

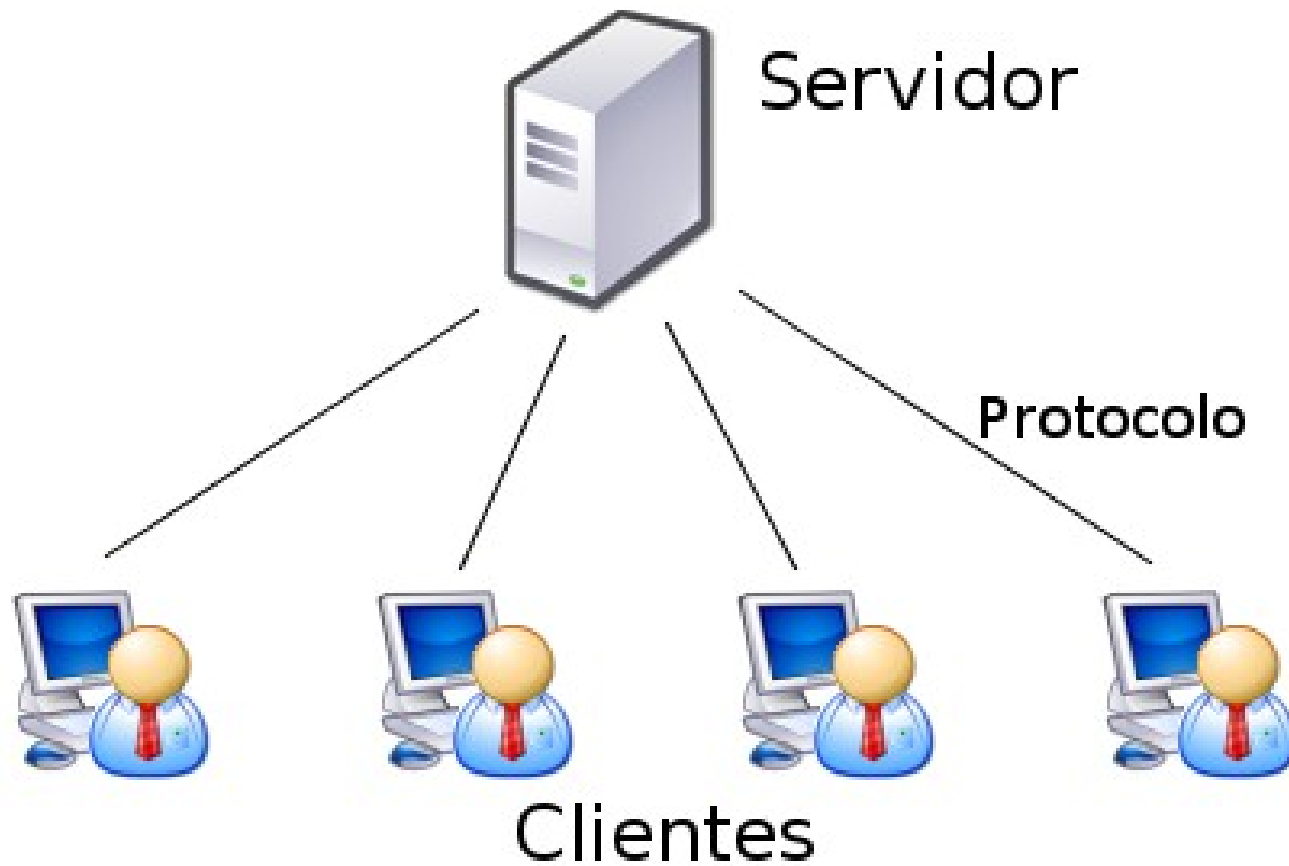
Desenvolvimento de Sistemas I

Rest

Professor: Jezer Machado de Oliveira



Cliente Servidor



REST

- Representational State Transfer
 - (Transferência de Estado Representacional)
- Orientado a recursos
- Não é um protocolo
- Estilo arquitetural de sistemas de informação distribuídos cliente/servidor
- Proposto por Roy Fielding em seu doutorado, um dos principais autores do HTTP



REST

- Focada nos recursos
- Identificação global (URI)
- Interfaces uniformes (Criar, Ler, Atualizar, Excluir)
- Interações sem estado (stateless)
- Cache
- Sistema em camadas

