Um guia para a REST e o design de API

ca technologies

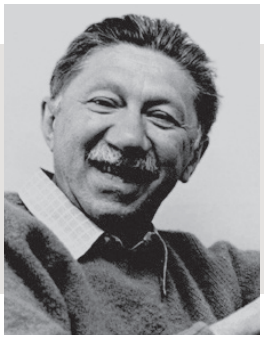
Se a única ferramenta que você tem for um martelo...

Em seu livro de 1966, "The Psychology of Science", o psicólogo Abraham Maslow defendeu a ideia de que no campo da psicologia o tratamento deve ser abordado de várias perspectivas para se obter novas ideias e não apenas continuar a usar as mesmas teorias e técnicas criadas por Freud e seus seguidores há tantos anos. Reconhecendo que pode ser difícil alterar um ponto de vista, Maslow escreveu: "Se você só tem um martelo, é tentador tratar tudo como um prego". Todos nós já tivemos essa experiência. Estamos tão acostumados com a maneira como as coisas eram feitas no passado, que algumas vezes não questionamos as razões de fazê-las.

Essa referência à psicologia em um trabalho sobre REST e Design de API pode ser curiosa, mas ela ajuda a ilustrar dois pontos distintos: (1) que todas as decisões de design, seja no software ou na arquitetura, devem ser feitas dentro do contexto de requisitos funcionais, comportamentais e sociais, não de tendências aleatórias; (2) quando você sabe fazer bem apenas uma coisa, tudo tende a parecer idêntico.

Em sua tese,"Estilos arquitetônicos e o design de arquiteturas de software com base na rede"[1], Roy Fielding define a REST (Representational State Transfer - Transferência de estado representacional): "Com frequência, vemos projetos de software começarem com a adoção da última onda em design arquitetônico e só mais tarde é que se descobre se os requisitos do sistema requerem ou não essa arquitetura."

Não tem tempo para ler toda a tese de Fielding? Não tem problema. Criamos esta visão geral de alto nível especialmente para você. Para começar, vamos examinar a REST detalhadamente.



"Se a única ferramenta que você tem for um martelo, tudo parece ser um prego."

—Abraham Maslow, The Psychology of Science

[1] Estilos arquitetônicos e o design de arquiteturas de software com base na rede

Estilo versus padrão

Um estilo arquitetônico é uma abstração, não algo concreto. Por exemplo, observe uma catedral gótica. A catedral é diferente do estilo arquitetônico gótico. O estilo gótico define os atributos ou as características que você vê em uma catedral construída naquele estilo.

Em comparação, o NIST (National Institute of Standards - Instituto nacional de padrões dos EUA) e o NEC (National Electrical Codes - Códigos elétricos nacionais dos EUA) são órgãos governamentais que produzem regras que reconhecemos como padrões. Se a parte elétrica da construção for feita incorretamente, o lugar pode queimar completamente. As pessoas geralmente confundem *padrões*, dos quais sabemos o que é certo ou errado, e *estilos*, um modo específico de expressão.

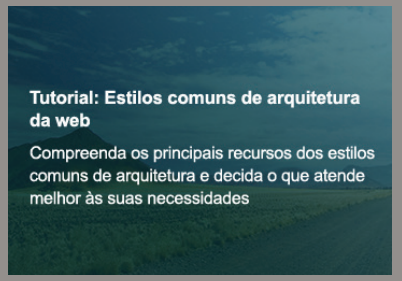
Na internet, a REST é um estilo, e o HTTP é um padrão. A REST conta com protocolos de comunicação sem estado, que podem ser armazenados em cache, como o HTTP, para facilitar o desenvolvimento de aplicativos. Aplicando os princípios do design da REST a um protocolo, como o HTTP, os desenvolvedores podem criar interfaces que podem ser usadas a partir de praticamente qualquer dispositivo ou sistema operacional.

Qual é o seu estilo?

Os três estilos comuns da arquitetura da web são:

- Encapsulamento (protocolo SOAP)
- Objetos (CRUD (Create/Read/Update/Delete - Criar/Ler/Atualizar/Excluir))
- Hipermídia (REST)

Assista a este vídeo[2] para compreender os recursos principais dos estilos arquitetônicos comuns e decidir qual deles atende melhor às suas necessidades.



