

Programação Para Internet

Aula: Serviços na Internet.



Prof. Anderson Augusto Bosing

Acesso a Sala de Aula



Google Classroom

Google Classroom



Formas de Avaliação da Disciplina

- **Trabalhos e Demais Atividades($\cong 3.0$)**
- **Avaliação Bimestral($\cong 7.0$)**



Meios de Comunicação



Google Classroom



Qual o objetivo da aula de hoje ?

- ✓ **Plano de Ensino**
- ✓ **O que é a internet ?**
- ✓ **Principais Protocolos de Comunicação da Internet.**



Introdução – Web Services



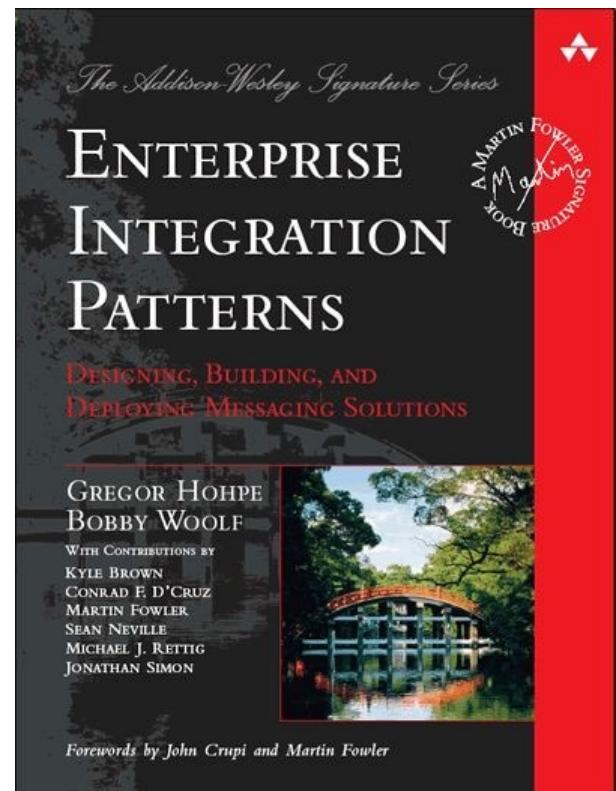
O que é Web Service?

**O que é?
Para que serve?
Resolve que problema?
Qual é o contexto?
Qual é aplicabilidade?**



O que é Web Service?

O livro “Enterprise Integration Patterns” classifica os web services como um estratégia de integração de soluções, no qual um sistema necessita acessar dados ou executar operações de outro sistema.



O que é Web Service?

- **Solução utilizada na integração de sistemas e na comunicação entre independentes aplicações.**
- **Tecnologia que possibilita a integração entre aplicações de forma que elas possam interagir entre si independentemente de plataforma, linguagem, paradigma de programação e fornecedor de tecnologia.**



O que é Web Service?

- São componentes que permitem às aplicações enviar e receber informações baseados em um formato pré estabelecido.
- Um serviço web é uma aplicação disponibilizada via (HTTP) intranet ou internet que através de uma URL pode ser acessado por aplicativos clientes usando protocolos acordados.



Para que serve Web Service?

Único e exclusivo objeto é:

**Interoperabilidade entre sistemas
independentemente de plataforma de
desenvolvimento, execução, provedor de
tecnologia, linguagem e paradigma de
desenvolvimento.**



Para que serve Web Service?

Interoperabilidade é a capacidade de um sistema informatizado ou não de se comunicar de forma transparente ou o mais próximo disso com outro sistema semelhante ou não.



Para que serve Web Service?

Interoperabilidade é a capacidade de um sistema informatizado ou não de se comunicar de forma transparente ou o mais próximo disso com outro sistema semelhante ou não.



Quando usar Web Services?

Aplicações corporativas abrangem uma série de cenários simples até os mais complexos que apresentam requisitos de interoperabilidade entre soluções corporativas dentro de um ambiente empresarial heterogêneo.



Cenários de Integração

- Grandes, médias e pequenas empresas possuem investimento significativos e crescentes em sistemas pertencentes a diferentes plataformas ao longo de sua história.
- Web services são a solução para fazer a integração entre o legado e as novas aplicações.



Cenários de Integração

- Na atualidade temos empresas adquirindo outras empresas e cada uma delas apresentando soluções próprias em diferentes plataformas.
- Web services são a solução para fazer a integração entre estas aplicações até então desconhecidas.



Cenários de Integração

- Muitas empresas têm expandido suas redes de negócios no estabelecimento de “parcerias” com outras empresas “sócios” de outros setores que unidas formam uma nova vertente de negócio.
- Web services são a solução para fazer a integração entre estas aplicações de cada empresa, garantindo eficiência no negócio e melhor experiência para o usuário final.



Cenários de Integração

- Muitas empresas têm a necessidade de expor suas soluções para ser acessadas remotamente em diferentes plataformas de execução(smartphones, tablet, TV etc) que são incompatíveis com a tecnologia adotadas.
- Web services são a solução utilizada para integrar aplicativos incompatíveis.

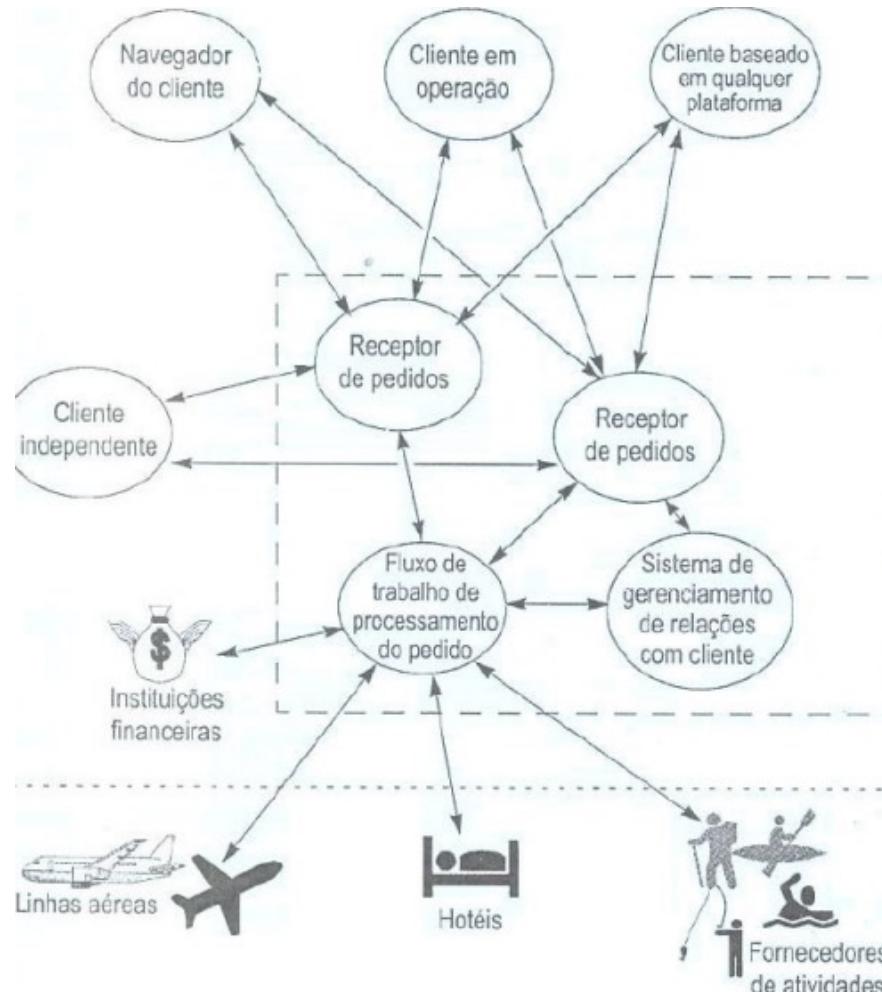


Cenários Real

- **Agência de Turismo**
 - Fornece catálogo de turismo.
 - Construir passeios customizados de acordo com o perfil dos clientes.
 - Rastrear status de pedidos.
 - Reservar passeios.
 - Todas as operações necessitam interagir com sistemas de outros sócios – linhas aéreas, hotéis, instrutores de passeios, instituições financeiras etc.



Cenários Real



Programação Para Internet

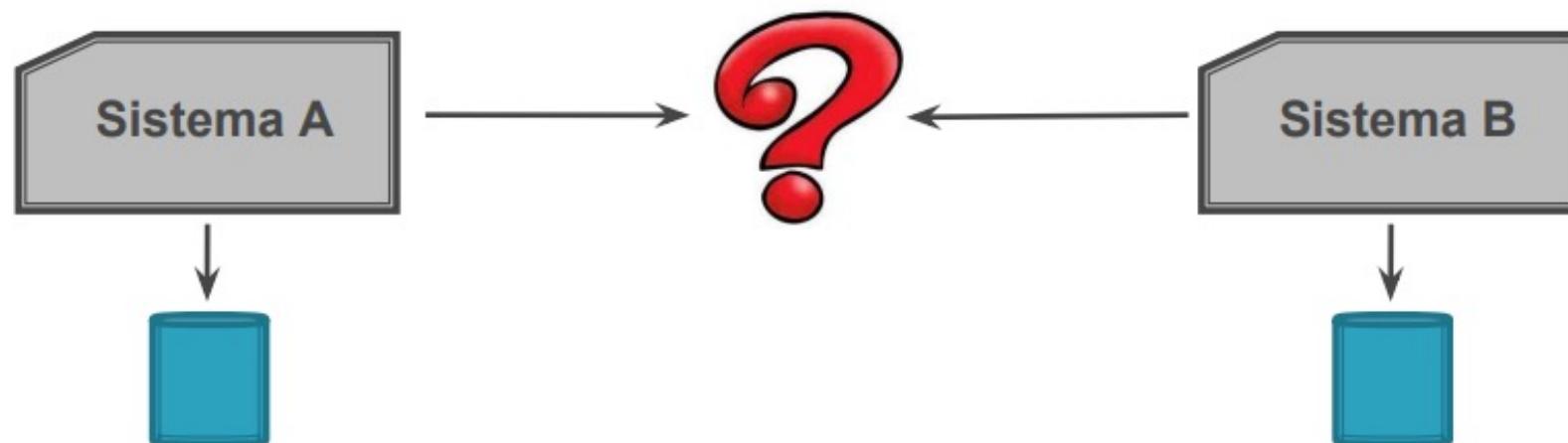
**Aula: Arquitetura de
Solução – Web Service.**



Prof. Anderson Augusto Bosing

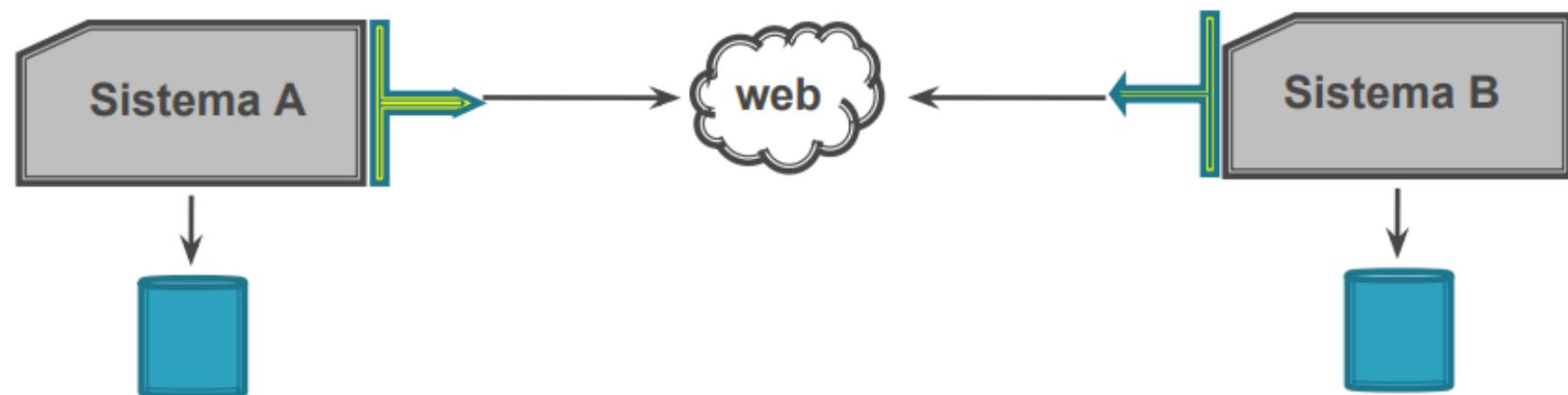
Arquitetura - Web Service

- Surge a necessidade de integrar 2 ou mais soluções diferentes dentro de uma mesma corporação:



