ROTAS & REST

ACH2006 – ENGENHARIA DE SISTEMAS DE INFORMAÇÃO
SIN5005 – TÓPICOS EM ENGENHARIA DE SOFTWARE

Daniel Cordeiro

Escola de Artes, Ciências e Humanidades | EACH | USP

REST (REPRESENTATIONAL STATE TRANSFER) — R. FIELDING, 2000

- Ideia: URI são nomes de *recursos*, não páginas ou ações
  - autocontido: qual o recurso, o que fazer com ele?
  - respostas incluem *hyperlinks* que permitem descobrir novos recursos RESTful
  - “uma descrição *post hoc* (depois do fato) das funcionalidades que fizeram a Web ser tão bem sucedida”
- Um serviço (no sentido de SOA) cujas operações seguem esses princípios é chamado de serviço RESTful
- Idealmente, URIs RESTful dão nome às operações

O QUE É REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

O QUE É REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

1. Um *estilo arquitetural* para definição de sistemas fracamente acoplados
   - definido por um conjunto de restrições gerais (princípios)
   - a Web (URL/HTTP/HTML/XML) é uma instância desse estilo

O QUE É REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

1. Um *estilo arquitetural* para definição de sistemas fracamente acoplados
   - definido por um conjunto de restrições gerais (princípios)
   - a Web (URL/HTTP/HTML/XML) é uma instância desse estilo
2. A Web usada corretamente (ou seja, não como um protocolo de transporte)
   - HTTP foi construído usando princípios RESTful
   - serviços que são construídos usando padrões Web sem utilizá-los incorretamente
   - mais importante, HTTP é um protocolo de aplicação (e não de transporte)

O QUE É REST?

REST (*Representational State Transfer* ou Transferência de Estado Representacional) possui 3 definições populares:

1. Um *estilo arquitetural* para definição de sistemas fracamente acoplados
   - definido por um conjunto de restrições gerais (princípios)
   - a Web (URL/HTTP/HTML/XML) é uma instância desse estilo
2. A Web usada corretamente (ou seja, não como um protocolo de transporte)
   - HTTP foi construído usando princípios RESTful
   - serviços que são construídos usando padrões Web sem utilizá-los incorretamente
   - mais importante, HTTP é um protocolo de aplicação (e não de transporte)
3. Qualquer coisa que use HTTP e XML (que não seja SOAP)
   - XML–RPC foi a primeira tentativa para conseguir isso
   - viola REST porque não há interface uniforme

Estilo arquitetural vs. Arquitetura

- Estilo arquitetural: princípios gerais que guiam a criação de uma arquitetura
- Arquitetura: projeto com a solução para um determinado problema dado um conjunto de restrições
- Estilos arquiteturais informam e guiam a criação de arquiteturas

Exemplo
O **Louvre** é um *projeto* arquitetônico cujo *estilo arquitetural* é o estilo **Barroco**.

REST NÃO É ARQUITETURA

- REST é um estilo arquitetural
  - capturado da Web *a posteriori*
  - alguns dos padrões e práticas Web não são perfeitamente RESTful
- A Web é um sistema de informação que segue o REST
- É possível projetar outros sistemas de informação RESTful
  - podem existir interfaces diferentes que sejam uniformes (mas que não usem métodos HTTP)
  - diferentes representações (que não HTML ou XML)
  - diferentes identificadores (que não URIs)

O ESTILO ARQUITETURAL REST

- Um conjunto de restrições que caracterizam uma arquitetura:
  1. Identificação de recursos
  2. Interface uniforme
  3. Mensagens autodescritivas
  4. Estado de aplicação determinado pela hipermídia
  5. Interações sem estado (*stateless*)
- Objetivos: escalabilidade, usabilidade, acessibilidade, facilidade de composição

IDENTIFICAÇÃO DE RECURSOS

- Dê nome a tudo aquilo que você quiser mencionar
- “aquilo” aqui pode se referir a qualquer coisa:
  - produtos em uma loja *online*
  - categorias que são usadas para agrupar produtos
  - clientes, os quais espera-se que irão comprar algo
  - carrinhos de compras, onde os clientes armazenarão seus produtos
- O estado de uma aplicação é representado também como um recurso
  - os links “próximo” em um processo de submissão de múltiplas páginas
  - resultados paginados (à la Gooooooogle) com URIs identificando as páginas seguintes

