Web Services

Prof. Razer A N R Montaño

2017

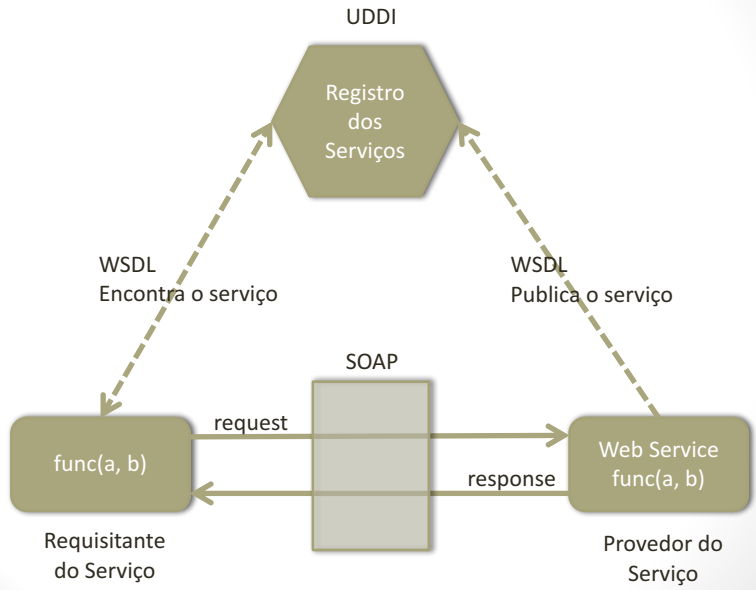
Conceitos Web Services

- Web Services:
  - Um serviço que pode ser acessado diretamente por outro sistema de qualquer plataforma, que esteja na rede/internet
- Arquitetura
  - Cada plataforma tem suas ferramentas
  - W3C oferece alguns padrões
  - Java API for XML-Based Web Services – JAX-WS
    - Padrão W3C (http://www.w3.org/TR/ws-arch/)
    - Depende de Java Architecture for XML Binding – JAXB
  - Java API for RESTful Web Services – JAX-RS
    - Representational State Transfer (REST)
    - Em geral mais fáceis de implementar e de evoluir
    - Usa JAXB para produzir XML ou JSON

Web Services

SOAP WEB SERVICES

Conceitos



Conceitos

- Passo
  1. Fornecedor de serviço contacta o UDDI, registrando e informando os serviços e onde encontrá-los
  2. Consumidor de um serviço utiliza métodos de busca oferecidos pelo UDDI para encontrar as informações que deseja sobre entidades e serviços
  3. Consumidor de um serviço se comunica diretamente com o fornecedor do serviço
- Toda interação é feita através de SOAP

Web Services Prof. Razer A N R Montaño

5

Conceitos

- Basicamente XML sobre HTTP
- Podem ser descobertos (UDDI)
- Elementos
  - SOAP – Simple Object Access Protocol
  - UDDI – Universal Description, Discovery and Integration
  - WSDL – Web Service Description Language
- W3C: http://www.w3.org/2002/ws/

Web Services Prof. Razer A N R Montaño

6

Conceitos

- SOAP
  - Simple Object Access Protocol
  - Protocolo de Comunicação
  - Formato para envio de mensagens
  - Comunicação sobre a internet
  - Independente de Plataforma
  - Independente de Linguagem
  - Baseado em XML
  - Simples e extensível
  - Padrão W3C

Web Services Prof. Razer A N R Montaño

7

SOAP - Pedido

```
POST /ServidorWS/TesteWS/endpoint HTTP/1.1
Host: localhost
Content-Type: text/xml; charset="utf-8"
Content-Length: 322
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:mostrar xmlns:ns2="http://ws.com/">
            <str>oi</str>
        </ns2:mostrar>
    </S:Body>
</S:Envelope>
```

Web Services Prof. Razer A N R Montaño

8

SOAP - Resposta

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 367

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:mostrarResponse xmlns:ns2="http://ws.com/">
            <return>Teste: oi</return>
        </ns2:mostrarResponse>
    </S:Body>
</S:Envelope>
```


Conceitos

- WSDL
  - Web Services Description Language
  - Baseado em XML
  - Usado para descrever WebServices
  - Usado para localizar WebServices
  - É um Padrão W3C

WSDL Exemplo

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://ws.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ws.com/"
name="TesteWS">
<types>
   <xsd:schema>
      <xsd:import namespace="http://ws.com/"
schemaLocation="http://localhost:8080/ServidorWS/TesteWS?xsd=1"/>
   </xsd:schema>
</types>
<message name="hello">
   <part name="parameters" element="tns:hello"/>
</message>
<message name="helloResponse">
   <part name="parameters" element="tns:helloResponse"/>
</message>
<message name="mostrar">
   <part name="parameters" element="tns:mostrar"/>
</message>
<message name="mostrarResponse">
   <part name="parameters" element="tns:mostrarResponse"/>
</message>
```

Web Services Prof. Razer A N R Montaño

11

WSDL - Exemplo

```
<portType name="TesteWS">
   <operation name="hello">
      <input wsam:Action="http://ws.com/TesteWS/helloRequest"
message="tns:hello"/>
      <output wsam:Action="http://ws.com/TesteWS/helloResponse"
message="tns:helloResponse"/>
   </operation>
   <operation name="mostrar">
      <input wsam:Action="http://ws.com/TesteWS/mostrarRequest"
message="tns:mostrar"/>
      <output wsam:Action="http://ws.com/TesteWS/mostrarResponse"
message="tns:mostrarResponse"/>
   </operation>
</portType>
```

Web Services Prof. Razer A N R Montaño

12

WSDL - Exemplo

```
<binding name="TesteWSPortBinding" type="tns:TesteWS">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
   <operation name="hello">
      <soap:operation soapAction=""/>
      <input>
          <soap:body use="literal"/>
      </input>
      <output>
          <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="mostrar">
       <soap:operation soapAction=""/>
       <input>
          <soap:body use="literal"/>
       </input>
       <output>
          <soap:body use="literal"/>
       </output>
    </operation>
</binding>
```

Web Services Prof. Razer A N R Montaño

13

WSDL - Exemplo

```
   <service name="TesteWS">
      <port name="TesteWSPort" binding="tns:TesteWSPortBinding">
          <soap:address
location="http://localhost:8080/ServidorWS/TesteWS"/>
      </port>
   </service>
</definitions>
```

Web Services Prof. Razer A N R Montaño

14

Conceitos

- UDDI
  - Universal Description, Discovery and Integration
  - Entidades que disponibilizam WS (organizações, empresas)
  - Um local para armazenar informações sobre Web Services
    - Diretório de busca
    - Define mecanismos para publicar e descobrir Web Services
    - Local onde clientes requisitantes encontram o serviço
  - Usado com Web Services baseados em WSDL
  - Comunicação via SOAP, onde *requests* e *responses* são objetos UDDI enviados via SOAP

Web Services Prof. Razer A N R Montaño

15

Formatos Consumo – XML / JSON

- XML

```
<pessoa>
    <nome>Razer Montano</nome>
    <descricao>Professor</descricao>
</pessoa>
```

- JSON – JavaScript Object Notation

```
{"nome": "Razer Montano", "descricao": "Professor"}
```

Web Services Prof. Razer A N R Montaño

16

JAXB

- Java Architecture for XML Binding
- Mapeamento Java <-> XML , Java <-> JSON
- Transforma objetos Java em texto XML/JSON e vice-versa
- Netbeans já cria códigos específicos

```
@XmlRootElement
public class Conta {
   private double saldo;
   private double limite;
   private Cliente cliente;
}
```


Web Services

RESTFUL WEB SERVICES

Introdução

- REST: Representational State Transfer
  - Tese de Doutorado de Roy Fielding
  - `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`
  - Meados de 2000
  - Estilo Arquitetural
- Interface web simples, sem abstrações de protocolos ou padrões de trocas de mensagens
- REST x RESTful
  - REST : Paradigma arquitetônico que explora a tecnologia existente e protocolos Web
  - RESTful : Um serviço Web que utiliza o paradigma REST

Web Services Prof. Razer A N R Montaño

19

Introdução

- Usar os mecanismos do HTTP
  - Identificação através de URIs
  - Métodos de acesso a recursos
    - GET, PUT, DELETE idempotentes
    - POST não idempotente
    - Transações
  - Códigos de retorno: 404, 201, 500, etc
- Acessível através de qualquer linguagem de programação
- Autenticação, criptografia, autorização: usar os recursos do HTTP
- API Java para manilupar este tipo de Serviço: **API for RESTful Web Services JAX-RS**

Web Services Prof. Razer A N R Montaño

20

Introdução - Recurso

- Toda informação disponível é um recurso : **Resource**
  - Uma imagem
  - Uma pessoa
  - Um documento
- Cada recurso deve ter um identificador único, para poder ser acessado
  - URI – *Universal Resource Identifier*
  - Exemplo: **`http://www.razer.net.br/clientes`**

Introdução - Media Types

- Recursos podem ser representados em vários formatos:
  - *Media Types*
  - **HTML**

```
<html><head><title>Razer A N R Montano</title><head>
<body>
<h1>Razer A N R Montano</h1>
<p>Professor de Java</p>
</body></html>
```

  - **XML**

```
<pessoa>
  <nome>Razer A N R Montano</nome>
  <descricao>Professor de Java</descricao>
</pessoa>
```

  - **JSON**

```
{"nome": "Razer A N R Montano", "descricao": "Professor de Java"}
```

Introdução - Métodos

- Recursos são manipulados por métodos do protocolo HTTP
- Recurso cliente:

`GET www.razer.net.br/clientes`	Retorna todos os clientes
`POST www.razer.net.br/clientes`	Insere um cliente

Introdução - Retorno

- As requisições em HTTP dão um código de retorno, usada em RESTful
- Códigos HTTP
  - 404 – não encontrado
  - 500 – Erro desconhecido do servidor
  - 201 - Criado

RESTful web services

PROTOCOLO HTTP

HTTP: Protocolo

- HTTP
  - HyperText Transfer Protocol
  - Desde 1990 – Tim Bernard-Lee : primeiro cliente/servidor
  - HTTP/0.9
  - 1996 : RFC-1945 HTTP/1.0
  - 1999 : RFC-2616 HTTP/1.1
  - Protocolo da Camada de Aplicação (Modelo OSI)
  - Baseado em Requisição e Resposta
  - Não mantém o estado (dados) entre requisições
  - Possui códigos de retorno