[2] Design de APIs, Lição 201: Estilos arquitetônicos de APIs da web, Academia de API

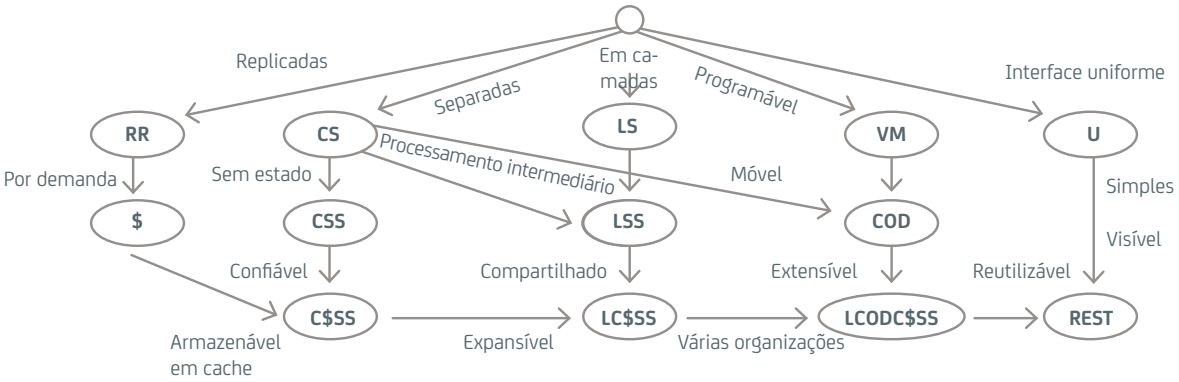
Os estilos são descritos pelas restrições

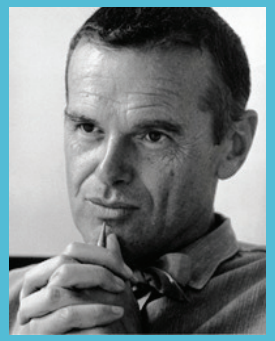
Um estilo arquitetônico é descrito pelos recursos que tornam uma construção ou outra estrutura notável e identificável. As formas características da arquitetura gótica, por exemplo, incluem o arco ogival, a abóboda de arcos cruzados, colunas, torres, pináculos e fachadas decoradas.

De maneira semelhante, a REST é descrita por um conjunto de restrições arquitetônicas que tentam minimizar a latência e as comunicações em rede e, ao mesmo tempo, maximizam a independência e a escalabilidade de implementações de componentes. As seis restrições da REST incluem:

1. **Cliente/servidor** — requer que um serviço ofereça uma ou mais operações e que os serviços esperem que os clientes solicitem essas operações.
2. **Sem estado** — exige que a comunicação entre o consumidor do serviço (cliente) e o provedor do serviço (servidor) seja sem estado.
3. **Cache** — exige que as respostas sejam rotuladas claramente como armazenáveis em cache ou não armazenáveis em cache.
4. **Interface uniforme** — exige que todos os provedores e consumidores de serviço em uma arquitetura compatível com a REST compartilhem uma única interface do usuário para todas as operações.
5. **Sistema em camadas** — exige a habilidade de adicionar ou remover intermediários em tempo de execução sem interromper o sistema.
6. **Codificação sob demanda (opcional)** — permite que a lógica em clientes (como navegadores da web) seja atualizada independentemente da lógica do servidor usando código executável fornecido de provedores de serviços para consumidores.

Figura: Derivação da REST por restrições de estilos





"Esta é uma das poucas chaves efetivas para o problema do design — a habilidade do designer de reconhecer o máximo de restrições possível — seu desejo e entusiasmo de trabalhar com essas restrições. Restrições de preço, tamanho, potência, equilíbrio, superfície, tempo e assim por diante."[3]

—Charles Eames, Eames Design.