INTERFACE UNIFORME

- Um mesmo conjunto pequeno de operações se aplica a *tudo*
- Um pequeno conjunto de *verbos* se aplica a um grande conjunto de *substantivos*
  - verbos são universais e não algo inventado para cada aplicação
  - se muitas aplicações precisarem de novos verbos, a interface uniforme pode ser estendida
  - as linguagens naturais funcionam da mesma forma (novos verbos dificilmente são incorporados a uma língua)
- Identifica as operações que são candidatas à serem otimizadas
  - `GET` e `HEAD` são operações seguras
  - `PUT` e `DELETE` são operações idempotentes
  - `POST` pode ter várias possibilidades e, portanto, pode causar efeitos colaterais
- Constrói funcionalidades baseadas nas propriedades úteis dessas operações

MENSAGENS AUTODESCRITIVAS

- Recursos são entidades abstratas (elas não podem ser utilizadas *per se*)
  - *Identificação de recursos* garantem que eles são identificados claramente
  - eles podem ser acessados usando uma *Interface Uniforme*
- Recursos são acessados via *representações de recursos*
  - representações de recursos devem ser suficientes para distinguir um certo recurso
  - comunica qual tipo de representação é usada
  - o formato para a representação pode ser negociado entre os envolvidos
- Representações de recursos podem ser baseados em restrições diferentes
  - XML e JSON podem representar o mesmo modelo para diferentes usuários
  - qualquer que seja a representação, ela deve permitir o uso de *links*

ESTADO DE APLICAÇÃO DETERMINADO PELA HIPERMÍDIA

- Representação de recursos possuem links para recursos identificados
- Recursos e o estado podem ser usados por links navegáveis
  - links tornam recursos interconectados navegáveis
  - sem navegação, a identificação de novos recursos é algo específico do serviço
- Aplicações RESTful *navegam* ao invés de *chamar*
  - representações contém informações sobre formas diferentes de percorrer os links
  - a aplicação navega para o próximo recurso dependendo da semântica do link
  - a navegação pode ser delegada já que todos os links usam identificadores

INTERAÇÕES SEM ESTADO (STATELESS)

- Essa restrição não significa “**Aplicações** sem Estado”!
  - em muitas aplicações RESTful, estado é uma parte essencial
  - a ideia de RESTful é evitar transações longas *nas aplicações*
- “Sem estado” significa mover o estado para os clientes ou recursos
  - consequência mais importante: evitar manter o estado da aplicação no lado do servidor
- O estado do recurso é gerenciado no servidor
  - é o mesmo para todos os clientes que interagem com o serviço
  - quando um cliente mudar o estado do recurso, outros clientes também verão a mudança

INTERAÇÕES SEM ESTADO (STATELESS)

- O estado do cliente é gerenciado no cliente
  - é específico para cada cliente e, portanto, deve ser mantido no cliente
  - pode afetar o *acesso* aos recursos do servidor, mas não afeta os recursos em si
- Problemas com segurança se tornam mais importantes quando o estado está no cliente
  - clientes podem (tentar) enganar o servidor mentindo sobre seu estado
  - manter o estado do cliente no servidor é caro (mas pode valer a pena)

O QUE É A WEB?

- Web = URL + HTTP + (HTML | XML)
- a Web RESTful usa métodos HTTP como sua interface uniforme
  - a web não-RESTful usa **GET**/**POST** e chamadas RPC
  - uma “web RESTful diferente” usa WebDAV
- Imagine sua aplicação sendo utilizada em “10 navegadores”
  - os recursos com os quais ela interage devem ser identificados e “linkados”
  - o tamanho de fonte preferido do usuário pode ser modelado como o estado do cliente
- Imagine sua aplicação sendo utilizada em “10 abas do navegador”
  - não faz diferença contanto que o estado do cliente seja baseado na representação
  - cookies são compartilhados entre as janelas do mesmo navegador

IDENTIFICAÇÃO DE RECURSOS NA WEB

- Essencial para implementar um mecanismo de Identificação de Recursos
- URIs são um esquema de identificação universal legível para identificar “coisas”
  - muitos esquemas de identificação não podem ser lidos por humanos (strings binárias ou hex)
  - muitos sistemas baseados em RPC não possuem esquema de identificação universal para seus objetos
- É importante fazer com que todas as coisas possam ser identificadas unicamente
  - isso evita a necessidade de identificadores não universais usados dentro de um escopo
  - permite referenciar todas as coisas exatamente do mesmo jeito
- Coisas identificadas por URI devem possuir um modo de interação bem definido