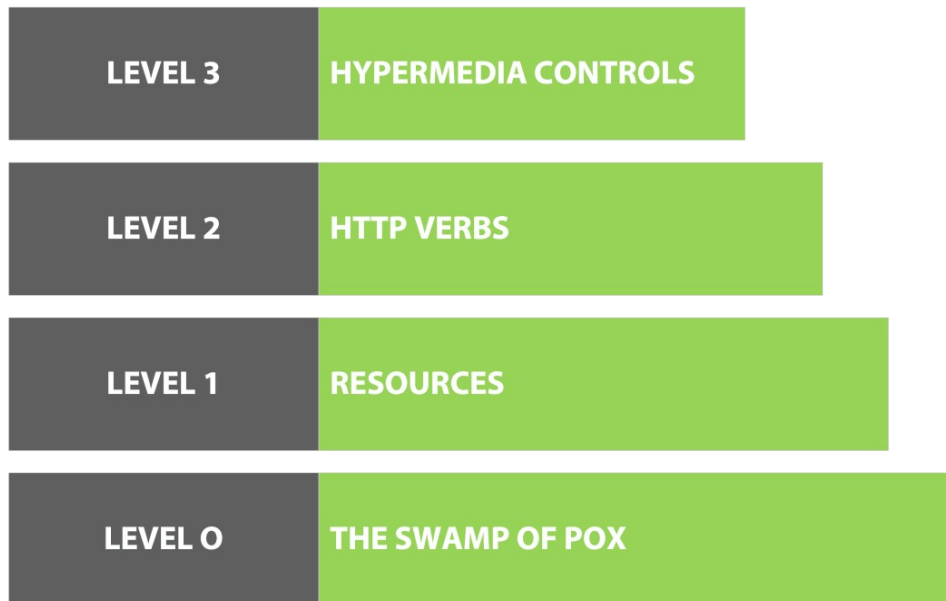


REST

Modelo de Maturidade de Richardson



GLORY OF REST



INSTITUTO FEDERAL
Rio Grande do Sul



REST vs RESTful

- REST
 - Conjunto de princípios de arquitetura
- RESTful
 - Capacidade de determinado sistema aplicar os princípios de REST



REST

- Definições
 - Camada de comunicação
 - Interface
 - Modelo de dados



REST

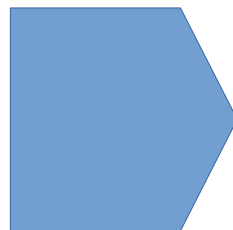
- Camada de comunicação
- Interface
 - Identificação global (URI)
 - Interfaces uniformes (Criar, Ler, Atualizar, Excluir)
 - Interações sem estado (stateless)
 - Cache



