

РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

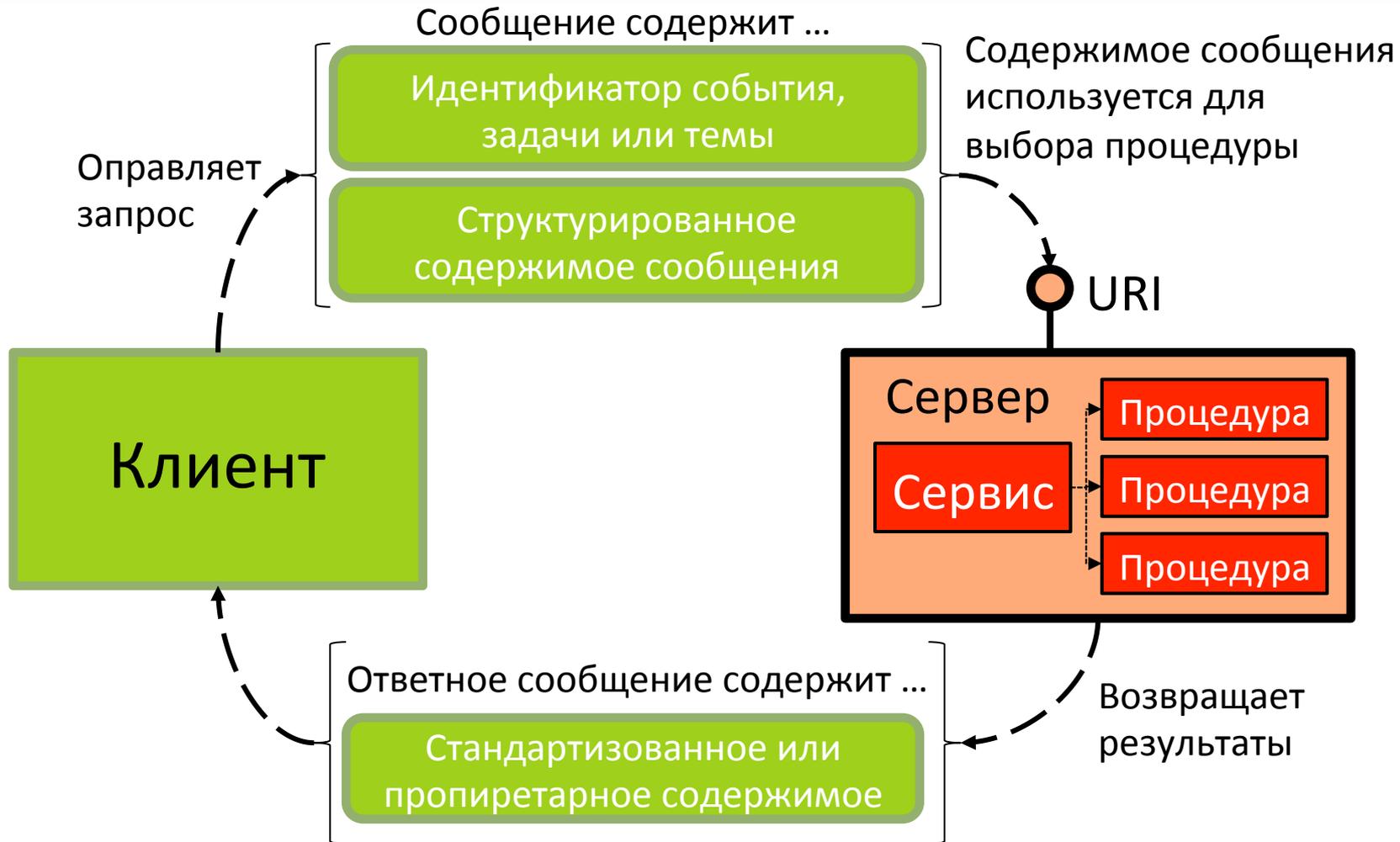
Сервис-ориентированная архитектура

API СООБЩЕНИЙ

Стили API ВЕБ-СЕРВИСОВ: API СООБЩЕНИЙ

- ⊙ Если API сервиса основывается на реализуемых процедурах, сложно обрабатывать изменение такого API, т.к. необходимо менять код на всех клиентах. Если клиенты и сервисы управляются разными организациями, такой подход может стать не приемлемым.
- ⊙ **Решение** - необходимо определить сообщения для обмена, которые бы не раскрывали интерфейс удаленных процедур.
- ⊙ Данные сообщения должны содержать информацию об определенных событиях, задачах, которые необходимо выполнить.
- ⊙ Клиент должен отправить сообщение с этой информацией по указанному URI, а сервер, на основе анализа сообщения должен выполнить определенные процедуры.