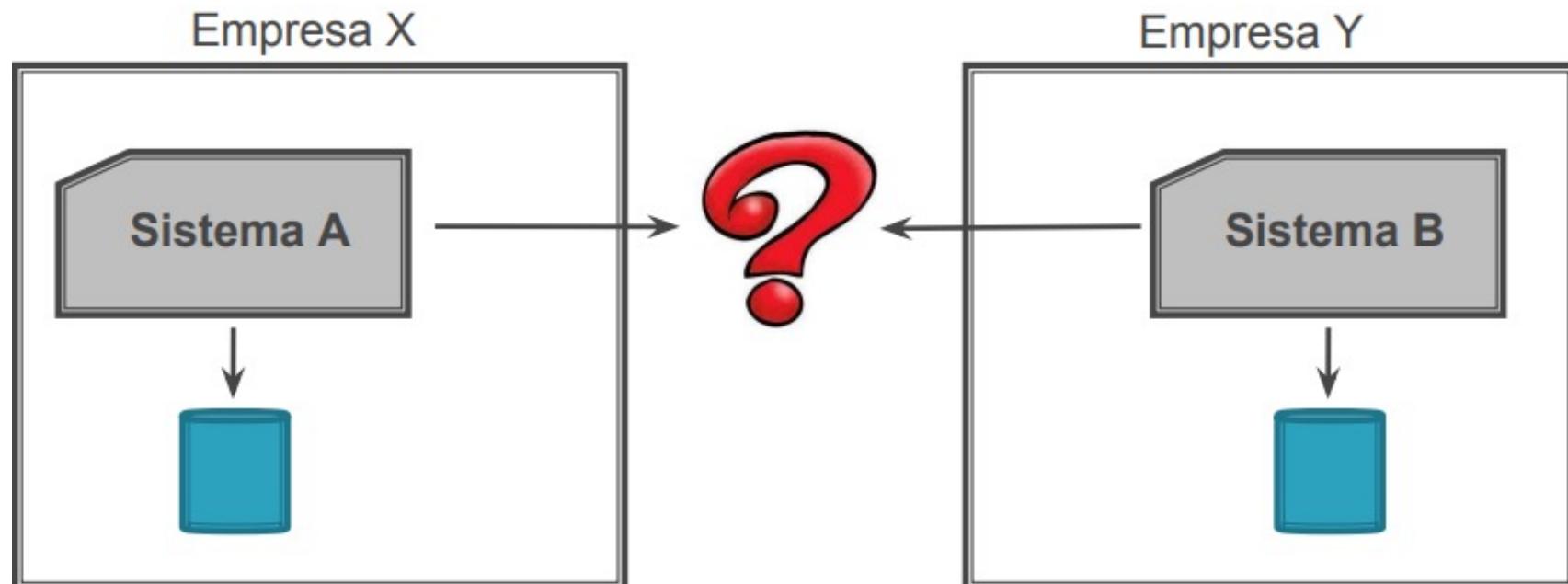
Arquitetura - Web Service

- É acrescentado em cada solução uma “camada de comunicação” que expõe as operações existentes como serviço:



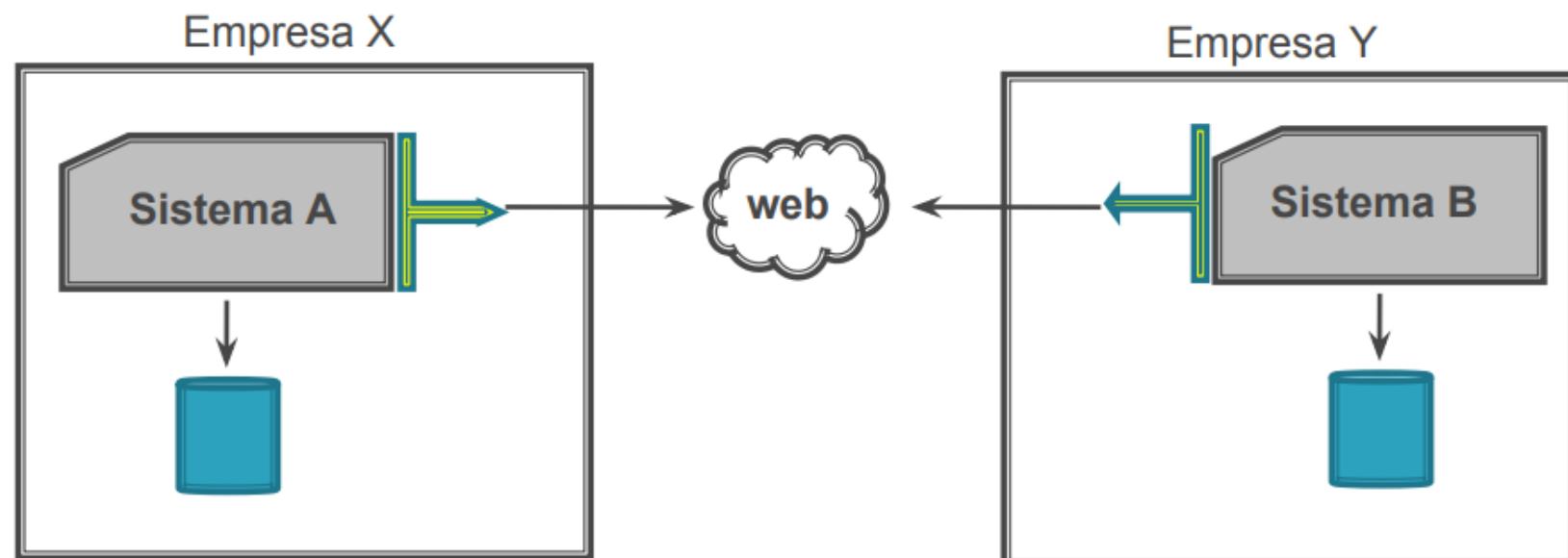
Arquitetura - Web Service

- Surge a necessidade de integrar duas ou mais soluções de diferentes corporações:



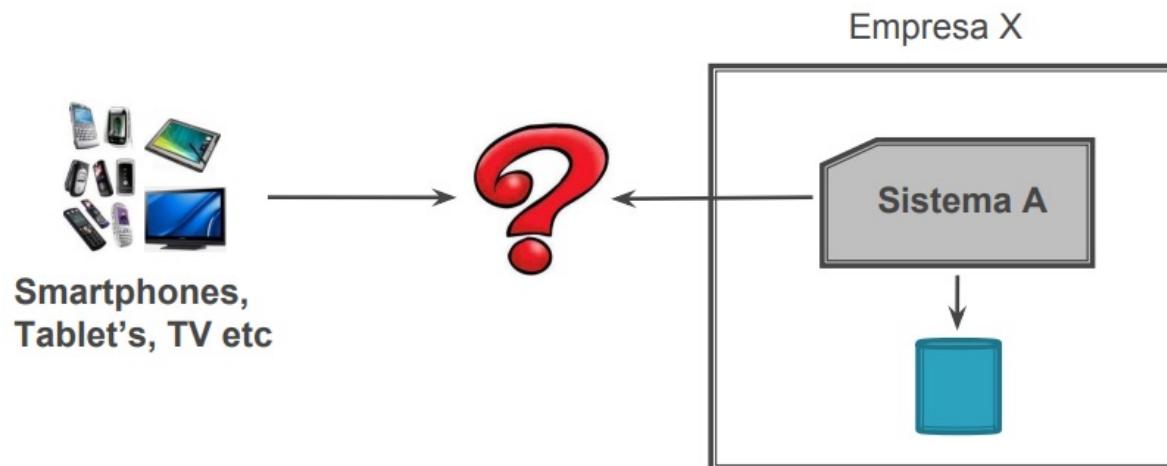
Arquitetura - Web Service

- É acrescentado nas soluções uma “camada de comunicação” que expõe as operações existentes como serviço:



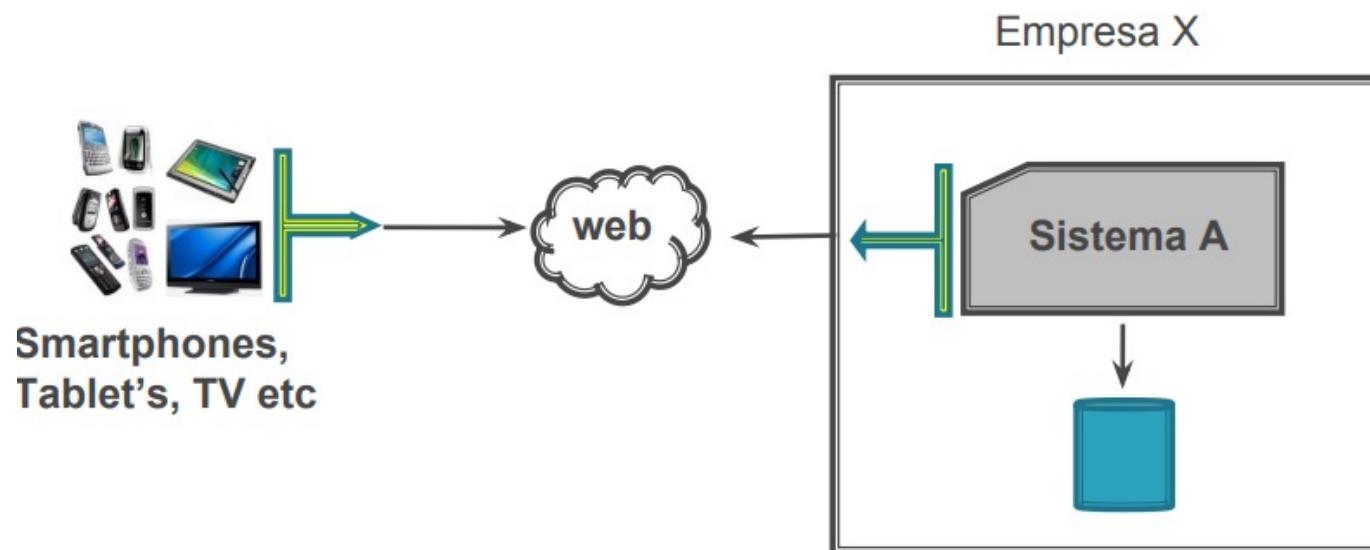
Arquitetura - Web Service

- Surge a necessidade de integrar soluções de dispositivos móveis em uma solução corporativa:



Arquitetura - Web Service

- É acrescentado na solução da corporação e nas soluções nativas de cada dispositivos uma “camada de comunicação”:



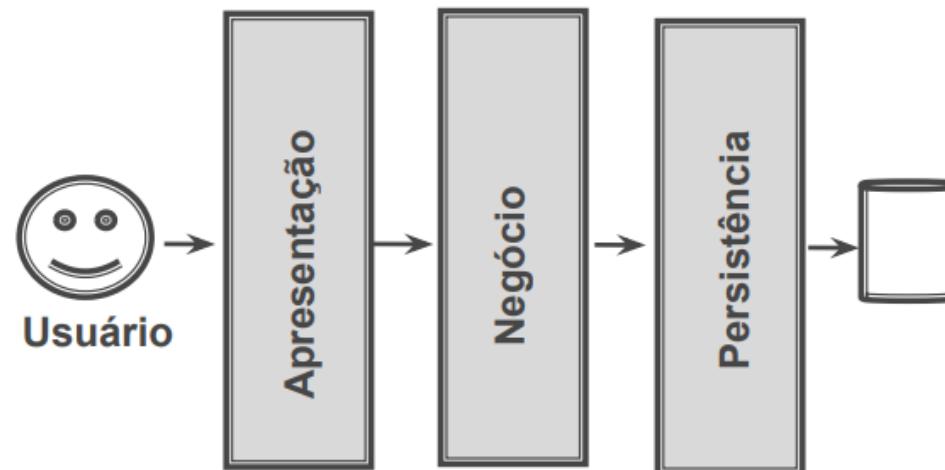
Arquitetura - Web Service

- A aplicação que expõe funcionalidades via web service é chamada de “serviço”.
- A aplicação que invoca as funcionalidade de um serviço é chamada de “consumidor”.



