Web Services

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Engenharia de Software

Sumário

Serviços web – *web services*

Serviço web (*web services*)

é uma tecnologia de chamada remota de objetos que utiliza protocolos da web (p. ex. HTTP) como meio de transporte e comunicação.

Web services RESTful (Representational State Transfer)

utiliza URIs e métodos HTTP para disponibilizar acesso aos recursos da aplicação.

Acessando web services REST

Um serviço web REST disponibiliza dados ou recursos a partir de uma URI.

Resultado do acesso a `www.thomas-bayer.com/sqlrest/`

```
<resource xmlns:xlink="http://www.w3.org/1999/xlink">
    <CUSTOMERList xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/">
                                                        CUSTOMER</CUSTOMERList>
    <INVOICEList xlink:href="http://www.thomas-bayer.com/sqlrest/INVOICE/">
                                                        INVOICE</INVOICEList>
    <ITEMList xlink:href="http://www.thomas-bayer.com/sqlrest/ITEM/">
                                                        ITEM</ITEMList>
    <PRODUCTList xlink:href="http://www.thomas-bayer.com/sqlrest/PRODUCT/">
                                                        PRODUCT</PRODUCTList>
</resource>
```

Acessando web services REST

- O exemplo indica que existem 4 web services disponíveis:
  1. **CUSTOMER**: dados de cliente.
  2. **INVOICE**: dados do pedido.
  3. **ITEM**: dados do item do pedido.
  4. **PRODUCT**: dados do produto.

Acessando web services REST

Para acessar um serviço web de clientes:
`http://www.thomas-bayer.com/sqlrest/CUSTOMER/`.

- Aparecerá a lista de clientes.

Acesse o cliente 10:
`http://www.thomas-bayer.com/sqlrest/CUSTOMER/10/`

```
<CUSTOMER xmlns:xlink="http://www.w3.org/1999/xlink">
   <ID>10</ID>
   <FIRSTNAME>Sue</FIRSTNAME>
   <LASTNAME>Fuller</LASTNAME>
   <STREET>135 Upland Pl.</STREET>
   <CITY>Dallas</CITY>
</CUSTOMER>
```

Framework Simple

Framework Simple

Servidor embutido em Java capaz de tratar requisições HTTP.

Para utilizar o framework em seu projeto:

- `http://www.simpleframework.org/`
- Descomprimir o arquivo simple-6.0.1.jar
- Adicionar o diretório criado no *build path* do projeto.

Framework Simple

A interface Container descreve o comportamento do servidor web. Método handle(Request request, Response response) trata requisição HTTP.

```
public class HTTPServer implements Container {

public void handle(Request request, Response response) {
    try {
        PrintStream body = response.getPrintStream();
        response.setValue("Content-Type", "text/plain");
        body.println("Olá, você requisitou: " + request.getPath());
        body.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
...
```

Framework Simple

Método main inicia o servidor, que ficará "ouvindo" uma porta.

```
public static void main(String[] list) throws Exception {
    int porta = 880;

    // Configura uma conexão soquete para o servidor HTTP.
    Container container = new HTTPServer();
    ContainerSocketProcessor servidor =
                                new ContainerSocketProcessor(container);
    Connection conexao = new SocketConnection(servidor);
    SocketAddress endereco = new InetSocketAddress(porta);
    conexao.connect(endereco);

    System.out.println("Tecle ENTER para interromper o servidor...");
    System.in.read();

    conexao.close();
    servidor.stop();
}
}
```

Framework Simple

Container container = **new** HTTPServer();

Instancia uma classe que implementa a interface Container. Essa classe irá tratar (*handle*) as requisições HTTP.

ContainerSocketProcessor servidor = **new** ContainerSocketProcessor(container);

SocketProcessor recebe os dados enviados pelo protocolo HTTP e os converte em um objeto Request, e converte o objeto Response em um fluxo de dados enviado ao cliente.

Abrir a conexão em uma porta pré-definida

```
Connection conexao = new SocketConnection(servidor);
SocketAddress endereco = new InetSocketAddress(porta);
conexao.connect(endereco);
```

Trabalhando com dados de formulário

Classe Query

Armazena os dados do formulário em um dicionário (Map).

```
Query query = request.getQuery();
String value = query.get(key);
```

Principais métodos:

```
List<String> lista = query.getAll(key);
String primeiro = list.get(0);
String segundo = list.get(1);
int inteiro = query.getInteger(key);
int real = query.getFloat(key);
boolean logico = query.getBoolean(key);
```

Trabalhando com dados de formulário

```
public void handle(Request request, Response response) {
    try {
        Query query = request.getQuery();
        PrintStream body = response.getPrintStream();
        long time = System.currentTimeMillis();

        response.setValue("Content-Type", "text/plain");
        response.setValue("Server", "Formulario");
        response.setDate("Date", time);
        response.setDate("Last-Modified", time);

        // pode ser testado com a URL:
        // http://127.0.0.1:880/?nome=Ze;idade=32
        String nome = query.get("nome");
        int idade = query.getInteger("idade");

        body.println("Teste de requisição com dados de formulário.");
        body.println("NOME: " + nome + "\nIDADE: " + idade + "");
        body.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Roteamento de requisições

- O caminho da requisição (*path*) permite recuperar o recurso solicitado pelo cliente.
- Decomposição do *path*:

```
Path path = request.getPath();
String directory = path.getDirectory();
String name = path.getName();
String[] segments = path.getSegments()
```

JavaScript Object Notation

JavaScript Object Notation – JSON

É um formato leve para troca de informação. Baseado na linguagem JavaScript, é fácil de ser entendido pelo ser humano e pela máquina.

Baseado em duas estruturas:

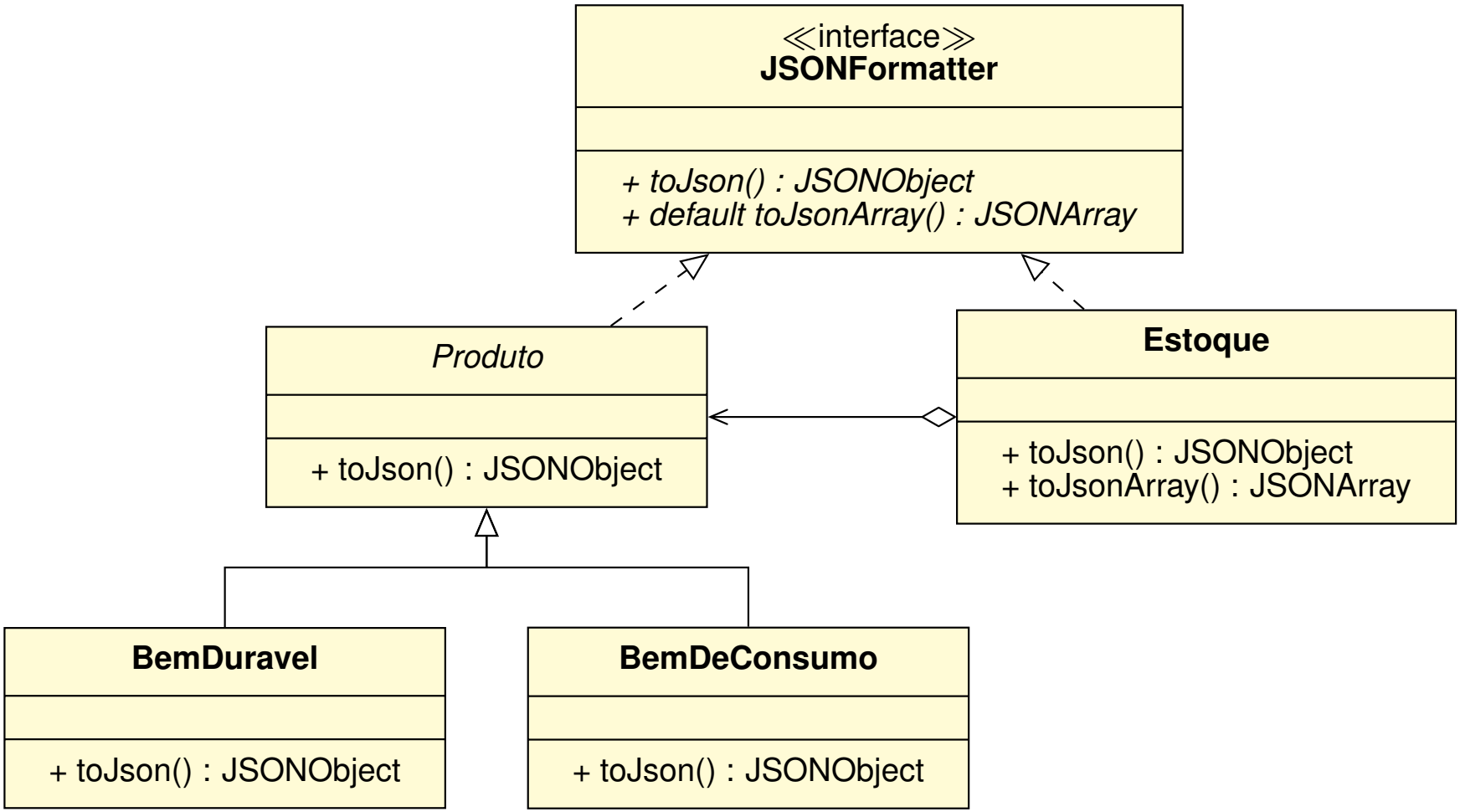
- `JSONObject`: uma coleção de `pares <nome, valor>`, tais como um dicionário ou Map, em Java.
- `JSONArray`: uma lista ordenada de valores, ou Array.

Mapeamento entre JSON e entidades Java

JSON	Java
string	java.lang.String
number	java.lang.Number
true|false	java.lang.Boolean
null	**null**
array	java. util . List
object	java. util .Map

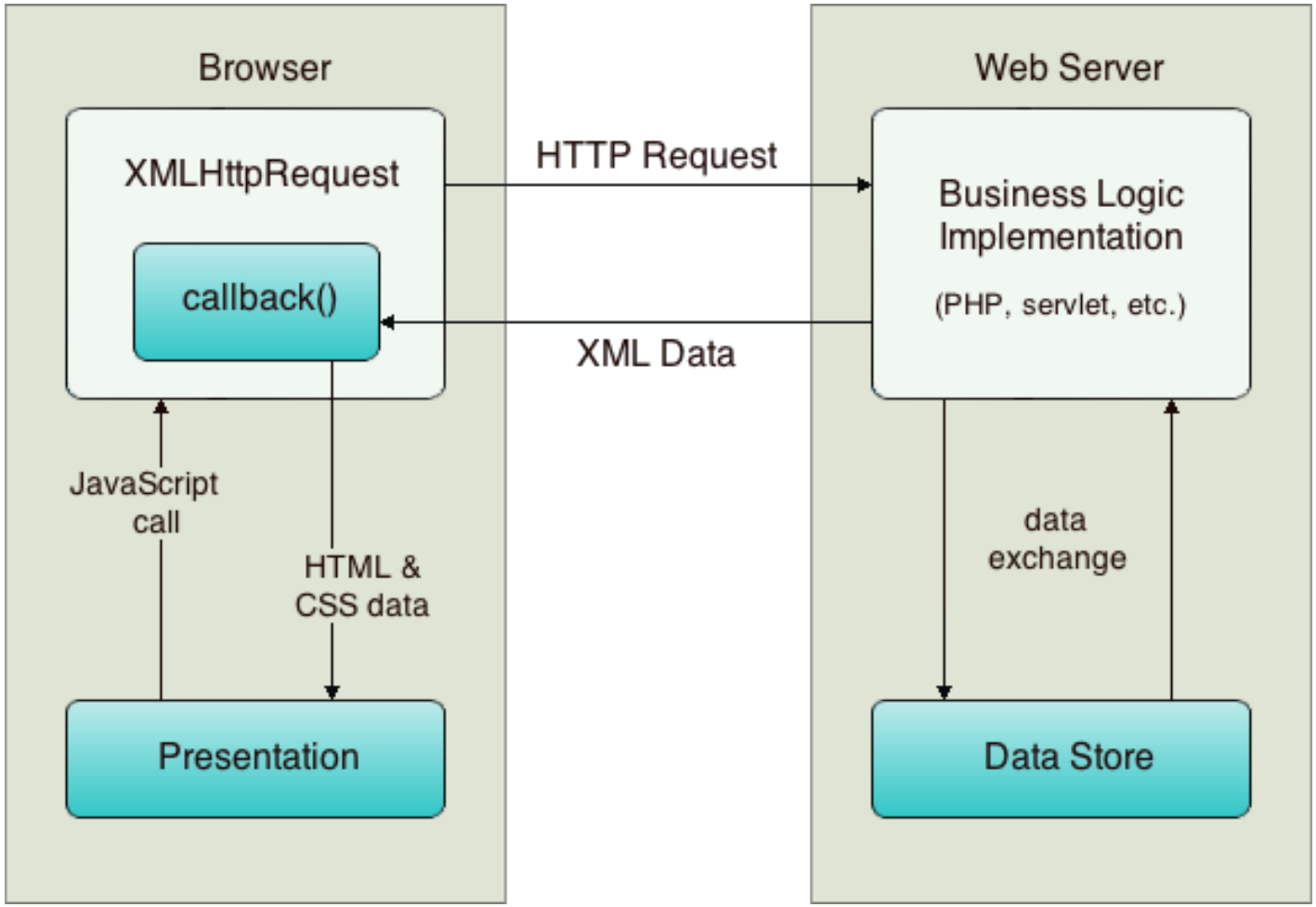
```
JSONObject obj = new JSONObject();