Стили API ВЕБ-СЕРВИСОВ: API для СООБЩЕНИЙ



Стили API ВЕБ-СЕРВИСОВ: API СООБЩЕНИЙ

- ◎ Фактически, такой подход строится на использовании тех же технологий, что и RPC API. Отличается метод проектирования системы:
 - ◎ Проектирование начинается с определения формата сообщений, которыми обмениваются удаленные компоненты.
 - ◎ Они могут быть определены на некотором промежуточном языке, например посредством XML Schema Language или Google Protocol Buffers IDL.
- ◎ Сервисы создаются после того, как сообщения были определены.
- ◎ Отличием сервисов для API сообщений от сервисов для RPC является количество аргументов в интерфейсе сервиса. Чаще всего, при использовании API сообщений, каждый метод имеет только 1 параметр.

СООБЩЕНИЯ

- ◎ Выделяют следующие типы сообщений:
 - ◎ *Командные сообщения*: используются для инициализации определенных действий на сервере (например, обработать запись).
 - ◎ *Сообщения о событиях*: оповещают получателя о произошедших у клиента событиях (например, закончилось место на складе).
 - ◎ *Документы-сообщения*: содержат информацию о тех или иных бизнес-сущностях (например, бланк заказа).
- ◎ Ответы сервера также определяются сообщениями: либо полноценным сообщением с ответом, либо простейшим подтверждением о получении сообщения.

API СООБЩЕНИЙ - ПРИМЕР

```
[ServiceContract( Name = "Dropbox" )]
public interface IDropbox {
    [OperationContract(
        Action = "http://acmeCorp.org/Dropbox/Invoice",
        IsOneWay = true)]
    void Invoice( InvoiceMessage invoice);

    [OperationContract(
        Action = "http://acmeCorp.org/Dropbox/Contractor",
        IsOneWay = true)]
    void Contractor( ContractorMessage contractor);
}
```