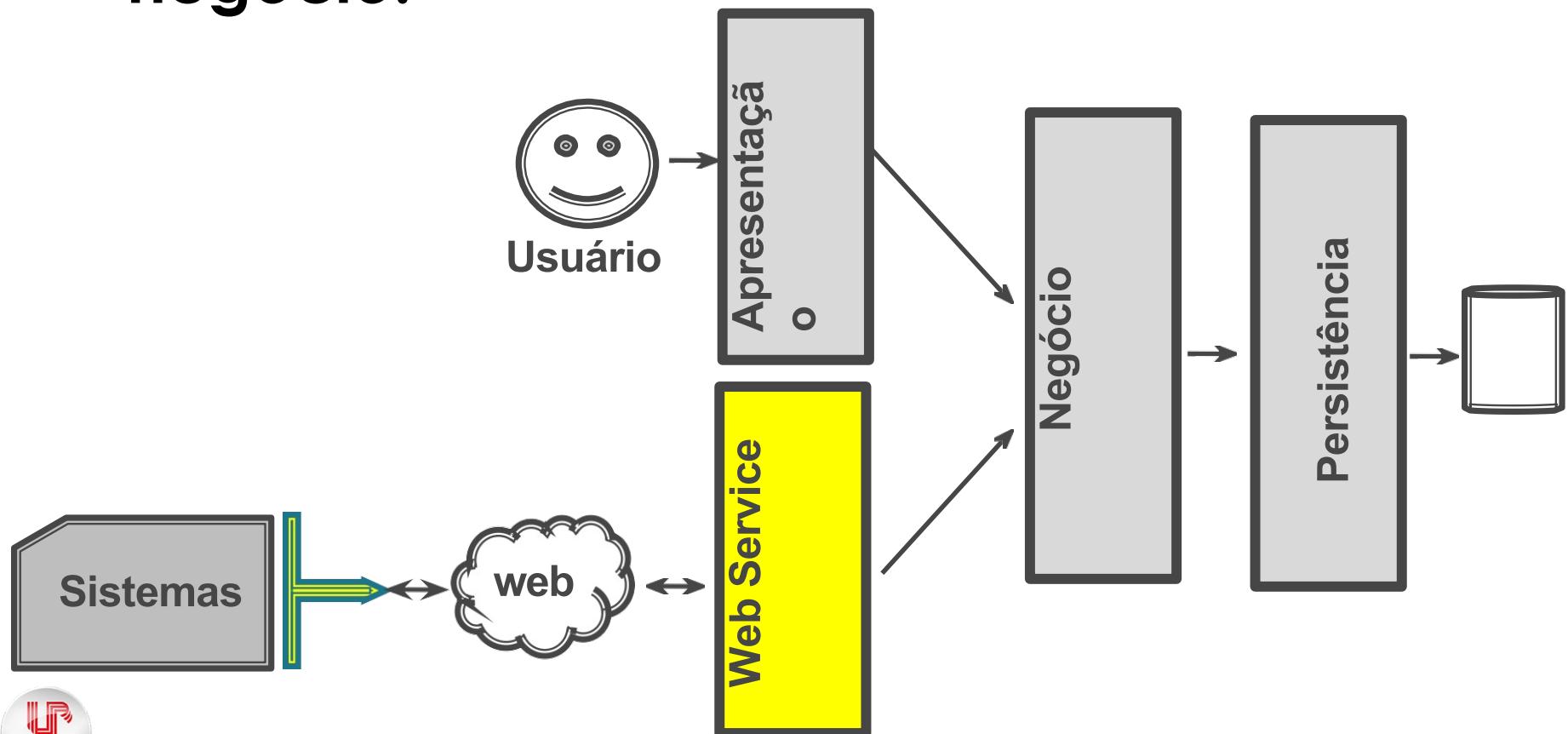
Arquitetura - Web Service

- A solução devidamente projetada está dividida arquiteturalmente em camadas lógicas que organiza suas responsabilidades.



Arquitetura - Web Service

- Com web services é criado uma camada (responsável por encapsular os detalhes WS) com dependência para a camada de negócio:



Benefícios

- **Interoperabilidade em ambientes heterogêneo:**
- **Como as empresas se desenvolvem ao passar dos tempos, elas adicionam sistemas e soluções requerem diferentes plataformas de diferentes provedores que não se comunicando com as outras.**
- **Permite que diferentes serviços distribuídos sejam executados em variedade de plataformas de software e de arquitetura.**



Benefícios

- **Interoperabilidade em ambientes heterogêneo:**
- **Permite que eles sejam escritos em diferentes linguagens e filosofias de programação, independentemente de fornecedor de tecnologia.**



Benefícios

- **Serviços de negócios através da web:**
- **Uma empresa pode usar web services para alavancar as vantagens em âmbito mundial.**
- **Ex: Uma empresa de catálogo de produtos pode colocar disponível seu estoque aos seus fornecedores através de web services de modo a conseguir melhor gerenciamento da cadeia de suprimentos.**



Benefícios

- **Integração com sistemas existentes:**
- **Uma empresa pode usar web services para alavancar as vantagens em âmbito mundial.**
- **A maioria das empresas possuem uma quantidade enorme de informações armazenados no sistemas de informações existentes e o custo para substituir isso é tão absurdo que não exista possibilidades de descartar sistemas legados.**



Benefícios

- **Liberdade de opção:**
- **O uso de padrões, convenções e diretrizes no desenvolvimento de web services abriram um grande mercado de ferramentas, produtos e tecnologias.**
- **Isso oferece aos desenvolvedores uma ampla variedade de opções (free, open source e comercial) na seleção de um produto para a criação de web services.**



Benefícios

- Suportam qualquer tipo de aplicativos clientes:
- Suportam qualquer tipo de dispositivo como aplicativo cliente de um web service devido independência de plataforma, tecnologias e provedores.



Programação Para Internet

**Aula: Arquitetura de
Solução – Web Service -
SOAP.**



Prof. Anderson Augusto Bosing

O que é SOAP?

O que é?
Para que serve?
O que eu faço?



Java Web Service - SOAP?

Digamos que tem um amigo chinês, solteiro.



Java Web Service - SOAP?

E você se lembrou que tem uma amiga
indiana, solteira.



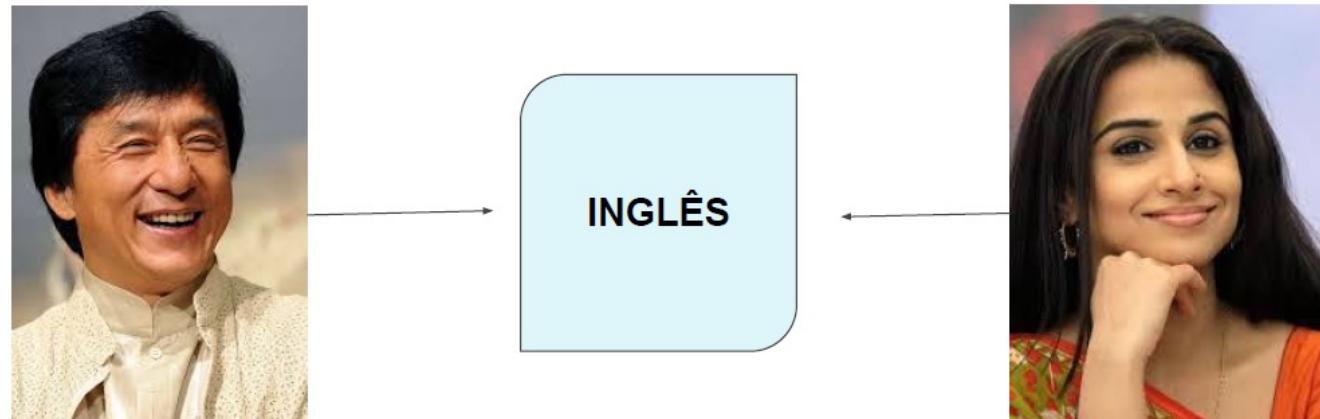
Java Web Service - SOAP?

Como você faria para que os dois pudessem trocar informações?



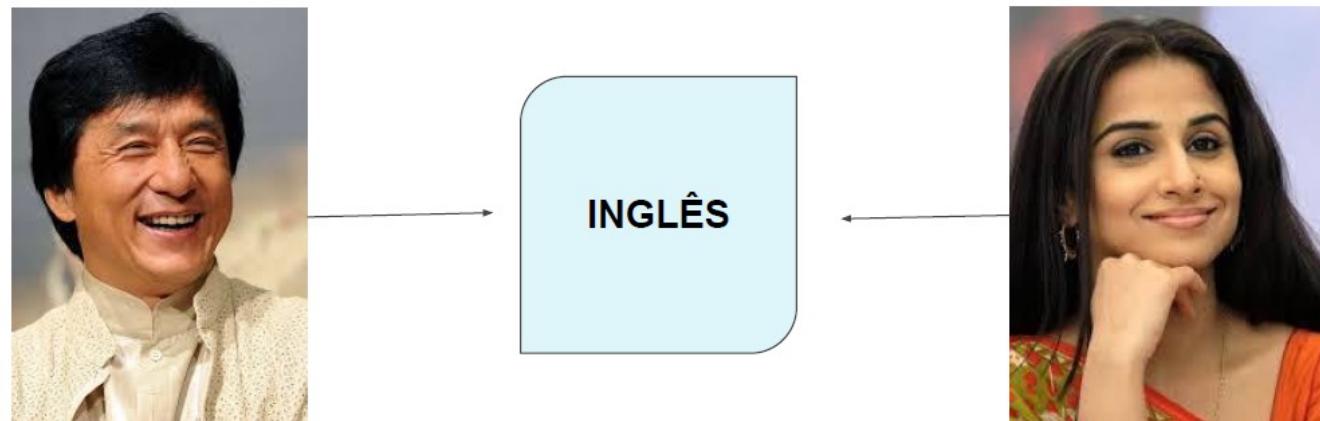
Java Web Service - SOAP?

**Seria necessário um acordo de “linguagem”
em comum entre os 2.**



Java Web Service - SOAP?

O inglês seria o protocolo de acordo entre eles para troca de informações.



Java Web Service - SOAP?

SOAP é o acrônimo de Simple Object Access Protocol - Protocolo Simples de Acesso a Objetos.

É um protocolo utilizado troca de informações estruturadas descentralizada e distribuída entre diferentes plataformas de desenvolvimento e execução.



Java Web Service - SOAP?

SOAP é uma especificação para a troca de informação entre sistemas, ou seja, uma especificação de formato de dados para envio de estruturas entre serviços, estabelecendo um padrão de mercado oficial padrão e mundialmente aceito para permitir a interoperabilidade entre eles.

Os órgãos responsáveis pelas especificações são:

- W3C (World Wide Web Consortium)**
- WS-I (Web Services Interoperability Industry Consortium)**



Java Web Service - SOAP?

