hillel КОМПЬЮТЕРНАЯ ШКОЛА

Web Services & Тестирование API

Web Services

Web сервисы

Web-сервис (служба) – программа, которая организовывает взаимодействие между сайтами. Информация с одного портала передается на другой.

Web сервисы

Например, есть авиакомпания. У нее много рейсов, соответственно, много билетов. Информацию через веб-службу она передает сайту-агрегатору тур-путешествий. Пользователь, который заходит на агрегатор, сможет прямо там купить билеты этой авиакомпании.

Web сервисы

Другой пример веб-сервисов — это сайт отслеживания погоды, который содержит сведения о метеоусловиях в конкретном городе или по стране в целом. Данная информация также часто используется сторонними приложениями.

Web сервисы

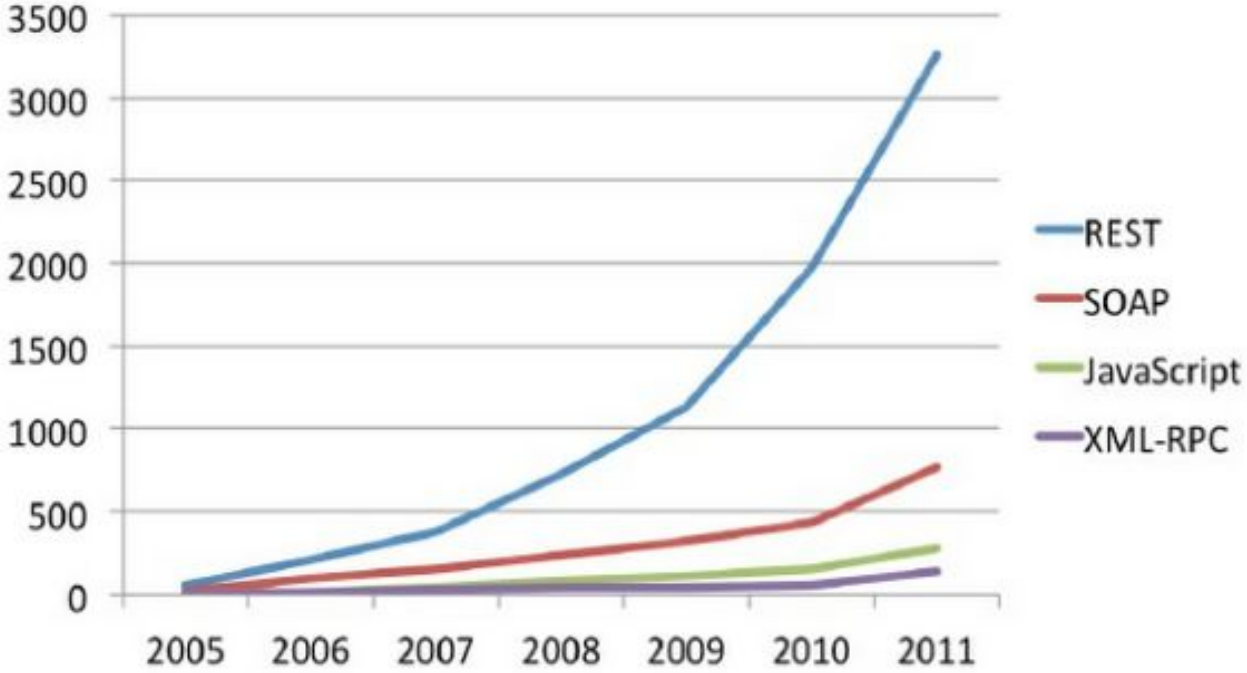
Информация в интернете разнородна. Сайты управляются разными системами. используются разные протоколы передачи и шифрования. Веб-сервисы упрощают обмен информацией между разными площадками.

Архитектура и протоколы Web-сервисов

На сегодняшний день наибольшее распространение получили следующие протоколы реализации веб-сервисов:

- SOAP (Simple Object Access Protocol)
- REST (Representational State Transfer)

Архитектура и протоколы Web-сервисов



Архитектура и протоколы Web-сервисов

SOAP более применим в сложных архитектурах, где взаимодействие с объектами выходит за рамки теории CRUD, а вот в тех приложениях, которые не покидают рамки данной теории, вполне применимым может оказаться именно REST ввиду своей простоты и прозрачности.