API СООБЩЕНИЙ - ПРИМЕР

```
[MessageContract( IsWrapped = false)]
public class InvoiceMessage {
    [MessageBodyMember]
    public Invoice Invoice { get; set; }
}

[DataContract(Name = "Invoice")]
public class Invoice {
    [DataMember( IsRequired = true, Order = 1)]
    public int ContractorId { get; set; }

    [DataMember( IsRequired = true, Order = 2)]
    public string PurchaseOrder { get; set; }

    [DataMember( IsRequired = true, Order = 3)]
    public DateTime StartDate { get; set; }

    [DataMember( IsRequired = true, Order = 4)]
    public DateTime EndDate { get; set; }

    [DataMember( IsRequired = true, Order = 5)]
    public decimal Hours { get; set; }
}

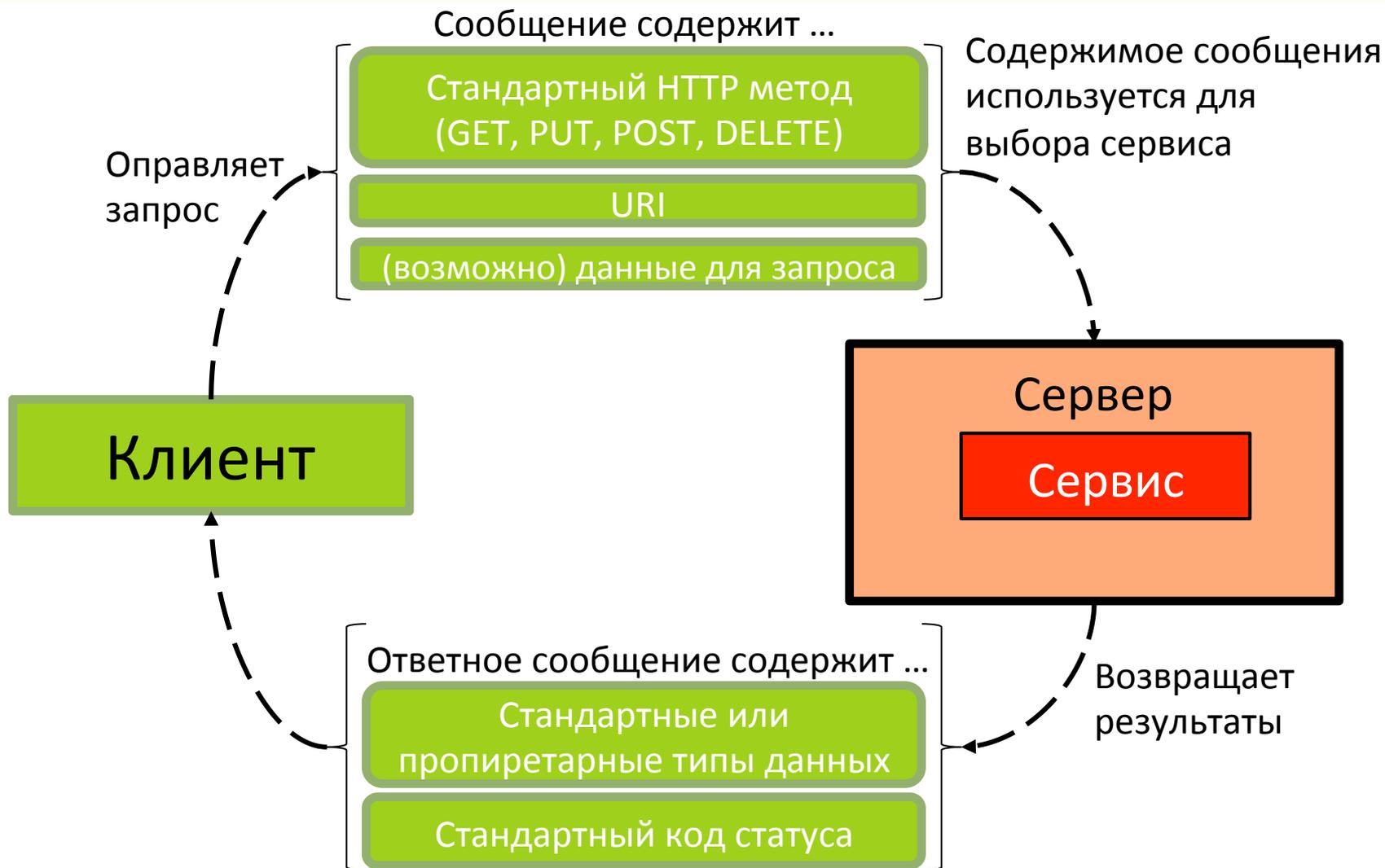
public class MessageService : IDropbox
{
    void IDropbox.Invoice (InvoiceMessage invoice) {
        // select procedure to process message here
    }
    void IDropbox.Contractor( ContractorMessage contractor) {
        // select procedure to process message here
    }
}
```

API РЕСУРСОВ

Стили API ВЕБ-СЕРВИСОВ: API РЕСУРСОВ

- ⊙ Если основа сервиса – это использование и модификация ресурсов, расположенных на удаленной системе, использование RPC может привести к разбуханию интерфейса (NewPerson, EditPerson, DeletePerson, GetPerson и др.)
- ⊙ **Решение** - необходимо использовать стандартные глаголы HTTP (GET, PUT, POST, DELETE) для работы с ресурсами удаленных систем.
- ⊙ Каждой процедуре, сущности предметной области, файлу назначается URI.
- ⊙ Клиент должен использовать стандартные команды HTTP с указанием соответствующих URI, а сервер выполнять данные команды и использовать стандартные ответы HTTP там, где это возможно.

Стили API ВЕБ-СЕРВИСОВ: API для СООБЩЕНИЙ



- ◎ **Representational State Transfer (REST)** это архитектурный стиль, определяющий процесс работы с ресурсами в сети Интернет посредством стандартных HTTP запросов.
- ◎ Был представлен в 2000-м году **Роем Филдингом** в его докторской диссертации.