PROCESSAMENTO DE URI

- O processamento de uma URI não é tão trivial como parece
  - regras de normalização e escape de caracteres não é trivial
  - muitas implementações não estão corretas
  - se tornam ainda mais complicadas quando se tratam de um *Internationalized Resource Identifier* (IRI)
- URIs não são apenas strings
  - URIs são strings com um conjunto considerável de regras implícitas
  - implementar essas regras não é trivial
  - ... mas é crucial

APLICAÇÕES RESTFUL FALAM HTTP

- HTTP é essencial para a implementação de Interfaces Uniformes
    - HTTP define um conjunto pequeno de métodos para interagir com recursos identificados com uma URI
- Má utilização de HTTP torna uma aplicação não-RESTful
    - ela perde a capacidade de ser utilizada apenas aderindo aos princípios de REST
    - perceber que você precisa de uma linguagem de descrição de interface é um sinal de que há algo errado
- Estender o uso de HTTP transforma sua aplicação em uma aplicação RESTful mais especializada
    - pode ser apropriado quando mais operações são necessárias
    - mas reduz muito o número de potenciais clientes

MÉTODOS HTTP

- *Métodos seguros* podem ser ignorados ou repetidos sem causar efeitos colaterais
  - segurança aritmética: $41 \times 1 \times 1 \times 1 \times 1 \cdots$
  - na prática “sem efeitos colaterais” significa “sem efeitos colaterais significativos”
- *Métodos idempotentes* o resultado devolvido por uma requisição ou por n requisições é o mesmo
  - $41 \times 0 \times 0 \times 0 \times 0 \cdots$
- Métodos inseguros ou que não sejam idempotentes devem ser tratados com cuidado
- HTTP tem dois métodos seguros: `GET` e `HEAD`
- HTTP tem dois métodos idempotentes: `PUT` e `DELETE`

O QUE É IDENTIFICADO POR UMA URI?

- Essencial para implementar Mensagens autodescritivas
  - também devem permitir “estado de aplicação determinado pela hipermídia”
- Identificação de recursos fala sobre um *recurso abstrato*
  - recursos nunca são trocados ou processados diretamente
  - todas as interações usam representação de recursos
- Representações dependem de vários fatores:
  - a natureza do recurso
  - as funcionalidades do servidor
  - as funcionalidades da mídia de comunicação
  - as funcionalidades do cliente
  - requisitos e restrições do domínio da aplicação
  - negociação do “melhor” tipo de representação
- Representações são identificados pelo seu tipo de mídia (MIME type)

O BÁSICO DO BÁSICO SOBRE CONFIG/ROUTES.RB

Rotas: conectam URIs ao código

1. Rails recebe a requisição `get /patients/17`
2. então pergunta ao roteador se essa requisição casa com uma ação de controlador, se o primeiro casamento for algo como: `get '/patients/:id', to: 'patients#show'`
3. a requisição será despachada para a ação `show` do controlador `patients`

Veja guia básico em
`http://guides.rubyonrails.org/routing.html`

CRUD EM UM RECURSO RESTFUL

- Rails permite definir recursos que serão acessíveis usando as operações CRUD.
- Ao invés de criar rotas para `index`, `show`, `new`, `edit`, `create`, `update` e `destroy`, você define uma *resourceful route* com uma linha de código:

  ```
  resources :photos
  ```

- Isso cria 7 rotas diferentes automaticamente:

Verbo HTTP	Path	Controller#Action	Usado para
GET	/photos	photos#index	mostra uma lista com todas as fotos
GET	/photos/new	photos#new	devolve um form HTML para criar uma nova foto
POST	/photos/new	photos#create	cria uma nova foto
GET	/photos/:id	photos#show	mostra uma foto específica
GET	/photos/:id/edit	photos#edit	devolve um form HTML para editar uma foto
PATCH/PUT	/photos/:id	photos#update	atualiza uma foto específica
DELETE	/photos/:id	photos#destroy	apaga uma foto específica