REST

- Camada de comunicação

- Interface

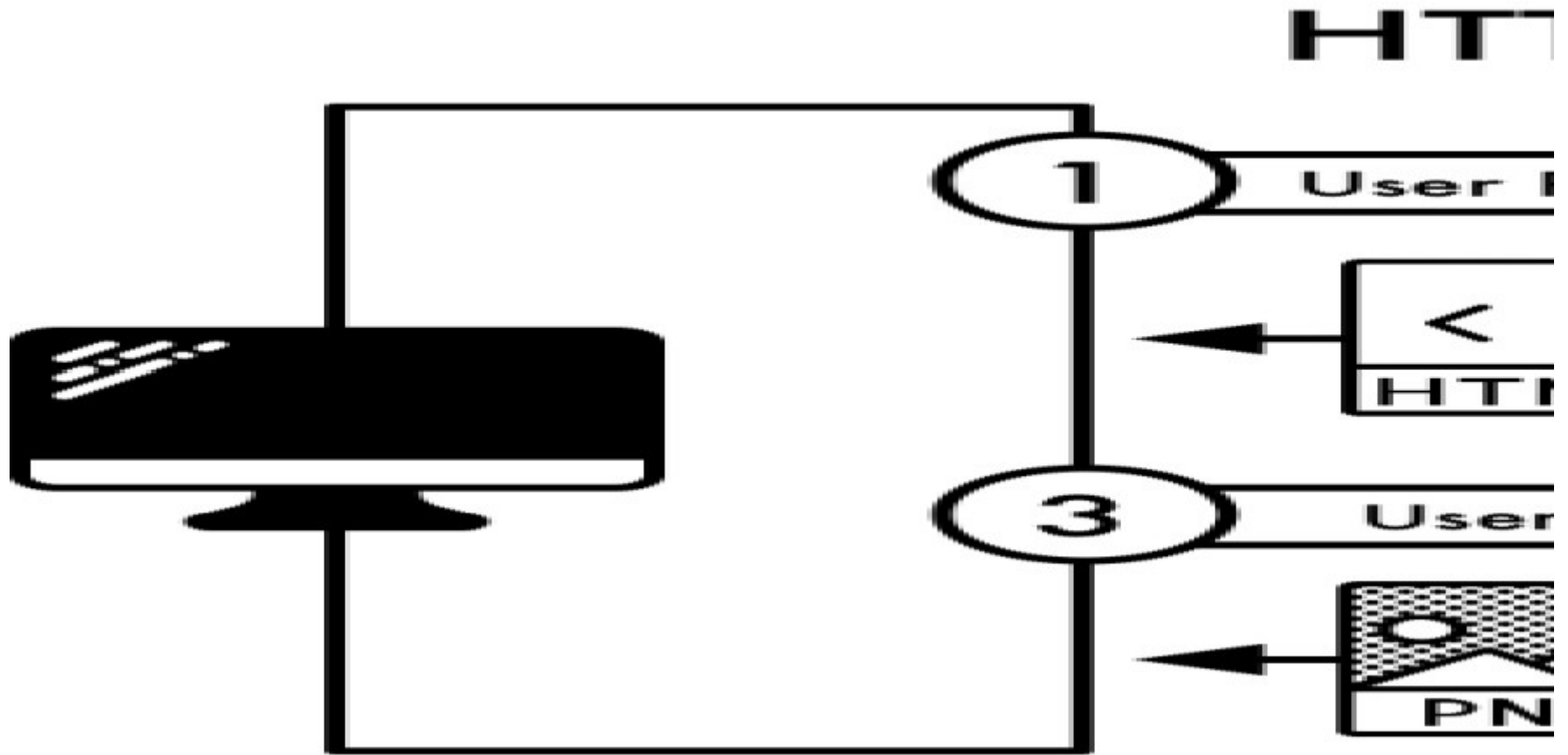


HTTP

- Identificação global (URI)
- Interfaces uniformes (Criar, Ler, Atualizar, Excluir)
- Interações sem estado (stateless)
- Cache



HTTP - Hypertext Transfer Protocol



HTTP - URI

Protocolo

Recurso dentro
do servidor
(URN)

<http://exemplo.com/img/foto.png>

Endereço
do servidor
(URL)



HTTP - URI

- **URI (Uniform Resource Identifier)**
 - Identificador de Recursos Universal
 - Formado pelo Protocolo+URL+URN
- **Protocolo**
 - Forma que o servidor e cliente se comunicam
- **URL (Uniform Resource Locator)**
 - Localizador de Recurso Universal
- **URN (Uniform Resource Name)**
 - Nome de Recurso Universal



HTTP

Requisição HTTP

POST /index.html HTTP/1.1

Host: restinga.ifrs.edu.br

Cache-Control: no-cache

Postman-Token: ed101df7-ef8e-5d8b-cab1-f4739196ff1b

Content-Type: multipart/form-data; boundary=----

Content-Disposition: form-data; name="nome"

teste



HTTP

Retorno HTTP

HTTP/1.1 200 OK

Date: Mon, 23 July 2017 22:38:34 GMT

Content-Type: text/html; charset=UTF-8

Content-Encoding: UTF-8

Content-Length: 81

Last-Modified: Wed, 08 Jan 2017 23:11:55 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

<html>

<body>

Hello World, this is a very simple HTML document.

</body>

</html>

HTTP

Retorno HTTP

```
HTTP/1.1 404 Not Found
Content-Length: 1635
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Tue, 04 May 2017 22:30:36 GMT
Connection: close
```



HTTP

Método	Função
GET	O método GET requisita uma representação do recurso especificado
HEAD	Variação do GET que recupera só o cabeçalho da resposta
POST	Envia dados para serem processados (formulário HTML)
PUT	Edita as informações de um determinado recurso
DELETE	Exclui o recurso
TRACE	Retorna os dados como um eco
OPTIONS	Recupera os métodos HTTP que o servidor aceita
CONNECT	Abre um túnel de comunicação