- ◎ Многие веб-сервисы используют сообщения, чтобы сформировать свои собственные API. Они часто реализуют общие логические команды CRUD:
 - ◎ Create Read Update Delete
- ◎ Тем не менее, может привести к разрастанию интерфейсов, даже в относительно небольших проблемных областях.
- ◎ REST предоставляют возможность манипулировать удаленными ресурсами, но **избежать прямой зависимости от удаленных процедур**, и свести к минимуму необходимость предметно-ориентированных API.
- ◎ Протокол HTTP позволяет клиентам относительно легко использовать логику удаленных процедур, при этом изолируя их от базовых технологий. Вместо того чтобы создавать предметно-ориентированный API, можно использовать стандарты, определенные в спецификации HTTP.

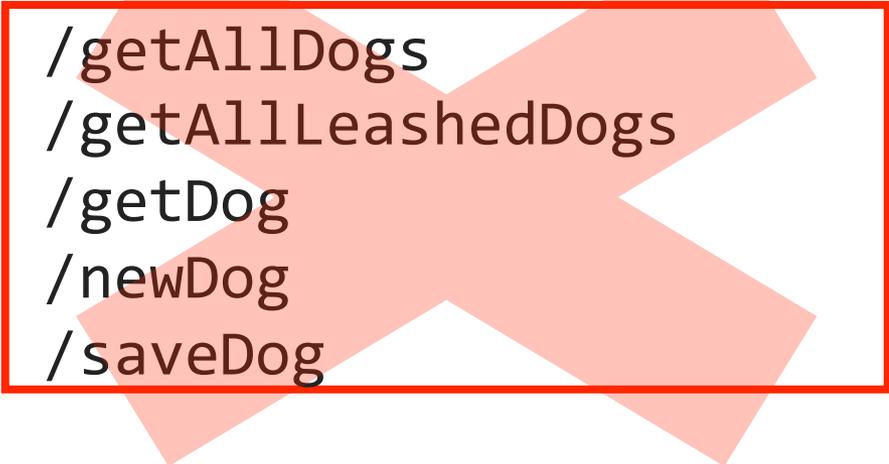
ВСЕ - ЭТО РЕСУРС

- ◎ Ресурс может быть текстовым файлом, медиа-файлом (например, изображения, видео, аудио), конкретная строка в таблице базы данных, набор связанных данных (например, продукты), логическая транзакция, очередь, загружаемое программа, бизнес-процесс (т.е. процедура) – практически все может быть ресурсом
 - ◎ `http://music.site/users/ivan99`
 - ◎ `http://music.site/albums/8`
- ◎ Коллекция ресурсов - также ресурс
 - ◎ `http://music.site/users`

- ◎ **Не зависимость от состояния**
 - ◎ Состояние клиента хранится только на клиенте
 - ◎ Вся информация, необходимая серверу для обработки запроса должна быть в запросе (самодокументированные сообщения)
- ◎ **Кэширования архитектура**
 - ◎ Ответ сервера может быть кэширован у клиента и использован повторно без новых обращений
- ◎ **Разделение клиент-сервер (слабосвязанность)**
 - ◎ Клиент знает все о интерфейс сервера, но ничего не знает о реализации сервера.

СОВЕТЫ ПО ИМЕНОВАНИЮ

- ⦿ Имена ресурсов должны быть существительными (желательно во множественном числе) а не глаголами.
- ⦿ Если вы ведете глагол в названии, скорее всего, это ошибка проектирования REST-интерфейса:



```
/getAllDogs  
/getAllLeashedDogs  
/getDog  
/newDog  
/saveDog
```

- ⦿ Вместо глаголов в названии ресурсов, необходимо использовать стандартные методы HTTP

API РЕСУРСОВ В REST

- ⊙ **PUT** используется для **создания** или **обновления** ресурсов.
- ⊙ **GET** используется для **получения** представления ресурса.
- ⊙ **DELETE** **удаляет** ресурс.
- ⊙ **POST** используется для **создания зависимого** ресурса.