obj.put("string", "Shulambs");
obj.put("int", new Integer(1));
obj.put("real", new Double(1.99));
obj.put(log", new Boolean(true));
```

Exemplo: Convertendo objetos Java para Json



Asynchronous JavaScript and XML (AJAX)

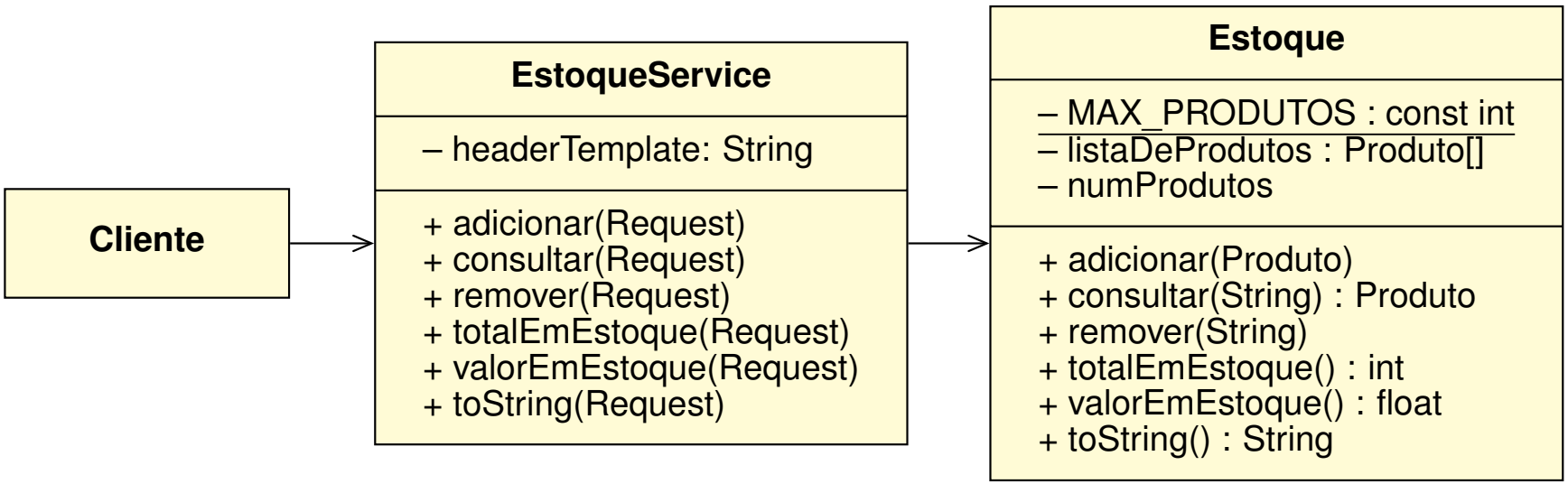
- Objeto XMLHttpRequest



AJAX e JSON juntos

1. Formulário utiliza função JavaScript para capturar uma conexão `XMLHttpRequest`.
2. Servidor retorna um objeto JSON.
3. Cliente realiza o *parsing* no objeto JSON e atualiza a tela.

Disponibilização do serviço na Web



AJAX e JSON juntos

- Classe EstoqueService é responsável por fazer a tradução `Java` $\rightarrow$ `JSON` e `JSON` $\rightarrow$ `Java`.
- Função FactoryXMLHttpRequest() cria uma conexão AJAX (XMLHttpRequest).
- Função GetIt(url) faz uma requisição ao servidor e realiza o *parsing* no JSON retornado.
- Documento HTML recebe o resultado da requisição.

Exemplo: Estoque AJAX + JSON