REST

- Identificação global (URI)
 - <http://exemplo.com.br/produto/156>
- Interfaces uniformes (CRUD)
 - Criar => POST
 - Ler => GET
 - Atualizar => PUT
 - Excluir => DELETE
- Não necessita implementação (Padrão mundial)
- Estrutura de cache definida



REST

Recurso é identificado pela URI

- <http://exemplo.com.br/pessoas/>
 - Representa todas as pessoas do serviço
- <http://exemplo.com.br/pessoas/32>
 - Representa a pessoa de id 32 do serviço
- <http://exemplo.com.br/pessoas/32/telefones>
 - Representa os telefones da pessoa de id 32 do serviço



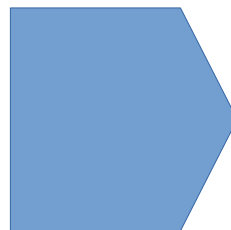
REST

URL	Método (verbo)			
	GET	POST	PUT	DELETE
http:// exemplo.com.br/ pessoas/	Lista todas as pessoas	Cria uma nova pessoa	Substitui a lista inteira de pessoas	Apaga a lista inteira de pessoas
http:// exemplo.com.br/ pessoas/15	Retorna a pessoa com a id = 15	Não utilizado	Altera os dados da pessoa com a id = 15	Apaga a pessoa com a id = 15



REST

- Camada de comunicação
- Interface
- Modelo de dados ?????

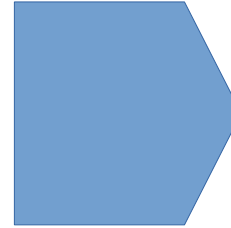


HTTP



REST

- Camada de comunicação
- Interface
- Modelo de dados



HTTP

XML ???



REST

- XML

- Verborrágico
- Difícil de ler para humanos
- Difícil de processar por software

```
<?xml version="1.0" encoding="utf-8"?>
```

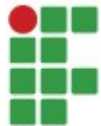
```
<produto>
```

```
  <id>1</id>
```

```
  <nome>Camiseta</nome>
```

```
  <valor>28.9</valor>
```

```
</produto>
```



REST

- JSON - JavaScript Object Notation

- Enxuto
- Sintaxe simples
- Fácil de processar
- Composto de pares atributo-valor

```
{"id":1, "nome":"Camiseta", "valor":28.9}
```



Json

- **JavaScript Object Notation**

- Notação de Objetos JavaScript

- **Formatação leve de troca de dados**

- Formato texto

- Independente de linguagem

- Para seres humanos, é fácil de ler e escrever

- Para máquinas, é fácil de interpretar e gerar

- <https://www.json.org/json-pt.html>



Json

- **Formado por tipos simples e duas estruturas**
- **Tipos simples**
 - Número, texto, lógico e nulo
- **Coleção de pares chave/valor**
 - Similar a objetos sem métodos
- **Lista ordenada de valores**
 - Similar a um array, vetor, lista ou sequência



Json

```
{  
  "nome":"Maria do Exemplo",  
  "idade":29,  
  "telefones":["6515814","6584684"],  
  "endereco":{  
    "logradouro":"Rua João da silva",  
    "numero":123,  
    "bairro":"centro",  
    "cidade":"Porto Alegre",  
    "cep":"93510-123"  
  }  
}
```