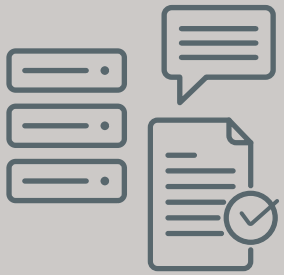
[3] http://www.eamesoffice.com/the-work/design-q-a-text/

Conector ≠ componente

De acordo com Fielding, "a [REST] é obtida colocando as restrições na semântica do conector onde outros estilos se concentraram na semântica do componente". O design de Fielding focaliza a restrição na maneira como as coisas se conectam umas com as outras, não na maneira como elas funcionam internamente, e ele aplica essa teoria a toda a rede. Ao criar aplicativos em grande escala, o conceito de que o conector não é igual ao componente sempre é negligenciado. Mas Fielding traz o conceito para o primeiro plano.

Quais são os componentes? Eles incluem:

- Bancos de dados
- Sistemas de arquivos
- Filas de mensagens
- Ferramentas de gerenciamento de transações
- Código-fonte



Esses diferem dos conectores que incluem:

- Servidores web
- Agentes de navegador
- Servidores proxy
- Cache compartilhado

Os componentes funcionam de maneiras exclusivas para resolver problemas. O MySQL funciona de maneira diferente do SQL Server, assim como o CouchDB ou o MongoDB. O mesmo pode ser dito sobre sistemas de arquivos que têm como base o UNIX, o Windows ou o Mac. A maneira como você coloca informações na fila e a maneira como você decide quando começa e quando termina uma transação são recursos locais do componente que podem ser manipulados pelo desenvolvedor. Eles são componentes do desenvolvedor, seu sistema operacional, ferramentas e linguagem, e são, portanto, privados.

Por outro lado, os conectores são públicos. Os conectores são uma série de pipes padronizados com os quais todos os desenvolvedores trabalham. Com base no princípio de Fielding, os desenvolvedores podem ser tão criativos ou mundanos quanto desejarem dentro de seus componentes privados, desde que concordem em receber e enviar informações usando conectores públicos padronizados.

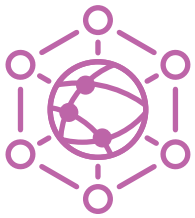
Mantenha os componentes e os conectores separados, facilitando seu intercâmbio mais tarde. Por exemplo, o código que você escreve para seu servidor web é criado para falar com muitos dispositivos na internet pública. Mas, o código que você escreve para seus componentes é criado para falar especificamente com as ferramentas que estão disponíveis para você.

Garantindo que os conectores funcionem em conjunto

Ao criar aplicativos com base no cliente ou serviços no servidor, o que torna o desenvolvimento desafiante e empolgante é a correspondência dos componentes privados com os conectores públicos, conectando-os e encadeando-os. Como você garante que um conector funcione? Como criar aplicativos expansíveis se eles estão se comunicando por meio de conectores?

Identificação de recursos — URIs, URLs e URNs como identificadores

Representações de recursos — tipos de mídia como maneiras de representar informações transmitidas entre partes

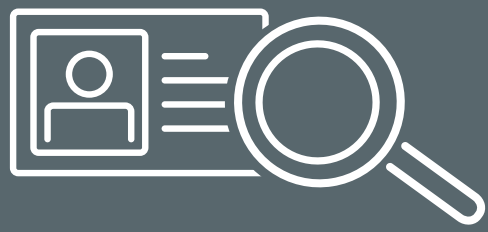


Mensagens autodescritivas — combinando metadados em cabeçalhos, bem como o corpo de uma mensagem, para criar uma resposta autodescritiva

Hipermídia — links e formulários como uma maneira de descrever para o cliente as ações disponíveis que são suportadas atualmente pelo serviço

A restrição da interface uniforme é fundamental para a criação de qualquer serviço REST. Ela simplifica e separa os conectores, o que permite que cada parte evolua de maneira independente. Por causa da maneira como a web é usada atualmente, as quatro restrições acima são as ferramentas essenciais que ajudam os desenvolvedores a obter a interface uniforme de Fielding. As próximas páginas explicam essas ferramentas mais profundamente.