```
function FactoryXMLHttpRequest() {
    if(window.XMLHttpRequest) {
        // Opera 8.0+, Firefox, Chrome, Safari
        return new XMLHttpRequest();
    }
    else if (window.XDomainRequest) {
        return new XDomainRequest(); // Antigo Safari
    } else if(window.ActiveXObject) {
        var msxmls = new Array(// Internet Explorer
        'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP',
        'Msxml3.XMLHTTP', 'Msxml2.XMLHTTP.7.0',
        'Msxml2.XMLHTTP.6.0', 'Msxml2.XMLHTTP.5.0',
        'Msxml2.XMLHTTP.4.0', 'Msxml2.XMLHTTP.3.0');
        for (var i = 0; i < msxmls.length; i++) {
            try {
                return new ActiveXObject(msxmls[i]);
            } catch (e) {       }
        }
    } else throw new Error("Could not instantiate XMLHttpRequest");
}
```

Exemplo: Estoque AJAX + JSON

```
function GetIt(url) {
    var xmlhttp = new FactoryXMLHttpRequest();

    xmlhttp.onreadystatechange = function(){
        if (xmlhttp.readyState == 4) {
            var jsonObj = JSON.parse(xmlhttp.responseText);
            document.getElementById("id").innerHTML = jsonObj.id;
            document.getElementById("descricao").innerHTML =
                                                    jsonObj.descricao;
            document.getElementById("preco").innerHTML = jsonObj.preco;
            document.getElementById("quant").innerHTML = jsonObj.quant;
            document.getElementById("dataFabricacao").innerHTML =
            jsonObj.dataFabricacao;
        }
    }
    if (xmlhttp) {
        xmlhttp.open('get', url, true);
        xmlhttp.send();
    }
}
```

Exemplo: Estoque AJAX + JSON

