Representational state transfer (REST)

Подготовил: Радченко В.И., студент гр. ВМИ-456, ЮУрГУ

REST

Рой Филдинг (род. 1965)

- HTTP
- Apache
- «Architectural Styles and the Design of Network-based Software Architectures»

REST

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем.

REST is a set of principles that define how Web standards, such as HTTP and URIs, are supposed to be used.

REST – это не протокол и не стандарт, а архитектурный стиль.

Ключевые принципы REST
(Representational state transfer)

- Разделены ответственности клиента и сервера (client-server model)
- Работа сервиса не зависит от текущего состояния клиента (statelessness)
- Для работы с ресурсами используются общепринятые технологии и методики (standardized interface)

Ключевые принципы взаимодействия в REST

- Источники данных называются ресурсами. Каждый ресурс обладает уникальным идентификатором (global identifier)
- Клиент при обращении получает представление данных (representation)
- Информация, содержащаяся в запросе, должна быть достаточной для обработки самого запроса (self-descriptive messages)

Второстепенные принципы REST

- Данные могут быть кэшированы (cacheable data)
- Многослойность серверной стороны должна быть прозрачной для пользователя (layered system)
- Выдаваемые данные могут быть представлены несколькими способами (multiple representations)
- На смежные данные должны быть предоставлены ссылки (linking things together)
- ...

Принципы REST: Standardized interface

- Для представления возвращаемой информации может использоваться XML, HTML, JSON…
- Запрос клиента должен содержать достаточно информации для обработки данного запроса
- Стандартный набор методов - CRUD

Принципы REST: Standardized interface

CRUD — `CREATE, READ, UPDATE, DELETE`

SQL: `INSERT, SELECT, UPDATE, DELETE`

HTTP: `POST, GET, PUT, DELETE`

Простота использования REST
Правильный вариант

`GET /book/` — получить список всех книг

`GET /book/3/` — получить книгу

`PUT /book/` — добавить книгу

`POST /book/3` - изменить книгу

`DELETE /book/3` - удалить книгу

Простота использования REST
Неправильный вариант

```
GET /book/3          GET /book/3/show

PUT /book            PUT /book/create

POST /book/3         POST /book/3/alter

DELETE /book/3       DELETE /book/3/remove
```

Принципы REST: Standardized client interface

`http://example.com/resources/item17`

GET	**Получить (Retrieve)** *состояние представления* заданного элемента
PUT	**Заменить (Replace)** заданный элемент другим. Если такого элемента не существует, **создать (Create)** его
POST	Обычно не используется
DELETE	**Удалить (Delete)** указанный элемент

Принципы REST: Standardized client interface

`http://example.com/resources`

GET	**Перечислить (List)** URI и, возможно, некоторые другие сведения о содержиом коллекции
PUT	**Заменить (Replace)** коллекцию другой коллекцией
POST	**Создать (Create)** новую запись в коллекцию
DELETE	**Удалить (Delete)** коллекцию.

Принципы REST:
Standardized server interface

`http://example.com/гардероб/куртка_107`

GET | 200 - OK («Вот ваша куртка»)
PUT | 201 - Created («Смотрите не потеряйте бирку»)

GET | 400 - Bad request («Вы сломали бирку»)
GET | 403 - Forbidden («Это не ваша куртка»)
GET | 404 - Not Found («Куртка не найдена»)
GET | 500 - Internal server error («У нас перерыв»)
GET | 502 - Bad gateway («Вам в другое окошко»)

Примеры REST API
Google Translate

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"
            }
        ]
    }
}
```

Примеры REST API PayPal

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 EMxItHE7Zl4cMdkvMg-
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."
    }
  ]
}'
```

Примеры REST API PayPal

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": {
    "payment_method": "credit_card",
    "funding_instruments": [
      {
        "credit_card": {
          "type": "visa",
          "number": "xxxxxxxxxxxx0331",
          "expire_month": "11",
          "expire_year": "2018",
          "first_name": "Joe",
          "last_name": "Shopper",
          "billing_address": {
            "line1": "52 N Main ST",
            "city": "Johnstown",
            "state": "OH",
            "postal_code": "43210",
            "country_code": "US"
          }
        }
      }
    ]
  },
```

```
"transactions": [
    {
      "amount": {
        "total": "7.47",
        "currency": "USD",
        "details": {
          "tax": "0.03",
          "shipping": "0.03"
        }
      },
      "description": "This is the payment
transaction description.",
      "related_resources":