HTTP: Requisição / Resposta

- Requisição

```
GET /sistema/teste.html HTTP/1.1
Host: www.empresa.com.br
```

- Resposta

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 71

<html><head><title>Teste</title></head>
<body>
Oi Mundo
</body>
</html>
```

Web Services Prof. Razer A N R Montaño

27

HTTP: Métodos

- **GET**: Requisita um representação do recurso especificado (HTML, XML ou JSON).
- **HEAD**: Retorna os cabeçalhos de uma resposta (sem o corpo contendo o recurso)
- **POST**: Envia uma entidade e requisita que o servidor aceita-a como subordinada do recurso identificado pela URI
- **PUT**: Requisita que um entidade seja armazenada embaixo da URI fornecida. Se a URI se refere a um recurso que já existe, ele é modificado; se a URI não aponta para um recurso existente, então o servidor pode criar o recurso com essa URI.
- **DELETE**: Apaga o recurso especificado.
- **TRACE**: Ecoa de volta a requisição recebida para que o cliente veja se houveram mudanças e adições feitas por servidores intermediários
- **OPTIONS**: Retorna os métodos HTTP que o servidor suporta para a URL especificada
- **CONNECT**: Converte a requisição de conexão para um túnel TCP/IP transparente, usualmente para facilitar comunicação criptografada com SSL (HTTPS) através de um proxy HTTP não criptografado.
- **PATCH**: Usado para aplicar modificações parciais a um recurso.

Web Services Prof. Razer A N R Montaño

28

HTTP: Métodos

- Características
  - **Idempotência**: Um método é **Idempotente** se a mesma requisição feita diversas vezes sempre resulta no mesmo efeito
  - **Segurança**: Um método é **Seguro** se não provoca alterações nos dados contidos

	Idempotente	Seguro
GET	X	X
POST		
PUT	X	
DELETE	X	
HEAD	X	X
OPTIONS	X	X

HTTP: Passagem de Parâmetros

- Os métodos suportam dois tipos de passagem de parâmetros
  - *Query Parameters*
  - *Body Parameters*
- ***Query Parameters*** : Parâmetros são passados na própria URL da requisição

  http://www.empresa.com.br/clientes.html?id=10&nome=Maria

```
GET clientes.html?id=10&nome=Maria HTTP/1.1
Host: www.empresa.com.br
```

  - Espaços são transformados para + ou %20
  - Caracteres especiais são codificados como Ascii / UTF-8 em hexadecimal

HTTP: Passagem de Parâmetros

- ***Body Parameters*** : Parâmetros são passados no corpo da requisição

```
POST clientes.html HTTP/1.1
Host: www.empresa.com.br
Content-Length: 16

id=10&nome=Maria
```


HTTP: Cabeçalhos

- Contém meta-informações sobre a requisição/resposta. Não são obrigatórios e podem ser criados outros
  - **Host** : mostra qual foi o DNS utilizado para chegar a este servidor
  - **User-Agent** : fornece informações sobre o meio utilizado para acessar este
  - endereço
  - **Accept** : realiza negociação com o servidor a respeito do conteúdo aceito
  - **Accept-Language** : negocia com o servidor qual o idioma a ser utilizado na resposta
  - **Accept-Encoding** : negocia com o servidor qual a codificação a ser utilizada na resposta
  - **Connection** : ajusta o tipo de conexão com o servidor (persistente ou não)

HTTP: *Media Types*

- Indicam qual é o tipo de informação que está sendo trafegada
  - Divididos em Tipos e Subtibos (tipo/subtipo)
- Tipos mais comuns
  - **`application`** : tráfego de dados específicos de aplicações
  - **`audio`**
  - **`image`**
  - **`text`**
  - **`video`**
  - **`vnd`** : tráfego de softwares específicos (ex. MS Word)
- Em REST os tipos/subtipos mais comuns são
  - **`application/xml`**
  - **`application/json`**

Web Services Prof. Razer A N R Montaño

33

HTTP: *Media Types*

- **Requisição**: Header **`Accept`** informa o tipo dos dados que deseja receber

```
GET /foto/1 HTTP/1.1
Host: www.empresa.com.br
Accept: image/*
```

- **Resposta**: Header **`Content-Type`** informa o tipo de dado que está sendo enviado

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Content-Length: 10248
```

Web Services Prof. Razer A N R Montaño

34

HTTP: Código de Status

- Para toda requisição feita, é retornado um código de status
  - 1xx – Informacionais
  - 2xx – Códigos de sucesso
  - 3xx – Códigos de redirecionamento
  - 4xx – Códigos causados pelo cliente
  - 5xx – Erros originados no servidor

HTTP: Código de Status

200	OK	**401**	Unauthorized
201	Created	**403**	Forbidden
202	Accepted	**404**	Not Found
204	No Content	**405**	Method Not Allowed
206	Partial Content	**409**	Conflict
301	Moved Permanently	**410**	Gone
303	See other	**412**	Precondition Failed
304	Not Modified	**415**	Unsupported Media Type
307	Temporary Redirect	**500**	Internal Server Error
400	Bad Request	**503**	Server Unavailable

HTTP: Código de Status

- Deve-se utilizar corretamente o código de Status
  - Nem sempre se retorna 200 (OK)
  - Muitas vezes deve-se retornar 201 (Created) e no header `Location` a URL do recurso criado
  - Em caso de conflitos (ex, criação de recursos com chaves duplicadas) deve-se retornar 409 (Conflict), e no header Location – se possível – retornar a URL do recurso que originou o conflito

RESTful web services

RESTFUL

Recursos

- Todo serviço REST é baseado em recursos
  - Entidades bem definidas, que possuem URI
  - Exemplo
    - Para a aplicação : **`empresa.com.br`**
    - Um recurso para retornar todos os clientes pode ser:

    `empresa.com.br/clientes`

- Identificadores
  - Se for necessário interagir com um recurso específico, usa-se identificadores
  - Exemplo:
    - Cliente código 10
    - Para buscar o cliente de código 10, usa-se:

    `empresa.com.br/clientes/10`

Web Services Prof. Razer A N R Montaño

39

Métodos do HTTP

- Para interagir com os recursos, usa-se métodos do HTTP:
  - GET : recupera os dados identificados na URL
  - POST : cria um novo recurso
  - PUT : atualiza um recurso
  - DELETE : apaga um recurso
- Assim, o CRUD ficaria:
  - Obter todos os clientes: **`GET empresa.com.br/clientes`**
  - Obter um cliente: **`GET empresa.com.br/clientes/10`**
  - Criar um novo cliente: **`POST empresa.com.br/clientes`**
  - Atualizar um cliente: **`PUT empresa.com.br/clientes/10`**
  - Remover um cliente: **`DELETE empresa.com.br/clientes/10`**

Web Services Prof. Razer A N R Montaño

40

RESTFUL

TIPOS DE DADOS

Tipos de Dados

- Formato no qual os dados são transferidos de/para o WS
- XML
  - API JAXB
- JSON
  - API JAXB
  - JSON-LIB
  - Jackson

XML

- XML : Extensible Markup Language
- Baseada em tags – podem representar qualquer informação
- Exemplo

```
<aluno>
   <nome>Razer</nome>
   <idade>18</idade>
</aluno>
```

- Podem ser padronizadas : XML Schemas (arquivo **`.xsd`**)
- API para trabalhar em Java é o JAXB
  - Anotações que indicam como converter um XML em Objeto
  - De um esquema, consegue-se gerar todas as classes que representam os dados
    - Usa-se o aplicativo **`xjc`** do Java

XML – Exemplo de Classes

```
@XmlRootElement
public class PessoaFisica extends Pessoa {
}

@XmlSeeAlso({
    PessoaFisica.class
})
public abstract class Pessoa {
        private String nome;
        private List<Endereco> endereco;
        private Long id;

        @XmlAttribute(name = "id")
        public Long getId() {
                return id;
        }
   // getters e setters
}

public class Endereco {
   private String cep;
   private String logradouro;
}
```

XML

- Exemplo de conversão de Classe para XML

```
PessoaFisica pessoaFisica = new PessoaFisica();
pessoaFisica.setCpf("12345678909");
pessoaFisica.setNome("Razer");

Endereco endereco = new Endereco();
endereco.setCep("12345-678");

pessoaFisica.getEndereco().add(endereco);

JAXB.marshal(pessoaFisica, System.out);
```


XML

- Resultado da conversão de Classe para XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pessoaFisica xmlns:ns2="http://exemplo.com.br/endereco/v1"
   xmlns:ns3="http://exemplo.com.br/pessoa/v1">
   <ns3:nome>Razer</ns3:nome>
   <ns3:endereco>
      <ns2:cep>12345-678</ns2:cep>
   </ns3:endereco>
   <ns3:cpf>12345678909</ns3:cpf>
</pessoaFisica>
```

JSON

- JSON: *JavaScript Object Notation*
  `http://www.json.org`
- Formato para troca de dados
- Tamanho reduzido em relação ao XML
- É texto e não necessida JavaScript
- RFC-4627
- Validador de JSON:
  `http://codebeautify.org/jsonvalidate`
- Media Type : **`application/json`**

Web Services Prof. Razer A N R Montaño

47

JSON - Exemplos

```
{
   "nome" : "Razer",
   "cpf" : "123.456.789-09"
}
```

Web Services Prof. Razer A N R Montaño

48

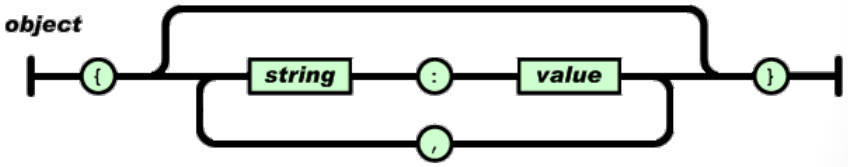
JSON - Exemplos

```
{
  "empregados":[
    {"nome":"José", "sobrenome":"Silva"},
    {"nome":"Maria", "sobrenome":"Costa"},
    {"nome":"João", "sobrenome":"Souza"}
  ]
}
```


JSON - Exemplos