Para alcançar 100% de interoperabilidade, os órgãos responsáveis especificaram os seguintes recursos:

- 1. Linguagens de comunicação.**
- 2. Formato de intercâmbio de mensagens.**
- 3. Protocolo de tráfego.**
- 4. Descrição de serviços.**



Java Web Service - SOAP?

1. Linguagens de Comunicação:

Comunicação entre diferentes plataformas obrigam uma terminologia em comum. Uma linguagem através do qual os requisitantes e o serviço possam se comunicar.

Uma linguagem independente de plataforma que fosse capaz de trocar informações neutras.



Java Web Service - SOAP?

1. Linguagens de Comunicação:

XML - eXtensible Markup Language é uma linguagem de marcação para o transporte de dados organizados hierarquicamente.

Independente de plataforma e de hardware uma vez que é representado como simples arquivo texto.



Java Web Service - SOAP?

1. Linguagens de Comunicação:

Classificada como extensível porque permite definir os elementos de marcação.

Recomendada pela W3C.



Java Web Service - SOAP?

2. Formato de Intercâmbio de Mensagens:

Para uma comunicação efetiva as partes devem ser capazes de trocar mensagens de acordo com um formato padrão.

Diante deste formato, a partes estranhas e incompatíveis podem então se comunicar eficientemente.



Java Web Service - SOAP?

2. Formato de Intercâmbio de Mensagens:

SOAP - Simple Object Access Protocol (Protocolo Simples de Acesso a Objetos) é um protocolo usando o paradigma de objetos, criado especialmente para troca de informações estruturadas em uma plataforma descentralizada e distribuída para web services.



Java Web Service - SOAP?

2. Formato de Intercâmbio de Mensagens:

SOAP é construído usando XML para seu formato de mensagem e tem o objetivo de formar a camada base de uma pilha de protocolos de web services, fornecendo um framework de mensagens básico sob o qual os serviços web podem ser construídos.



Java Web Service - SOAP?

2. Formato de Intercâmbio de Mensagens:

SOAP consiste basicamente em três partes:

Envelope: Toda mensagem SOAP deve contê-lo. É o elemento raiz do documento XML. O Envelope pode conter declarações de namespaces e também atributos adicionais como o que define o estilo de codificação.



Java Web Service - SOAP?

2. Formato de Intercâmbio de Mensagens:

Header: É um cabeçalho opcional. Ele carrega informações adicionais. Quando utilizado, o Header deve ser o primeiro elemento do Envelope.

Body: Elemento obrigatório e contém a informação a ser transportada para o seu destino final. O elemento Body pode conter um elemento opcional Fault, usado para carregar mensagens de status e erros retornadas pelos web services.



Java Web Service - SOAP?

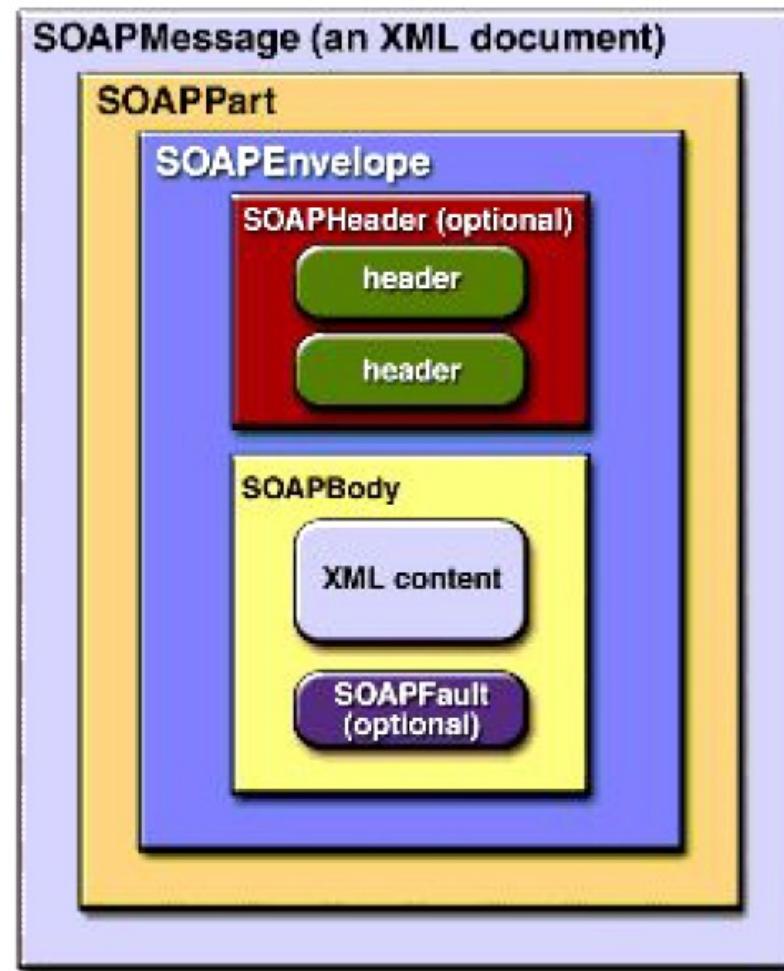
2. Formato de Intercâmbio de Mensagens:

Header: É um cabeçalho opcional. Ele carrega informações adicionais. Quando utilizado, o Header deve ser o primeiro elemento do Envelope.

Body: Elemento obrigatório e contém a informação a ser transportada para o seu destino final. O elemento Body pode conter um elemento opcional Fault, usado para carregar mensagens de status e erros retornadas pelos web services.



Java Web Service - SOAP?



Java Web Service - SOAP?

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:c="http://www.acmeOrders.com/OrderService"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <c:OrderMessage>
            <localElement>
                <FirstName>John</FirstName>
                <LastName>Smith</LastName>
                <Street>High Street</Street>
                <City>London</City>
                <ZipCode>W1A1AA</ZipCode>
                <PartNumber>ABC1234</PartNumber>
                <Quantity>1</Quantity>
            </localElement>
        </c:OrderMessage>
    </soap:Body>
</soap:Envelope>
```



Java Web Service - SOAP?

3. Protocolo de tráfego:

SOAP foi especificado como protocolo de serviço, com o objetivo de ser independente do protocolo de comunicação. Assim SOAP pode trafegado dentro de qualquer tipo de protocolo – HTTP, SMTP, FTP etc...

Para web services corporativos, foi selecionado o HTTP como protocolo de tráfego oficial, justamente pelo popularização e facilidades da internet.



Java Web Service - SOAP?

4. Descrição de Serviços:

Além dos formatos de mensagem e linguagem de marcação padrão, deve haver uma formato em que todos os fornecedores de web services possam utilizar para especificar detalhes de seus serviços.



Java Web Service - SOAP?

4. Descrição de Serviços:

- Como por exemplo:
- Quais as operações disponíveis?
- Como acessar cada uma delas?
- Quais parâmetros enviar em cada operação?
- O que será retornado caso sucesso na comunicação?
- O que será retornado caso falha da comunicação?



Java Web Service - SOAP?

4. Descrição de Serviços:

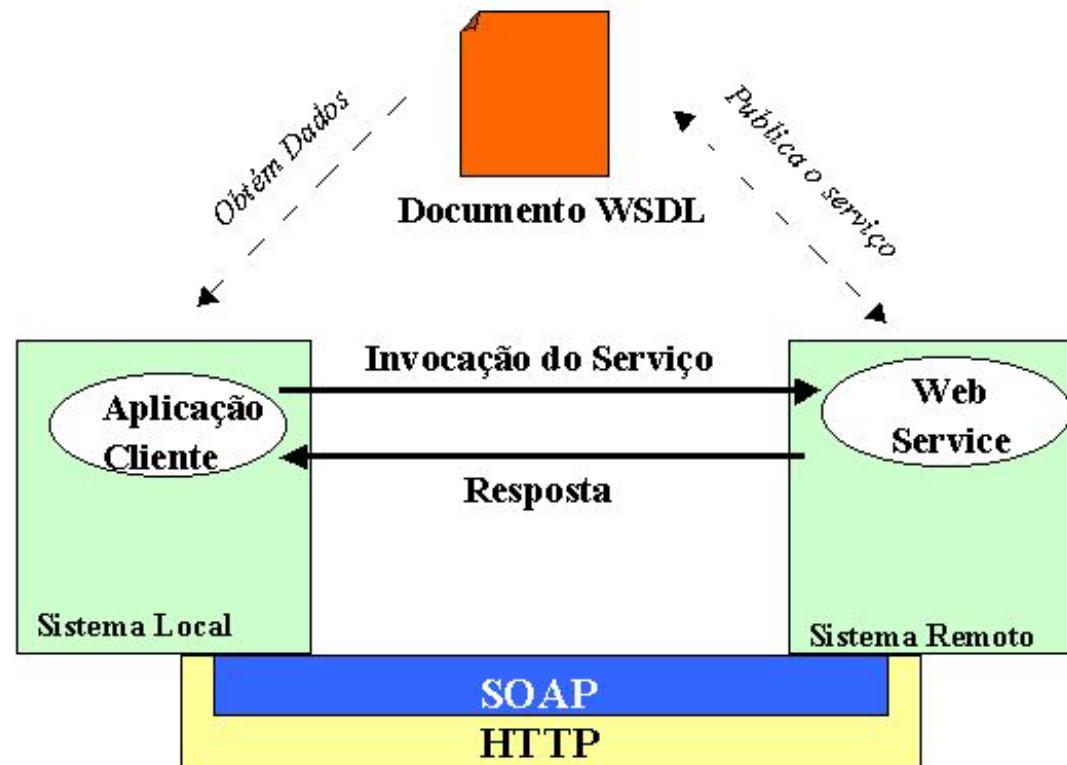
WSDL - Web Services Description Language é uma linguagem baseada em XML utilizada para descrever Web Services funcionando como um contrato do serviço. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.



Java Web Service - SOAP?

```
- <wsdl:definitions name="AdderService" targetNamespace="http://adder.remote.ipojofelix.apache.org/">
  - <wsdl:types>
    - <xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      targetNamespace="http://adder.remote.ipojofelix.apache.org/">
        <xsd:element name="add" type="tns:add"/>
    - <xsd:complexType name="add">
      - <xsd:sequence>
        <xsd:element name="arg0" type="xsd:int"/>
        <xsd:element name="arg1" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="addResponse" type="tns:addResponse"/>
  - <xsd:complexType name="addResponse">
    - <xsd:sequence>
      <xsd:element name="return" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
</wsdl:types>
- <wsdl:message name="addResponse">
  <wsdl:part element="tns:addResponse" name="parameters"> </wsdl:part>
</wsdl:message>
- <wsdl:message name="add">
  <wsdl:part element="tns:add" name="parameters"> </wsdl:part>
</wsdl:message>
- <wsdl:portType name="AdderServicePortType">
  - <wsdl:operation name="add">
    <wsdl:input message="tns:add" name="add"> </wsdl:input>
    <wsdl:output message="tns:addResponse" name="addResponse"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
- <wsdl:binding name="AdderServiceSoapBinding" type="tns:AdderServicePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="add">
    <soap:operation soapAction="" style="document"/>
    - <wsdl:input name="add">
      <soap:body use="literal"/>
    </wsdl:input>
    - <wsdl:output name="addResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Java Web Service - SOAP?