URIs para identificação

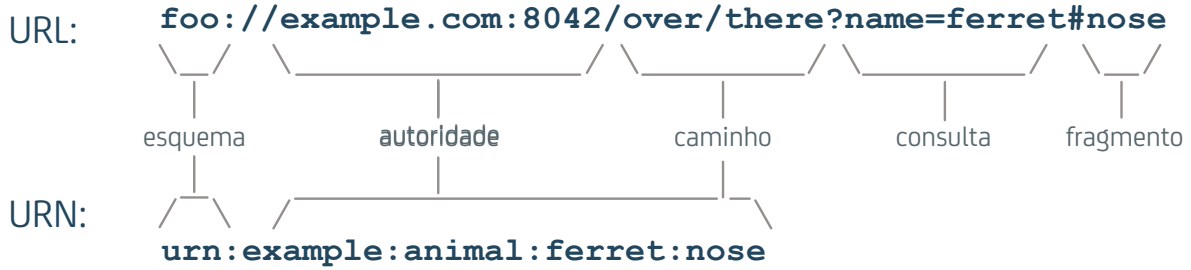


Como a RFC2396[4] descreve, "um URI (Uniform Resource Identifier) é uma cadeia de caracteres compacta para identificar um recurso abstrato ou físico". Esse identificador pode ser obtido de duas maneiras, uma URL (Uniform Resource Locator) ou um URN (Uniform Resouce Name). As URLs (por exemplo, http://example.org/users/mike) são usadas para identificar o local online de um recurso individual, enquanto os URNs (por exemplo, urn:user:mike) são identificadores persistentes independentes de local. O URN funciona como o nome de uma pessoa, e uma URL é semelhante ao endereço dessa pessoa. Ou seja, o URN define a identidade de um item ("o nome do usuário é mike"), e a URL fornece um método de localizá-lo ("mike pode ser localizado em example.org/users/").

Os componentes de um URI incluem:

- **Nome do esquema** — identifica o protocolo (por exemplo, FTP:, HTTP:, HTTPS:, IRC:)
- **Parte hierárquica** — destinada a manter informações hierárquicas por natureza
  - **Autoridade** — refere-se à resolução real do DNS do servidor (por exemplo, o nome do domínio ou o endereço IP)
  - **Caminho** — refere-se a uma sequência de segmentos separados por uma barra ("/")
- **Consulta** — contém informações adicionais que não são hierárquicas por natureza e geralmente são separadas por um ponto de interrogação ("?")
- **Fragmento** — fornece direção para um recurso secundário dentro do primário identificado pela Autoridade e o Caminho separados do restante por um ("#")

Estrutura dos URIs



[4] https://tools.ietf.org/html/rfc2396

Tipos de mídia para representação

De acordo com a RFC2046[5], os identificadores do tipo MIME (tipos de mídia) devem ser usados para "especificar a natureza dos dados no corpo de uma entidade MIME, junto com quaisquer informações auxiliares que possam ser necessárias". Os tipos MIME foram usados primeiro para transmissões de email, como é evidenciado por seu nome completo: Multipurpose Internet Mail Extensions. Atualmente, os tipos MIME permitem que as pessoas troquem diferentes tipos de dados pela internet: áudio, vídeo, texto, imagens e programas de aplicativos.

Os tipos MIME (ou tipos de mídia) identificam a natureza dos dados e as informações auxiliares. Na web, os tipos de mídia também identificam regras de processamento para a mensagem. A cadeia de caracteres do identificador do tipo MIME tem um tipo e um subtipo separados por uma barra (por exemplo, texto/simples, imagem/gif etc.).

Além das cadeias de caracteres do tipo MIME padrão (por exemplo, aplicativo/json), os identificadores podem ser criados usando as seguintes convenções:

- Use vnd. como um prefixo para o subtipo dos tipos MIME específicos ao fornecedor que fazem parte de um produto comercial (por exemplo, vnd.bigcompany.report/json).
- Use prs. como um prefixo para o subtipo de tipos MIME pessoais/personalizados que não fazem parte de um produto comercial (por exemplo, prs.smith.data/json).

Registro do tipo MIME

Uma lista completa dos tipos MIME oficiais atribuídos pela IANA (Internet Assigned Number Authority - Autoridade para Atribuição de Números da Internet) pode ser encontrada aqui.

[5] https://tools.ietf.org/html/rfc2396

Cabeçalhos+corpo para mensagens autodescritivas

A RFC2616[6] declara que [no HTTP] as mensagens consistem em uma linha de início, zero ou mais campos de cabeçalho (também conhecidos como "header"), uma linha vazia (por exemplo, uma linha com nada antes do CRFL) indicando o final dos campos de cabeçalho e possivelmente um corpo da mensagem.

