Webservers e Webservices

Prof.: Maiquel de Brito

BLU3024
CAC - Departamento de Engenharia de Controle, Automação e Computação
UFSC Blumenau

Sumário

Preâmbulo - APIs

APIs

No princípio:

- Resultados da execução de software eram consumidos por humanos através de alguma interface humanamente legível
  - imagem na tela, mensagem etc

APIs

No princípio:

- Resultados da execução de software eram consumidos por humanos através de alguma interface humanamente legível
  - imagem na tela, mensagem etc

A evolução:

- Resultados da execução de software são consumidos por outro software
- Necessidade: interface legível por software

APIs

No princípio:

Resultados da execução de software eram consumidos por humanos através de alguma interface humanamente legível

imagem na tela, mensagem etc

A evolução:

Resultados da execução de software são consumidos por outro software

Necessidade: interface legível por software

API: Application Programmig Interface

Interaface que permite programas de computador interagirem entre si.

APIs

API: Application Programmig Interface

Analogia: tomadas - interface que permite dispositivos diversos consumirem o serviço fornecido pela comanhia de energia

- Serviço pode ser acessado por qualquer consumidor adaptado (dispositivo elétrico, programa de computador);
- Consumidores podem consumir serviços em diferentes locais sem adaptação
- O consumidor não precisa saber detalhes sobre a implementação dos serviço fornecido
- O fornecedor não se preocupa sobre como o consumidor irá usar o serviço

APIs

API - Application Programming Interface

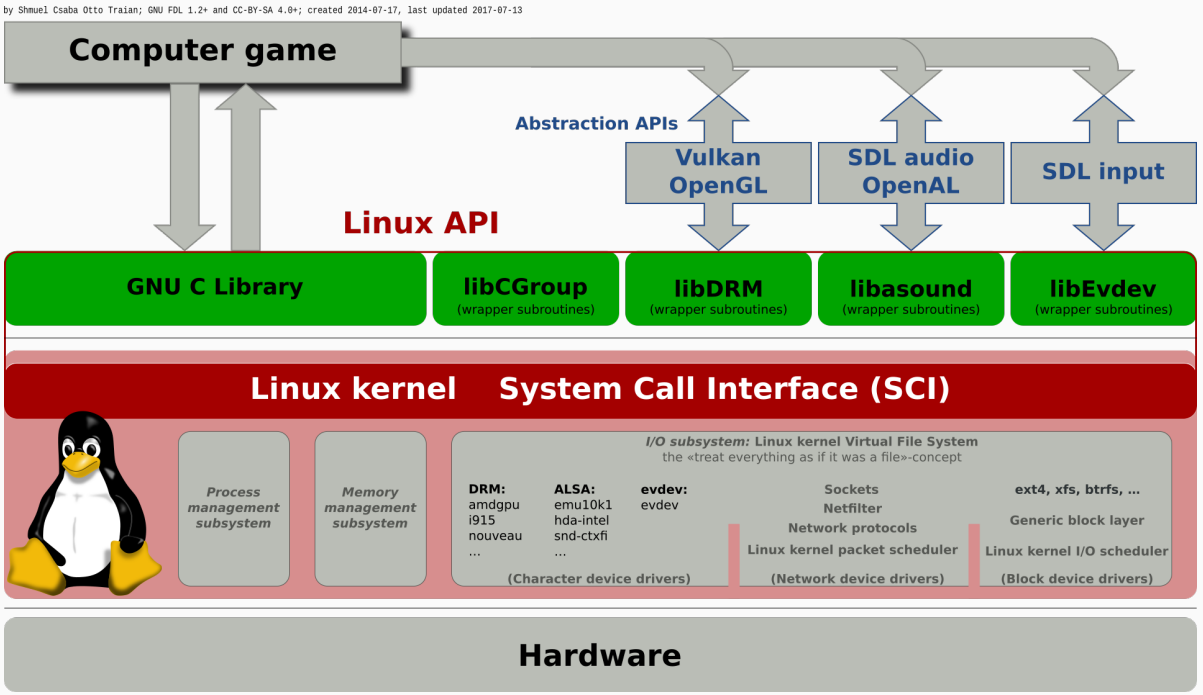
Conjunto de definições, protocolos e funções fornecidas por um software para que suas funcionalidades possam ser utilizadas por outros sistemas.

Permitem que um software acesse funções de outro software sem saber como este foi implementado.

Exemplos:

- Sistemas operacionais
- Aplicações na *web*
- Dispositivos físicos

Exemplo: API do Linux Kernel



- *kernel* desenvolvido em C
- bibliotecas em C fornecem funções que acessam o *kernel*
- Exemplo disponível aqui

Web services - Introdução

Contexto

Antes: aplicações *desktop* isoladas
web destinada a acessar páginas HTML

Hoje: aplicações comunicam-se entre si
Aplicações modernas rodam via *web*
Desenvolvimento em Java, .Net, Angular JS, Node.js etc

Problema: integrar aplicações
heterogêneas (desenvolvidas para diferentes propósitos),
baseadas em diferentes tecnologias (diferentes linguagens de programação etc)
distribuídas (executando em diferentes máquinas de uma rede)

Contexto

O desenvolvimento de aplicações na atualidade:

"... break from the traditional design of monoliths.... We no longer extend an existing application's functionality by creating or importing a library into the application... Instead, we build ... and integrate it with the rest of the application using endpoint type interfacing (such as HTTP) and event type interfacing (such as a messaging platform)."

