
Sistemas Web III

REST

Tiago Cruz de França

tcruz.franca@gmail.com

REST

@ REpresentational State Transfer

- @ Aqui você encontra a descrição inicial sobre o REST:
<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

@ Quem usa

- @ Grandes empresas como Google, Yahoo, Facebook, Twitter
 - ▶ Sim, nosso cliente Twitter que publica e pega dados no Twitter faz isso usando os RECURSOS da API REST do Twitter

REST

@ É orientado a recurso

- @ Sua aplicação disponibiliza **recursos**
- @ Para disponibilizar recursos, o REST define um conjunto de **princípios arquiteturais**
 - ▶ Esses princípios arquiteturais vão orientá-lo na disponibilização dos seus recursos
- @ Não depende de uma linguagem de programação
 - ▶ Nós vamos usar Java para implementar
- @ **Sistemas cliente-servidor**
 - ▶ Cliente(s) e servidor não precisam usar as mesmas tecnologias de implementação

REST

- Ⓜ **Existem outras abordagens (arquiteturas e princípios) para se desenvolver Web Services, MAS**
 - Ⓜ REST tem sido amplamente adotado
 - ▶ Antes do REST, o SOAP (outra abordagem) figurava como abordagem mais proeminente de desenvolvimento de Web Services
- Ⓜ **O REST não atraiu muita atenção em 2000 quando Roy Fielding o introduziu**
 - Ⓜ Mas ele ganhou notoriedade nos últimos anos

REST

@ Existem 4 princípios REST básicos

- ⌚ Uso de métodos do protocolo para definir o CRUD da aplicação
 - ▶ HTTP permite isso (POST, GET, PUT, DELETE)
 - E ainda tem o OPTIONS e HEAD
- ⌚ Não manter estado (*stateless*)
- ⌚ Expor recursos na forma de URIs
 - ▶ Fáceis de lerem e serem relacionadas a funcionalidade em questão
 - ▶ Evite verbos, use substantivos na identificação do recurso
 - Ex: meuendereco.ufrr.br/**encontrar/tiago** (errado)
 - Ex: meuendereco.ufrr.br/**usuario/tiago** (certo)
- ⌚ Suportar envio de qualquer tipo de representação de recurso
 - ▶ Principalmente XML e/ou JSON

Interface bem Definida

@ A principal característica

- @ As funcionalidades são vinculadas ao método do protocolo
 - ▶ Na prática, do protocolo HTTP, visto que estamos falando de Web

CRUD	Método HTTP	Descrição da Funcionalidade baseada no método
CREATE	POST	Criar um recurso no servidor
READ	GET	Recuperar um recurso (texto, imagem, etc.)
UPDATE	PUT	Mudar o estado de um recurso ou atualizá-lo
DELETE	DELETE	Apagar um recurso

- @ Muitas aplicações Web não respeitam o uso dos métodos
 - ▶ Ex: usar o GET para criar um novo recurso no servidor
 - E ainda se dizem REST

Exemplo de Interface que não segue os Princípios REST

@ GET para recuperar informações

@ Ex. de requisição que não segue os princípios REST

```
GET /adduser?name=Tiago HTTP/1.1
```

- ▶ **Observe o nome do recurso**
 - Adicionar usuário com método GET?
- ▶ **Apesar ser simples de entender, essa URL não segue os princípios REST como deveria**
 - Considere também a existência de cache que retornariam uma resposta ao usuário sem realizar a inserção de um novo usuário
- ▶ **A semântica da interface bem definida REST não foi seguida**
 - GET é usado para recuperar recurso, não para criar, alterar ou apagar

Exemplo de Interface que não segue os Princípios REST

@ GET é pra consulta (recuperação)

- @ Evite problemas com cache, *crawlers*, mecanismos de consultas...