Cada solicitação de cliente e resposta de servidor é uma mensagem, e os aplicativos compatíveis com a REST esperam que cada mensagem seja autodescritiva. Isso significa que cada mensagem deve conter todas as informações necessárias para concluir a tarefa. Outras maneiras de descrever esse tipo de mensagem são "sem estado" ou "sem contexto". Cada mensagem passada entre o cliente e o servidor pode ter um corpo e metadados.

As implementações da REST também dependem da noção de um conjunto restrito de operações que são totalmente compreendidas pelo cliente e pelo servidor no início. No HTTP, as operações são descritas na "linha de início", e as seis operações mais amplamente usadas no HTTP são:

- **GET** — retorna qualquer informação identificada pelo URI de solicitação
- **HEAD** — idêntica ao GET, exceto que o servidor não precisa retornar um corpo de mensagem na resposta, apenas os metadados
- **OPTIONS** — retorna informações sobre as opções da comunicação disponíveis na cadeia de solicitação/resposta identificada pelo URI de solicitação
- **PUT** — solicita que a entidade embutida seja armazenada sob o URI de solicitação fornecido
- **POST** — solicita que o servidor de origem aceite a entidade embutida na solicitação como um novo subordinado do recurso identificado pelo URI de solicitação
- **DELETE** — solicita que o servidor de origem exclua o recurso identificado pelo URI de solicitação

As três primeiras são operações somente leitura, enquanto as três últimas são operações de gravação. No HTTP, há regras bem-definidas de como se espera que os clientes e os servidores se comportem ao usar esses operadores. Os nomes e os significados dos elementos que acompanham os metadados (cabeçalhos) também são bem-definidos. Os aplicativos compatíveis com a REST que executam por HTTP compreendem e seguem essas regras muito cuidadosamente.

[6] https://tools.ietf.org/html/rfc2396

Exemplo de troca de "GET" via HTTP

Para recuperar um arquivo em http://www.somehost.com/path/file.html, abra um soquete para o host www.somehost.com, use a porta padrão 80, porque nenhuma está especificada na URL, e envie o seguinte pelo soquete:

```
GET/path/file.html HTTP/1.0
From: someuser@jmarshall.com
User-Agent: HTTPTool/1.0
[linha em branco aqui]
```

Enviado de volta pelo mesmo soquete, o servidor deve responder com:

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Feliz Ano Novo!</h1>
(mais conteúdo do arquivo)
</body>
</html>
```

Links e formulários para hipermídia

Links e formulários são usados em um tipo de mídia para oferecer suporte à restrição de hipermídia de Fielding. Por exemplo, há inúmeras funcionalidades no HTML, e o navegador web comum compreende as regras para todas elas. Os links e formulários em uma mensagem HTML são fáceis de reconhecer, mas o que pode não ser tão claro são as regras de processamento ou as semânticas que estão associadas a eles.

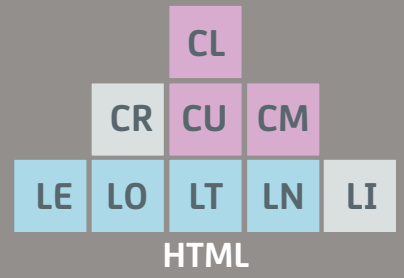
Os links e formulários dão a você a habilidade de executar uma ação. Eles são *funcionalidades*. O HTML tem um conjunto bem-definido de funcionalidades. Algumas funcionalidades permitem que você grave dados, como um formulário que tenha a propriedade de método definida como "POST." Algumas funcionalidades permitem que você extraia dados de um local remoto para exibir dentro do documento HTML atual, com o elemento IMG. O elemento A do HTML é uma funcionalidade para navegação para um novo local na web.

De acordo com Fielding, "a hipermídia é definida pela presença de informações de controle do aplicativo embutidas dentro da apresentação das informações ou em uma camada acima". Para Fielding, a REST oferece aos provedores de serviço a habilidade de enviar informações de controle (links e formulários) a aplicativos de cliente de todo o mundo enviando funcionalidades, que são os controles de hipermídia.

Fatores H

Ao comparar tipos de mídia, pode ser útil documentar os Fatores H existentes em um gráfico visual simples. No exemplo a seguir, a linha inferior identifica fatores básicos de link — os fatores de hipermídia mais notáveis — enquanto as duas linhas superiores identificam fatores de dados de controle.

Fatores de hipermídia



Suporte de links
[LE] Links embutidos
[LO] Links de saída
[LT] Consultas modeladas
[LN] Atualizações não idempotentes
[LI] Atualizações idempotentes

Suporte de dados de controle
[CR] Dados de controle para solicitações de leitura
[CU] Dados de controle para solicitações de atualização
[CM] Dados de controle para métodos da interface
[CL] Dados de controle para links

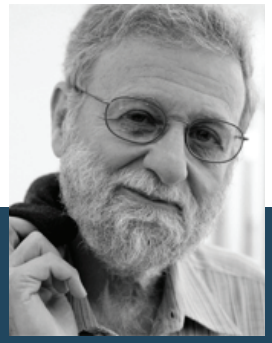
Para obter mais informações sobre Fatores H: http://amundsen.com/hypermedia/hfactor/.

Software que amplia o tempo de vida

A arquitetura física de qualidade amplia os tempos de vida. Prédios que foram construídos há centenas de anos podem ser tão cheios de vida, tão úteis, tão vibrantes e confortáveis hoje como eram quando as pessoas entraram neles pela primeira vez, até mesmo quando esses prédios são transformados em museus ou salões de reunião em prédios de apartamentos. Por quê? Porque esses prédios atemporais contam com padrões arquitetônicos universais que atravessam o tempo e o espaço. Uma entrada é uma entrada, uma janela é uma janela. A maneira como esses elementos são implementados depende dos materiais disponíveis. A maneira como eles são representados é diferente em cada caso local, mas eles ainda podem ser identificados ano após ano.

No desenvolvimento de software, o conceito é o mesmo. Algumas vezes, os arquitetos e desenvolvedores de software desejam criar aplicativos que durem por um longo tempo. A REST, com seu conjunto de restrições universais (como padrões arquitetônicos), é uma maneira de conseguir isso.

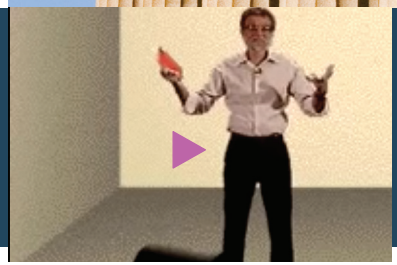
Mas Fielding não disse que essa é a única maneira de obter êxito. Quando ele escreveu sua tese, não incluiu todas as possibilidades, nem todas as respostas. Na verdade, muitas partes foram deixadas inacabadas. O que Fielding fez, no entanto, foi documentar sua abordagem de criar um estilo arquitetônico para software em rede que era baseado na identificação de uma série de restrições para atingir seu objetivo de minimizar a latência e maximizar a escalabilidade. Na verdade, ele usou restrições da mesma maneira como Charles Eames as usou, porque elas ajudaram a atingir seu objetivo.



Podemos tirar proveito da experiência de Fielding para nos ajudar a ver as coisas de maneira um pouco diferente. Podemos usar as ferramentas que ele forneceu para experimentar com restrições e expressar nosso próprio estilo e, no processo, criar aplicativos de software notáveis que podem, se desejarmos, resistir ao teste do tempo.

"O valor de um objeto bem-criado é quando ele tem um conjunto tão rico de funcionalidades que as pessoas que o usam podem fazer coisas que o designer nunca imaginou."[7]

—Donald Norman, 1994



[7] https://www.youtube.com/watch?v=NK1Zb_5VxuM

Saiba mais sobre o CA API Management

Visite **ca.com/br/API**

A CA Technologies (NASDAQ: CA) cria software que acelera a transformação das empresas e permite que elas aproveitem as oportunidades da economia dos aplicativos. O software está no cerne de todas as empresas, em todos os setores. Do planejamento ao desenvolvimento e do gerenciamento à segurança, a CA está trabalhando com empresas de todo o mundo para mudar a maneira como vivemos, fazemos negócios e nos comunicamos – usando dispositivos móveis, as nuvens privada e pública e os ambientes distribuídos e de mainframe. Obtenha mais informações em ca.com/br.

CS200-110010

ca technologies