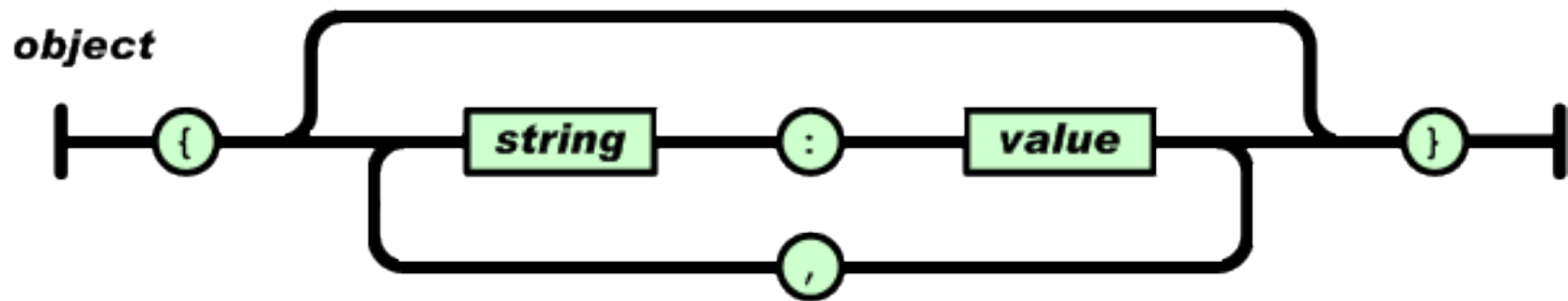


Json - Objeto

- Um objeto é um conjunto desordenado de pares nome/valor.
- Um objeto começa com { (chave de abertura) e termina com } (chave de fechamento).
- Cada nome é seguido por : (dois pontos) e os pares nome/valor são seguidos por , (vírgula).



Json - Objeto

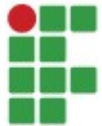
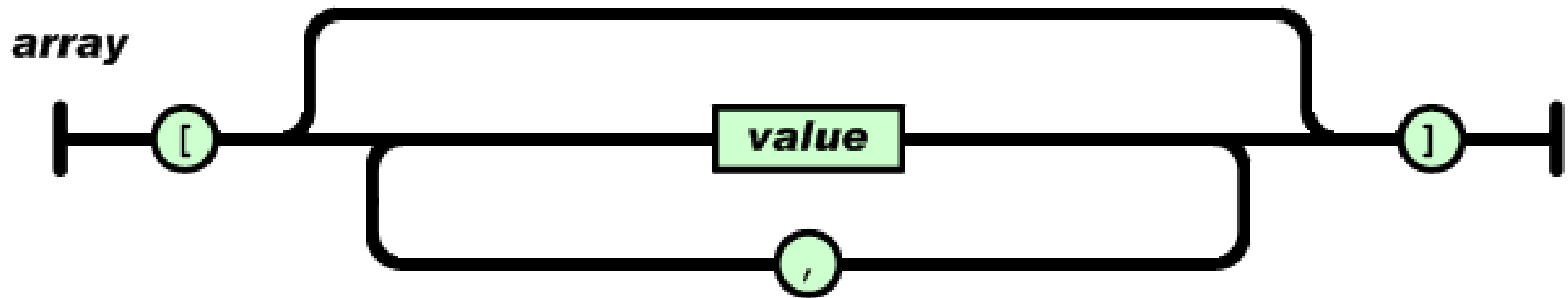


Json - Array

- Uma array é uma coleção de valores ordenados.
- O array começa com [(conchete de abertura) e termina com] (conchete de fechamento).
- Os valores são separados por , (vírgula).



Json - Array



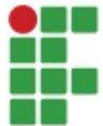
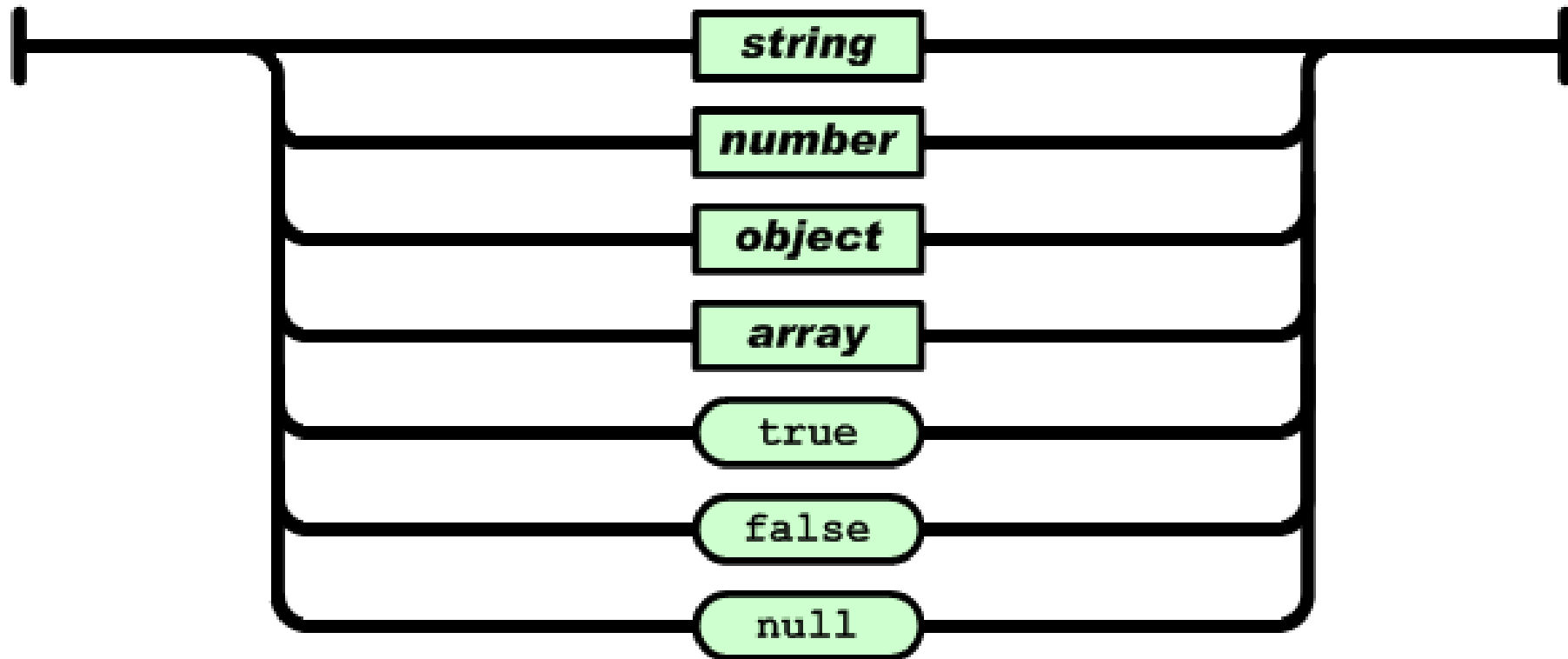
Json - Valor