```
<button onclick="GetIt('http://127.0.0.1:880')">Carregar</button>

<table border="1">
<tr>
  <td>Produto id:</td><td><span id="id">Vazio</span></td>
</tr>
<tr>
  <td>Descricao:</td><td><span id="descricao">Vazio</span></td>
</tr>
<tr>
  <td>Preco:</td><td><span id="preco">Vazio</span></td>
</tr>
<tr>
  <td>Quantidade:</td><td><span id="quant">Vazio</span></td>
</tr>
<tr>
  <td>Dt. fab.:</td><td><span id="dataFabricacao">Vazio</span></td>
</tr>
</table>
```

Aplicação RESTful - Jersey/Grizzly

Infraestrutura:

- Servidor Web: Grizzly 2 (Java SE).
- Framewrok REST: Jersey REST Framework / JAX-RS API.
- Gerenciador de configuração: Maven.

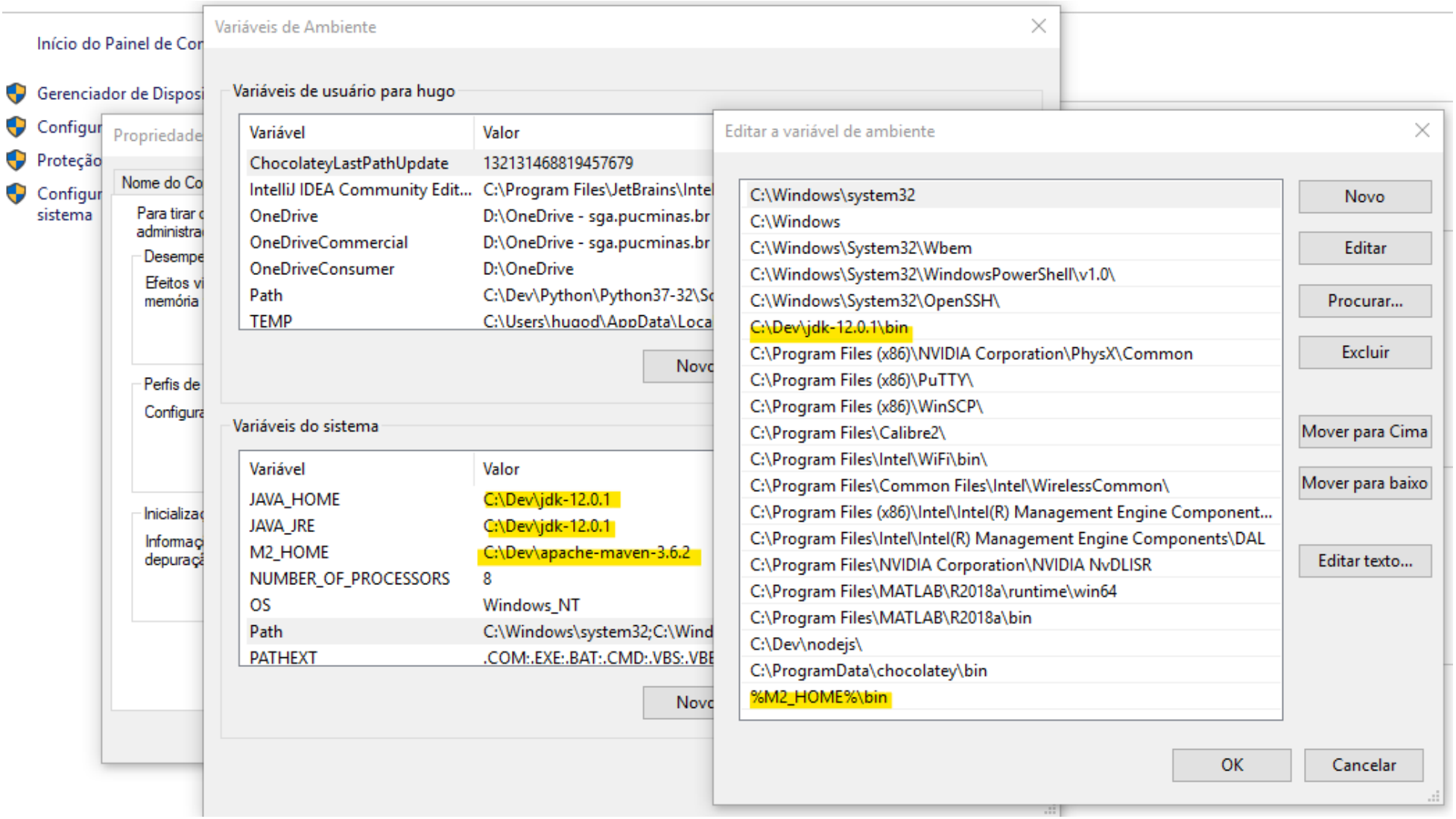
Processo:

- Servidor web recebe **requests** e encaminha **responses**.
- **Annotations** convertem métodos java em URLs REST.

Configuração do Maven

- Maven é uma ferramenta de gerenciamento de projeto.
- Controla configuração e dependências.
- Instalação:
  `https://maven.apache.org/download.cgi`
  - Baixar e descomprimir o arquivo zip.
  - Adicionar o diretório **bin** ao **PATH**.