Операция	Оператор в языке SQL	Операция в протоколе HTTP
Создание (create)	INSERT	POST
Чтение (read)	SELECT	GET
Редактирование (update)	UPDATE	PUT или PATCH
Удаление (delete)	DELETE	DELETE

Архитектура и протоколы Web-сервисов

Если любым объектам вашего сервиса не нужны более сложные взаимоотношения, кроме: «Создать», «Прочитать», «Изменить», «Удалить» (как правило — в 99% случаев этого достаточно), возможно, именно REST станет правильным выбором.

Кроме того, REST по сравнению с SOAP, может оказаться и более производительным, так как не требует затрат на разбор сложных XML команд на сервере (выполняются обычные HTTP запросы — PUT, GET, POST, DELETE). Хотя SOAP, в свою очередь, более надежен и безопасен.

Архитектура и протоколы Web-сервисов

Важно понимать, что REST – это не протокол и не стандарт, а архитектурный стиль. У этого стиля есть свои принципы.

1. Give every “thing” an ID.

   Очень желательно.

1. Link things together.

   Например, в страницу (представление) о Mercedes C218 хорошо бы добавить ссылку на страницу конкретно о двигателе данной модели, чтобы желающие могли сразу туда перейти, а не тратить время на поиск этой самой страницы.

Архитектура и протоколы Web-сервисов

3. Use standard methods.

Имеется в виду, экономьте свои силы и деньги заказчика, используйте стандартные методы HTTP, например GET http://www.example.com/cars/00345 для получения данных вместо определения собственных методов вроде getCar?id=00345.

4. Resources can have multiple representations.

Одни и те же данные можно вернуть в XML или JSON для программной обработки или обернутыми в красивый дизайн для просмотра человеком.

Архитектура и протоколы Web-сервисов

5. Communicate statelessly.

Да, RESTful сервис должен быть как идеальный суд – его не должно интересовать ни прошлое подсудимого (клиента), ни будущее – он просто выносит приговор (отвечает на запрос).

RESTful (веб-)сервис всего лишь означает сервис, реализованный с использованием принципов REST

Сравнение подходов SOAP и REST

1. SOAP – это целое семейство протоколов и стандартов, откуда напрямую вытекает, что это более тяжеловесный и сложный вариант с точки зрения машинной обработки. Поэтому REST работает быстрее.
2. SOAP используют HTTP как транспортный протокол, в то время как REST базируется на нем.
3. Есть мнение, что разработка RESTful сервисов намного проще.
4. SOAP - только XML, REST - любые типы данных, например удобный JSON
5. «REST vs SOAP» можно перефразировать в «Простота vs Стандарты»
6. SOAP не кэшируется на сервере (так как использует HTTP как транспортный протокол)

Архитектура и протоколы Web-сервисов

Приведу пару примеров на понимание разницы между подходами.

Букмекерская контора заказала сервис для работы с футбольной статистикой. Пользовательский функционал – получить список матчей, получить детали о матче. Для редакторов – редактировать (Create, Edit, Delete) список матчей, редактировать детали матча.

Для такой задачи однозначно надо выбирать подход REST и получать бенефиты от его простоты и естественности во взаимодействии с HTTP.

Что происходит при обращении с использованием методов HTTP

URL	GET	POST	PUT	DELETE
example.com/matches	Получаем список матчей	Создаем заново список матчей	Обновляем список матчей	Мстим директору букмекерской конторы
example.com/matches/28	Получаем детали матча с ID=28	Создаем информацию о матче	Обновляем детали матча	Удаляем матч

Архитектура и протоколы Web-сервисов

Все очень просто! Теперь пример посложнее.

Та же букмекерская контора захотела API для ставок на live матчи. Эта процедура включает в себя **многочисленные проверки, например, продолжает ли ставка быть актуальной, не изменился ли коэффициент, не превышена ли максимальная сумма ставки для маркета. После этого происходит денежная транзакция, результаты которой записываются в основную и в резервные базы данных. Лишь после этого клиенту приходит ответ об успешности операции.**

Здесь явно прослеживается ориентация на операции, имеются повышенные требования к безопасности и устойчивости приложения, поэтому целесообразно использовать SOAP.

XML & JSON

JSON (англ. JavaScript Object Notation) — формат обмена данными, легко читаем людьми, легко обрабатывается и генерируется программами. Основан на подмножестве языка JavaScript.

XML (eXtensible Markup Language) — расширяемый язык разметки.

XML & JSON

Преимущества JSON:

- Удобочитаемость кода.
- Простота создания объекта данных на стороне сервера.
- Простота обработки данных на стороне клиента.
- Простота расширения.

