#### **REST: теория и практика**

### Что вы узнаете

В этом материале вы познакомитесь с архитектурой и принципами REST. А ещё узнаете, какие методы в ней применяются, что такое RESTful-сервисы и как они используются, а также научитесь создавать и тестировать такие сервисы.

### Что такое REST и почему он так популярен

****REST****(****RE****presentational ****S****tate ****T****ransfer) — это архитектурный стиль, который описывает основы построения распределённых сетевых систем. Другими словами, REST — это набор правил, который помогает программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать. Принципы REST сформулировал [Рой Филдинг](https://en.wikipedia.org/wiki/Roy_Fielding" \t "/home/parviz/Documents\\x/_blank) в 2000 году.

В отличие от других подходов, например SOAP, REST обладает гибкостью и универсальностью. Это позволяет программистам комбинировать разные модули в системе.

Ключевые элементы REST:

* ****сервер****, который хранит полезные данные или осуществляет полезные функции; сервер предлагает API как средство доступа к своим данным или функциям;
* ****клиент****— программное обеспечение, которое работает на компьютере, смартфоне или другом устройстве пользователя и обращается к серверу по предоставленному API;
* ****ресурс**** — контент, который сервер может предоставить клиенту (например, видео или текстовый файл).

Чтобы получить доступ к ресурсу, клиент отправляет HTTP-запрос к серверу.  
В ответ сервер возвращает HTTP-ответ с закодированными данными о ресурсе. Особенность REST в том, что оба типа сообщений (запрос и ответ) являются информативными, то есть содержат информацию о том, как их интерпретировать и обрабатывать.

Наиболее часто информация в запросах и ответах передаётся в виде JSON.

****JSON****(JavaScript Object Notation) — это легковесный формат обмена данными. С помощью JSON можно передать информацию о числах и строках, а также объектах и их коллекциях.

Для примера попробуем записать информацию о пользователе онлайн-университета:

{  
    "firstName": "Иван",  
    "lastName": "Петров",  
    "age": 25,  
    "courses": [  
        {"name": "Программирование", "duration": 4},  
        {"name": "Система контроля версий Git", "duration": 2}  
    ]  
}

В JSON данные записываются в формате «ключ — значение». Ключ — это название параметра объекта, а значение — значение этого параметра. Также можно записать коллекцию (заключается в квадратные скобки) и сделать значением некоторый объект или коллекцию объектов (как информация о курсах).

Важное преимущество JSON — отсутствие лишней информации. Например, в сравнении с XML в JSON отсутствует ссылка на схему XSD и каждый элемент не нужно заключать в парные теги. Это облегчает чтение JSON-запросов и ответов. А ещё JSON намного легче читается людьми, это существенно облегчает разработку.

Теперь представим, каким может быть наш первый REST-запрос и ответ на него. Допустим, есть сервер онлайн-университета, который хранит информацию обо всех курсах, и нам нужно получить эту информацию с сервера. Отправим запрос с нашего клиента (браузера или мобильного приложения):

GET [https://skillbox.ru/code/](https://skillbox.ru/code/" \t "/home/parviz/Documents\\x/_blank)

В ответ мы можем получить список курсов по программированию:

{  
    "courses": [  
        {"id":1, "name":"Java с нуля", "duration":4},  
        {"id":2, "name":"Система контроля версий Git", "duration":2}  
        {"id":3, "name":"Python Basic", "duration":3}  
    ]  
}

Как видите, программист сможет легко понять содержание запроса (Get/Programming/Courses) и ответа и прочитать их. Это важное преимущество архитектуры REST.

### Виды запросов и ответов в REST

Для корректного взаимодействия клиент и сервер должны понимать, где хранится та или иная информация. В REST для этого используют структуру HTTP-запросов, она помогает передавать максимум информации в минимальном объёме.

Типичный REST-запрос всегда использует какой-то ****HTTP-метод****. Тип метода определяет, что мы собираемся делать с ресурсом сервера. Наиболее распространённые методы:

* POST, чтобы создать ресурс;
* GET, чтобы получить ресурс;
* PUT, чтобы обновить содержимое ресурса;
* PATCH, частичное обновление;
* DELETE, чтобы удалить ресурс.