Graham Charters, Sebastian Daschner, Pratik Patel, Steve Poole. Developing Open Cloud Native Microservices. 1 ed. O'Reilly Media, Inc. Sebastopol, CA, USA. 2019. (Livro com acesso aberto disponível aqui)

Tecnologias para computação distribuída

CORBA (OMG)

Padrão de desenvolvimento, independente de linguagem de programação. Poderoso mas limitado. Concepção complicada para o desenvolvimento de aplicações *web*

DCOM (Microsoft)

Plataforma para integração de componentes desenvolvidos pela MS, tais como OLE, COM e ActiveX.

RMI (Sun Microsystems)

Plataforma Java para objetos distribuídos. Permite que objetos Java interajam entre si através da invocação de métodos mesmo que estes objetos esteja distribuídos em uma rede.

Web Services

...

Web Services

A Web Programável

A web contém

- dados (textos, imagens, vídeos, preços, opiniões, *likes* etc)
- serviços (mecanismos de busca, lojas virtuais, jogos, calculadoras etc)

Dados e serviços são acessados remotamente:

web tradicional: através de um navegador

web programável: através de um software (ou *serviços*)

Web Services

A web: simples e ubíqua

Websites são simples para usuários
Webservices são simples para programadores

Com webservices, os *surfers* são programas de computador

Web Services

Definição:
Um **web service** é uma entidade computacional baseada em padrões bem definidos, independente de linguagem de programação, que aceita requisições de outras entidades, possivelmente remotamente localizadas, através de protocolos de comunicação adequados, produzindo respostas dentro do contexto de uma aplicação específica.

Informalmente:
Web services são um meio para que aplicações desenvolvidas em diferentes linguagens, possivelmente distribuídas em uma rede de computadores, troquem informações através da *web* .

Vantagens de Web Services

Baixo acoplamento

Cada serviço é independente dos demais. Partes da aplicação (i.e. os serviços) podem ser modificados sem impacto aos demais.

Facilidade de integração

Web services funcionam como uma "cola" que une diferentes partes de programas distribuídos

Reuso

Um mesmo serviço pode ser reutilizado por diferentes clientes

Web Services

Aspecto comum a todos os tipos de web services:
trocam dados e acessam e disponibilizam serviços através de HTTP

Há diferentes arquiteturas de webservices
Principais diferenças:

- Informação de método
- Escopo da informação

Informação de método

Como o cliente informa "o que deseja" ao servidor?
Como o servidor "sabe" que o cliente deseja obter dados?

em vez de excluir, alterar etc os dados existentes

Abordagens:

1. Identificar o método na URI
2. Incorporar método no *body* e no *header*

Web Services - Informação de método

Abordagem #1: Identificar o método na URI

```
http://www.flickr.com/services/rest?method=flickr.photos.search&api_key=xxx&tag=penguins
http://www.flickr.com/services/rest?method=flickr.photos.comments.deleteComment&
                                                    api_key=xxx&comment_id=9862

http://www.flickr.com/services/rest?method=flickr.photos.addTags&api_key=xxx&
                                                      photo_id=123&tags=animal
```

- Requisições GET
- O servidor não "sabe" o que o cliente quer fazer
- O significado da requisição depende da interpretação do webservice

Web Services - Informação de método

Abordagem #2: Incorporar método no body e no header

Exemplo (SOAP RPC call):

```
POST search/beta2 HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml
SOAPAction: urn:GoogleSearchAction
<?xml version="1.0"encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/≫
 <soap:Body>
  <gs:doGoogleSearch xmlns:gs="urn:GoogleSearch≫
   <q>REST</q>
   ...
  </gs:doGoogleSearch>
 </soap:Body>
</soap:Envelope>
```

- O servidor "sabe" o que o cliente quer fazer (GET, POST, PUT, DELETE etc)

Escopo da informação

Como o cliente informa ao servidor sobre qual porção de dados deseja atuar?

Abordagens:

1. Delimitar o escopo na URI
2. Delimitar o escopo no corpo da requisição

Web Services - Escopo da informação

Abordagem #1: Delimitar o escopo na URI

Exemplo:
`http://www.flickr.com/services/rest?method=flickr.photos.search&api_key=xxx&tag=penguins`

- O programador do webservice precisa decodificar a URI e usar a informação

Web Services - Escopo da informação

Abordagem #1: Delimitar o escopo no corpo da requisição

Exemplo (SOAP RPC call):

```
POST search/beta2 HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml
SOAPAction: urn:GoogleSearchAction
<?xml version="1.0"encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/≫
 <soap:Body>
  <gs:doGoogleSearch xmlns:gs="urn:GoogleSearch≫
   <q>REST</q>
   ...
  </gs:doGoogleSearch>
 </soap:Body>
</soap:Envelope>
```

- O servidor pode decodificar a informação

REST

A Web

A web "tradicional"

HTTP usado para troca de documentos na web

Os clientes HTTP são humanos e as informações trocadas são legíveis por humanos

Ao acessarmos páginas web, um servidor envia páginas HTML que são renderizadas em nosso browser. As páginas podem ser resultado da execução de programas complexos no servidor (ex. *google sheets*), mas do ponto de vista do cliente, são apenas HTML.