…

…

…

…

…
```

REST vs SOAP

REST

- Архитектурный стиль
- XML, JSON, HTML, JPG, MP3…
- HTTP - база всей архитектуры
- Ключевое понятие - **ресурс**

SOAP

- Семейство протоколов
- XML. Точка.
- HTTP - транспортный протокол
- Ключевое понятие - **операция (RPC)**

Università della Svizzera italiana

Facoltà di scienze informatiche

Freedom of Choice
Freedom from Choice

Architectural Decision and AAs	REST	WS-*
Integration Style	1 AA	2 AAs
Shared Database		
File Transfer		
Remote Procedure Call	✓	✓
Messaging		✓
Contract Design	1 AA	2 AAs
Contract-first		✓
Contract-last		✓
Contract-less	✓	
Resource Identification	1 AA	n/a
Do-it-yourself	✓	
URI Design	2 AA	n/a
"Nice" URI scheme	✓	
No URI scheme	✓	
Resource Interaction Semantics	2 AAs	n/a
Lo-REST (`POST`, `GET` only)	✓	
Hi-REST (4 verbs)	✓	
Resource Relationships	1 AA	n/a
Do-it-yourself	✓	
Data Representation/Modeling	1 AA	1 AA
XML Schema	(✓)[a]	✓
Do-it-yourself	✓	
Message Exchange Patterns	1 AA	2 AAs
Request-Response	✓	✓
One-Way		✓
Service Operations Enumeration	n/a	≥3 AAs
By functional domain		✓
By non-functional properties and QoS		✓
By organizational criterion (versioning)		✓
Total Number of Decisions, AAs	**8**, 10	**5**, ≥10

[a] Optional

Table 2: Conceptual Comparison Summary

Architectural Decision and AAs	REST	WS-*
Transport Protocol	1 AA	≥7 AAs
HTTP	✓	✓[a]
waka [13]	(✓)[b]	
TCP		✓
SMTP		✓
JMS		✓
MQ		✓
BEEP		✓
IIOP		✓
Payload Format	≥6 AAs	1 AA
XML (SOAP)	✓	✓
XML (POX)	✓	
XML (RSS)	✓	
JSON [10]	✓	
YAML	✓	
MIME	✓	
Service Identification	1 AA	2 AA
URI	✓	✓
WS-Addressing		✓
Service Description	3 AAs	2 AAs