- É necessário que a variável **JAVA_HOME** esteja configurada.

Variáveis de ambiente



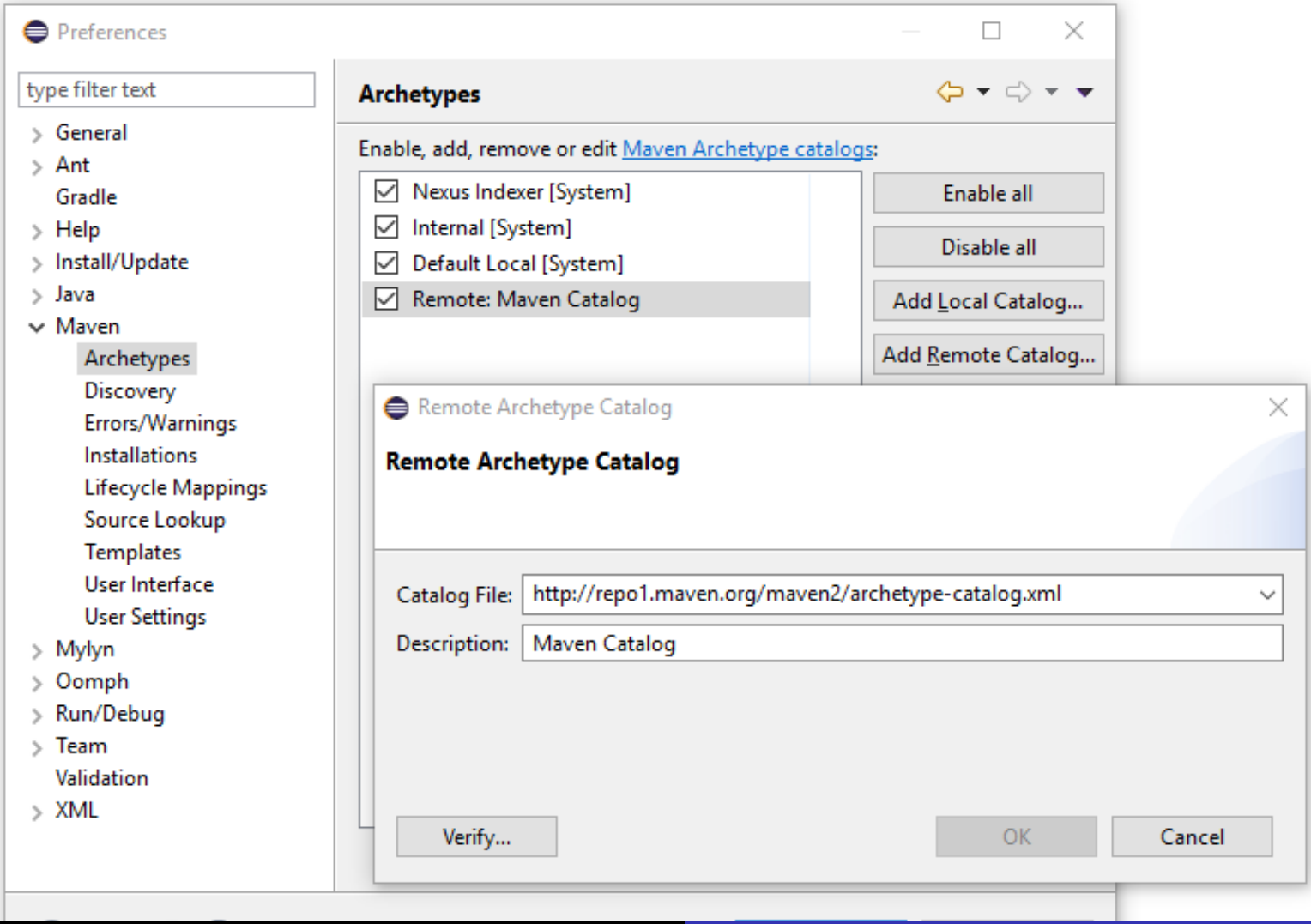
Maven Archetypes

Maven Archetypes (arquétipos)

São modelos (*templates)* que já possuem o esqueleto básico de uma aplicação.

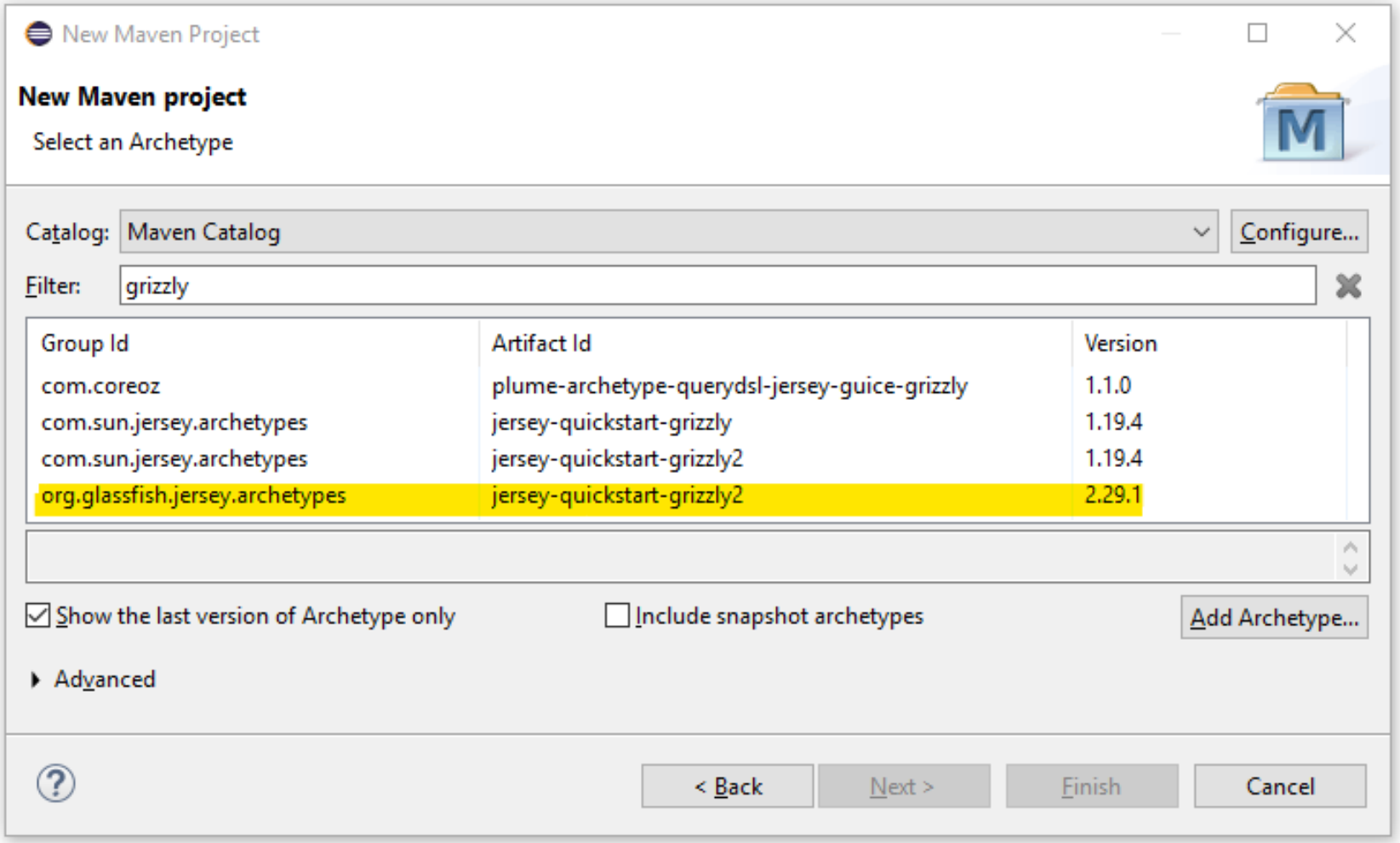
- Iremos utilizar o `jersey-quickstart-grizzly2`, disponível nos repositórios Maven.
- É necessário adicionar o repositório ao Eclipse.
  `Window` → `Preferences` → `Maven` → `Archetypes` → `Add remote catalog`.
- Adicionar: `http://repo1.maven.org/maven2/archetype-catalog.xml`

Configuração do catálogo Maven no Eclipse



Criando um projeto Maven

- No Eclipse: `New` → `Maven Project`



Configuração do projeto

- **archetypeArtifactId**:Especifica o tipo do projeto a ser criado.
- **archetypeGroupId**: Especifica o grupo em que o arquétipo está definido.
- **groupId**: Usado para identificar unicamente o projeto. Normalmente baseado no domínio. Usado como o nome do pacote.
- **artifactId**: O nome do seu projeto.

Código padrão da Aplicação RESTful – Jersey/Grizzly

- Classe Main dispara a aplicação, executando o servidor web.