A web "programável"

HTTP usado para troca de informações

Os clientes são entre *programas de computador*

Em vez de HTML, os dados trafegam em formatos apropriados para troca de informação entre programas de computador (XML, JSON etc)

A web “programável” - exemplo

GET: obtém informações sobre algum recurso gerenciado pelo servidor
as informações são retornadas em formato legível por computador

Exemplos:

```
curl
http://viacep.com.br/ws/89036002/xml
```

```
<xmlcep>
   <cep>89036-002</cep>
   <logradouro>Rua João Pessoa</logradouro>
   <complemento>de 1267 a 1671 - lado ímpar</complemento>
   <bairro>Velha</bairro>
   <localidade>Blumenau</localidade>
   <uf>SC</uf>
   <unidade></unidade>
   <ibge>4202404</ibge>
   <gia></gia>
</xmlcep>
```

```
curl
http://viacep.com.br/ws/89036002/json
```

```
{
   "cep": "89036-002",
   "logradouro": "Rua João Pessoa",
   "complemento": "de 1267 a 1671 - lado ímpar",
   "bairro": "Velha",
   "localidade": "Blumenau",
   "uf": "SC",
   "unidade": ,
   "ibge": "4202404",
   "gia":
}
```

fonte: http://viacep.com.br

A web “programável” - exemplo

Os dados trafegam na web em outros formatos XML, JSON

POST: envia dados ao servidor http

possivelmente retornando algum resultado

Exemplos:

```
curl -d x=5 -d y=2 -X POST https://postman-echo.com/post
```

fonte: `https://viacep.com.br/`

A web “programável” - exemplo

```
GET https://api.bcb.gov.br/
    dados/serie/bcdata.sgs.10813/dados/ultimos/1?formato=xml
```

```
GET https://api.bcb.gov.br/
   dados/serie/bcdata.sgs.10813/dados/ultimos/1?formato=json
```

fonte: https://dadosabertos.bcb.gov.br/dataset/10813-taxa-de-cambio---livre---dolar-americano-compra/resource/d5bdc538-d5cc-4a6a-9827-3de7f208738f

REST

Estilo (ou arquitetura) de programação baseado em padrões da *web* e no protocolo HTTP

Padrões que estruturam APIs web

API web: API que é acessada através do protocolo HTTP, fazendo requisições a URLs e recebendo dados (não necessariamente páginas HTML) como resposta

Servidores REST fornecem acesso aos recursos

Clientes REST acessam os recursos

Cada recurso é acessado através de métodos do HTTP:

- **GET**: requisita um recurso
- **PUT**: cria ou atualizar um recurso conforme os dados fornecidos
- **DELETE**: exclui o recurso especificado
- **POST**: submete dados para serem processados pelo recurso especificado

Dados em REST: texto, XML, JSON

O que é REST?

REpresentational State Transfer
Ideias fundamentais:

- quanto mais simples, melhor
- a *web* funciona e funciona muito bem
- serviços deveriam seguir o "jeito *web*" de ser
- recursos acessíveis via web
- requisições HTTP equivalem à chamada de um método de um objeto localizado no servidor

Utilização de métodos de um protocolo consolidado (HTTP)
Recursos com múltiplas representações

XML, JSON

REST

Tudo é um endereço web

`http://www.ecommerce.com/cliente/3245`

`http://www.ecommerce.com/cliente/3245/endereco`

Pelo endereço, sabe-se:

- O protocolo (como comunicar)
- O *host*
- O caminho para o recurso

Vantagens do REST

- Menos *overhead* (comparado ao SOAP)
- Ausência de duplicidade semântica (SOAP utiliza outros métodos para GET, PUT, POST, DELETE)
- Padronização: HTTP é um protocolo consolidado
- Facilidade de teste: os retornos podem ser acessados pelo *browser*

Recursos com múltiplas representações

HTTP headers negociam:

- CONTENT-TYPE: especifica o tipo do corpo da mensagem
- ACCEPT: lista de um ou mais tipos que o cliente espera receber como resposta
- Exemplo: cliente requisitando dados em XML ou JSON:

```
GET /customers/33323

ACCEPT: application/xml,application/json
```

JSON

- JavaScript Object Notation
- Sintaxe enxuta para representar dados
- Alternativa ao XML

Exemplo

```
[{"Email":"bob@example.com","Name":"Bob"},
{"Email ":"mark@example.com","Name":"Mark"},
{"Email":"john@example.com","Name":"John"}]
```