@ A inserção anterior poderia ser feita assim:

```
POST /users HTTP/1.1
Host: myserver
Content-Type:
application/xml
<?xml version="1.0"?>
<user>
  <name>Tiago</name>
</user>
```

Método: POST

Conteúdo a ser inserido no corpo da mensagem

Foi usada uma representação XML

Interface Segundo os Princípios REST

@ Uma requisição realmente RESTfull

@ POST para inserção

▶ *Payload no corpo da mensagem*

- Não na URL, como quando usamos o GET

```
POST /users HTTP/1.1  
Host: myserver  
Content-Type: application/xml
```

```
<?xml version="1.0"?>  
<user>  
  <name>Tiago</name>  
</user>
```

@ /users identifica o recurso

@ Método **POST** define a operação a ser realizada (inserção)

@ **XML do corpo da mensagem HTTP**

- #### ▶ *Informações a serem inseridas (poderia usar outra representação)*

Interface Segundo os Princípios REST

@ Acessando novo usuário (recurso) recém criado

- @ Novo recurso usuário “Tiago” pode ser, depois de criado, acessado com uma URI lógica
- @ Veja a requisição

```
GET /users/Tiago HTTP/1.1  
Host: myserver  
Accept: application/xml
```

- @ A consulta (recuperação) deste recurso é idempotente
 - Ou seja, pode ser realizada várias vezes e o estado do recurso não será alterado

Interface Segundo os Princípios REST

@ Atualizando o recurso respeitando o princípio de interface REST

```
PUT /users/Tiago HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Franca</name>
</user>
```

- @ Observe que o recurso é o mesmo (/users/Tiago) no mesmo servidor da busca
 - ▶ Observe o método PUT e o novo nome no XML passado no corpo da mensagem HTTP
 - ▶ Recurso passa a ser acessado por meio da URL /users/Franca

Interface Segundo os Princípios REST

@ Atualizando o recurso respeitando o princípio de interface REST

```
PUT /users/Tiago HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Franca</name>
</user>
```

Identifica o Recurso a ser recuperado para ser alterado

@ Observe que o recurso é o mesmo (/users/Tiago) no mesmo servidor da busca


- ▶ Observe o método PUT e o novo nome no XML passado no corpo da mensagem HTTP
- ▶ Recurso passa a ser acessado por meio da URL /users/Franca

Interface Segundo os Princípios REST

@ Atualizando o recurso respeitando o princípio de interface REST

```
PUT /users/Tiago HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Franca</name>
</user>
```

Novo valor para o recurso em valor passado no corpo da mensagem



@ Observe que o recurso é o mesmo (/users/Tiago) no mesmo servidor da busca

- ▶ Observe o método PUT e o novo nome no XML passado no corpo da mensagem HTTP
- ▶ Recurso passa a ser acessado por meio da URL /users/Franca

Interface Segundo os Princípios REST

@ Usar o PUT para o UPDATE

- @ Interface semântica e adequada de acordo com os princípios REST
- @ Evita problemas que a adoção do GET poderia trazer
 - ▶ Semelhante aos problemas exemplificados para inserção
- @ Após a alteração, o acesso ao recurso com a antiga URI irá gerar um erro do tipo 404-Recurso não Encontrado
- @ O corpo da mensagem transfere o novo estado de um recurso (POST, carrega o estado do recurso a ser criado)
 - ▶ Dispensa a necessidade de novos recursos (ex: /updateusers)
 - Basta indicar /users indicando o método POST

Não Manter Estado (*stateless*)