Textual Documentation	✓	
XML Schema	(✓)[c]	✓
WSDL	✓[d]	✓
WADL [18]	✓	
Reliability	1 AA	4 AAs
HTTPR [38][e]	(✓)	(✓)
WS-Reliability		✓
WS-ReliableMessaging		✓
Native		✓
Do-it-yourself	✓	✓
Security	1 AA	2 AAs
HTTPS	✓	✓
WS-Security		✓

Transactions	1 AA	3 AAs
WS-AT, WS-BA		✓
WS-CAF		✓
Do-it-yourself	✓	✓
Service Composition	2 AAs	2 AAs
WS-BPEL		✓
Mashups	✓	
Do-it-yourself	✓	✓
Service Discovery	1 AAs	2 AAs
UDDI		✓
Do-it-yourself	✓	✓
Implementation Technology	many	many
...	✓	✓
Total Number of Decisions, AAs	**10**, ≥17	**10**, ≥25

[a] Limited to only the verb `POST`
[b] Still under development
[c] Optional
[d] WSDL 2.0
[e] Not standard

Table 3: Technology Comparison Summary

Architectural Principle and Aspects	REST	WS-*
Protocol Layering	yes	yes
HTTP as application-level protocol	✓	
HTTP as transport-level protocol		✓
Dealing with Heterogeneity	yes	yes
Browser Wars	✓	
Enterprise Computing Middleware		✓
Loose Coupling, aspects covered	yes, 2	yes, 3
Time/Availability		✓
Location (Dynamic Late Binding)	(✓)	✓
Service Evolution:		
Uniform Interface	✓	
XML Extensibility	✓	✓
Total Principles Supported	**3**	**3**

Table 1: Principles Comparison Summary

Достоинства REST

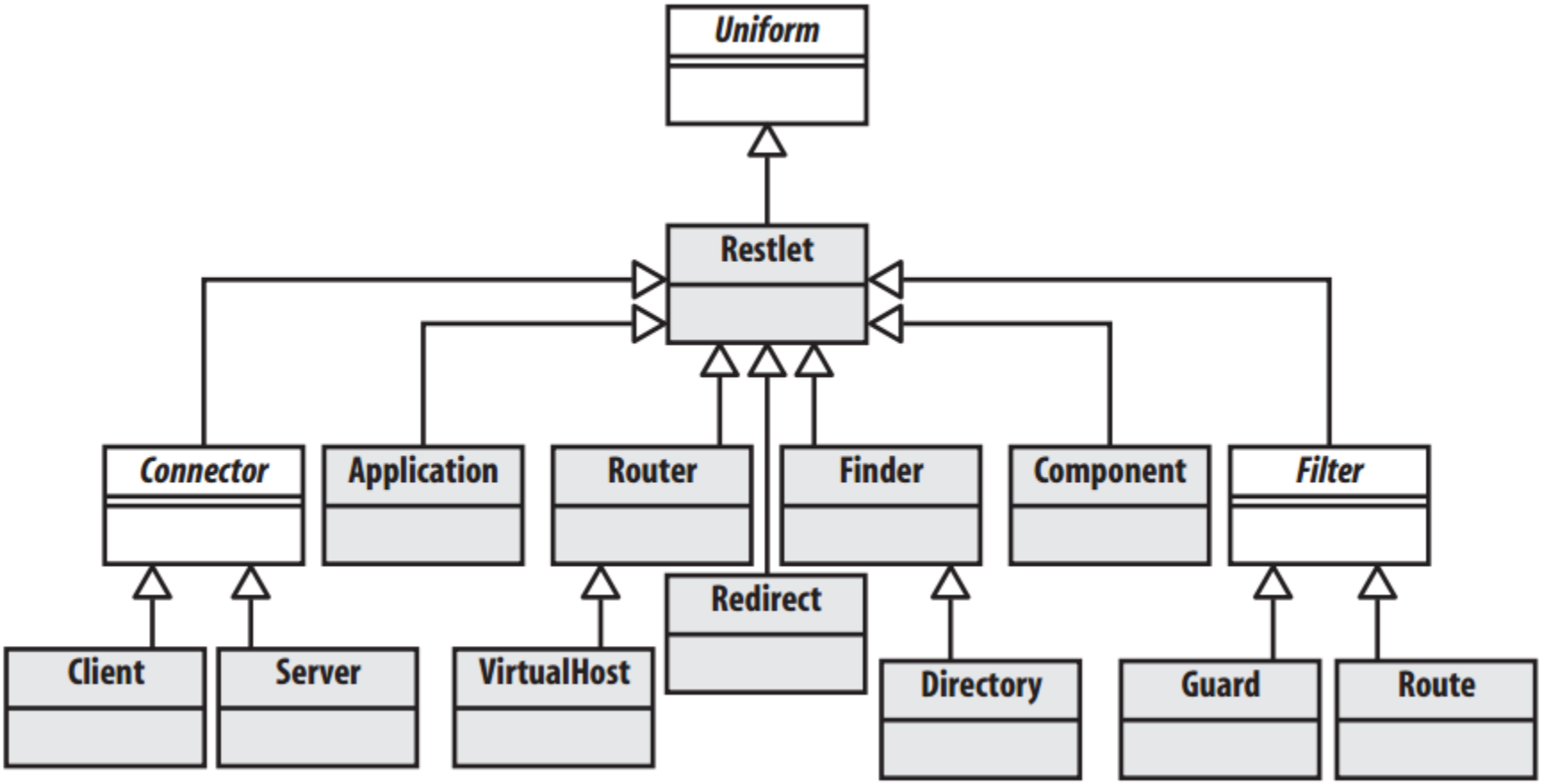
- Проще практически во всех отношениях
- Поддержка нескольких форматов данных
- Лучшая масштабируемость
- Кэширование можно легко встроить

Достоинства SOAP

- WS-Security
- WS-Automatic Transaction
- WS-ReliableMessaging
- WS-*

Инструменты для создания RESTful сервиса

- Ruby on Rails
- Java Restlet
- Python Django
- ...



Leonard Richardson, Sam Ruby. RESTful Web Services, p. 345