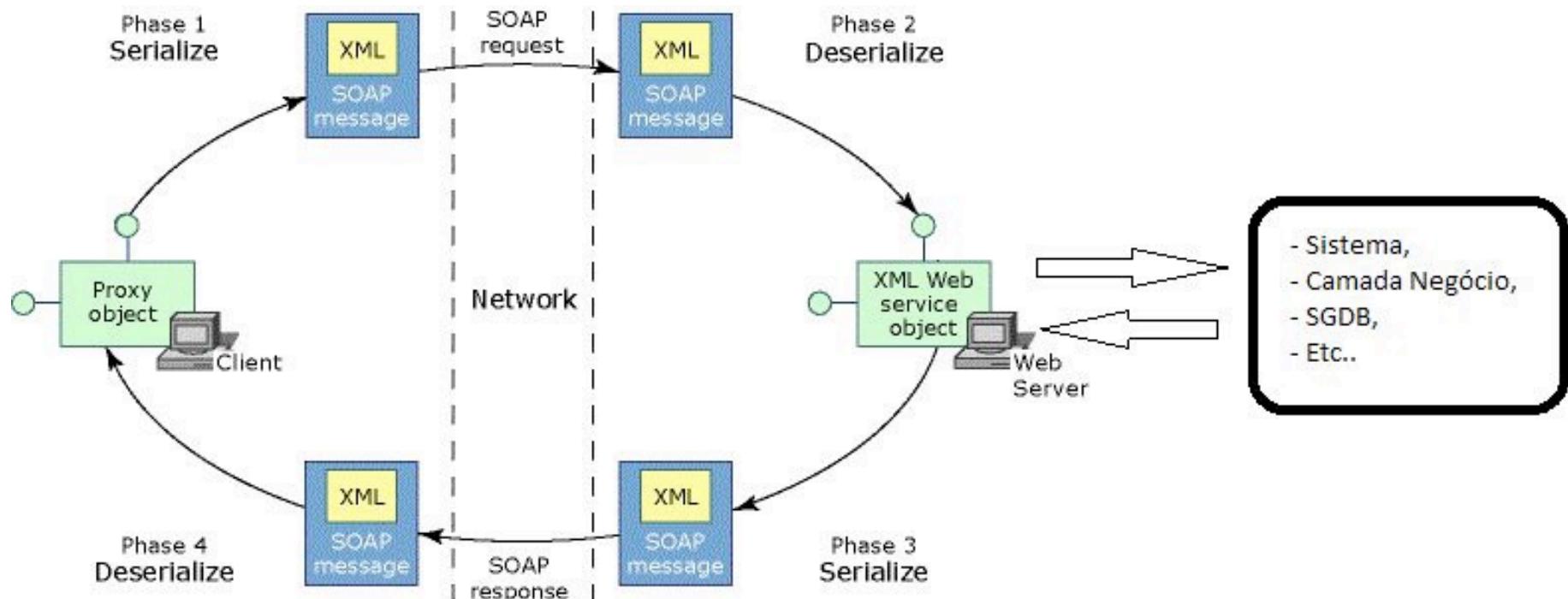


Arquitetura SOAP

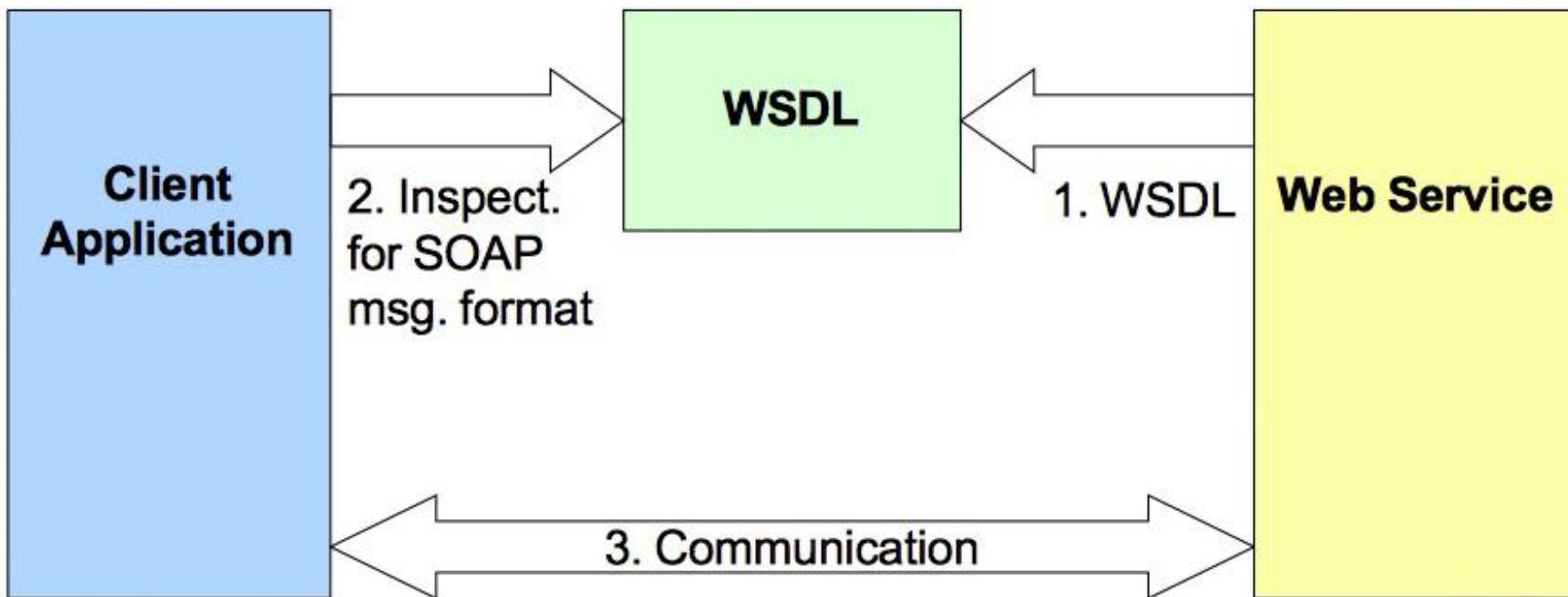
O que é?
Como se inicia?
Como funciona?



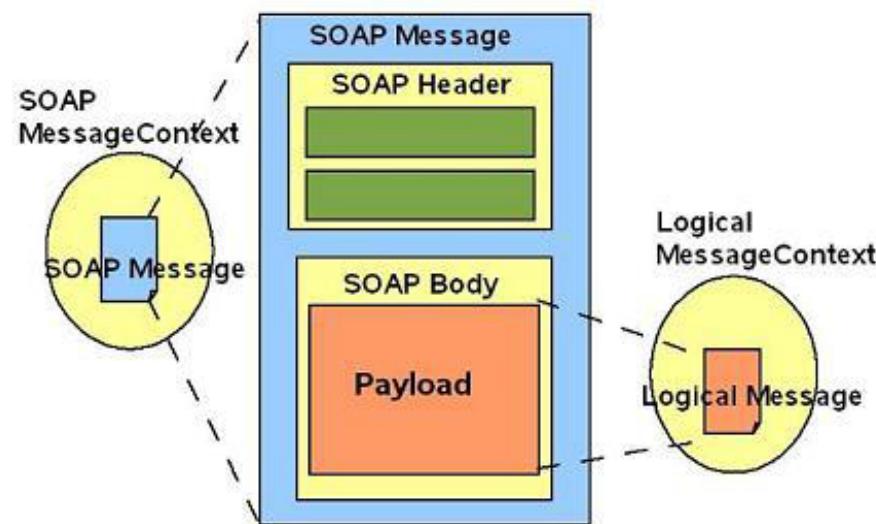
Arquitetura SOAP



Arquitetura SOAP



Arquitetura SOAP



Como eu faço para criar um web service SOAP então?



Java Web Service - SOAP

- 1. De posse de uma aplicação existente, defina quais serão os serviços expostos para serem invocados por outros sistemas.**
- 2. Aprenda XML e XSD.**
- 3. Aprenda a especificação SOAP e seus XSDs.**
- 4. Aprenda a especificação WSDL e seus XSDs.**
- 5. Gere o arquivo WSDL correspondente ao seu serviço, declarando as operações disponíveis.**



Java Web Service - SOAP

- Implementar uma aplicação web que:
- Receba requisições HTTP.
- Processe a requisição, lendo o SOAP de requisição.
- Execute as regras de negócio de serviço - camada de negócio.
- Processe a resposta, gerando a SOAP de resposta.



**Quanto tempo eu demoraria para
aprender todas as especificações e
implementar toda a infraestrutura
necessária ?**



**Existem produtos prontos no Java
para eu não perder meu tempo com
infraestrutura de SOAP e seus XSDs?**



Java Web Service - SOAP

Opções de produtos soap Java proprietários:

- Axis –<http://axis.apache.org/axis2/java/core/>
- CXF – <http://cxf.apache.org/>
- Spring SOAP Web Services -
<http://www.concretepage.com/spring-4/spring-4-soapweb-service-producer-consumer-example-with-tomcat>
- Groovy-wslite - <https://github.com/jwagenleitner/groovy-wslite>
- Muitos outros...



Java Web Service - SOAP

Suporte para o desenvolvimento de web services SOAP com Java está oficialmente disponível na especificação desde o JSE 6 sobre o nome de:

JAX-WS: Java API for XML and Java Web Services.



Java Web Service - SOAP

Opções de providers de JAX-WS:

- Metro – <https://javaee.github.io/metro/>
- Axis – <http://axis.apache.org/axis2/java/core/>
- CXF – <http://cxf.apache.org/>



Java Web Service - SOAP

A implementação de referência de JAX-WS é desenvolvida como um projeto de código aberto e é parte do projeto GlassFish, um servidor de aplicações Java EE de código aberto.

Chamada de Metro, a JAX-WS é uma RI e destinada a ser uma implementação de qualidade referência para a comunidade.
Link oficial - <https://javaee.github.io/metro/>



Java Web Service - SOAP

Desde a versão JDK6, a JSE padrão já vem com o provider Metro de JAX-WS (jars), não sendo necessário baixar implementação para esse produto.

Baixe somente em casos no qual se deseje usar outro provider em específico.



O que são servidores de aplicação?

O servidor de aplicação é um software voltado para desenvolvedores web. Também chamados de middleware, eles têm como objetivo conceder uma plataforma para que outras ferramentas sejam instaladas e executadas.



Exemplos de servidores de aplicação.

- **Apache Tomcat**
- **IBM WebSphere**
- **GlassFish Server**
- **IIS**
- **Wildfly**
- **Projeto Payara**



Instalação das ferramentas

Netbeans 20

JDK 21

GlassFish Server 7.0.9

SOAP UI



Programação Para Internet

Aula: WSDL Overview



Prof. Anderson Augusto Bosing

WSDL Overview



WSDL Overview

Objetivo de WSDL:

- 1. Estabelecer um contrato entre o serviço e a aplicação cliente.**
- 2. Descrever detalhes de serviço – “o que”, “como” e “aonde”.**

XML do WSDL é composto por 5 sessões.



WSDL Overview

<types > - define ou importa dos tipos de dados XML via XSD utilizados pelo serviço.

<message > - define as mensagens disponíveis pelo serviço.

<portType> - nomeia abstratamente as mensagens disponíveis pelo serviço, como se fossem uma classe/interface Java.



WSDL Overview

<binding> - implementação concreta do serviço, fornecendo detalhes do serviço. Seria a implementação concreta da sessão **<portType>**.

<service> - define as conexões disponíveis e o endereço físico a ser usado como se fosse uma aplicação distribuída.



WSDL Overview

Para todas as informações acesse a especificação oficial -
<http://www.w3.org/TR/wsdl>



WSDL Overview

Criar um web service SOAP, como provedor de serviços consiste em:

- 1. Fazer o WSDL, estabelecendo o contrato e os serviços disponibilizados.**

- 2. Fazer uma aplicação web que implemente o contrato dos serviços, recebendo as chamadas e processando das respostas SOAP.**



WSDL Overview

Desenvolvedores Java não precisam se preocupar esses detalhes infraestruturais, pois o JAX-WS:

- 1. Gera 100% automático o arquivo WSDL a partir de do modelo de objetos POJO.**
- 2. Disponibiliza o WSDL via URL padrão on-line.**
- 3. Faz 100% automático parse de requisição SOAP, com base no modelo de objeto.**
- 4. Faz 100% automático parse da resposta SOAP, com base no modelo de objeto.**



WSDL Overview

Observação:

O arquivo de WSDL gerado pode conter variações de provider para provider, podendo apresentar grandes diferenças entre eles.