- Um valor (value) pode ser uma cadeia de caracteres (string), ou um número, ou true ou false, ou null, ou um objeto ou uma array.



Json - Valor

value

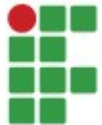
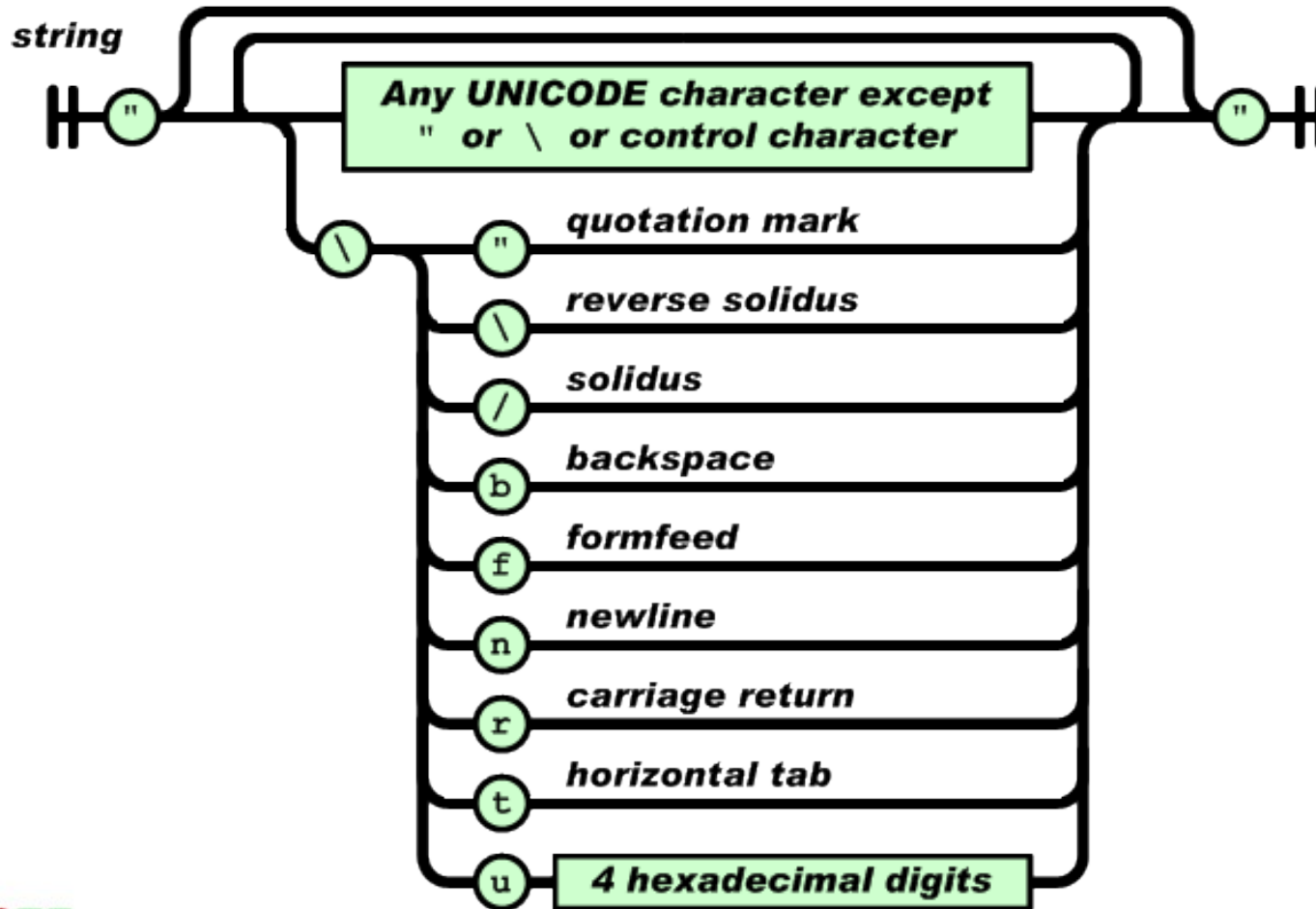