Преимущества XML:

- Отладка и исправление ошибок.
- Безопасность.

XML & JSON

Удобочитаемость кода.

XML

```
<person>
    <firstname>Subbu</firstname>
    <lastname>Allamaraju</lastname>
</person>
```

JSON

```
({
  "firstName" : "Subbu",
  "lastName" : "Allamaraju"
});
```

Задачи веб-сервисов

Веб-сервисы могут использоваться во многих сферах.

Задачи веб-сервисов

B2B-транзакции

Интеграция процессов идет сразу, без участия людей.

Например, пополнение каталога интернет-магазина новыми товарами. Их привозят на склад, и кладовщик отмечает в базе данных приход. Автоматически информация передается в интернет-магазин. И покупатель вместо пометки “Нет на складе” на карточке товара видит его количество.

Задачи веб-сервисов

Интеграция сервисов предприятий

Если в компании используются корпоративные программы, то веб-сервис поможет настроить их совместную работу.

Задачи веб-сервисов

Создание системы клиент-сервер

Сервисы используются, чтобы настроить работу клиента и сервера. Это дает преимущества:

- можно продавать не само программное обеспечение, а делать платным доступ к веб-сервису;
- легче решать проблемы с использованием стороннего ПО;
- проще организовывать доступ к контенту и материалам сервера.

Веб-сервис — это приложение, которое упрощает техническую настройку взаимодействия ресурсов.

Swagger

Swagger — это набор инструментов, которые помогают описывать API. Благодаря ему пользователи и машины лучше понимают возможности REST API без доступа к коду. С помощью Swagger можно быстро создать документацию и отправить ее другим разработчикам или клиентам.

Основные подходы

Swagger предлагает два основных подхода к генерированию документации:

- Автогенерация на основе кода.
- Самостоятельная разметка-написание.

Swagger

- Первый подход проще. Мы добавляем зависимости в проект, конфигурируем настройки и получаем документацию. Сам код из-за этого может стать менее читабельным, документация тоже не будет идеальной. Но задача минимум решена — код задокументирован.
- Чтобы пользоваться вторым подходом, нужно знать синтаксис Swagger. Описания можно готовить в формате YAML/JSON. Можно упростить эту задачу, используя Swagger Editor. Конечно, второй подход позволяет сделать документацию более качественной и кастомной для каждого конкретного проекта и его особенностей, к тому же все не так сложно как может показаться, это потребует минимальных дополнительных усилий.

Curl

- Что такое Curl ? Curl — это сокращение от “Client URL”. Утилита доступна в большинстве систем на основе Unix и предназначена для проверки подключения к URL-адресам. Кроме того команда Curl — отличный инструмент передачи данных.

- HTTP и HTTPS
- FTP и FTPS
- IMAP и IMAPS
- POP3 и POP3S
- SMB и SMBS и другие

Swagger UI

Не защищено | petstore.swagger.io/#/pet/addPet

Schemes

HTTPS

Authorize

pet Everything about your Pets

Find out more: http://swagger.io

POST /pet Add a new pet to the store

Parameters

Try it out

Name	Description
body * required *(body)*	Pet object that needs to be added to the store Example Value | Model

```
{
  "id": 12,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Parameter content type

application/json

Swagger UI

Не защищено | petstore.swagger.io/#/pet/addPet

Parameters

Cancel

Name	Description
body * required *(body)*	Pet object that needs to be added to the store

Edit Value | Model

1. Заполняем поля (я изменила id и name)

```
{
  "id": 25,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "test",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Cancel

2. Отправляем запрос

Parameter content type

application/json

Execute | Clear

Swagger UI

Не защищено | petstore.swagger.io/#/pet/addPet

Responses

Response content type: application/xml

Наш запрос через Curl

Curl

```
curl -X POST "https://petstore.swagger.io/v2/pet" -H "accept: application/xml" -H "Content-Type: application/json" -d "{ \"id\": 25, \"category\": { \"id\": 0, \"name\": \"string\" }, \"name\": \"test\", \"photoUrls\": [ \"string\" ], \"tags\": [ { \"id\": 0, \"name\": \"string\" } ], \"status\": \"available\"}"
```

Request URL

```
https://petstore.swagger.io/v2/pet
```

Server response

Ответ от сервера

Code	Details
200 *Undocumented*	Response body

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <Pet>
    <category>
      <id>0</id>
      <name>string</name>
    </category>
    <id>25</id>
    <name>test</name>
    <photoUrls>
      <photoUrl>string</photoUrl>
    </photoUrls>
    <status>available</status>
    <tags>
      <tag>
        <id>0</id>
        <name>string</name>
      </tag>
    </tags>
  </Pet>
```

Download

Response headers

