RPC, REST, SOAP

XML-RPC

- 1998
- Microsoft :)

Типичный пример запроса XML-RPC:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Типичный пример ответа на запрос XML-RPC:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

+ :

- простота,
- краткость сообщений,
- минимальная проверка формата данных,

— :

- недостаточная строгость,
- требуется отдельное описание сервиса.

«…понравился толпе простых незамороченных программистов. Этим ребятам нужно было простое и надежное средство обмена информацией между системами, они не хотели заморачиваться на такую хрень как красивый УРЛ, схема документа и прочие академизмы. Первое, что они хотели получить — простую работающую систему. »

REST

- Рой Филдинг (2000г)
- В самом протоколе HTTP уже есть ряд методов работы с объектами:
  - GET (получить),
  - POST (отправить/создать),
  - PUT (обновить),
  - DELETE (удалить)

- REST - Representational State Transfer
- Это не протокол! Это стиль построения архитектуры приложения!

- Вместо

  POST/api/object.php?object_id=445&action=delete&user_id=255&auth_key=oJf4tet5 HTTP/1.1

- Так:

  DELETE /objects/445 HTTP/1.1

- Или

  POST /objects/445 HTTP/1.1

- (при этом delete передается параметром)

XML-RPC использует из HTTP-протокола только транспортную часть для передачи XML-ки запроса/ответа

REST задействует HTTP по-полной: авторизационные заголовки, content-negotiation — предпочтения по формату, языку, кодировке и виду ответа, различные служебные заголовки, простая передача бинарных данных и т.п. Ошибки хорошо описываются кодами HTTP 4xx и 5xx.

Тело сообщения:

- XML
- JSON

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress":
   "Московское ш., 101,
   кв.101",
    "city": "Ленинград",
    "postalCode": 101101
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
  ]
}
```

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101,
     кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-
     1234</phoneNumber>
    <phoneNumber>916 123-
     4567</phoneNumber>
  </phoneNumbers>
</person>
```

+ : гибкость, простота, скорость обработки (особенно важно для крупных сайтов), органичность протоколу, мультиформатность, компактность.
— : отсутствие строго контроля данных, из практических соображений приходится выходить за рамки идеальной модели.

SOAP

- SOAP — Simple Object Access Protocol
- Microsoft)
- 2003
- Наследник XML-RPC

Запрос:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
   <req:echo
xmlns:req="http://localhost:8080/axis2/services/MyService/">
     <req:category>classifieds</req:category>
   </req:echo>
  </soapenv:Body>
</soapenv:Envelope>
```

Ответ:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
            schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
  <wsa:ReplyTo>
   <wsa:Address>schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
  </wsa:ReplyTo>
  <wsa:From>
   <wsa:Address>localhost:8080/axis2/services/MyService</wsa:Address>
  </wsa:From>
  <wsa:MessageID>ECE5B3F187F29D28BC114339056662036</wsa:MessageID>
 </soapenv:Header>
 <soapenv:Body>
  <req:echo xmlns:req="http://localhost:8080/axis2/services/MyService/">
   <req:category>classifieds</req:category>
  </req:echo>
 </soapenv:Body>
</soapenv:Envelope>
```

JAX

- Java EE
- Java API for XML ….

- JAX-WS (Java API for XML Web Services)
- JAX-RS (Java API for RESTful Web Services)
- JAXB (Java Architecture for XML Binding)
- JAX-RPC (Java API for XML-based RPC)

JAX-RS

JAX-RS provides some annotations to aid in mapping a resource class as a web resource.

@Path specifies the relative path for a resource class or method.

@GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP request type of a resource.

@Produces specifies the response Internet media types (used for content negotiation).

@Consumes specifies the accepted request Internet media types.

Jersey

- Основные классы javax.ws.rs.*

```
package ru.spbau.antonk.jersey.first;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

//Sets the path to base URL + /hello
@Path("/hello")
public class Hello {

  // This method is called if TEXT_PLAIN is request
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello() {
    return "Hello Jersey";
  }

  // This method is called if XML is request
  @GET
  @Produces(MediaType.TEXT_XML)
  public String sayXMLHello() {
    return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
  }

  // This method is called if HTML is request
  @GET
  @Produces(MediaType.TEXT_HTML)
  public String sayHtmlHello() {
    return "<html> " + "<title>" + "Hello Jersey" + "</title>"
        + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
  }

}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>de.vogella.jersey.first</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>ru.spbau.antonk.jersey.first</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Задание

- Создать RESTful сервис, который предоставляет доступ к неким данным (курс валют, прогноз погоды, результаты матчей…)
- Должны поддерживаться «задания». Описание некого набора параметров, по которым выдается результат (например, для результатов матчей – название Команды и период). Задание создается методом PUT. POST – для обновления. DELETE – для удаления. У каждого задания есть id.
- При этом должны поддерживаться разные типы клиентов (различия по типу содержимого) – html, JSON, txt, XML. При запросе из браузера должны должны отображаться таблицы, картинки.
- В базе данных должна сохраняться статистика (для выдачи средних значений) и история запросов.

Что почитать

http://www.vogella.com/articles/REST/article.html