```
// YahooSearch.java
import org.restlet.Client;
import org.restlet.data.Protocol;
import org.restlet.data.Reference;
import org.restlet.data.Response;
import org.restlet.resource.DomRepresentation;
import org.w3c.dom.Node;

/**
 * Searching the web with Yahoo!'s web service using XML.
 */
public class YahooSearch {
    static final String BASE_URI =
    "http://api.search.yahoo.com/WebSearchService/V1/webSearch";

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("You need to pass a term to search");
        } else {
            // Fetch a resource: an XML document full of search results
            String term = Reference.encode(args[0]);
            String uri = BASE_URI + "?appid=restbook&query=" + term;
            Response response = new Client(Protocol.HTTP).get(uri);
            DomRepresentation document = response.getEntityAsDom();

            // Use XPath to find the interesting parts of the data structure
            String expr = "/ResultSet/Result/Title";
            for (Node node : document.getNodes(expr)) {
                System.out.println(node.getTextContent());
            }
        }
    }
}
```

```
from datetime import datetime
from django.db import models
from django.contrib.auth.models import User

class Tag(models.Model):
    name = models.SlugField(maxlength=100, primary_key=True)

class Bookmark(models.Model):
    user                = models.ForeignKey(User)
    url                 = models.URLField(db_index=True)
    short_description   = models.CharField(maxlength=255)
    long_description    = models.TextField(blank=True)
    timestamp           = models.DateTimeField(default=datetime.now)
    public              = models.BooleanField()
    tags                = models.ManyToManyField(Tag)
```

```
from django.conf.urls.defaults import *
from bookmarks.views import *

urlpatterns = patterns('',
    (r'^users/([\w-]+)/$',              bookmark_list),
    (r'^users/([\w-]+)/tags/$',         tag_list),
    (r'^users/([\w-]+)/tags/([\w-]+)/', tag_detail),
    (r'^users/([\w-]+)/(.*)',           BookmarkDetail()),
)
```

Leonard Richardson, Sam Ruby. RESTful Web Services, p. 355

```
from bookmarks.models import Bookmark
from django.contrib.auth.models import User
from django.core import serializers
from django.http import HttpResponse
from django.shortcuts import get_object_or_404

def bookmark_list(request, username):
    u = get_object_or_404(User, username=username)
    marks = Bookmark.objects.filter(user=u, public=True)
    json = serializers.serialize("json", marks)
    return HttpResponse(json, mimetype="application/json")
```

Leonard Richardson, Sam Ruby. RESTful Web Services, p. 358

```
class BookmarkDetail:

    def __call__(self, request, username, bookmark_url):
        self.request = request
        self.bookmark_url = bookmark_url

        # Look up the user and throw a 404 if it doesn't exist
        self.user = get_object_or_404(User, username=username)

        # Try to locate a handler method.
        try:
            callback = getattr(self, "do_%s" % request.method)
        except AttributeError:
            # This class doesn't implement this HTTP method, so return
            # a 405 ("Method Not Allowed") response and list the
        #allowed methods.
            allowed_methods = [m.lstrip("do_") for m in dir(self)
                                                 if m.startswith("do_")]
            return HttpResponseNotAllowed(allowed_methods)

        # Check and store HTTP basic authentication, even for methods that
        # don't require authorization.
        self.authenticate()

        # Call the looked-up method
        return callback()
```

Leonard Richardson, Sam Ruby. RESTful Web Services, p. 361

```
def do_GET(self):
    # Look up the bookmark (possibly throwing a 404)
    bookmark = get_object_or_404(Bookmark,
        user=self.user,
        url=self.bookmark_url
    )

    # Check privacy
    if bookmark.public == False and self.user != self.authenticated_user:
        return self.forbidden()

    json = serializers.serialize("json", [bookmark])
    return HttpResponse(json, mimetype="application/json")
```

Leonard Richardson, Sam Ruby. RESTful Web Services, p. 362

Безопасность в REST

- Аутентификация: токены (OAuth)
- Передаваемые данные: HTTPS
- Анонимность: луковая маршрутизация (помним о принципе прозрачности)

Источники

- https://en.wikipedia.org/wiki/Representational_state_transfer
- http://habrahabr.ru/post/131343/
- http://habrahabr.ru/post/134303/
- http://habrahabr.ru/post/38730/
- http://www.infoq.com/articles/rest-introduction
- http://spf13.com/post/soap-vs-rest
- http://developer.mindtouch.com/REST/REST_for_the_Rest_of_Us
- http://www.restlet.org
- http://www.djangobook.com
- Leonard Richardson, Sam Ruby. RESTful Web Services
- Subbu Allamaraju. RESTful Web Services Cookbook