```
{
    "nome": "Razer",
    "enderecos": [
        {
            "cep": "12345-678",
            "logradouro": "Rua Um"
        },
        {
            "cep": "87654-321",
            "logradouro": "Rua Dois"
        }
    ],
    "cpf": "123.456.789-09"
}
```

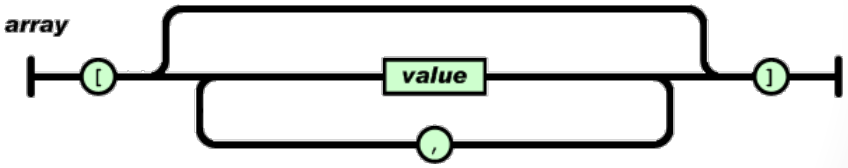
JSON



Web Services Prof. Razer A N R Montaño

51

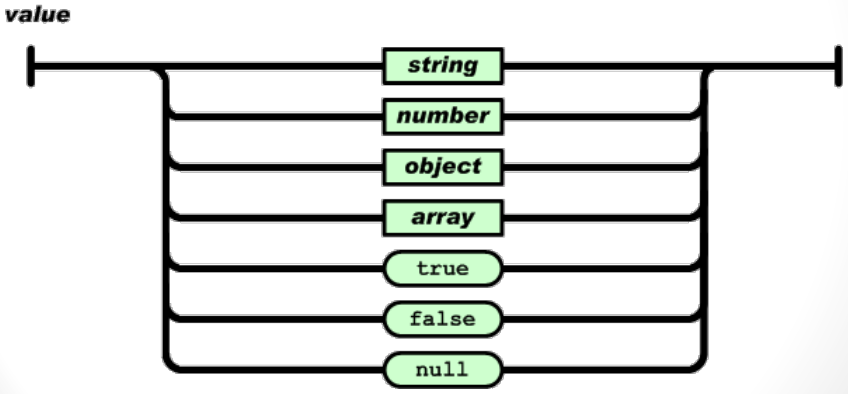
JSON



Web Services Prof. Razer A N R Montaño

52

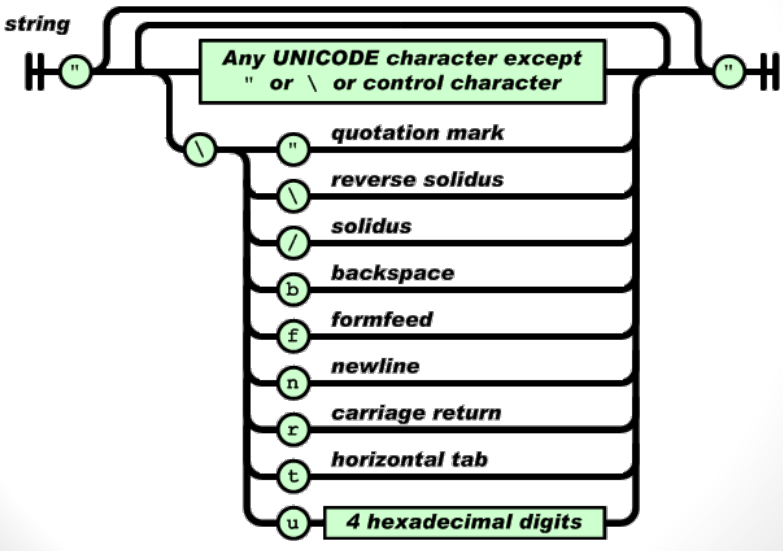
JSON



Web Services Prof. Razer A N R Montaño

53

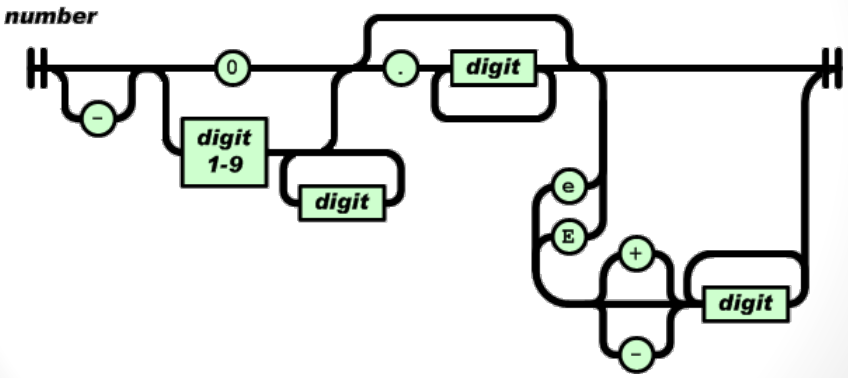
JSON



Web Services Prof. Razer A N R Montaño

54

JSON



Web Services Prof. Razer A N R Montaño

55

Web Services

IMPLEMENTAÇÃO

Web Services Prof. Razer A N R Montaño

56

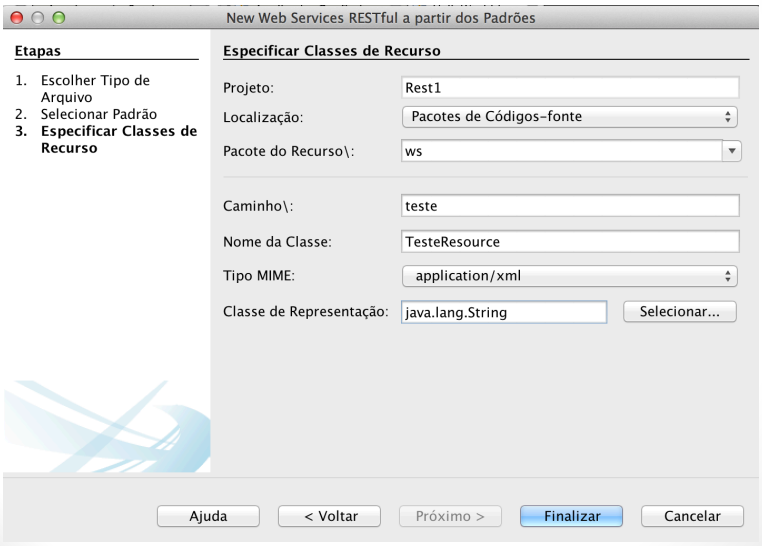
Como Implementar

- Usando Servlets
  - Fazer segmentação de URL na mão
  - Tratar métodos HTTP
  - Tratar retornos
  - Processo complicado
- Usando JAX-RS (Implementação de referência - Jersey)
  - API de fácil uso
  - Anotações para configurar uma classe como um Serviço
  - Anotações para configurar métodos para tratar métodos HTTP
  - Plugins/Wizards do Netbeans

Como Implementar - Netbeans

- Criar novo Projeto Web
- Botão direito no Projeto
  - Novo | Outros
  - Escolher **Web Services**, do lado esquerdo
  - Escolher **Web Services RESTful a partir dos Padrões**, do lado direito
  - Selecionar Recurso Raiz Simples
  - Digitar
    - Pacote do Recurso
    - Caminho : será o nome do recurso para ser acessado
    - Nome da Classe : dar um nome significativo
    - Tipo MIME : o tipo de dado usado, escolher **application/json**

Como Implementar - Netbeans



Web Services Prof. Razer A N R Montaño

59

Como Implementar - Netbeans

- Classe criada

```
@Path("teste")
public class TesteResource {
```

- Métodos criados

```
@GET
@Produces("application/json")
public String getJson() {
      ...
}
```

Web Services Prof. Razer A N R Montaño

60

Como Implementar - Netbeans

- Dentro do método **`getJson()`** coloque:

```
return "{\"pessoaFisica\" : {\n" +
       "    \"nome\" : \"Razer\",\n" +
       "    \"cpf\" : \"123.456.789-09\"\n" +
       "    }\n" +
       "}";
```

Web Services Prof. Razer A N R Montaño

61

Como Testar

- Três formas de testar
  - Se for via GET : Endereço do serviço na URL do navegador

```
http://localhost:8080/Rest1/webresources/teste
```

  - Extensão do Chrome : POSTMAN
  - Aplicação no Netbeans

Web Services Prof. Razer A N R Montaño

62

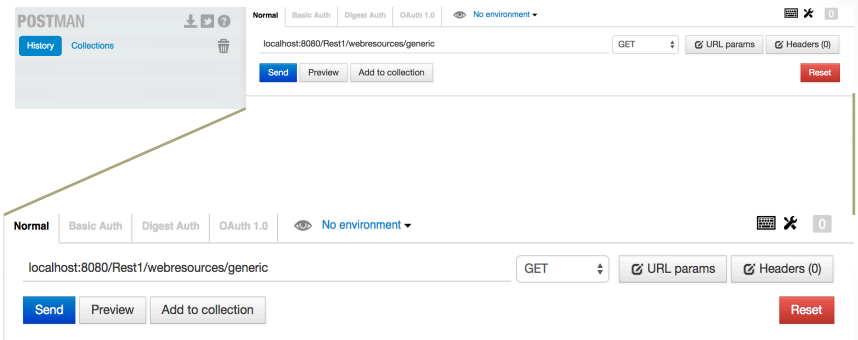
Como Testar – Extensão Chrome

- Instalar uma extensão no navegador
  - Chrome : POSTMAN
  - **`https://chrome.google.com/webstore`**
- Depois de instalado, inicie a aplicação
  - Não esqueça de implantar o projeto WS no Netbeans
  - **`chrome://apps`**
  - Escolha a App POSTMAN

Web Services Prof. Razer A N R Montaño

63

Como Testar – Extensão Chrome



Web Services Prof. Razer A N R Montaño

64

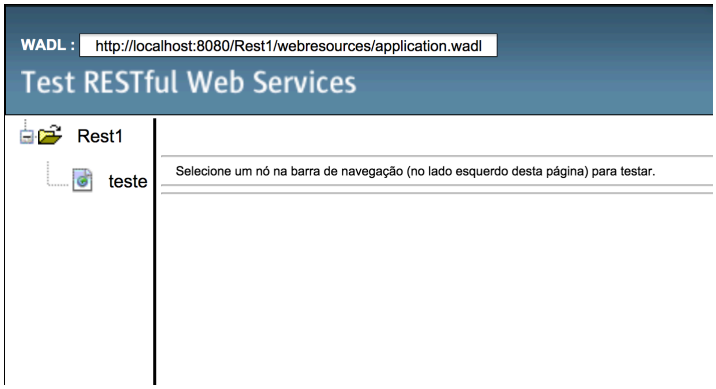
Como Testar – Extensão Chrome

- No campo para o nome do recurso digite
  - `localhost:8080/Rest1/webresources/teste`
  - Sendo **Rest1** o nome do seu projeto e **teste** o nome que está dentro da anotação `@Path`
- Do lado direito, selecione o método a ser invocado, no caso GET
- Pressione SEND e veja abaixo o resultado

Como Testar – App Netbeans

- Netbeans cria uma aplicação
  - Botão direito sobre o Projeto
  - Escolher “Testar Web Services RESTful”
  - Escolher “Cliente de Teste Web no Projeto”
  - Escolha um projeto para conter o teste, pode ser o mesmo projeto do WS
  - O projeto será reimplantado e a aplicação de teste será aberta

Como Testar – App Netbeans



Como Testar – App Netbeans

- Teste
  - Selecione o WS a ser usado (teste)

Como Testar – App Netbeans

Como Testar – App Netbeans

- Teste
  - Selecione o método do WS a ser usado (GET)
  - Pressione “Testar” e veja o resultado abaixo

Exercício

- Implementar este primeiro WS e testá-lo tanto no Plugin como na App do Netbeans
- Alterar o retorno do WS para conter seus dados pessoais mais seu endereço de e-mail e telefone
- Alterar o retorno do WS para conter seus dados pessoais mais seu telefone e uma lista de endereços de e-mail
- Alterar o retorno do WS para conter seus dados pessoais mais seu telefone, uma lista de endereços de e-mail e uma lista de endereços físicos (logradouro, número e cep)

Web Services

JAX-RS

Introdução

- JAX-RS é a API do Java que manipula Web Services REST
- Qualquer informação disponível é um Recurso (Resource)
- Cada Recurso deve possuir uma identificação única
  - Será utilizada para acessar o recurso
  - É uma URI
- Recursos são representados por MediaTypes (HTML, XML, JSON)
- Para manipular recursos, definem-se métodos. Ex.:
  - GET – retorar todos os recursos
  - POST – criar um recurso
- Deve-se tratar os tipos de retorno das requisições
- Todas as importações são de:

```
javax.ws.rs
```

Web Services Prof. Razer A N R Montaño

73

WebService JAX-RS

Web Services Prof. Razer A N R Montaño

74

Recursos

- Em JAX-RS, define-se um Web Resource como uma classe Java
  - Anotada com **@Path**, que define parcialmente sua URI

```
@Path("/clientes")
public class ClienteResource {
```

- O parâmetro de **@Path** pode ou não começar (terminar) com "/"
- Recursos são acessados como