Json - String

- Uma string é uma coleção de nenhum ou mais caracteres Unicode, envolvido entre aspas duplas usando barras invertidas como caracter de escape.
- Uma string json é parecida com uma string em C ou Java.



Json - String



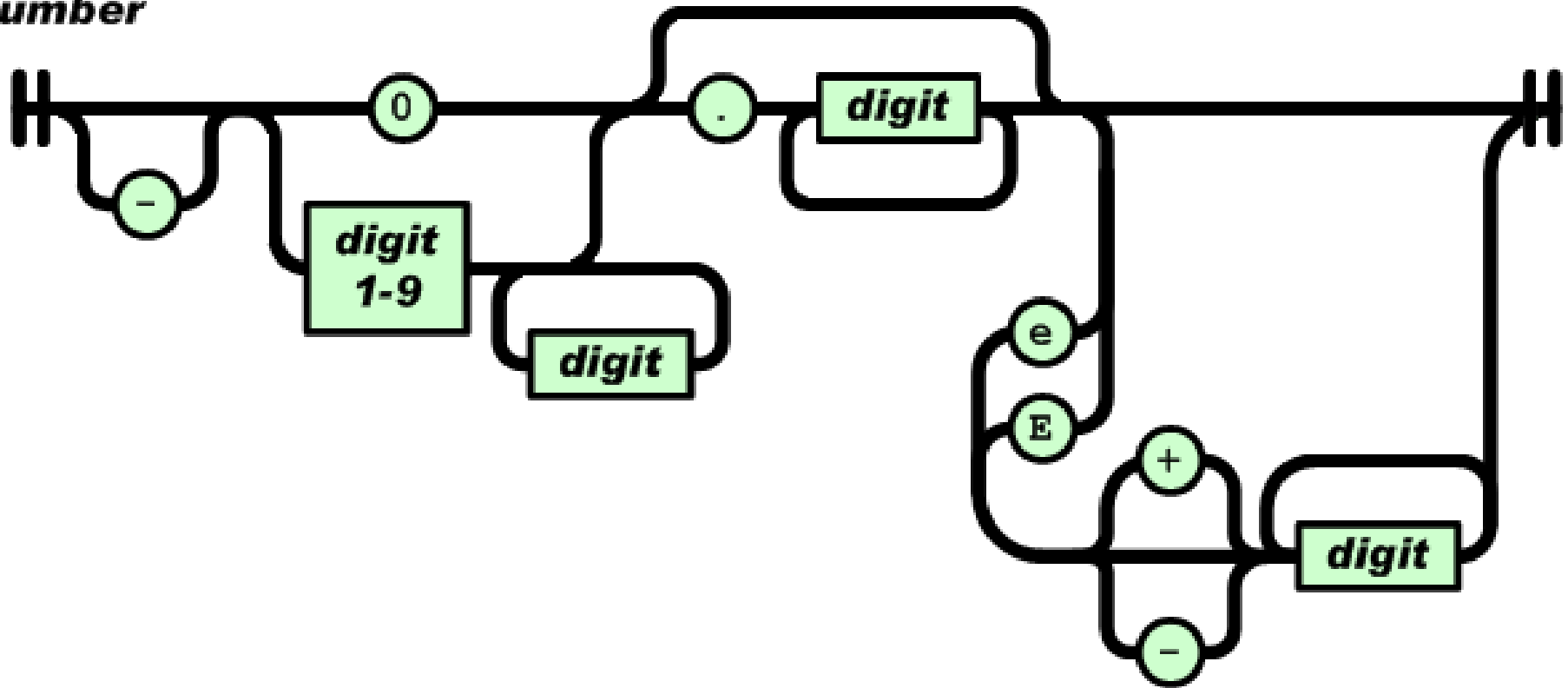
Json - Número

- Um número é similar a um número em C ou Java
- Aceita somente decimais, não tem suporte a números octais ou hexadecimais.



Json - Número

number







Json

```
{  
  "nome":"Maria do Exemplo",  
  "idade":29,  
  "telefones":["6515814","6584684"],  
  "endereco":{  
    "logradouro":"Rua João da silva",  
    "numero":123,  
    "bairro":"centro",  
    "cidade":"Porto Alegre",  
    "cep":"93510-123"  
  }  
}
```



REST - Exemplo

v		produtos
		id : int(11)
		nome : varchar(100)
		valor : float

{"id":1, "nome":"Camiseta", "valor":28.9}



REST - API

- URI

- <URL>/produtos/

- Interfaces

- GET - lista ou pesquisa produto

- POST - cria novo produto

- PUT - atualiza um produto

- DELETE - apaga um produto



REST

URL	Método (verbo)			
	GET	POST	PUT	DELETE
http://localhost:8080/produtos/	Lista todos os produtos	Cria um novo produto	Substitui a lista inteira de produtos	Apaga a lista inteira de produtos
http://localhost:8080/produtos/15	Retorna o produto com a id = 15	Não utilizado	Altera os dados do produto com a id = 15	Apaga o produto com a id = 15



REST - GET

- Requisição (http://localhost:8080/produtos/)

GET /produtos/ HTTP/1.1

host: localhost:8080

- Resposta

HTTP/1.1 200 OK

status: 200

content-type: application/json;charset=UTF-8

```
[{"id":1,"nome":"Camiseta","valor":28.9},  