Этот набор операций принято называть CRUD-операциями (****C****reate, ****R****etrieve, ****U****pdate, ****D****elete). Подавляющее большинство веб-сервисов работают с этими операциями.

Далее необходимо указать, где мы будем работать с ресурсом. Для этого нужно указать ****эндпоинт****— уникальный идентификатор ресурса в Сети. Обычно в качестве эндпоинта служит URL сервера.

****Путь****до ресурса также очень важен. С помощью пути мы можем передать информацию о назначении API. Также в пути могут передаваться идентификаторы ресурсов.

****Заголовки****хранят информацию, относящуюся как к клиенту, так и к серверу. В основном заголовки предоставляют данные аутентификации, такие как ключ API, имя или IP-адрес компьютера, на котором установлен сервер, и информацию о формате ответа (как правило, это application/JSON).

****Тело запроса**** используется для передачи дополнительной информации на сервер. Например, это может быть часть данных, которую вы хотите добавить или заменить.

У ответа сервера тоже могут быть свои особенности. В ответ на запрос сервер отправляет не сам искомый ресурс, а его представление — машиночитаемое описание его текущего состояния. Один и тот же ресурс может быть представлен в разных форматах. Например, кроме используемого нами JSON, это может быть XML.

Ранее мы получили информацию о курсах по программированию в онлайн-университете через REST-запрос. Попробуем обновить информацию о длительности курса по Python. Для этого нужно обратиться к серверу и указать идентификатор курса:

PUT /course/python-basic-customtariffs/1  
HOST: https://skillbox.ru  
Content-Type: application/json  
Accept: application/json  
{  
    "duration": 8  
}

В данном случае мы обратились к серверу и попросили его обновить (PUT) информацию о курсе по программированию с id=1. Отразим это в таблице, чтобы легче запомнить.

|  |  |  |
| --- | --- | --- |
| ****Зона ответственности**** | ****Часть запроса**** | ****Пример**** |
| Что делаем? | Метод | PUT |
| Где делаем и с чем? | Адрес | [https://skillbox.ru/course/python-basic-customtariffs/](https://skillbox.ru/course/python-basic-customtariffs/" \t "/home/parviz/Documents\\x/_blank)/1 |
| Дополнительная информация | Тело и заголовки | {  "duration": 8  } |

И давайте рассмотрим несколько рекомендаций при работе с REST:

1. Используйте существительные, но не глаголы в URL.

Для простоты понимания используйте эту структуру для каждого ресурса:

|  |  |
| --- | --- |
| /courses | /courses/456 |
| GET (чтение) | Вернёт список курсов | Вернёт курс с id 456 |
| POST (создание) | Создаёт новый курс | ****Такой метод невозможен**** |
| PUT (обновление) | Обновляет множество переданных курсов | Обновляет данные курса 456 |
| PATCH (частичное обновление) | Частично обновляет множество курсов | Частично обновляет данные курса 456 |
| DELETE (удаление) | Удаляет все курсы | Удаляет курс 456 |

1. Не рекомендуется использовать GET-методы для изменения любого состояния.

Допустим, нам необходимо активировать курс через REST, и, возможно, придёт мысль сделать такой запрос:

  GET /courses/456/activate

Получается, GET-запрос изменяет состояние курса. Для таких случаев подходит PUT:

  PUT /courses/456/activate

1. Используйте множественные формы существительных.

Не перемешивайте существительные единственного и множественного числа. Используйте только существительные во множественном числе для всех ресурсов:

* /cars, а не /car
* /users, а не /user
* /products, а не /product
* /settings, а не /setting

1. Используйте вложенные ресурсы.

Если у курса необходимо получить список модулей:

GET /courses/456/modules

Если у курса необходимо получить конкретный модуль:

GET /courses/456/modules/9

### Заключение

В материале мы подробно рассмотрели основные принципы архитектурного подхода REST. Этот подход важен в современных распределённых системах и непременно пригодится вам в дальнейшем.