```
http://servidor:porta/app/clientes
```

Web Services Prof. Razer A N R Montaño

75

Métodos

- Os métodos do HTTP são mapeados para métodos desta classe
  - Anotações **@GET, @POST, @PUT, @DELETE**
  - Indicam que método da classe será invocado quando o recurso for acessado pelo método HTTP

```
@Path("/clientes")
public class ClienteResource {

    @GET
    public String getClientes(){
    }
}
```

Web Services Prof. Razer A N R Montaño

76

Media Type

- Pode-se definir o *Media Type* nos métodos
  - Anotação **@Produces** e enumerador **MediaType**
  - Anotação **@Consumes** e enumerador **MediaType**
- **MediaType**
  - **APPLICATION_JSON:** "application/json"
  - **APPLICATION_XML :** "application/xml"
  - **TEXT_PLAIN :** "text/plain"
  - **TEXT_HTML :** "text/html"

```
@Path("/clientes")
public class ClienteResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getClientes() {
    }
}
```

Web Services Prof. Razer A N R Montaño

77

Media Type

```
@Path("/clientes")
public class ClienteResource {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa getClientes() {
    }
}
```

Web Services Prof. Razer A N R Montaño

78

Media Type

```
@Path("/clientes")
public class ClienteResource {

    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Pessoa insertCliente(Pessoa p) {
    }
}
```


Media Type

```
@Path("/clientes")
public class ClienteResource {

    @POST
    @Produces("application/json")
    @Consumes("application/json")
    public Pessoa insertCliente(Pessoa p) {
    }
}
```

Media Type

```
@Path("/clientes")
public class ClienteResource {

    @POST
    @Produces({"application/xml",
               "application/json"})
    @Consumes("application/json")
    public Pessoa insertCliente(Pessoa p) {
    }
}
```

- Para retornar JSON, o cliente deve adicionar no Header:

```
Accept: application/json
```


Media Type

```
@Path("/clientes")
public class ClienteResource {

    @POST
    @Produces({"application/xml",
               "application/json"})
    @Consumes({"application/xml",
               "application/json"})
    public Pessoa insertCliente(Pessoa p) {
    }
}
```

- Para consumir JSON, o cliente deve adicionar no Header:

```
Content-type: application/json
```

Cliente JAX-RS

Cliente

- Para criar um novo cliente

```
Client client = ClientBuilder.newClient();
```

- Para fazer uma chamada
  - **`client.target(url)`** : URL a ser invocada, retorna um WebTarget
  - **`WebTarget.request(media-type)`** : Indica o MediaType de retorno da invocação, retorna um Invocation.Builder
  - **`Invocation.Builder.header(header, valor)`** : Seta algum header, retorna o próprio InvocationBuilder
  - **`Invocation.Builder.get()`** : Executa o método GET, retorna um Response
  - **`Response.getStatus()`** : Retorna o Status Code
  - **`Response.readEntity(String.class)`** : Retorna o resultado da chamada

Cliente

- Invocar método do HTTP, métodos de **`Invocation.Builder`**:
  - **`get()`** : Executa o método GET, retorna um **Response**
  - **`get(classe_retorno)`** : Executa o método GET, retorna um tipo da classe de retorno
  - **`put(entidade)`** : Executa o método PUT, retorna um **Response**
  - **`put(entidade, classe_retorno)`** : Executa o método PUT, retorna um tipo da classe de retorno
  - **`post(entidade)`** : Executa o método POST, retorna um **Response**
  - **`post(entidade, classe_retorno)`** : Executa o método POST, retorna um tipo da classe de retorno
  - **`delete()`** : Executa o método DELETE, retorna um **Response**
  - **`delete(classe_retorno)`** : Executa o método DELETE, retorna um tipo da classe de retorno

Cliente - Chamada GET

- Para facilitar, pode-se encadear as chamadas
  - Retornando um Response

```
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.core.MediaType;

...

Client client = ClientBuilder.newClient();
Response resp = client.target(
    "http://localhost:8080/Rest1/webresources/teste/10")
    .request(MediaType.APPLICATION_JSON)
    .get();
```

Cliente - Chamada GET

```
Client client = ClientBuilder.newClient();
Response resp = client.target("uri do WS")
                .request(MediaType.APPLICATION_JSON)
                .get();
```

Tipo do retorno do WS, no caso JSON

URI: http://localhost:8080/Rest1/webresources/teste/10

Chamada GET. Retorna um objeto Response.

Web Services Prof. Razer A N R Montaño

87

Cliente - Chamada GET

- Para facilitar, pode-se encadear as chamadas
  - Retornando um Objeto

```
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.core.MediaType;

...

Client client = ClientBuilder.newClient();
Atendimento a = client.target(
    "http://localhost:8080/Rest1/webresources/teste/10")
    .request(MediaType.APPLICATION_JSON)
    .get(Atendimento.class);
```

Web Services Prof. Razer A N R Montaño

88

Cliente - Chamada G

```
Client client = ClientBuilder.newClient();
Atendimento a = client.target("uri do WS")
                .request(MediaType.APPLICATION_JSON)
                .get(Atendimento.class);
```

URI: http://localhost:8080/Rest1/webresources/teste/10

Tipo do retorno do WS, no caso JSON

Chamada GET. Indica qual o tipo retornado. Como é JSON e retorna um Atendimento, a conversão é automática

Web Services Prof. Razer A N R Montaño

89

Cliente - Chamada PUT

```
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.core.MediaType;

...
Atendimento a = new Atendimento();
a.setXXX(...);
...

Client client = ClientBuilder.newClient();
Atendimento x = client.target("uri do ws")
                .request(MediaType.APPLICATION_JSON)
                .put(Entity.json(a), Atendimento.class);
```

Web Services Prof. Razer A N R Montaño

90

Cliente - Chamada

URI: http://localhost:8080/Rest1/webresources/teste

```
Client client = ClientBuilder.newClient();
Atendimento x = client.target("uri do ws")
                .request(MediaType.APPLICATION_JSON)
                .put(Entity.json(a), Atendimento.class);
```

Tipo do retorno do WS, no caso JSON

Chamada PUT.
Indica o parâmetro passado, no caso o objeto 'a' convertido para JSON.
Indica qual o tipo retornado. Como é JSON e retorna um Atendimento, a conversão é automática

Web Services Prof. Razer A N R Montaño

91

Exemplos e Exercícios

Web Services Prof. Razer A N R Montaño

92

Exemplos e Exercícios

- Dois Projetos
  - Projeto SERVIDOR :
    - Implementação do Web Service
    - Classe Java Bean : contém a classe com os dados a serem transmitidos
  - Projeto CLIENTE :
    - index.html : com um link que ao ser clicado invoca uma Servlet
    - Servlet : contém os códigos de cliente que invocam a WS. Após, redireciona para mostrar.jsp
    - mostrar.jsp : apresenta o resultado
    - Classe Java Bean : contém a classe com os dados a serem transmitidos

SERVIDOR/CLIENTE: Exemplo: Pessoa

```
package beans;

public class Pessoa implements Serializable {
    private String nome;
    private String email;

    public Pessoa() {}
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

SERVIDOR: Exemplo – WS

```
package ws;

import beans.Pessoa;
import javax...

@Path("pessoas")
public class PessoaResource {


    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa getJson() {
        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        return p;
    }
}
```

Web Services Prof. Razer A N R Montaño

95

CLIENTE: Exemplo - index.html

```
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <a href="Controlador">Chamar WS</a>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

96

CLIENTE: Exemplo – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {
. . .
  protected void processRequest(HttpServletRequest request,
                                HttpServletResponse response)
            throws ServletException, IOException {

    Client client = ClientBuilder.newClient();

    Pessoa p = client
         .target(
              "http://localhost:12488/REST_Server1/webresources/pessoas")
         .request(MediaType.APPLICATION_JSON)
         .get(Pessoa.class);

    request.setAttribute("pessoa", p);
    request.getRequestDispatcher("mostrar.jsp")
                    .forward(request, response);
 }
}
```

Web Services Prof. Razer A N R Montaño

97

CLIENTE: Exemplo - mostrar.jsp

```
<body>
       <h1>Dados</h1>
       <table>
           <tr>
               <th>Nome</th>
               <th>E-mail</th>
           </tr>
           <tr>
               <td>${pessoa.nome}</td>
               <td>${pessoa.email}</td>
           </tr>
       </table>
</body>
```

Web Services Prof. Razer A N R Montaño

98

Exercícios com GET

- Implementar e executar a aplicação anterior
- Alterar a aplicação anterior para que o WS retorne um objeto do tipo Pessoa contendo seu Endereço, além dos dados já existentes
- Alterar a aplicação anterior para que o WS retorne um objeto do tipo Pessoa contendo todos os dados já existentes, mas ao invés de um e-mail, retorne uma lista de e-mails (Lista de Strings)
- Alterar a aplicação anterior para que o WS retorne um objeto do tipo Pessoa contendo:
  - nome : String
  - endereco : Objeto do tipo Endereco contendo logradouro (String) e numero (int)
  - emais : Lista de e-mails (String)
- Alterar a aplicação anterior para que o WS retorne um objeto do tipo Pessoa contendo:
  - nome : String
  - enderecos: Lista de objetos do tipo Endereco contendo logradouro (String) e numero (int)
  - emais : Lista de e-mails (String)

Web Services Prof. Razer A N R Montaño

99

Solução - Objetos

```
public class Endereco implements Serializable {
    private String logradouro;
    private int numero;

    public Endereco() {}
    public String getLogradouro() {
        return logradouro;
    }
    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }
}
```

Web Services Prof. Razer A N R Montaño

100

Solução - Objetos

```
public class Pessoa implements Serializable {
    private String nome;
    private List<String> emails = new ArrayList<String>();
    private List<Endereco> enderecos = new ArrayList<Endereco>();
    public Pessoa() {}
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public List<Endereco> getEnderecos() {
        return enderecos;
    }
    public void setEnderecos(List<Endereco> enderecos) {
        this.enderecos = enderecos;
    }
    public List<String> getEmails() {
        return emails;
    }
    public void setEmails(List<String> emails) {
        this.emails = emails;
    }
}
```

Web Services Prof. Razer A N R Montaño

101

Solução - WS

```
@Path("pessoas")
public class PessoaResource {
    @GET
    @Produces("application/json")
    public Pessoa getJson() {
        List<Endereco> lista = new ArrayList<Endereco>();
        Endereco e = new Endereco();
        e.setLogradouro("Rua X");
        e.setNumero(500);
        lista.add(e);
        e = new Endereco();
        e.setLogradouro("Rua y");
        e.setNumero(600);
        lista.add(e);

        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEnderecos(lista);
        p.getEmails().add("razer.anthom@gmail.com");
        p.getEmails().add("razer@ufpr.br");

        return p;
    }
}
```

Web Services Prof. Razer A N R Montaño

102

Exemplo – WS Cliente (Servlet)

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {
. . .
  protected void processRequest(HttpServletRequest request,
                                HttpServletResponse response)
            throws ServletException, IOException {

    Client client = ClientBuilder.newClient();