{"id":2,"nome":"Calça","valor":90.0},  
{"id":3,"nome":"Bermuda","valor":52.2}]
```



REST - GET

- Requisição (http://localhost:8080/produtos/2)

GET /produtos/2 HTTP/1.1

host: localhost:8080

- Resposta

HTTP/1.1 200 OK

status: 200

content-type: application/json;charset=UTF-8

{"id":2,"nome":"Calça","valor":90.0}



REST - POST

- Requisição (http://localhost:8080/produtos/)

POST /produtos/ HTTP/1.1

host: localhost:8080

content-type: application/json

content-length: 31

{"nome":"blusão", "valor":150}

- Resposta

HTTP/1.1 201 CREATE

status: 201

content-type: application/json;charset=UTF-8

{"id":4,"nome":"blusão","valor":150.0}



REST - PUT

- Requisição (http://localhost:8080/produtos/2)

PUT /produtos/2 HTTP/1.1

host: localhost:8080

content-type: application/json

content-length: 47

{ "nome": "Calça Jeans", "valor": 90 }

- Resposta

HTTP/1.1 204 No Content

status: 204



REST - DELETE

- Requisição (http://localhost:8080/produtos/2)

DELETE /produtos/2 HTTP/1.1

host: localhost:8080

- Resposta

HTTP/1.1 204 No Content

status: 204



REST - Spring

```
@RestController  
public class Rest {
```

```
    @RequestMapping(path = "/produtos/", method = RequestMethod.GET)  
    public ArrayList<Produto> listar() {  
        return produtos;  
    }
```

```
    @RequestMapping(path = "/produtos/{id}", method = RequestMethod.GET)  
    public Produto recuperar(@PathVariable int id) {  
        for (Produto produto : produtos) {  
            if (produto.getId() == id) {  
                return produto;  
            }  
        }  
        return null;  
    }
```