@ Melhorar o desempenho e ser escalável

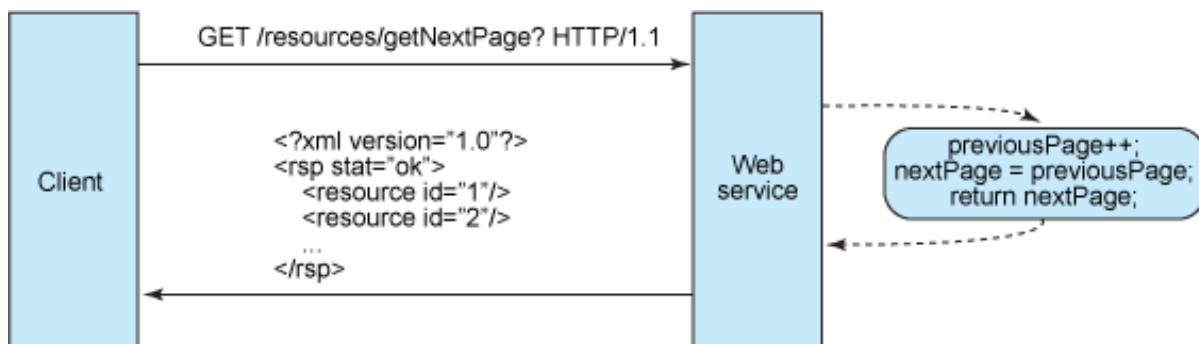
@ Suportar grandes demandas

▶ Reduzir o uso de recursos do servidor

@ Demanda pode ser distribuída entre vários servidores

▶ Requisições são autocontidas, qualquer servidor de um cluster pode tratar a requisição

– Menor complexidade para a distribuição do serviço



Ex. visão cliente:

1. faça um busca no Google
2. Copie a URL e cole em outro navegador

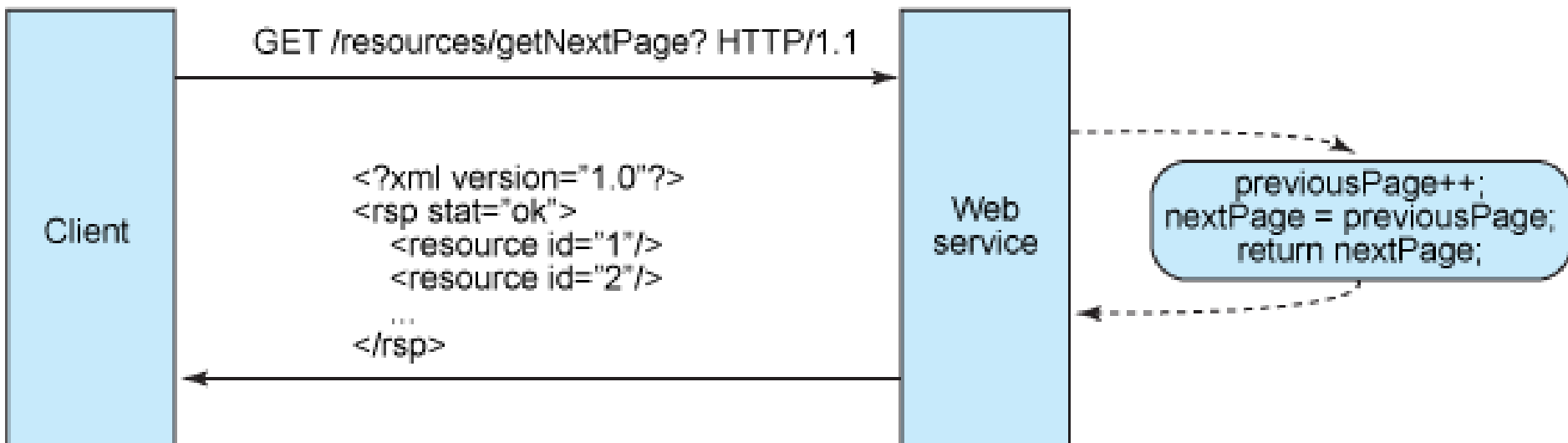
Ex. visão servidor:

1. Qualquer servidor pode responder a requisição com tal URL

Fonte da imagem: <http://www.ibm.com/developerworks/library/ws-restful/>

Não Manter Estado (*stateless*)

Toda informação necessário para ser tratada no servidor é enviada na requisição (na URL ou no corpo da mensagem)



Fonte da imagem: <http://www.ibm.com/developerworks/library/ws-restful/>

Não Manter Estado (*stateless*)