```
package br.pucminas.produto_service;

public class Main {
  // Base URI the Grizzly HTTP server will listen on
  public static final String BASE_URI =
                                  "http://localhost:8080/estoque/";
```

- BASE_URI especifica a localização da sua aplicação. Qualquer método que você criar, será adicionado a partir dessa URI (nesse exemplo `/estoque`).
- O método HttpServer startServer() instancia o servidor web.
- A classe ResourceConfig especifica qual pacote contém as classes anotadas pelo Jersey que serão carregadas.

Código padrão da Aplicação RESTful – Jersey/Grizzly

ResourceConfig define o pacote em que serão armazenadas as classes que irão tratar as requisições HTTP.

```
public static HttpServer startServer() {
    // create a resource config that scans for JAX-RS resources
    // and providers in br.pucminas.ws_03_produto_service package
    final ResourceConfig rc = new
            ResourceConfig().packages("br.pucminas.ws_03_produto_service");

    // create and start a new instance of grizzly http server
    // exposing the Jersey application at BASE_URI
    return GrizzlyHttpServerFactory
                .createHttpServer(URI.create(BASE_URI), rc);
}
```

Código padrão da Aplicação RESTful – Jersey/Grizzly

Método Main inicia e interrompe o servidor web.

```
public static void main(String[] args) throws IOException {
    final HttpServer server = startServer();
    System.out.println(String.format("Jersey app started with WADL "
    + "available at %sapplication.wadl\nHit enter to stop it...",
    BASE_URI));
    System.in.read();
    server.shutdownNow();
}
```

Classe MyResource – Jersey/Grizzly

- Contém os métodos que serão publicados no web server.
- *Annotations* configuram os métodos. Por exemplo:

  @GET indica que responderá ao método **HTTP GET**.

  @Produces(MediaType.TEXT_PLAIN) indica que a resposta do método será enviada no formato texto.

```
@Path("webservice")
public class MyResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Serviço Web funcionando.";
    }
}
```

Métodos (ou verbos) de requisição HTTP

- Principais métodos HTTP que podem ser usados em um serviço web.

GET	Solicita um recurso. Pode passar parâmetro na própria URI.
POST	Cria um recurso. Dados encaminhados no corpo da requisição, e não na URI.
PUT	Atualiza um recurso caso já exista ou cria se não existir.
DELETE	Exclui um recurso.

Roteamento de requisições

- é importante definir padrões para a criação de URIs.
- Em uma API REST, cada modelo (recurso que se deseja persistir), cria um conjunto de rotas associado às operações CRUD de armazenamento de dados.

CRUD	**REST (Método / URI)**
Create	PUT /modelName
	POST /modelName
Read (Retrieve)	GET /modelName?filter=...
Update (Modify)	PUT /modelName
Delete (Destroy)	DELETE /modelName/modelID

Criando rotas com annotations

- Cada recurso deve ter uma rota raiz (@Path):

```
@Path("duravel")
public class BemDuravelService { ...
```

- Cada método HTTP deve ser declarado e o tipo de dados consumido/produzido precisa ser especificado.
  Rota **all**/ (GET):

```
@GET
@Path("/all")
@Produces(MediaType.APPLICATION_JSON)
public List<BemDuravel> getAllProducts() { ...
```

  Rota /**{id}** (GET):

```
@GET
@Path("{id}")
@Produces(MediaType.APPLICATION_JSON)
public BemDuravel getProduto(@PathParam("id") String id) { ...
```

Criando rotas com annotations

Rota /**add** (POST):

```
@POST
@Path("/add")
@Consumes(MediaType.APPLICATION_JSON)
public Response addProduct(BemDuravel produto) {
```

Rota /**update** (PUT):

```
@PUT
@Path("/update")
@Consumes(MediaType.APPLICATION_JSON)
public Response updateProduct(BemDuravel produto) {
```

Rota /**delete{id}** (DELETE):

```
@DELETE
@Path("/{id}")
public Response deleteProduct(@PathParam("id") String id) {
BemDuravel produto = Estoque.bemDuravelDao.get(id);
```

Criando o pacote de execução

- Adicionar plugins ao arquivo `pom.xml` para garantir que o pacote contenha todas as dependências.

```
<artifactId>maven-assembly-plugin</artifactId>
<artifactId>maven-dependency-plugin</artifactId>
```

- Adicionar plugins ao arquivo `pom.xml` para garantir que o pacote contenha todas as dependências.