```
content-type: application/xml
```

Swagger UI

Не защищено | petstore.swagger.io/#/pet/addPet

Тип запроса

pet Everything about your Pets

Find out more: http://swagger.io

POST `/pet` Add a new pet to the store

URL метода

Parameters

Try it out

Name	Description
body * required *(body)*	Pet object that needs to be added to the store

Тело сообщения

Example Value | Model

```
{
  "id": 25,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "test",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Parameter content type

application/json

Responses

Response content type application/xml

Swagger
Supported by SMARTBEAR

https://petstore.swagger.io/v2/swagger.json

Swagger Petstore 1.0.0

[Base URL: petstore.swagger.io/v2]

https://petstore.swagger.io/v2/swagger.json

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. `special-key` to test the authorization filters.

Terms of service

Contact the developer

Apache 2.0

Find out more about Swagger

Postman

File Edit View Help

New Import Runner My Workspace Invite Upgrade

Filter

History Collections

Trash

Petstore.swagger.io
3 requests

POST Add pet

GET Find pet by ID

GET findByStatus

Postman Echo
37 requests

Reuse variables
2 requests

Swagger Petstore
2 requests

POST Add pet

GET Find pet by ID

Task_17_fail_but_runner_green

POST Add pet | GET Find pet by ID

No Environment

POST petstore.swagger.io/v2/pet Send Save

Pretty Raw Preview JSON

```
1  {
2      "id": 11,
3      "category": {
4          "id": 0,
5          "name": "string"
6      },
7      "name": "monkey",
8      "photoUrls": [
9          "string"
10     ],
11     "tags": [
12         {
13             "id": 0,
14             "name": "string"
15         }
16     ],
17     "status": "available"
18 }
```

Build Browse

Swagger UI

Не защищено | petstore.swagger.io/#/pet/getPetById

GET `/pet/{petId}` Find pet by ID

Returns a single pet

Parameters Try it out

Name	Description
petId * required `integer($int64)` *(path)*	ID of pet to return

Responses Response content type application/xml

Code	Description
200	*successful operation* Example Value | Model

```
<?xml version="1.0" encoding="UTF-8"?>
<Pet>
        <id>0</id>
        <Category>
                <id>0</id>
                <name>string</name>
```

Postman

File Edit View Help

New Import Runner My Workspace Invite Upgrade

Filter

History Collections

Trash

Petstore.swagger.io
3 requests

POST Add pet
GET Find pet by ID
GET findByStatus

Postman Echo
37 requests

Reuse variables
2 requests

Swagger Petstore
2 requests

POST Add pet
GET Find pet by ID

Task_17_fail_but_runner_green

POST Add pet | GET Find pet by ID

No Environment

Find pet by ID

Examples (0)

GET petstore.swagger.io/v2/pet/11 Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code

KEY	VALUE	DESCRIPTION	••• Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results Status: 200 OK Time: 310 ms Size: 432 B Save Download

Pretty Raw Preview JSON


```
{
    "id": 11,
    "category": {
        "id": 0,
        "name": "string"
    },
    "name": "monkey",
    "photoUrls": [
        "string"
    ],
    "tags": [
        {
            "id": 0,
            "name": "string"
```


Build Browse

Postman

File Edit View Help

New Import Runner My Workspace Invite Upgrade

Filter

History Collections

Trash

Petstore.swagger.io
3 requests

POST Add pet
GET Find pet by ID
GET findByStatus

Postman Echo
37 requests

Reuse variables
2 requests

Swagger Petstore
2 requests

POST Add pet
GET Find pet by ID

Task_17_fail_but_runner_green

POST Add pet | GET Find pet by ID

No Environment

Find pet by ID

Examples (0)

GET petstore.swagger.io/v2/pet/25 Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code

KEY	VALUE	DESCRIPTION	••• Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results Status: 200 OK Time: 326 ms Size: 430 B Save Download

Pretty Raw Preview JSON


```
{
    "id": 25,
    "category": {
        "id": 0,
        "name": "string"
    },
    "name": "test",
    "photoUrls": [
        "string"
    ],
    "tags": [
        {
            "id": 0,
            "name": "string"
```


Build Browse

Домашнее задание работа со Swagger

- Swagger

- Выполнить несколько запросов по Сваггеру!
- Опишу в ДЗ более подробнее