Qualquer servidor de um cluster
poder tratar a requisição de
forma apropriada, visto que todo
conteúdo necessário pode ser
encontrado na requisição

Consultas ou exclusões na URL

Criação ou alteração no corpo da
requisição

GET /resources/getNextPage? HTTP/1.1

```
<?xml version="1.0"?>
<rsp stat="ok">
  <resource id="1"/>
  <resource id="2"/>
  ...
</rsp>
```

Web
service

```
previousPage++;
nextPage = previousPage;
return nextPage;
```

Fonte da imagem: <http://www.ibm.com/developerworks/library/ws-restful/>

Não Manter Estado (*stateless*)

@ Serviço *statefull*

@ Totalmente sem manutenção de estado

Servidor não precisa guardar, gerenciar e sincronizar sessão.

GET /...

Client

```
<?xml version="1.0"?>
<rsp stat="ok">
  <resource id="1"/>
  <resource id="2"/>
  ...
</rsp>
```

Web service

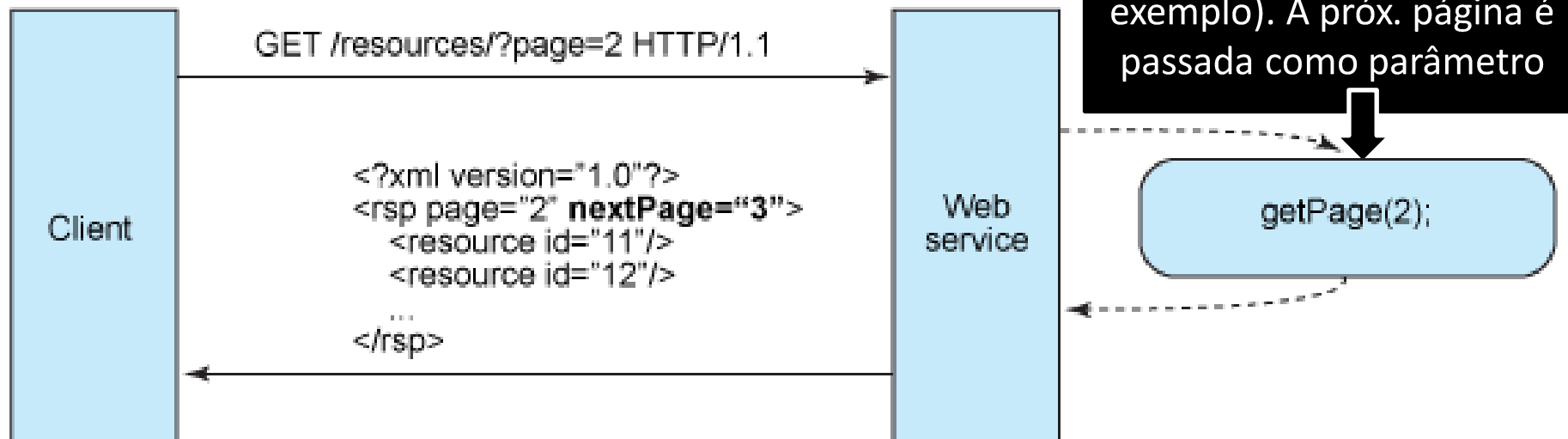
```
previousPage++;
nextPage = previousPage;
return nextPage;
```

Fonte da imagem: <http://www.ibm.com/developerworks/library/ws-restful/>

Não Manter Estado (*stateless*)

@ Serviço *statefull*

@ Totalmente sem manutenção de estado



Fonte da imagem: <http://www.ibm.com/developerworks/library/ws-restful/>

Não Manter Estado (*stateless*)

@ Características do servidor

- @ Recursos retornados para o cliente possuem *links* para outros recursos
 - ▶ Navegabilidade
- @ Recursos recebem indicação se devem ser armazenados em cache
 - ▶ Possibilidade de redução de requisição para o mesmo recurso
 - Armazena pelo menos parte dos recursos
 - ▶ Mecanismo em cache pode ser tratado com GET condicional do HTTP