```
<artifactId>jersey-media-moxy</artifactId>
```

Criando o pacote de execução

- Executar os comandos a seguir no diretório do projeto:

```
mvn clean compile
mvn package
```

- Executar servidor:

```
java -jar
     ws_03_produto_service-0.1-jar-with-dependencies.jar
```

- Verificar o arquivo `pom.xml` do projeto `produto-form` para configurar as dependências e plugins.

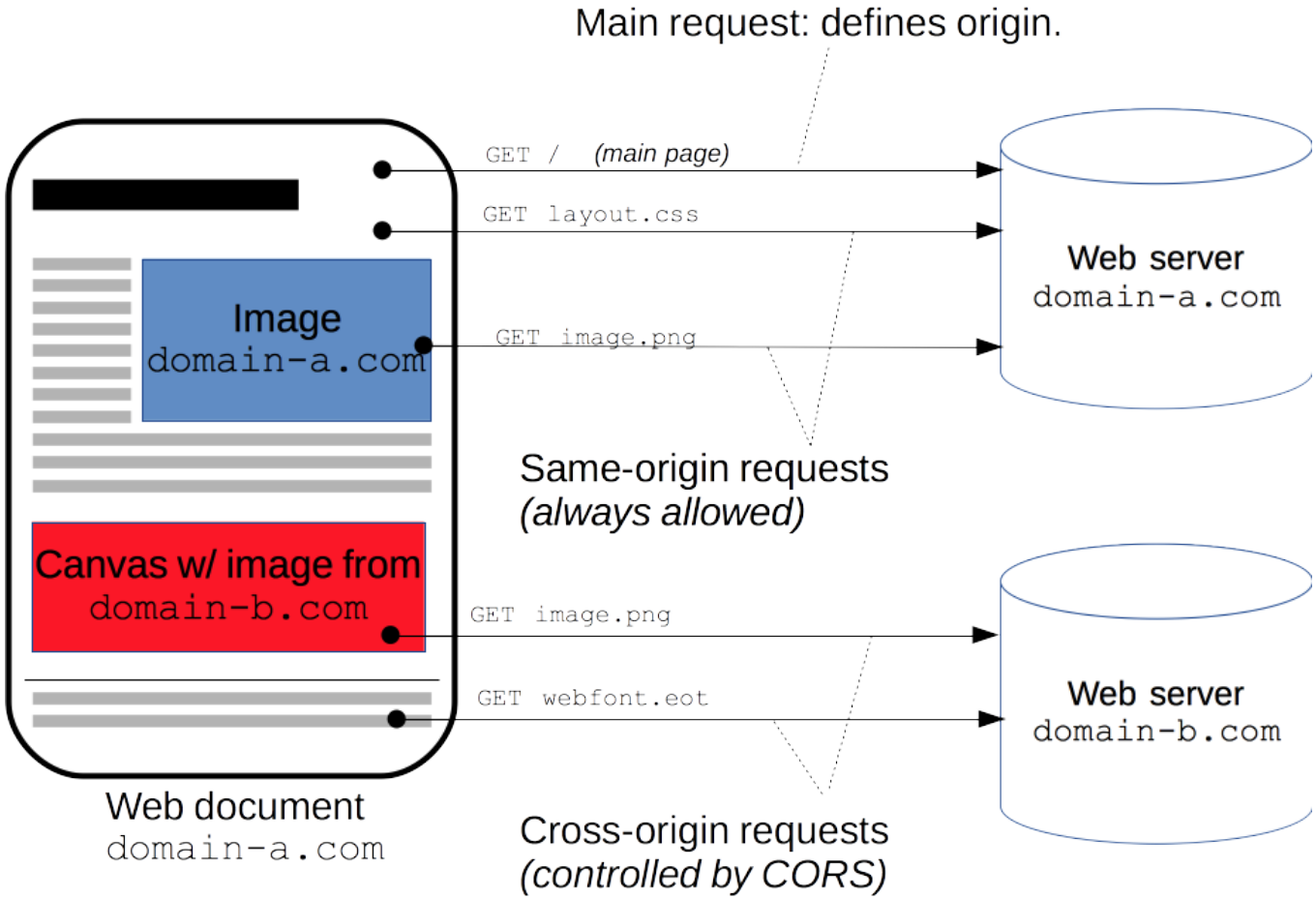
CORS Filter

Cross-Origin Resource Sharing (CORS)

Mecanismo de segurança do HTTP que utiliza o cabeçalho de uma página (*head*) para avisar que a aplicação está executando em uma origem.

Uma aplicação executa uma **cross-origin HTTP request** quando solicita recursos de outra origem.

CORS Filter



Fonte: `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`

CORS Filter

```
@Provider
public class CorsFilter implements ContainerResponseFilter {

  @Override
  public void filter(final ContainerRequestContext requestContext,
            final ContainerResponseContext cres) throws IOException {
    cres.getHeaders().add("Access-Control-Allow-Origin", "*");
    cres.getHeaders().add("Access-Control-Allow-Headers",
                          "origin, content-type, accept, authorization");
    cres.getHeaders().add("Access-Control-Allow-Credentials", "true");
    cres.getHeaders().add("Access-Control-Allow-Methods",
                          "GET, POST, PUT, DELETE, OPTIONS, HEAD");
    cres.getHeaders().add("Access-Control-Max-Age", "1209600");
  }

}
```