GET

PUT

DELETE

POST

=

=

=

=

READ

UPDATE

DELETE

CREATE

СТАНДАРТИЗОВАННЫЙ ИНТЕРФЕЙС ЭЛЕМЕНТА

<http://example.com/resources/item17>

GET	Получить состояние элемента
PUT	Заменить этот элемент с другим элементом. Если такой элемент не существует, создать такой элемент
POST	Обычно не используется
DELETE	Удалить элемент

СТАНДАРТИЗОВАННЫЙ ИНТЕРФЕЙС КОЛЛЕКЦИИ

<http://example.com/resources>

GET

Перечислить URI и другую информацию о элементах в данной коллекции

PUT

Заменить эту коллекцию другой коллекцией

POST

Создать новый элемент в коллекции

DELETE

Удалить коллекцию

СТАНДАРТНЫЕ ДЕЙСТВИЯ REST

Правильный интерфейс REST

POST /albums – **добавить новый альбом**

GET /albums/2 – **получить альбом 2**

PUT /albums/2 – **обновить альбом 2**

DELETE /albums/2 – **удалить альбом 2**

НЕ ПРАВИЛЬНЫЙ REST ИНТЕРФЕЙС

21

POST /albums/**create**

GET /albums/**show**/2

POST /albums/**update**/2

GET /**delete**/albums/2

DELETE /albums/3/**remove**

HTTP: ОТВЕТЫ СЕРВЕРА

- ⦿ REST позволяют использовать стандартные типы данных и коды состояния.
- ⦿ Ответы сервера - это HTTP-коды, определяющие статус операции
 - ⦿ 200 – OK (“Вот ваша одежда”)
 - ⦿ 201 – Created (“Не потеряйте бирку”)
 - ⦿ 400 – Bad request (“Бирка сломана”)
 - ⦿ 403 – Forbidden (“Вам запрещено это делать”)
 - ⦿ 404 – Not found (“На вешалке с такой биркой ничего нет”)
 - ⦿ 500 – Server error (“У нас обед”)

ВЕРСИИ REST-API

ВЕРСИИ API

- ⦿ Когда вы делаете обновления, вы практически всегда вносите изменения во внутренние структуры данных и в модели.
- ⦿ Версионирование включает в себя изменения в структуре ресурсов (например, добавление или удаление столбцов в базе данных).
- ⦿ Если бы мы жили в идеальном мире, все клиенты автоматически бы обновлялись до новых версий протокола. Но это не так в реальном мире. Приходится одновременно поддерживать несколько версий.

РАЗДЕЛЕНИЕ С ИСПОЛЬЗОВАНИЕМ URL

- ◎ api.example.com/v1/feeds будет использоваться версией 1 приложения
- ◎ api.example.com/v2/feeds будет использоваться версией 2.
- ◎ Несмотря на то, что звучит это все неплохо, это приводит к чрезмерному количеству URL для каждого изменения в формате возвращаемых данных.
- ◎ Такой подход рекомендуется для использования только в случае глобальных изменений в API.

РАЗДЕЛЕНИЕ С ИСПОЛЬЗОВАНИЕМ URL

- ◎ api.example.com/v1/feeds будет использоваться версией 1 приложения
- ◎ api.example.com/v2/feeds будет использоваться версией 2.
- ◎ Несмотря на то, что звучит это все неплохо, это приводит к чрезмерному количеству URL для каждого изменения в формате возвращаемых данных.
- ◎ Такой подход рекомендуется для использования только в случае глобальных изменений в API.

РАЗДЕЛЕНИЕ НА ОСНОВЕ МОДЕЛИ (СТРУКТУРЫ ДАННЫХ)

- ⦿ При каждом изменении структуры данных необходимо изменять версию протокола.
- ⦿ Идентификатор протокола можно указать в поле “UserAgent” в поле HTTP-запроса:

```
Request
/login
Headers
Authorization: Token XXXXX
User-Agent: MyGreatApp /1.0
Accept: application/json
Accept-Encoding: compress, gzip
Parameters Encoding type - application/x-www-form-urlencoded
token - "Facebook Auth Token" (mandatory) profileInfo = "json string
containing public profile information from Facebook" (optional)
```

РАЗДЕЛЕНИЕ НА ОСНОВЕ МОДЕЛИ (СТРУКТУРЫ ДАННЫХ)

- ⊙ В этом случае, логика создания объекта-обработчика в зависимости от протокола может быть реализована на основе паттерна «Фабричный метод»:

```
Feed myFeedObject = Feed.createFeedObject("1.0");  
myFeedObject.populateWithDBObject(feedDaoObject);
```

- ⊙ Внесение изменений не будет нарушать имеющиеся соглашения. Просто создавайте новые структуры данных (модели), вносите изменения в «Фабричный метод» для создания экземпляра новой модели для новой версии и сразу несколько версий приложения могут работать одновременно с одним сервером.

