:**

REST может позволить расширить функциональность клиента за счёт загрузки кода с сервера в виде апплетов или скриптов. Филдинг утверждает, что дополнительное ограничение позволяет проектировать архитектуру, поддерживающую желаемую функциональность в
общем случае, но, возможно, за исключением некоторых контекстов.

- *Апплет - несамостоятельный компонент программного обеспечения, работающий в контексте другого, полновесного приложения, предназначенный для одной узкой задачи и не имеющий ценности в отрыве от базового приложения.*
- *Скрипт - высокоуровневый язык сценариев — кратких описаний действий, выполняемых системой.*

HTTP Request-Response Structure

⚙️ HTTP - *HyperText Transfer Protocol* — «протокол передачи гипертекста» — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

HTTP сообщения - это обмен данными между сервером и клиентом. Есть два типа сообщений: ***запросы***, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера, и ***ответы*** от сервера.

Запросы HTTP

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.

Загаловки

Загаловки запросы 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, 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:

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, 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, значением которого является `chunked`.
- Многоресурсные тела (Multiple-resource bodies), состоящие из многокомпонентного тела, каждая часть которого содержит свой сегмент информации. Они относительно редки.

Коды ответа

1 Информационные -** Данная группа отвечает за передачу данных. Коды этого типа свидетельствуют о том, что запрос принят сервером и обрабатывается.

2 Успешные -** Коды группы сообщают, что запрос не только принят сервером, но и успешно обработан.

3 Перенаправления -** Данная группа кодов состояния сообщает о перенаправлении пользователя с его согласием или без него.

4 Клиентские ошибки -** Коды состояний данной группы сообщают об ошибках клиента, при которых сервер не может вызвать запрашиваемый результат.

5 Серверные ошибки -** В эту группу входят коды ошибок со стороны сервера, когда по тем или иным причинам он не способен обработать запрос или выполнить требуемую операцию.

Сообщения HTTP играют ключевую роль в использовании HTTP; они имеют простую
структуру и хорошо расширяемы. Механизм фреймов в HTTP/2 добавляет ещё один промежуточный уровень между синтаксисом HTTP/1.x и используемым им транспортным протоколом, не проводя фундаментальных изменений: создаётся надстройка над уже зарекомендовавшими себя методами.