    Pessoa p = client
          .target(
               "http://localhost:12488/REST_Server1/webresources/pessoas")
          .request(MediaType.APPLICATION_JSON)
          .get(Pessoa.class);

    request.setAttribute("pessoa", p);
    request.getRequestDispatcher("mostrar.jsp")
                     .forward(request, response);
 }
}
```

Web Services Prof. Razer A N R Montaño

103

Solução – Cliente JSP

```
<body>
    <h1>Dados</h1>
    <table>
        <tr>
            <th>Nome</th>
            <th>Endereço</th>
            <th>E-mails</th>
        </tr>
        <tr>
            <td>${pessoa.nome}</td>
            <td>
                <c:forEach var="end" items="${pessoa.enderecos}">
                    ${end.logradouro},${end.numero} <br/>
                </c:forEach>
            </td>
            <td>
                <c:forEach var="e" items="${pessoa.emails}">
                    ${e} <br/>
                </c:forEach>
            </td>
        </tr>
    </table>
</body>
```

Web Services Prof. Razer A N R Montaño

104

Solução – Cliente JSP

- Servlet e index.html não se alteram

Com POST

- Diferenças para os exemplos com GET
  - A classe Pessoa recebeu um atributo id
  - Projeto SERVIDOR :
    - O método do WS recebe como parâmetro um objeto
  - Projeto CLIENTE :
    - Na Servlet, deve-se construir um objeto antes da invocação

SERVIDOR/CLIENTE: Exemplo: Pessoa

```
package beans;

public class Pessoa implements Serializable {
    private int id;
    private String nome;
    private String email;

    // construtor sem parâmetros
    // setters/getters
}
```


SERVIDOR: Exemplo POST1 - WS

```
@Path("pessoas")
public class PessoaResource {

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void processarPessoa(Pessoa pessoa) {
        System.out.println(
                "Pessoa recebida: " +
                pessoa.getNome() + " - " + pessoa.getEmail());
    }
}
```

CLIENTE: Exemplo POST1 – index.html

```
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <a href="Controlador">Chamar WS</a>
    </body>
</html>
```


CLIENTE: Exemplo POST1 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        Client client = ClientBuilder.newClient();

        client
            .target(
                  "http://localhost:12488/REST_Server2/webresources/pessoas")
            .request(MediaType.APPLICATION_JSON)
            .post(Entity.json(p));

        request.setAttribute("resposta", "OK");
        request.getRequestDispatcher("mostrar.jsp").forward(request, response);
    }
}
```

CLIENTE: Exemplo POST1 – JSP

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>${resposta}</h1>
    </body>
</html>
```


SERVIDOR: Exemplo POST2 – WS

```
@Path("pessoas")
public class PessoaResource {

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa inserirPessoa(Pessoa p) {

        // inserção no BD e obtenção do ID
        p.setId(10);
        return p;
    }
}
```

SERVIDOR: Exemplo POST2 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        Client client = ClientBuilder.newClient();

        Pessoa retorno = client
                .target(
                     "http://localhost:12488/REST_Server3/webresources/pessoas")
                .request(MediaType.APPLICATION_JSON)
                .post(Entity.json(p), Pessoa.class);

        request.setAttribute("pessoa", retorno);
        request.getRequestDispatcher("mostrar.jsp").forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

113

SERVIDOR: Exemplo POST2 – JSP

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <table>
            <tr>
                <th>ID</th>
                <th>Nome</th>
                <th>E-mail</th>
            </tr>
            <tr>
                <td>${pessoa.id}</td>
                <td>${pessoa.nome}</td>
                <td>${pessoa.email}</td>
            </tr>
        </table>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

114

Exercícios com POST

- Implementar e executar a aplicação anterior
- Alterar a aplicação anterior para que o WS receba um objeto do tipo Pessoa contendo:
  - nome : String
  - enderecos: Lista de objetos do tipo Endereco contendo logradouro (String) e numero (int)
  - emais : Lista de e-mails (String)

E mostre esses dados (System.out.println)

CHARSET

CHARSET

- Setar a conversação para UTF-8, por exemplo
- No servidor:
  - Deve-se informar que o retorno (@Produces) será em JSON UTF-8

```
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
```


CHARSET

- No cliente:
  - Deve-se enviar a entidade em JSON UTF-8
  - Ao invés de **`Entity.json()`**, usa-se **`Entity.entity()`**

```
Pessoa p = client
    .target("<uri>")
    .request(MediaType.APPLICATION_JSON)
    .post(
        Entity.entity(
             p,
             MediaType.APPLICATION_JSON+"; charset=utf-8"),
        Pessoa.class);
```

Sub-recursos e Parâmetros

Exemplo – Pessoa (Server/Client)

```
package beans;

public class Pessoa implements Serializable {
    private String nome;
    private String email;

    // construtor sem parâmetros
    // getters/setters
}
```

JAX-RS – Sub-recursos

- Pode-se definir sub-recursos
  - Anotação **@Path** em métodos

```
@Path("/clientes")
public class ClienteResource {

    @GET
    @Path("/PessoaFisica")
    @Produces(MediaType.TEXT_PLAIN)
    public String getClientesPF(){
        ...
    }
}
```

- Sub-recursos são acessados:

```
http://servidor:porta/app/clientes/PessoaFisica
```

Web Services Prof. Razer A N R Montaño

121

JAX-RS – Sub-recursos

- Pode-se ter vários métodos HTTP iguais com subrecursos diferentes

```
@Path("/clientes")
public class ClienteResource {

    @GET
    @Path("/PessoaFisica")
    @Produces(MediaType.APPLICATION_JSON)
    public PessoaFisica getClientesPF(){
      ...
    }
    @GET
    @Path("/PessoaJuridica")
    @Produces(MediaType.APPLICATION_JSON)
    public PessoaJuridica getClientesPJ(){
      ...
    }
}
```

```
http://servidor:porta/app/clientes/PessoaFisica
http://servidor:porta/app/clientes/PessoaJuridica
```

Web Services Prof. Razer A N R Montaño

122

JAX-RS – Parâmetros

- Várias formas de passagem de parâmetros
  - **@PathParam** : /clientes/10
  - **@MatrizParam**: /clientes;id=10
  - **@QueryParam**: /clientes?id=10
  - **@FormParam**: Dados vindos diretamente de Formulários
  - **@HeaderParam**: Dados vindos no Header da requisição
  - **@CookieParam**: Dados vindos de Cookies

JAX-RS – @PathParam

- Na anotação **@Path** do método indica-se o parâmetro entre chaves
- No parâmetro do método usa-se **@PathParam** para atribuí-lo à variável

http://servidor/app/webresources/clientes/10

```
@Path("/clientes")
public class ClienteResource {

    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Cliente getClientes (
                @PathParam("id") String x){
        …
    }
}
```

JAX-RS – @PathParam

- Pode ser definido no **@Path** da classe também
  - Todos os métodos possuem um parâmetro

```
@Path("/clientes/{id}")
public class ClienteResourse {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Cliente getClientes (
                @PathParam("id") String x){
            …
    }
}
```

Web Services Prof. Razer A N R Montaño

125

JAX-RS – @PathParam

- Vários parâmetros são tratados da mesma forma

```
http://servidor/app/webresources/clientes/25/12/2016
```

```
@Path("/clientes")
public class ClienteResource {

    @GET
    @Path("/{dia}/{mes}/{ano}")
    @Produces(MediaType.APPLICATION_JSON)
    public Cliente getClientes (
                @PathParam("dia") String d,
                @PathParam("mes") String m,
                @PathParam("ano") String a){
            …
    }
}
```

Web Services Prof. Razer A N R Montaño

126

JAX-RS – @QueryParam

- Parâmetros no final da URL/URI
  - Sinal '?' : define o início dos parâmetros
  - Sinal '&' : separação entre parâmetros

```
http://servidor/app/webresources/clientes?id=10&nome=Razer
```

- Anotação **@QueryParam(param)** no parêmetro do método
  - **param** : o nome do parâmetro passado
  - Se o dado não for passado, recebe **null** ou valor default dos tipos primitivos
- Pode-se definir um valor default
  - Com a anotação **@DefaultValue**

JAX-RS – @QueryParam

```
http://servidor/app/webresources/clientes?id=10&nome=Razer
```

```
@Path("/clientes")
public class ClienteResourse {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Cliente getClientes (
                @DefaultValue("0")
                @QueryParam("id") String i,
                @QueryParam("nome") String n){
        …
    }
}
```

JAX-RS – @QueryParam

- Pode-se passar uma Lista como parâmetro
  - Parâmetros com o mesmo nome

```
http://servidor/app/webresources/clientes?dado=Razer&dado=Anthom
```

- O parâmetro **dado** é instanciado como **List**

```
@Path("/clientes")
public class ClienteResourse {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Cliente getClientes (
                @QueryParam("dado") List<String> x){
        ...
    }
}
```

Web Services Prof. Razer A N R Montaño

129

Exemplos e Exercícios

Web Services Prof. Razer A N R Montaño

130

SERVIDOR: Exemplo 1 – WS

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa obterPessoa(@PathParam("id") int numero ) {
        Pessoa p = new Pessoa();

        p.setId(numero);
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        return p;
    }
}
```


CLIENTE: Exemplo 1 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();

        Pessoa retorno = client
                .target(
"http://localhost:12488/REST_Server4/webresources/pessoas/500")
                .request(MediaType.APPLICATION_JSON)
                .get(Pessoa.class);

        request.setAttribute("pessoa", retorno);
        request.getRequestDispatcher("mostrar.jsp")
                            .forward(request, response);
    }
}
```

SERVIDOR: Exemplo 2 – WS

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Path("/{dia}/{mes}/{ano}")
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa obterPessoa(
                @PathParam("dia") int dia,
                @PathParam("mes") int mes,
                @PathParam("ano") int ano
            ) {
        Pessoa p = new Pessoa();
        p.setId(10);
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");
        Calendar cal = Calendar.getInstance();
        cal.set(ano, mes-1, dia);
        Date dt = cal.getTime();
        p.setData(dt);

        return p;
    }
}
```

Web Services Prof. Razer A N R Montaño

133

CLIENTE: Exemplo 2 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();

        Pessoa retorno = client
                .target(
"http://localhost:12488/REST_Server4/webresources/pessoas/25/12/2015")
                .request(MediaType.APPLICATION_JSON)
                .get(Pessoa.class);

        request.setAttribute("pessoa", retorno);
        request.getRequestDispatcher("mostrar.jsp")
                        .forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

134

CLIENTE: Exemplo 2 – JSP

```
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html>
    <head><title>JSP Page</title></head>
    <body>
        <table>
           <tr>
              <th>ID</th>
              <th>Nome</th>
              <th>E-mail</th>
              <th>Data</th>
           </tr>
           <tr>
              <td>${pessoa.id}</td>
              <td>${pessoa.nome}</td>
              <td>${pessoa.email}</td>
              <td>
                  <fmt:formatDate value="${pessoa.data}" pattern="dd/MM/yyyy" />
              </td>
           </tr>
        </table>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

135

SERVIDOR: Exemplo 3 – WS

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa obterPessoa(
                @QueryParam("dia") @DefaultValue("1") int dia,
                @QueryParam("mes") @DefaultValue("1") int mes,
                @QueryParam("ano") @DefaultValue("2015") int ano
            ) {
        Pessoa p = new Pessoa();

        p.setId(10);
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");
        Calendar cal = Calendar.getInstance();
        cal.set(ano, mes-1, dia);
        Date dt = cal.getTime();
        p.setData(dt);

        return p;
    }
}
```