ПРИМЕРЫ REST API

TWITTER REST API v1.1

GET `statuses/retweets/:id`

Вернет до 100 ретвитов твита с номером `id`

GET `statuses/show/:id`

Вернет отдельный твит «`id`»

GET `statuses/destroy/:id`

Удалить твит «`id`»

GET `statuses/update`

Обновить статус (создать новый твит)

GOOGLE TRANSLATE REST API

31

IN:

```
GET https://www.googleapis.com/language/translate/v2?  
key=INSERT-YOUR-KEY&source=en&target=de&q>Hello%20world
```

OUT :

```
200 OK
```

```
{  
  "data": {  
    "translations": [  
      {  
        "translatedText": "Hallo Welt"  
      }  
    ]  
  }  
}
```

PAYPAL REST API

IN: `https://api.paypal.com/v1/payments/payment`

```
curl -v https://api.sandbox.paypal.com/v1/payments/payment \-H "Content-Type:application/json" \-H "Authorization:Bearer EMxltHE7Zl4cMdkvMg-f7c63GQgYZU8FjyPWKQlpsqQP" \-d '{ "intent":"sale", "payer": { "payment_method":"credit_card", "funding_instruments":[{" credit_card": { "number":"4417119669820331", "type":"visa", "expire_month":11, "expire_year":2018, "cvv2":"874", "first_name":"Joe", "last_name":"Shopper", "billing_address": {"line1":"52 N Main ST", "city":"Johnstown", "country_code":"US", "postal_code":"43210", "state":"OH" } } ] } }, "transactions":[ { "amount":{"total":"7.47", "currency":"USD", "details": { "subtotal":"7.41", "tax":"0.03", "shipping":"0.03" } }, "description":"This is the payment transaction description." } ] }'
```

PAYPAL REST API

OUT:

200 OK

```
{ "id": "PAY-17S8410768582940NKEE66EQ", "create_time":  
"2013-01-31T04:12:02Z", "update_time":  
"2013-01-31T04:12:04Z", "state": "approved", "intent":  
"sale", "payer": {
```

...

РАЗРАБОТКА RESTFUL-СЕРВИСА

РАЗРАБОТКА СЕРВЕРА

- ◎ Ruby on Rails
 - ◎ Много магии
- ◎ Java – JAX-RS
 - ◎ Очень популярный язык
 - ◎ Очень популярная платформа для сервисов
- ◎ Python – Django
 - ◎ Попробуете на лабораторной.

JAVA SERVICE

```
@Path("/stores")
public class StoreService {

    @GET
    @Produces("application/xml")
    public JAXBElement <Stores> getStoresAsXML()    {
        Stores stores = Stores.getStores();
        return new JAXBElement <Stores>
            ( new QName("Stores"), Stores.class, stores);
    }

    @Path("/{id}")
    @GET
    @Produces("application/xml")
    public Store getStoreAsXML(@ PathParam("id") String id) {
        // implementation here
    }
}
```

JAVA SERVICE

```
@POST
@Consumes("application/xml")
@Produces("application/xml")
public Store createStore(JAXBElement <Store> store)    {
    // implementation here
}

@Path("/{id}")
@PUT
@Produces("application/xml")
public Store updateStore(@PathParam("id") String id)    {
    // implementation here
} }
```

JAVA CLIENT

Для создания клиента можно использовать пакет **Jersey**. Jersey предоставляет REST-клиент и REST-сервер.

```
public class Test {
public static void main(String[] args) throws ClientProtocolException,
IOException {
    Client client = Client.create();
    WebResource r = client.resource("http://localhost:8080/xyz");
    MultivaluedMap<String, String> params = new MultivaluedMapImpl();
    params.add("foo", "x");
    params.add("bar", "y");
    // getting XML data: http://localhost:8080/xyz/abc?foo=x&bar=y
    System.out.println(r.path("abc").
queryParams(params).accept(MediaType.APPLICATION_XML).get(String.class));
    // getting JSON data: http://localhost:8080/xyz/abc?foo=x&bar=y
    System.out.println(r.path("abc").
queryParams(params).accept(MediaType.APPLICATION_JSON).get(String.class));
    }
}
```