WSDL - JAX-WS

Exercício 1:

- ▶ Implementar um serviço expondo operações de calculadora.
- ▶ Acessar o WSDL gerado.

?tester

?wsdl



O que são annotations no Java ?

Annotations são funções que podem ser usadas em classes, métodos ou atributos.

As annotations tem a seguinte assinatura @ seguido do nome da anotação.

As annotations podem servir para diferentes casos, como no exemplo acima a annotation @Table() serve para dizer que essa classe se refere a tabela de nome pessoa. Enquanto a annotation @id diz ao banco que aquele campo é uma chave primária e por fim a annotation @IsNotNull diz ao banco que aquela coluna é não nula.



Anotações JAX- WS

JAX-WS - Conceitos

Web services SOAP em Java são chamados – Endpoint Interface .

Endpoint interface em português “interface endpoint de serviço”, é um termo usado na plataforma Java Enterprise Edition ao expor qualquer componentes Java como um serviço web SOAP.

A maneira adequada para implementar um Endpoint SOAP em Java é:



Anotações JAX- WS

JAX-WS - Conceitos

1. Criando uma interface que defina os métodos que serão expostos no WSDL. Esta interface é chamada de SEI: Interface Endpoint Service.
2. Criando um objeto POJO que implemente polimorficamente a interface, fornecendo a implementação concreta dos métodos que serão expostos como serviços. Este objeto é chamada de SIB: Service Implementation Bean.



Anotações JAX- WS

Para habilitar o web service, usar as anotações do JAX-WS localizados no pacote javax.jws* na SEI e SIB para configurar declarativamente os detalhes e comportamento na exposição do web services. Veja as anotações básicas e suas funções:

Nas versões mais atuais do java as anotações do JAX-WS se encontram no pacote: jakarta.jws.*



Anotações JAX- WS

@WebService – indica que a classe ou interface implementa um web service. Deve ser especificado no SEI e SIB.

Todos os métodos public serão automaticamente publicado no serviço.



Anotações JAX- WS

@WebService.targetNamespace – customiza o namespace do WSDL gerado. Usa o nome da classe totalmente qualificado com padrão. Deve ser especificado no SEI e SIB.

@WebService. serviceName – customiza o nome do serviço no WSDL gerado. Usa o nome da classe totalmente qualificado com padrão. Deve ser especificado no SEI e SIB.

@WebService.endpointInterface – indica que o SIB implementa a interface SEI. Deve ser especificado somente no SIB, amarrando com a interface SEI.



Anotações JAX- WS

@WebMethod - Personaliza um método exposto como uma operação de serviço Web. O método associado deve ser público e seus parâmetros retornam valor, e as exceções devem seguir as regras definidas pelo padrão SOA. Deve ser especificado somente no SEI, amarrando com a interface SIB.

@WebResult - Personaliza o mapeamento do valor de retorno para uma parte WSDL e um elemento XML. Deve ser especificado somente no SEI, amarrando com a interface SIB.



Anotações JAX- WS

@WebParam - Customiza o mapeamento de um parâmetro individual para uma parte de mensagem para um elemento XML de serviço da Web.

Aplique esta anotação aos métodos em um SEI (Service Endpoint Interface) de cliente ou servidor ou em uma classe de implementação de terminal de servidor.

Propriedade header = true determina que será recepcionado dado no header da chamada.



Falhas de SOAP

Seguindo a especificação SOAP, quando uma requisição a um serviço soap falha por erro de negócio, ele deve gerar um SOAP FAULT e gerar na resposta do SOAP.

Desenvolvedores java não precisam se preocupar com isso, sendo que o JAX-WS mapeia automaticamente qualquer exception padrão Java para a especificação SOAP FAULT, tanto no servidor quanto no cliente, fazendo o parse de server e de cliente de forma transparentemente.



Anotações JAX- WS

@WebFault - A anotação @WebFault mapeia falhas de WSDL em exceções Java. Ela é utilizada para capturar o nome da falha durante a serialização do tipo JAXB que é gerado a partir de um elemento global mencionado por uma mensagem de falha de WSDL. Ela pode ser utilizada também para customizar o mapeamento de exceções específicas do serviço para falhas de WSDL.

Essa anotação pode ser aplicada apenas a uma classe de implementação de falha no cliente ou servidor.



Falhas de SOAP

Exercicio:

- ▶ Implementar um serviço de cadastro de produtos que gere falhas de soap quando o nome e valor do produto não for corretamente preenchido.

- ▶ Acessar o WSDL gerado.

- ▶ Verifique o tipo do <fault message> da operação de retorno.



Anotações JAX- WS

Exercício 2:

Implementar um web service de Livraria On-Line, customizando o nome do serviço WSDL.

Acessar o WSDL gerado.

Consumir o WSDL no SOAP UI.



Anotações JAX- WS

Observação:

Veja que o desenvolvedor Java também não precisa se preocupar com a criação do XSD de objeto de negócio trafegado dentro do protocolo SOAP, sendo que o JAX-WS gera o XSD no WSDL. Acesse o <xsd:import> do WSDL no exercício anterior e veja o XSD online do objeto Livro.



Anotações JAX- WS

Para excluir métodos use na SEI: Use a anotação @WebMethod(exclude=true) somente naquele(s) métodos que se deseje excluir.

Eles não serão gerados no WSDL e não serão invocados pelo SOAP.



Handler Framework JAX- WS

O JAX-WS encapsula toda complexidade do camada SOAP, escondendo como infra-estrutura. O desenvolver então fica livre para se concentrar na implementação das regras dos serviços, não se preocupando com detalhes das especificações (infra-estrutura).



Handler Framework JAX- WS

Mas em alguns casos é interessante que o desenvolvedor tenha acesso direto ao protocolo HTTP e SOAP caso necessite implementar alguma coisa customizada.

Para isso, o JAX-WS disponibiliza o framework chamado de “Handler” que dá acesso a manipulação direto aos recursos infra-estruturais.



Handler Framework JAX- WS

O que é?

- ▶ Recursos oferecidos pela JAX-WS com o objetivo de interpor as requisições e as respostas, tanto no cliente como no servidor.
- ▶ Eles permitem que você centralize um determinado comportamento comum que corte várias partes do seu aplicativo, evitando espalhar essa determinada intenção em vários lugares diferentes.



Handler Framework JAX- WS

Para que serve?

- ▶ Ter um ponto de implementação global que ofereca a solução um ponto de cross-cutting concern (https://pt.wikipedia.org/wiki/Cross-cutting_concern) relacionado ao gerenciamento do conteúdo das requisições, no qual exista a possiblidade de acessar e alterar itens do próprio conteúdo trafegado



Handler Framework JAX- WS

Para quem conhece...

- ▶ Mesma coisa de um servlet filter.
- ▶ Mesma coisa de um serviço AOP.
- ▶ Mesma coisa de um pattern decorator



Handler Framework JAX- WS

Para que serve no Servidor?

- ▶ Executando alguma coisa de interesse da solução, acessando, manipulando, alterando o conteúdo da requisição, antes de ser enviado ao cliente.

Para que serve no Cliente?

- ▶ Executando alguma coisa de interesse da solução, acessando, manipulando, alterando o conteúdo da requisição, antes de ser enviado ao servidor.



Handler Framework JAX- WS

Existem 2 tipos de handlers:

- 1. SOAPHandler:** usado para manipular a mensagem SOAP, tendo acesso aos detalhes de SOAP e payload da mensagem.

- 2. LogicalHandler:** usado para manipular a mensagem independente de protocolo, tendo acesso somente a algumas propriedades genéricas e payload da mensagem.



Handler Framework JAX- WS

Os handlers funcionam como fosse uma “cadeia de filtros” que são aplicados dinamicamente antes da entrada e saída das mensagens SOAP. (Da mesma forma que os Servlet Filter) O desenvolvedor pode, de forma declarativa e dinâmica adicionar e retirar quantos handlers forem necessários.



Handler Framework JAX- WS

Qual a Aplicabilidade do handler framework servidor/cliente?

- ▶ Não duplicar código genérico nos métodos de serviços.
- ▶ Validação coisas conteúdo SOAP, HTTP, etc.
- ▶ Autenticação.
- ▶ Autorização.
- ▶ Logg.
- ▶ Auditoria
- ▶ Inicialização e fechamento de recursos da solução.



Handler Framework JAX- WS

Como criar um SOAPHandler?

1. Crie uma classe implementando a interface

`javax.xml.ws.handler.soap.SOAPHandler.`

2. Sobreponha o método `public boolean handleMessage(SOAPMessageContext context)`



Handler Framework JAX- WS

O que é e para que serve o SOAPMessageContext?

- ▶ Dar acesso e controle ao conteúdo da corpo SOAP da requisição e da resposta.

Veja javadoc:

<https://docs.oracle.com/javase/7/docs/api/javax/xml/ws/handler/soap/SOAPMessageContext.html>.



Handler Framework JAX- WS

Como configurar SOAPHandle no servidor?

1. São configurados usando um xml específico do JAX-WS.
2. Adicionando a seguinte anotação na interface SIB, indicando o path do xml:
`@HandlerChain(file="pacote1/pacote1/handlers.xml").`



Handler Framework JAX- WS

Exercício:

- ▶ Implementar um serviço hora/data que retorne a data e hora do servidor.
- ▶ Implemente e configure um SOAPHandler simulando LOG de acesso.
- ▶ Acessar o WSDL gerado.



E como eu conecto meu WebService no banco de dados?

**Através do pool de conexões
que o servidor de aplicação
nos disponibiliza.**



1. Baixar o driver JDBC do PostgreSQL:

- Acesse o site oficial do PostgreSQL: <https://jdbc.postgresql.org/> ou o mvn repository.
- Selecione a versão do driver compatível com sua versão do PostgreSQL.



2. Copiar o driver JDBC para o Glassfish:

- Acesse a pasta de instalação do Glassfish.
- Navegue até a pasta glassfish/domains/[nome_do_seu_dominio]/lib.
- Copie o arquivo JAR do driver JDBC baixado para esta pasta.



3. Reiniciar o Glassfish:

- Reinicie o servidor Glassfish para que o driver JDBC seja carregado.



4. Criar um pool de conexões no Glassfish:

- Acesse o console de administração do Glassfish.
- Faça login com o usuário administrador.
- Navegue até Recursos > JDBC > Pools de conexões JDBC.
- Clique em Novo para iniciar o assistente de criação de pool de conexões.
- Digite um nome para o pool de conexões (por exemplo, postgresPool).
- Selecione java.sql.Driver como tipo de recurso.
- Em Fornecedor do banco de dados, selecione PostgreSQL.
- Clique em Avançar.



5. Configurar as propriedades do pool de conexões:

- Na próxima tela, configure as propriedades do pool de conexões:
 - URL do JDBC: `jdbc:postgresql://localhost:5432/nome_do_banco_de_dados`
 - Nome de usuário: O nome de usuário do PostgreSQL.
 - Senha: A senha do PostgreSQL.
 - Classe do driver: `org.postgresql.Driver`



6. Criar um recurso JDBC no Glassfish:

- Navegue até Recursos > JDBC > Conexões JDBC.
- Clique em Novo para criar um novo recurso JDBC.
- Digite um nome para o recurso JDBC (por exemplo, jdbc/postgres).
- Selecione o pool de conexões criado anteriormente (por exemplo, postgresPool) como pool de conexões.
- Clique em Salvar.



Programação Para Internet

**Aula: Java Web Service -
REST**



Prof. Anderson Augusto Bosing

O que é REST?

Contexto?

O que é?

Para que serve?

Como?