Web Services Prof. Razer A N R Montaño

136

CLIENTE: Exemplo 3 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();

        Pessoa retorno = client
                .target(
"http://localhost:12488/REST_Server4/webresources/pessoas?dia=25&mes=12&
ano=2015")
                .request(MediaType.APPLICATION_JSON)
                .get(Pessoa.class);

        request.setAttribute("pessoa", retorno);
        request.getRequestDispatcher("mostrar.jsp")
                        .forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

137

CLIENTE: Exemplo 3 – JSP

```
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html>
    <head><title>JSP Page</title></head>
    <body>
        <table>
           <tr>
              <th>ID</th>
              <th>Nome</th>
              <th>E-mail</th>
              <th>Data</th>
           </tr>
           <tr>
              <td>${pessoa.id}</td>
              <td>${pessoa.nome}</td>
              <td>${pessoa.email}</td>
              <td>
                  <fmt:formatDate value="${pessoa.data}" pattern="dd/MM/yyyy" />
              </td>
           </tr>
        </table>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

138

CLIENTE: Exemplo 3 – Servlet - Default

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();

        Pessoa retorno = client
                .target(
"http://localhost:12488/REST_Server4/webresources/pessoas")
                .request(MediaType.APPLICATION_JSON)
                .get(Pessoa.class);

        request.setAttribute("pessoa", retorno);
        request.getRequestDispatcher("mostrar.jsp")
                        .forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

139

SERVIDOR: Exemplo 4 – FormParam WS

```
@Path("pessoas")
public class PessoaResource {

    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Pessoa atualizar(@FormParam("nome") String nome,
                            @FormParam("email") String email)
{

        Pessoa p = new Pessoa();
        p.setNome(nome);
        p.setEmail(email);

        return p;
    }
}
```

Web Services Prof. Razer A N R Montaño

140

CLIENTE: Exemplo 4 – FormParam index.html

```
<body>
    <form
action="http://localhost:12488/REST_Server5/webresources/pesso
a" method="post">
        Nome: <input type="text" name="nome" /><br/>
        E-mail: <input type="text" name="email" /><br/>
        <input type="submit" value="Ok" />
    </form>
</body>
```


Exercícios

- Implementar as aplicações anteriores

Melhorando a Passagem de Parâmetros

Adicionar Caminho

- No **`target()`**, usa-se o método **`path()`** para acrescentar um segmento à URI
- Método **`path()`** retorna um **`WebTarget`**

```
String uriWS = "http://localhost:8080/app";
Client client = ClientBuilder.newClient();
Pessoa p = client.target(uriWS)
            .path("clientes")
            .request(MediaType.APPLICATION_JSON)
            .get(Pessoa.class);
```

Adicionar Caminho

- Caminho com sub-recurso

```
String uriWS = "http://localhost:8080/app";
Client client = ClientBuilder.newClient();
Pessoa p = client.target(uriWS)
            .path("clientes/pessoafisica")
            .request(MediaType.APPLICATION_JSON)
            .get(Pessoa.class);
```


Adicionar Caminho

- Caminho com sub-recurso

```
String uriWS = "http://localhost:8080/app";
Client client = ClientBuilder.newClient();
Pessoa p = client.target(uriWS)
            .path("clientes")
            .path("pessoafisica")
            .request(MediaType.APPLICATION_JSON)
            .get(Pessoa.class);
```

Passagem de PathParam

- Método **`resolveTemplate()`** retorna um **`WebTarget`**
- Usado para parametrizar o PathParam

```
String uriWS = "http://localhost:8080/app";
Client client = ClientBuilder.newClient();
Pessoa p = client.target(uriWS)
           .path("clientes/{id}")
           .resolveTemplate("id", "1010")
           .request(MediaType.APPLICATION_JSON)
           .get(Pessoa.class);
```

Web Services Prof. Razer A N R Montaño

147

Passagem de QueryParam

- Método **`queryParam()`** retorna um **`WebTarget`**
- Usado para passar parâmetros QueryParam

```
String uriWS = "http://localhost:8080/app";
Client client = ClientBuilder.newClient();
Pessoa p = client.target(uriWS)
           .path("clientes")
           .queryParam("id", "1010")
           .request(MediaType.APPLICATION_JSON)
           .get(Pessoa.class);
```

Web Services Prof. Razer A N R Montaño

148

Retorno do WS

Retorno Padrão: SUCESSO

- Métodos do WS que retornam um objeto possuem dois retornos
  - **200 – OK :** Quando retornam uma entidade
  - **204 – NO CONTENT :** Quando retornam NULL
- Métodos do WS que retornam **void** sempre retornam:
  - **204 – NO CONTENT**

```
@GET
public Pessoa obterPessoa() {
…
}
@POST
public void inserirPessoa() {
…
}
```

Retorno Padrão: ERRO

- Erros padrão:
  - **`400 – BAD REQUEST:`** Se no POST ou PUT, o JAX-RS não consegue interpretar o JSON ou XML
  - **`404 – NOT FOUND:`** Se a URI do serviço estiver errada
  - **`405 – METHOD NOT ALLOWED:`** Quando um método que não está implementado para a URI é invocado
    - Também quando foi omitido um sub-recurso necessário, como um parâmetro
  - **`406 – NOT ACCEPTABLE:`** Quando o Media Type não é compatível

Web Services Prof. Razer A N R Montaño

151

Exceções

- Pode-se retornar um erro de duas formas:
  - Construindo um objeto Response
  - Levantando uma exceção
- Exceções precisam de mapeamento
  - Exceção -> Retorno HTTP
  - Usam-se as exceções padrão
  - Pode-se construir um mapeamento de exceções
- A exceção padrão (que não necessita de mapeamento):
  `javax.ws.rs.WebApplicationException`
- Pode-se lançá-la no WS com um status de retorno

```
throw new WebApplicationException(
                    Response.Status.NOT_FOUND);
```

Web Services Prof. Razer A N R Montaño

152

Exceções

- JAX-RS 2.0 introduziu uma hierarquia de exceções
  - Todas herdam de **`WebApplicationException`**

  - **`BadRequestException`** – 400
  - **`NotAuthorizedException`** – 401
  - **`ForbiddenException`** – 403
  - **`NotFoundException`** – 404
  - **`NotAllowedException`** – 405
  - **`NotAcceptableException`** – 406
  - **`NotSupportedException`** – 415
  - **`InternalServerErrorException`** – 500
  - **`ServiceUnavailableException`** - 503

Mapeamento de Exceções

- Converte uma exceção em um resposta HTTP
- Devem ser anotadas com @Provider
- Quando ocorre uma exceção:
  - JAX-RS primeiro tenta encontrar um ExceptionMapper para a exceção
  - Se não encontrar, tenta encontrar para sua superclasse
  - E assim por diante
- Exemplo:
  - Exceção do JPA
  - **javax.persistence.EntityNotFoundException**
  - Quando ocorrer, deve-se ter um retorno: NOT FOUND

Mapeamento de Exceções

```
@Provider
public class EntityNotFoundMapper
     implements ExceptionMapper<EntityNotFoundException> {

   public Response toResponse(EntityNotFoundException e) {
      return Response
                  .status(Response.Status.NOT_FOUND)
                  .build();
   }
}
```


Response

- O retorno do WS pode ser controlado através do objeto Response
- Feito no método do Servidor que implementa o WS / Método
- Classe
  `javax.ws.rs.core.Response`
- Objeto encapsula a resposta
- Pode-se definir o **Status** de Retorno
- Pode-se definir o **Objeto** de retorno
- Métodos retornam **`Response.ResponseBuilder`**

Response

- Método o WS agora retorna um **Response**

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response obterPessoas(
                          @PathParam("id") String id) {
        ...
    }
}
```

Web Services Prof. Razer A N R Montaño

157

Response

- Dentro do método, usa-se métodos que criam um **Response.ResponseBuilder** e retornam um **Response**

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response obterPessoas(
                          @PathParam("id") String id) {
        return Response
                    .serverError()
                    .build();
    }
}
```

Web Services Prof. Razer A N R Montaño

158

Response

- No Cliente, a invocação do WS deve :
  - Invocar métodos que retornam um Response (os que não possuem o parâmetro de tipo da classe de retorno)
  - Esperar um **`Response`**

```
Client client = ClientBuilder.newClient();
Response resp = client
                .target("URL DO SERVICO")
                .request(MediaType.APPLICATION_JSON)
                .get();
```


Response.ResponseBuilder

- Métodos de **`Response`**:
  - **`status(<status de retorno>)`** : cria um **ResponseBuilder** com o status informado
  - **`serverError()`** : cria um **ResponseBuilder** com um estado de erro do servidor
  - **`ok([<entidade>[, <tipo>]])`** : cria um **ResponseBuilder** com estado OK retornando a entidade passada como parâmetro
  - **`created(<URI>)`** : cria um **ResponseBuilder** com estado CREATED e a localização (URI) do recurso criado
- Métodos de **`Response.ResponseBuilder`**:
  - **`header(<nome>, <valor>)`** : adiciona um dado de cabeçado, retorna um ResponseBuilder
  - **`cookie(<cookie>)`** : adiciona cookies à resposta
  - **`status(<estado>)`** : seta o estado do ResponseBuilder
  - **`link(<links>)`** : adiciona um header do tipo Link
  - **`entity(<entidade>)`** : seta a entidade a ser retornada
  - **`build()`** : cria uma instância de Response (para ser retornada) a partir do ResponseBuilder que foi invocado

Response

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response obterPessoas(
                          @PathParam("id") String id) {

        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");
        return Response
                   .ok(p)
                   .build();
    }
}
```

Web Services Prof. Razer A N R Montaño

161

Response - Status

- Enumerador

  Response.Status
- Valores
  - **Response.Status.OK** : 200
  - **Response.Status.CREATED** : 201
  - **Response.Status.FORBIDDEN** : 403
  - **Response.Status.NOT_FOUND** : 404
- Por exemplo:

```
return Response
           .status(Response.Status.NOT_FOUND)
           .build();
```

Web Services Prof. Razer A N R Montaño

162

Response - Status

- Códigos de Status do HTTP
  - 1xx – Informacionais
  - 2xx – Códigos de sucesso
  - 3xx – Códigos de redirecionamento
  - 4xx – Códigos causados pelo cliente
  - 5xx – Erros originados no servidor

Response - Status

200	OK	**401**	Unauthorized
201	Created	**403**	Forbidden
202	Accepted	**404**	Not Found
204	No Content	**405**	Method Not Allowed
206	Partial Content	**409**	Conflict
301	Moved Permanently	**410**	Gone
303	See other	**412**	Precondition Failed
304	Not Modified	**415**	Unsupported Media Type
307	Temporary Redirect	**500**	Internal Server Error
400	Bad Request	**503**	Server Unavailable