Não Manter Estado (*stateless*)

@ Características do cliente

- @ Matém cópia de um recurso no cache local com base na informação de controle de cache do cabeçalho da resposta HTTP
 - ▶ Envia mensagens em GET condicional
- @ Envia requisições autocontidas e independentes
 - ▶ Usa o cabeçalho e corpo das mensagens para seguir os princípios REST de forma adequada
 - Requisições não devem depender de qualquer informação sobre o requisições prévias mantidas no servidor

Recursos são Disponibilizados por URIs

@ URIs devem ser intuitivas e fáceis

- @ Desenvolvedor tem que ser criativo e preciso
- @ A URI define a interface da aplicação
 - ▶ URIs bem definidas facilitarão o entendimento dos recursos disponíveis no sistema
 - ▶ Dicas:
 - Use estruturas hierárquicas para as URIs do sistema
 - Ex:

`http://www.ufrrj.br/demat/si/alunos/fulano`

- » No departamento de matemática, curso de sistema de informação, entre os alunos, aquele com nome “fulano”

Recursos são Disponibilizados por URIs

@ Dicas sobre URI

- @ Oculte as extensões dos scripts do servidor
 - ▶ Ex: .php, .asp, etc.
 - ▶ Você pode mudar a tecnologia sem mudar a interface (URIs)
- @ Mantenha tudo em minúscula
- @ Substitua espaços em branco por “-” ou “_”
 - ▶ Padronize: use um ou outro
- @ Evite *strings* de busca o máximo possível
- @ Use páginas de erro padronizadas para substituir erros 404
 - ▶ Ao invés de informar que o recurso não foi encontrado, use uma página com uma mensagem padrão

Representações das Respostas

@ XML e JSON são formatos comuns

@ Use um ou ambos

- ▶ Usuário pode especificar qual tipo de retorno deseja receber na resposta (se indicar ambos, adote um padrão, como, por exemplo, XML antes de JSON)

```
<?xml version="1.0"?>
<discussion date="{date}" topic="{topic}">
  <comment>{comment}</comment>
  <replies>
    <reply from="joe@mail.com" href="/discussion/topics/{topic}/joe"/>
    <reply from="bob@mail.com" href="/discussion/topics/{topic}/bob"/>
  </replies>
</discussion>
```

@ Use qualquer representação

@ Isso é possível (um HTML, por exemplo)

Respostas: além da representação

@ **Usuário pode também indicar outras preferências**

@ Ex: se o recurso requisitado está disponível em mais de um idioma, usuário pode especificar qual deseja receber

► Observe que, se nenhum idioma for especificado, um idioma padrão deve ser definido

— A resposta a requisição é retornada com o idioma padrão

@ **Um recurso recuperado pelo cliente representa o estado daquele recurso**

@ Estado do recurso pode ser a idade de um aluno calculada a partir da data de seu nascimento

Conclusão

@ Entre os princípios, há

- @ A interface é auto-descritiva
 - ▶ As URIs (e, frequentemente, descrições textuais) são suficientes para criação do cliente
- @ Propostas para adoção de descritores de serviços
 - ▶ No caso do REST, o WADL
 - Web Application Description Language
 - ▶ Não é muito aceito, pois vai contra a filosofia do REST

@ REST facilita a construção de *mashups* Web

- @ Ex: app que usa twitter + googlemaps
 - ▶ Existem ferramentas para criação de mashups Web

Refs

- ① <http://www.ibm.com/developerworks/library/ws-restful/>
- ① <http://www.infoq.com/br/articles/rest-introduction>
- ① <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>
- ① <http://www.oracle.com/technetwork/articles/javase/index-137171.html>
- ① [http://shop.oreilly.com/product/9780596529260.d
o](http://shop.oreilly.com/product/9780596529260.do)