Java Web Service - REST

Era em um época no qual implementávamos web services usando estilo SOAP -

[https://pt.wikipedia.org/wiki/SOAP.](https://pt.wikipedia.org/wiki/SOAP)

Nesse momento, REST foi anunciado como como um estilo de web services concorrente ao SOAP, no qual apresentava melhorias, leveza, facilidades e maior escalabilidade.



Java Web Service - REST

E assim, o REST foi ao poucos sendo adicionadas plataformas e passou a ser utilizado com mais uma opção na criação de web services.

A partir de então, os desenvolvedores possuíam duas diferentes opções para fazer web services SOAP ou REST.



Java Web Service - REST

“Representational State Transfer” (REST) foi a sigla apresentada pela tese de doutorado de Roy Fielding, autor do protocolo HTTP, que defende a utilização do protocolo HTTP puro como web service sem a necessidade de especificações de protocolos adicionais como o SOAP e etc.

No REST, Roy estabeleceu um conjunto de princípios que definem como os recursos padrões de comunicação existentes no HTTP devem ser usados para ser utilizado como solução de web services.



Java Web Service - REST

Assim, REST não criou e nem inventou nada de novo, simplesmente passou a usar o antigo e velho conhecido protocolo HTTP como solução final para criação de web services para oferecer integração 100% interoperável.



Java Web Service - REST

REST é:

Uma técnica de engenharia de engenharia de software que faz uso do protocolo HTTP como um estilo arquitetural utilizado para comunicação (troca de informações) entre sistemas distribuídos que lidam simultaneamente com diversos formatos textos, imagens, vídeos, etc.



Java Web Service - REST

RESTFul é nome dado para as soluções que seguem os princípios de REST como solução de web services.



Java Web Service - REST

Para poder criar web services REST com eficiência é necessário conhecer pelo menos básico do protocolo

HTTP:

Wiki:

https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Especificação:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>



Java Web Service - REST

Para usar HTTP como web services, é necessário seguir os cinco princípios fundamentais:

1. Dê a todas as coisas um identificador [URL].
2. Utilize protocolo HTTP para definir as ações e suas respostas[HTTP].
3. Recursos com múltiplas representações [MIME].
4. Comunique sem estado [Stateless].
5. Vincule as coisas [links].



Java Web Service - REST

1. Dê a todas as coisas um identificador:

Todos os serviços oferecidos no seu web services rest deverá ter um identificador único na web. Use URLs para identificar tudo o que precisar ser identificado, especifique todos os recursos de "alto nível" que seu aplicativo oferece, se eles representam itens individuais, conjuntos de itens, objetos virtuais e físicos, ou resultados de computação.



Java Web Service - REST

1. Dê a todas as coisas um identificador:

Assim, uma solução cliente pode interagir com um recurso conhecendo apenas seu identificador (URL) não necessitando conhecer a infraestrutura que a faz funcionar.

Isso é: baixo acoplamento, independência de plataforma e nenhuma necessidade de protocolos adicionais.



Java Web Service - REST

1. Dê a todas as coisas um identificador:

Exemplos:

`http://vendas.com/cliente/1234`

`http://vendas.com/vendas/2007/10/776654`

`http://vendas.com/produtos/4554`

`http://vendas.com/boleto/5248`



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Use os métodos do protocolo HTTP para especificar quais as ações serão efetuadas sobre os recursos disponibilizado nas URI's.



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Mapeando para um CRUD básico, ficaria assim os quatro mais utilizados:

- ▶ Create - POST
- ▶ Read - GET
- ▶ Update - PUT
- ▶ Delete - DELETE



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Segue exemplos crud:

- ▶ POST `http://vendas.com/cliente` = gravar dados do cliente que será enviado dentro do corpo do protocolo HTTP.
- ▶ GET `http://vendas.com/cliente/1234` = obter dados do cliente 1234
- ▶ PUT `http://vendas.com/cliente/1234` = atualizar dados do cliente que será enviado dentro do corpo do protocolo HTTP.
- ▶ DELETE `http://vendas.com/cliente/1234` = deletar do sistema o cliente 1234



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

Use status HTTP que define a resposta ao cliente daquela determinada requisição:

- ▶ HTTP 1xx – Informativo.
- ▶ HTTP 2xx – Sucesso.
- ▶ HTTP 3xx – Redirecionamento.
- ▶ HTTP 4XX – Erro do cliente
- ▶ HTTP 5XX – Erro do servidor.



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

- ▶ SUCESSO 200 – OK – indica que o pedido foi processado com sucesso. Usado com GET para retornar o(s) recurso(s) requerido(s).

- ▶ SUCESSO 201 – CRIADO – indica que o(s) recurso(s) do pedido foram criadas com sucesso. Deve retornar no header da resposta a URI para o(s) recurso(s) criado(s). Usado com POST E PUT.



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

- ▶ SUCESSO 202 – APROVADO – indica que o pedido foi aprovado mas não processado. Utilizado para processamento assíncrono.
- ▶ SUCESSO 204 – SEM CONTEÚDO – indica que o pedido foi processado com sucesso. Usado com UPDATE e DELETE sem resposta (não tem BODY).
- ▶ ERRO CLIENTE 400 – PEDIDO INVÁLIDO – indica que o cliente enviou o pedido com sintaxe errada.



Java Web Service - REST

2. Utilize protocolo HTTP para definir as ações:

- ▶ **ERRO CLIENTE 401 – NÃO AUTORIZADO** – indica que o pedido foi feito por requerer credencial não existente ou inválidas.
- ▶ **ERRO CLIENTE 404 – NÃO ENCONTRADO** – indica que o servidor não encontrou nada que corresponda à URI do pedido.
- ▶ **ERRO SERVIDOR 500 – ERRO INTERNO DO SERVIDOR** – o servidor encontrou uma condição inesperada que o impediu de cumprir a solicitação.



Java Web Service - REST

3. Recursos com múltiplas representações:

Use o cabeçalho padrão de MIME do protocolo HTTP para especificar pelo cliente qual será o tipo do resultado esperado por ele na resposta das URI's, fazendo o próprio cliente decidir qual é o tipo do resultado que ele espera receber.



Java Web Service - REST

3. Recursos com múltiplas representações:

Use o cabeçalho padrão de MIME do protocolo HTTP para especificar pelo servidor qual será o tipo do resultado da resposta disponibilizado nas URI's fazendo com que um mesmo recurso possa ter dinamicamente múltiplos representações diferentes.



Java Web Service - REST

3. Recursos com múltiplas representações:

Exemplos: Resposta de pedido do cliente em txt:

HTTP-Version: HTTP/1.0 200 OK

Content-Length: 3012

Content-Type: text/txt

<body>

Ricardo, 25 anos de idade, engenheiro.

</body>



Java Web Service - REST

4. Comunique sem estado:

Crie seu web services totalmente STATELESS, sem armazenar estado do cliente no servidor. Isto permite alguns benefícios como:

- ▶ **Visibilidade:** não é necessário verificar a requisição para determinar sua natureza;
- ▶ **Fiabilidade:** a tarefa de recuperação a falhas parciais se torna fácil;
- ▶ **Escalabilidade:** não necessitando armazenar estados entre as requisições, o servidor pode rapidamente liberar recursos facilitando seu gerenciamento e ter escalabilidade horizontal fácil e rápida.



Java Web Service - REST

5. Vincule as coisas:

Sempre que necessário e quando houver aplicabilidade, use links para referenciar outros recursos que possam ser identificadas dinamicamente, permitindo que a hipermídia funcione como máquina de estado da solução, a partir da própria aplicação, sem interferência humana.

Conceito complexo chamado de "HATEOAS":



Java Web Service - REST

“Na teoria eu sei que tenho que tentar seguir, mas na prática eles realmente funcionam?”



Princípios - REST

Princípios são ideais e valores almejados, mas que na práticas e mundo real, em certos casos podem não ser aplicáveis.

Sendo assim, tente fazer sua solução web service rest o mais RESTFUL possível, seguindo esses princípios.

Se começar um projeto novo, do zero, siga eles!



Princípios - REST

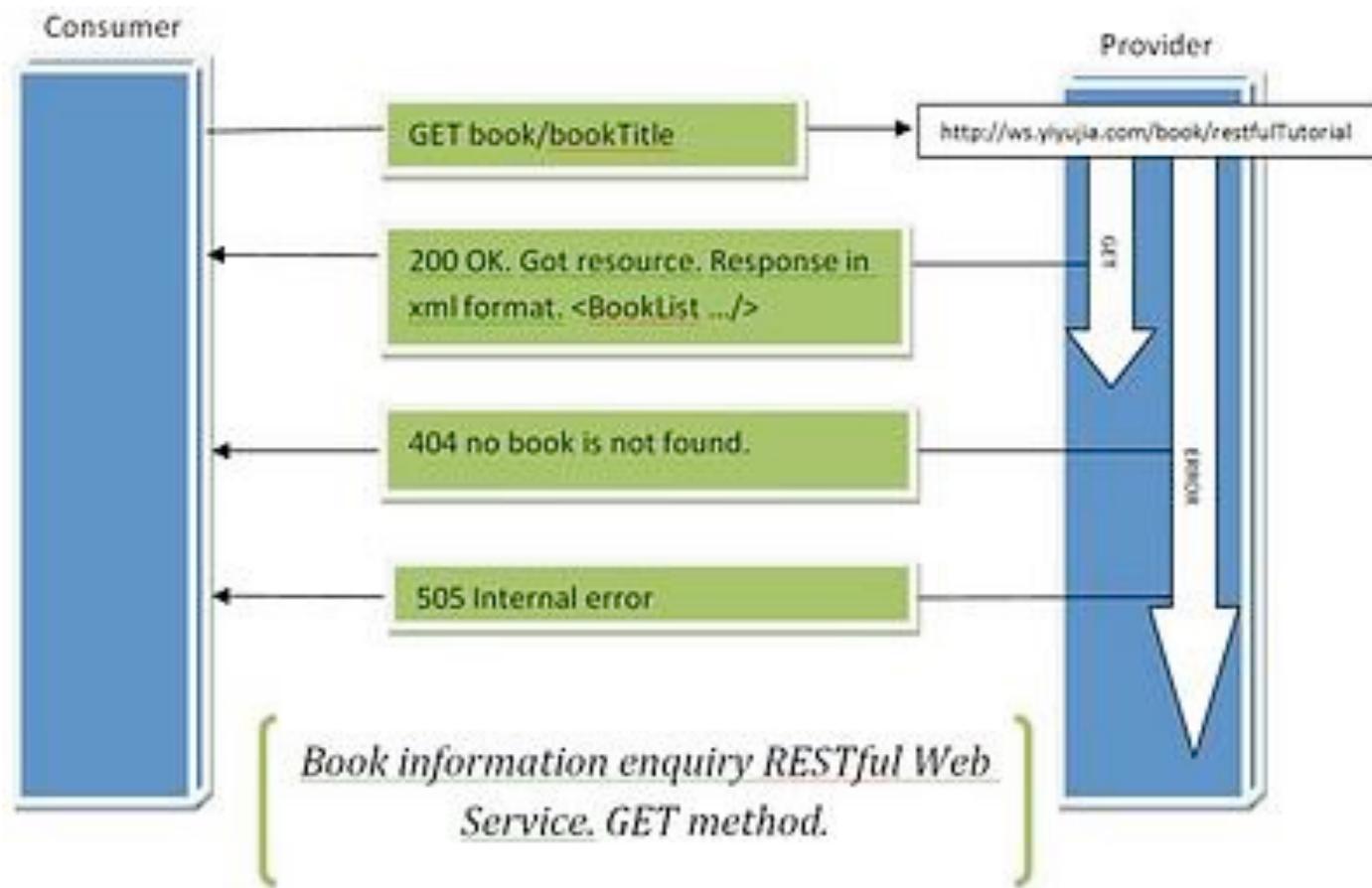
Dica do Professor:

- ▶ Na prática, conseguimos usar os 3 primeiros princípios que são fundamentais e básicos para fazer um rest real.