CORS

- Requisições seguem a política *Same-Origin Policy*
  - Limita requisições entre domínios distintos
  - Só envia requisições para o mesmo domínio de onde está sendo enviado
  - Evitar ataques *XSS – Cross-site Scripting*
- CORS – *Cross-Origin Resource Sharing*
  - Mecanismo W3C
  - Pode-se enviar um header no servidor indicando quais domínios têm permissão de fazer requisições *cross-domain*
  - Header é: **`Access-Control-Allow-Origin`**
    - Ex.: **`http://www.razer.net.br`**
    - Ex.: **`*`**
- Melhor implementar com Filtros

Web Services Prof. Razer A N R Montaño

165

Resposta e CORS

- Exemplo no WS:

```
return Response
          .serverError()
          .header("Access-Control-Allow-Origin",
"http://www.razer.net")
          .build();
```

- Exemplo no WS:

```
return Response
            .status(Response.Status.OK)
            .header("Access-Control-Allow-Origin", "*")
            .entity(p)
            .build();
```

Web Services Prof. Razer A N R Montaño

166

SERVIDOR: Exemplo 1 - WS

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getJson() {

        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        return Response.ok(p).build();
    }
}
```


CLIENTE: Exemplo 1 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();
        Response resp = client

.target("http://localhost:12488/REST_Server6/webresources/pessoas")
                .request(MediaType.APPLICATION_JSON)
                .get();

        request.setAttribute("status", resp.getStatus());
        request.setAttribute("pessoa", resp.readEntity(Pessoa.class));
        request.getRequestDispatcher("mostrar.jsp")
                                  .forward(request, response);
    }
}
```

CLIENTE: Exemplo 1 - JSP

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Status: ${status}</h1>
        <table>
            <tr>
                <th>Nome</th>
                <th>E-mail</th>
            </tr>
            <tr>
                <td>${pessoa.nome}</td>
                <td>${pessoa.email}</td>
            </tr>
        </table>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

169

SERVIDOR: Exemplo 2 - WS

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Produces("application/json")
    public Response getJson() {

        return Response.serverError().build();

    }
}
```

Web Services Prof. Razer A N R Montaño

170

CLIENTE: Exemplo 2 - Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();
        Response resp = client

.target("http://localhost:12488/REST_Server7/webresources/pessoas")
                .request(MediaType.APPLICATION_JSON)
                .get();

        request.setAttribute("status", resp.getStatus());
        request.getRequestDispatcher("mostrar.jsp")
                        .forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

171

CLIENTE: Exemplo 2 - JSP

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Status: ${status}</h1>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

172

SERVIDOR: Exemplo 3 - WS

```
@Path("pessoas")
public class PessoaResource {

    @GET
    @Produces("application/json")
    public Response getJson() {
        Pessoa p = new Pessoa();
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        return Response
                .status(Response.Status.OK)
                .header("Access-Control-Allow-Origin", "*")
                .entity(p)
                .build();
    }
}
```

Web Services Prof. Razer A N R Montaño

173

CLIENTE: Exemplo 3 – Servlet

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();
        Response resp = client

.target("http://localhost:12488/REST_Server8/webresources/pessoas")
                .request(MediaType.APPLICATION_JSON)
                .get();
        request.setAttribute("status", resp.getStatus());
        request.setAttribute("pessoa", resp.readEntity(Pessoa.class));

        request.getRequestDispatcher("mostrar.jsp")
                            .forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

174

CLIENTE: Exemplo 3 - JSP

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Status: ${status}</h1>
        <table>
            <tr>
                <th>Nome</th>
                <th>E-mail</th>
            </tr>
            <tr>
                <td>${pessoa.nome}</td>
                <td>${pessoa.email}</td>
            </tr>
        </table>
    </body>
</html>
```


Retorno de Listas Genéricas

- Ao retornar tipos genéricos, deve-se empacotá-los e tipá-los com
  `javax.ws.rs.core.GenericEntity`
  `javax.ws.rs.core.GenericType`

- Empacota-se da seguinte forma:

```
List<Pessoa> p = new ArrayList<>();
…
GenericEntity<List<Pessoa>> gt =
   new GenericEntity<List<Pessoa>( p ) {};
```

Retorno de Listas Genéricas

- No Servidor WS, empacota-se um **List<Pessoa>**
  - Usa-se **GenericEntity**

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response buscarPessoas() {
    List<Pessoa> pessoas = PessoaDAO.buscarTodos();

    GenericEntity<List<Pessoa>> lista =
         new GenericEntity<List<Pessoa>>(pessoas) {};

    return Response
                .ok()
                .entity(lista)
                .build();
}
```


Retorno de Listas Genéricas

- No cliente, recebe-se um **List<Pessoa>**
  - Para designar o tipo genérico empacotado usa-se **GenericType**

```
Client client = ClientBuilder.newClient();
Response resp = client
       .target("http://localhost:8080/app/pessoas")
       .request(MediaType.APPLICATION_JSON)
       .get();
List<Pessoa> lista =
            resp.readEntity(
               new GenericType<List<Pessoa>>() {}
             );
```

HATEOAS

HATEOAS

- *Hypermedia As The Engine Of Application State*
- Evita que clientes designem URIs "na mão"
- Provê informação de navegação entre os recursos, de forma dinâmica
- Usar links como referências a ações que podem ser tomadas a partir da entidade atual
- Links podem ser:
  - **Transacionais** : indicam ações que o cliente pode efetuar utilizando o recurso
  - **Estruturais** : indicam elementos da própria estrutura do conteúdo

HATEOAS

- Solicitação de Compra

```
<compra>
   <item>
      <tablet id="1">iPad Air</cerveja>
      <quantidade>1</quantidade>
   </item>
</compra>
```

- Resposta da Compra

```
<compra id="111">
   <item>
      <tablet id="1">iPad Air</cerveja>
      <quantidade>1</quantidade>
   </item>
   <link rel="pagamento" href="/pagamento/111" />
</compra>
```

Web Services Prof. Razer A N R Montaño

181

HATEOAS

- Links de Pessoa

```
{
       "id" : "10",
       "nome" : "Razer",
       "cpf" : "123456790",
       "links" : [
          {
             "uri" : "/pessoa/10",
             "rel" : "self"
          },
          {
             "uri" : "/pessoa/10",
             "rel" : "edit"
          },
          {
             "uri" : "/pessoa/10/likes",
             "rel" : "likes"
          }

       ]
}
```

Web Services Prof. Razer A N R Montaño

182

HATEOAS

- A tag **`link`** possui os seguintes atributos importantes:
  - **`href`** : URL onde o recurso está localizado
  - **`rel`** : informação semântica, relacionamento entre o recurso atual e o que está especificado no link
  - **`title`** : descrição legível do recurso
  - **`method`** : indica os métodos suportados (separados por vírgula). Não usado
  - **`type`** : indica os *media types* suportados. Não usado

HATEOAS

- **`rel`** comumente usados:
  - **`item`** : aponta para um membro de uma coleção
  - **`collection`** : aponta para a coleção
  - **`edit`** : aponta para o recurso que pode ser usado para edição
  - **`latest-version`** : aponta para o recurso que tem a informação da última versão
  - **`self`**: aponta para si mesmo
  - Etc…
- Lista de rels
  - **`https://tools.ietf.org/html/rfc5988#section-6.2.2`**
  - **`http://www.iana.org/assignments/link-relations/link-relations.xhtml`**

HATEOAS

- Implementação:
  - No Servidor WS: gerar os links
    - **`UriInfo`** : Dá acesso à URI da aplicação, gera o construtor de URI
    - **`UriBuilder`** : Gerador de URIs
    - **`Response.link(<URI>, <rel>)`** : para passar a URI para o cliente
  - No Cliente: obter os links
    - Obter a resposta (**`Response`**) da invocação
    - **`Response.getLink(<rel>)`** : obtém a URI atrelada ao <rel>
    - **`URI.toString()`** : retorna a URI como uma string

HATEOAS

- No Servidor: Implementação – Gerar os Links
- Usar o Contexto de URI

```
@Context
private UriInfo context;
```

- Obter o Construtor de URIs

```
UriBuilder ub = context.getAbsolutePathBuilder();
```

- Criar a URI

```
URI uriSelf = ub.path(String.valueOf(p.getId())).build();
```

- Passar a URI no Response

```
return Response
         .status(Response.Status.OK)
         .link(uriSelf, "self")
         .link(uriSelf, "edit")
         .header("Access-Control-Allow-Origin", "*")
         .entity(p)
         .build();
```

HATEOAS

- No Cliente: Implementação – Obter os Links
- Obter o Response

```
Response resp = client

.target("http://localhost:12488/REST_Server9/webresources/pessoas")
                    .request(MediaType.APPLICATION_JSON)
                    .get();
```

- Obter as URIs

```
URI selfUri = resp.getLink("self").getUri();
URI editUri = resp.getLink("edit").getUri();
```

- Obter a String dos Links

```
request.setAttribute("self", selfUri.toString());
request.setAttribute("edit", editUri.toString());
```

Web Services Prof. Razer A N R Montaño

187

Exemplo

```
@Path("pessoas")
public class PessoaResource {
    @Context
    private UriInfo context;

    @GET
    @Produces("application/json")
    public Response obterPessoa() {
        Pessoa p = new Pessoa();
        p.setId(500);
        p.setNome("Razer");
        p.setEmail("razer.anthom@gmail.com");

        UriBuilder ub = context.getAbsolutePathBuilder();
        URI uriSelf = ub.path(String.valueOf(p.getId())).build();

        return Response
                    .status(Response.Status.OK)
                    .link(uriSelf, "self")
                    .link(uriSelf, "edit")
                    .header("Access-Control-Allow-Origin", "*")
                    .entity(p)
                    .build();
    }
}
```

Web Services Prof. Razer A N R Montaño

188

Exemplo

```
@WebServlet(name = "Controlador", urlPatterns = {"/Controlador"})
public class Controlador extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {

        Client client = ClientBuilder.newClient();

        Response resp = client

.target("http://localhost:12488/REST_Server9/webresources/pessoas")
                .request(MediaType.APPLICATION_JSON)
                .get();

        URI selfUri = resp.getLink("self").getUri();
        URI editUri = resp.getLink("edit").getUri();

        request.setAttribute("self", selfUri.toString());
        request.setAttribute("edit", editUri.toString());
        request.setAttribute("status", resp.getStatus());
        request.setAttribute("pessoa", resp.readEntity(Pessoa.class));

        request.getRequestDispatcher("mostrar.jsp")
                                        .forward(request, response);
    }
}
```

Web Services Prof. Razer A N R Montaño

189

Exemplo

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Status: ${status}</h1>
        <a href="${self}">Ver Pessoa</a> <br/>
        <a href="${edit}">Editar Pessoa</a> <br/>
        <table>
            <tr>
                <th>Nome</th>
                <th>E-mail</th>
            </tr>
            <tr>
                <td>${pessoa.nome}</td>
                <td>${pessoa.email}</td>
            </tr>
        </table>
    </body>
</html>
```

Web Services Prof. Razer A N R Montaño

190

Exercício

- Executar o exemplo anterior e ver os links no HTML

Web Services

JAXB - JSON

JAXB

- Especificação/API para conversão de XML em Java, e vice-versa
- Também usada pelo JAX-RS para conversão para JSON
- Se a classe não possui anotação nenhuma
  - Conversão direta
  - Nome das propriedades viram atributos no JSON
- Por default mapeia:
  - Atributos públicos
  - Atributos anotados
  - Propriedades (getNome() -> propriedade nome)
- Para alterar o tipo do mapeamento usa-se
  - **`@XmlAccessorType`**

JAXB

- Anotações na classe para indicar a conversão
  - **`@XmlRootElement`** : Indica que é uma classe que pode ser serializada para JSON/XML
  - **`@XmlElement`** : Explicitamente indica que o atributo será um atributo/tag no JSON/XML; Usado também para alterar o nome no mapeamento
  - **`@XmlTransient`** : Indica que não deve ser mapeado

JAXB

```
package beans;

public class Pessoa {
  private String nome;
  private String email;

  public String getNome() {
      return nome;
  }
  public void setNome(String nome) {
      this.nome = nome;
  }
  public String getEmail() {
      return email;
  }
  public void setEmail(String email) {
      this.email = email;
  }
}
```

```
{
  "nome": "Razer",
  "email": "razer@razer"
}
```

Web Services Prof. Razer A N R Montaño

195

JAXB

```
package beans;

public class Pessoa {
  private String nome;
  private String email;

  @XmlElement(name="nomeCompleto")
  public String getNome() {
      return nome;
  }
  public void setNome(String nome) {
      this.nome = nome;
  }
  public String getEmail() {
      return email;
  }
  public void setEmail(String email) {
      this.email = email;
  }
}
```

```
{
  "nomeCompleto": "Razer",
  "email": "razer@razer"
}
```

Web Services Prof. Razer A N R Montaño

196

Web Services

GSON

GSON

- JSON: http://www.json.org/
- Geração de JSON em Java
  - Biblioteca GSON do Google
  - https://code.google.com/p/google-gson/
  - Fazer o download da última versão
  - Consultar o User Guide que eles fornecem

GSON

- Objeto -> String JSON

```
Gson gson = new Gson();
String json = gson.toJson(objeto);
```

- String JSON -> Objeto

```
Gson gson = new Gson();
Pessoa x = gson.fromJson(json, Pessoa.class);
```


GSON

- Regras de Serialização/Desserialização
  - Atributos podem ser privados
  - Não há necessidade de usar anotações para indicar atributos a serem serializados
  - Atributos transientes/estáticos são ignorados
  - Valores NULL
    - Serialização : Atributo com valor NULL não é transformado para JSON
    - Desserialização : Um atributo que não está presente no JSON é setado para NULL
  - Em listas, sempre usar tipos definidos
- Classes com tratamentos específicos
  - Pode-se criar serialização/desserialização customizada

GSON

- **`@Expose`**
  - Para customizar que campos devem ser serializados/desserializados
  - Somente os campos marcados com `@Expose` serão usados
  - Deve-se criar a instância de Gson com:

```
Gson gson = new GsonBuilder().
                excludeFieldsWithoutExposeAnnotation().
                create();
```

Web Services Prof. Razer A N R Montaño

201

Exemplos: Objeto -> JSON

Web Services Prof. Razer A N R Montaño

202

Bean

```
public class Pessoa {
    private String cpf;
    private String nome;
    private int idade;
    private Date dtNascimento;

    // setters/getters
}
```


Objeto Simples -> JSON

```
@GET
@Path("/{id}")
@Produces("application/json")
public String getJson(@PathParam("id") String id) {
    Pessoa p = new Pessoa();
    p.setNome("Razer");
    p.setCpf("1010");
    p.setIdade(18);
    p.setDtNascimento(new Date());

    Gson gson = new Gson();
    String json = gson.toJson(p);

    return json;
}
```

Objeto Simples -> JSON

```
{
    "cpf": "1010",
    "nome": "Razer",
    "idade": 18,
    "dtNascimento": "Sep 10, 2014 12:12:09 PM"
}
```


Bean com Objetos

```
public class Atendimento {
    private Pessoa pessoa;
    private String observacao;

    // setters/getters
}
```

Objeto -> JSON

```
@GET
@Path("/{id}")
@Produces("application/json")
public String getJson(@PathParam("id") String id) {
    Pessoa p = new Pessoa();
    p.setNome("Razer");
    p.setCpf("1010");
    p.setIdade(18);
    p.setDtNascimento(new Date());
    Atendimento a = new Atendimento();
    a.setPessoa(p);
    a.setObservacao("Teste");

    Gson gson = new Gson();
    String json = gson.toJson(a);
    return json;
}
```


Objeto -> JSON

```
{
    "pessoa": {
        "cpf": "1010",
        "nome": "Razer",
        "idade": 18,
        "dtNascimento": "Sep 10, 2014 12:20:11 PM"
    },
    "observacao": "Teste"
}
```

Bean com Lista

```
public class Atendimento {
    private Pessoa pessoa;
    private String observacao;
    private ArrayList<String> categorias =
                                    new ArrayList<>();

    // setters/getters
}
```


Objeto com Lista -> JSON

```
Pessoa p = new Pessoa();
p.set...
ArrayList<String> cat = new ArrayList<>();
cat.add("cat 1");
cat.add("cat 2");
cat.add("cat 3");
Atendimento a = new Atendimento();
a.setPessoa(p);
a.setObservacao("Teste");
a.setCategorias(cat);

Gson gson = new Gson();
String json = gson.toJson(a);
return json;
```

Objeto com Lista -> JSON

```
{
    "pessoa": {
        "cpf": "1010",
        "nome": "Razer",
        "idade": 18,
        "dtNascimento": "Sep 10, 2014 12:23:24 PM"
    },
    "observacao": "Teste",
    "categorias": [
        "cat 1",
        "cat 2",
        "cat 3"
    ]
}
```

Web Services Prof. Razer A N R Montaño

211

Exemplos: JSON -> Objeto

Web Services Prof. Razer A N R Montaño

212

Bean

```
public class Pessoa {
    private String cpf;
    private String nome;
    private int idade;
    private Date dtNascimento;

    // setters/getters
}
```


JSON -> Objeto

```
 @PUT
 public String putJson(@FormParam("pessoa") String p) {

    Gson gson = new Gson();
    Pessoa x = gson.fromJson(p, Pessoa.class);
    return x.getCpf();
}
```

JSON -> Objeto

- Para testar, use o Postman
  - Implante o projeto no GlassFish
  - Setar o endereço para
    `localhost:8080/Rest1/webresource/teste`
  - Setar o método para **`PUT`**
  - Setar a codificação para **`x-www-form-urlencoded`**
  - Adicionar um parâmetro de nome `pessoa` com o valor

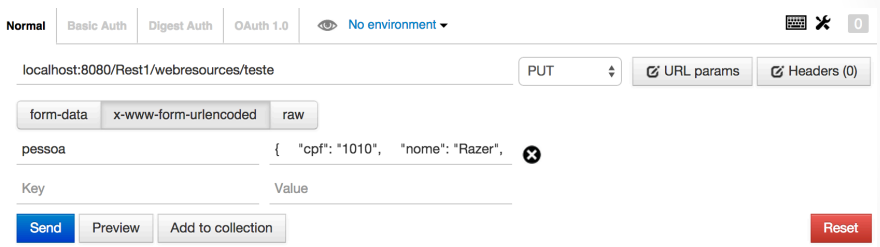
```
{
    "cpf": "1010",
    "nome": "Razer",
    "idade": 18,
    "dtNascimento": "Sep 10, 2014 12:12:09 PM"
}
```

Web Services Prof. Razer A N R Montaño

215

JSON -> Objeto

- No Postman fica:



Web Services Prof. Razer A N R Montaño

216

Bean com Objetos

```
public class Atendimento {
    private Pessoa pessoa;
    private String observacao;

    // setters/getters
}
```


JSON -> Objeto

```
@PUT
public String putJson(@FormParam("atendimento") String p) {

    Gson gson = new Gson();
    Atendimento x = gson.fromJson(p, Atendimento.class);
    return x.getPessoa().getNome();
}
```

JSON -> Objeto

- Para testar, use o Postman
  - Implante o projeto no GlassFish
  - Setar o endereço para
    localhost:8080/Rest1/webresource/teste
  - Setar o método para PUT
  - Setar a codificação para **x-www-form-urlencoded**
  - Adicionar um parâmetro de nome **atendimento** com o valor

```
{
    "pessoa": {
        "cpf": "1010",
        "nome": "Razer",
        "idade": 18,
        "dtNascimento": "Sep 10, 2014 12:20:11 PM"
    },
    "observacao": "Teste"
}
```


Bean com Lista

```
public class Atendimento {
    private Pessoa pessoa;
    private String observacao;
    private ArrayList<String> categorias =
                                    new ArrayList<>();

    // setters/getters
}
```

JSON -> Objeto

```
@PUT
public String putJson(@FormParam("atendimento") String p) {

    Gson gson = new Gson();
    Atendimento x = gson.fromJson(p, Atendimento.class);
    return x.getCategorias().get(1);
}
```


JSON -> Objeto

- Para testar, use o Postman
  - Implante o projeto no GlassFish
  - Setar o endereço para
    `localhost:8080/Rest1/webresource/teste`
  - Setar o método para **PUT**
  - Setar a codificação para **x-www-form-urlencoded**
  - Adicionar um parâmetro de nome **atendimento** com o valor

```
{
    "pessoa": {
        "cpf": "1010",
        "nome": "Razer",
        "idade": 18,
        "dtNascimento": "Sep 10, 2014 12:23:24 PM"
    },
    "observacao": "Teste",
    "categorias": [
        "cat 1",
        "cat 2",
        "cat 3"
    ]
}
```

Web Services

REFERÊNCIAS

Materiais

- Links

https://netbeans.org/kb/docs/websvc/jax-ws.html
https://netbeans.org/kb/docs/websvc/rest.html
https://code.google.com/p/google-gson/
http://json.org/
http://www.oracle.com/technetwork/pt/articles/java/java-ee7-y-jax-rs-2-2109106-ptb.html
https://jersey.java.net/documentation/latest/index.html

- REST x SOAP WebServices

http://www.infoq.com/articles/rest-soap-when-to-use-each
https://netbeans.org/kb/trails/web.html

Materiais

- Discussão

http://blog.caelum.com.br/os-7-habitos-dos-desenvolvedores-de-webservices-altamente-eficazes/

- Apostila

http://www.k19.com.br/downloads/apostilas/java/k19-k23-integracao-de-sistemas-com-webservices-jms-e-ejb

Livros

SAUDATE. REST: Construa API's Inteligentes de Maneira Simples. (http://www.casadocodigo.com.br/products/livro-rest)

BURKE. RESTful Java with JAX-RS 2.0. (http://www.amazon.com/RESTful-Java-JAX-RS-Bill-Burke/dp/144936134X/ref=la_B001IGJSZ8_1_1?s=books&ie=UTF8&qid=1410470076&sr=1-1)