- ▶ No caso do HATEOAS, somente em casos que existe aplicabilidade e ganho real de uso. Caso contrário, nem usamos isso. Eu não nunca precisei usar.



Arquitetura - REST



Princípios - REST

Workflow de execução:

1. A solução servidora disponibiliza a lista de recursos através de URI's e métodos HTTP.
2. A solução consumidora cria e envia uma requisição HTTP para o web service REST, usando uma URL e um específico método HTTP.



Princípios - REST

Workflow de execução:

3. A solução cliente envia informações via parâmetros HTTP dentro do protocolo ou na própria URL destino.
4. A solução servidora descobre o que o cliente quer pela URL e método HTTP. O processamento é feito retornando algum código de resposta HTTP válido e ou algum recurso dentro da resposta HTTP.



Princípios - REST

Workflow de execução:

5. A solução cliente consumidora descobre qual foi a resposta (sucesso, erro, redirecionamento, etc) pelo código da resposta HTTP, juntamente do conteúdo vindo do processamento dentro da resposta do protocolo HTTP.



Princípios - REST

URIs e métodos HTTP do processo de leilão

URI	Método	Formato	Efeito
/item/{id}	GET	Item	Busca um item.
	PUT	Item	Atualiza um item.
/item/{id}/ofertas	GET	Coleção de ofertas	Busca ofertas feitas sobre um item.
	POST	Oferta	Adiciona oferta a um item.
/oferta/{id}	GET	Oferta	Busca uma oferta.
	PUT	Oferta	Atualiza uma oferta.
	DELETE	-	Remove uma oferta.
/usuario	POST	Usuario	Cadastra um usuário.
/usuario/{id}	GET	Usuario	Busca um usuário.
	PUT	Usuario	Atualiza um usuário.
/usuario/{id}/avaliacoes	GET	Coleção de avaliações	Busca as avaliações recebidas por um usuário.
/usuario/{id}/itens	GET	Coleção de itens	Busca os itens anunciados por um determinado usuário.
	POST	Item	Usuário coloca novo item à venda.
/avaliacao/{id}	GET	Avaliação	Busca uma determinada avaliação.
/avaliacao/de/{id}/para/{id}	POST	Avaliação	Realização da avaliação de um usuário sobre outro.
/services	GET	Coleção de URIs	Consulta URIs e métodos HTTP disponíveis para acesso.

Protocolos de comunicação

Requisições HTTP

Requisições HTTP são mensagens enviadas pelo cliente para iniciar uma ação no servidor. Suas linhas iniciais contêm três elementos:

1) Um método HTTP, um verbo (como GET, PUT ou POST) ou um nome (como HEAD ou OPTIONS), que descrevem a ação a ser executada. Por exemplo, GET indica que um recurso deve ser obtido ou POST significa que dados são inseridos no servidor (criando ou modificando um recurso, ou gerando um documento temporário para mandar de volta).



Protocolos de comunicação

Requisições HTTP

2) O alvo da requisição, normalmente um URL, ou o caminho absoluto do protocolo, porta e domínio são em geral caracterizados pelo contexto da requisição. O formato deste alvo varia conforme o método HTTP.

Sintaxe geral de uma URL:

<protocolo>://<servidor>:<porta>/<caminho>/<recurso>

- A porta é opcional para serviços em portas default
- Caminho e recurso podem ser omitidos (URLs parciais)
- URLs podem conter dados depois do nome do recurso



Protocolos de comunicação

Requisições HTTP

Exemplos de URLs:

http://java.sun.com/docs/servlets/servlets.html

http://java.sun.com/docs/servlets/

http://java.sun.com/cgi-bin/reverse?string=fred

**http://localhost:8080/fred/servlets/ListaServlet?tipo
=superior&curso=334**



Protocolos de comunicação

Cabeçalhos HTTP

Cabeçalhos HTTP de uma requisição seguem a mesma estrutura básica de um cabeçalho HTTP: uma cadeia de caracteres insensível à caixa seguida de dois pontos (':') e um valor cuja estrutura depende do cabeçalho. O cabeçalho inteiro, incluindo o valor, consiste em uma única linha, que pode ser bem grande. Há numerosos cabeçalhos de requisição disponíveis.



Protocolos de comunicação

Cabeçalhos HTTP

POST / HTTP/1.1

```
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh; ... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,...,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345
```

Request headers

General headers

Entity headers

-12656974

(more data)



Protocolos de comunicação

Corpo HTTP

Corpo A parte final da requisição é o corpo. Nem todas as requisições tem um: as que pegam recursos, como GET, HEAD, DELETE, ou OPTIONS, usualmente não precisam de um. Algumas requisições enviam dados ao servidor a fim de atualizá-lo: é o caso frequente de requisições POST (contendo dados de formulário HTML)



Protocolos de comunicação

Respostas HTTP

A linha inicial de uma resposta HTTP, chamada de linha de status, contém a seguinte informação: A versão do protocolo, normalmente HTTP/1.1. Um código de status, indicando o sucesso ou falha da requisição. Códigos de status comuns são 200, 404, ou 302 Um texto de status. Uma descrição textual breve, puramente informativa, do código de status a fim de auxiliar o entendimento da mensagem HTTP por humanos. Uma linha de status típica se parece com: HTTP/1.1 404 Not Found.



Protocolos de comunicação

Verbos HTTP

A ideia geral é a seguinte: seu serviço vai prover uma url base e os verbos HTTP vão indicar qual ação está sendo requisitada pelo consumidor do serviço.



Verbos HTTP

DELETE

O método de requisição HTTP **DELETE** remove o recurso especificado.

Exemplo:

DELETE maxrest/rest/mbo/asset/1234 HTTP/1.1



Verbos HTTP

GET

O método HTTP GET solicita uma representação do recurso especificado. Solicitações usando GET só devem recuperar dados.

Exemplo:

GET maxrest/rest/mbo/asset/1234 HTTP/1.1



Verbos HTTP

POST

O método HTTP POST envia dados ao servidor. O tipo do corpo da solicitação é indicado pelo cabeçalho Content-Type.

Exemplo:

POST / HTTP/1.1

Host: foo.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 13

say=Hi&to=Mom



Verbos HTTP

PUT

O método de requisição HTTP PUT cria um novo recurso ou substitui uma representação do recurso de destino com os novos dados.

Exemplo:

PUT /new.html HTTP/1.1

Host: example.com

Content-type: text/html

Content-length: 16

<p>New File</p>



HTTP Response Status Code

100 Continue

O Status HTTP 100 Continue indica que até o momento tudo está OK e que o cliente pode continuar com a requisição ou ignorar caso já tenha terminado.



HTTP Response Status Code

200 OK

O código HTTP 200 OK é a resposta de status de sucesso que indica que a requisição foi bem sucedida. Uma resposta 200 é cacheável por padrão.

O significado de sucesso depende do método de requisição HTTP:
GET: O recurso foi carregado e transmitido no corpo da mensagem.

POST: O recurso descrevendo o resultado da ação é transmitido no corpo da mensagem.



HTTP Response Status Code

200 OK

O resultado de sucesso de um PUT ou DELETE geralmente não são 200 OK, e sim 204 No Content (ou 201 Created quando o recurso é carregado pela primeira vez).



HTTP Response Status Code

201 Created

O status HTTP "201 Created" é utilizado como resposta de sucesso, indica que a requisição foi bem sucedida e que um novo recurso foi criado. Este novo recurso é efetivamente criado antes do retorno da resposta e o novo recurso é enviado no corpo da mensagem (pode vir na URL ou na header Location).

Comumente, este status é utilizado em requisições do tipo POST.



HTTP Response Status Code

400 Bad Request

O código de status de resposta HTTP 400 Bad Request indica que o servidor não pode ou não irá processar a requisição devido a alguma coisa que foi entendida como um erro do cliente (por exemplo, sintaxe de requisição mal formada, enquadramento de mensagem de requisição inválida ou requisição de roteamento enganosa).



HTTP Response Status Code

401 Unauthorized

O código de resposta de status de erro do cliente HTTP 401 Unauthorized indica que a solicitação não foi aplicada porque não possui credenciais de autenticação válidas para o recurso de destino.



HTTP Response Status Code

403 Forbidden

O código de resposta de status de erro do cliente HTTP 403 Forbidden indica que o servidor entendeu o pedido, mas se recusa a autorizá-lo.

Esse status é semelhante ao 401 , mas neste caso, a re-autenticação não fará diferença. O acesso é permanentemente proibido e vinculado à lógica da aplicação (como uma senha incorreta).



HTTP Response Status Code

404 Not Found

A resposta de erro 404 Not Found indica que o servidor não conseguiu encontrar o recurso solicitado. Normalmente, links que levam para uma página 404 estão quebrados ou desativados.

Um código 404 não indica se o recurso está indisponível temporariamente ou se o recurso foi permanentemente removido.



HTTP Response Status Code 405 Method Not Allowed

Este status de resposta indica que o verbo HTTP utilizado não é suportado, por exemplo: a requisição ocorre por meio de um get, porém o único método disponível é o post.Curiosidade: Existem um método chamado OPTIONS que retorna todos os verbos suportados naquela requisiçãoobs: ele também pode não ser permitido



HTTP Response Status Code

500 Internal Server Error

Quando o servidor retorna um código de erro (HTTP) 500, indica que encontrou uma condição inesperada e que o impediu de atender à solicitação.

Essa resposta de erro é uma resposta genérica "abrangente". Às vezes, os arquivos log de servidores podem responder com um status code 500 acompanhado de mais detalhes sobre o request para evitar que no futuro erros desse tipo possam voltar a acontecer.



HTTP Response Status Code 503 Service Unavailable

O código de resposta de erro de servidor 503 Service Unavailable do HTTP indica que o servidor não está pronto para lidar com a requisição.

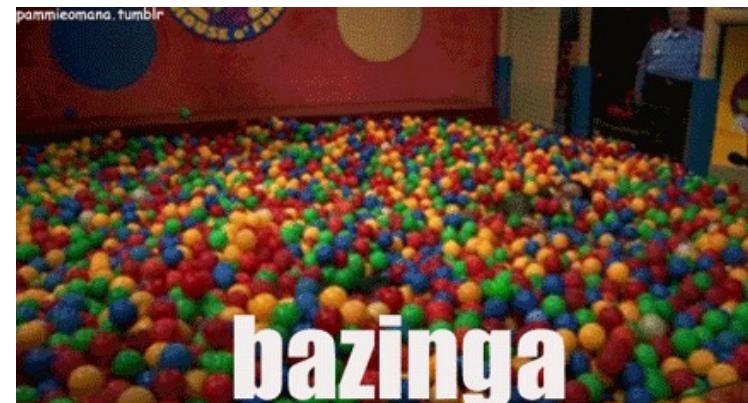
Causas comuns são um servidor que está em manutenção ou sobrecarregado. Esta resposta deve ser usada para condições temporárias



HTTP Response Status Code

418 I'm a teapot

O código de erro HTTP para o cliente 418 I'm a teapot indica que o servidor se recusa a preparar café por ser um bule de chá. Este erro é uma referência ao Hyper Text Coffee Pot Control Protocol, que foi uma piada de 1º de abril de 1998.



JSON x XML

- Ambos são formatos de dados para recebimento e envio de dados com servidores web.
- Os dois modelos representam informações no formato texto.
- Ambos possuem natureza auto-descritiva (ou seja, basta “bater o olho” em um arquivo JSON ou em um arquivo XML para entender o seu significado). etc.
- Ambos são capazes de representar informação complexa, difícil de representar no formato tabular. Alguns exemplos: objetos compostos (objetos dentro de objetos), relações de hierarquia, atributos multivvalorados, arrays, dados ausentes, etc.



JSON x XML

- Ambos podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627.
- Ambos são independentes de linguagem. Dados representados em XML e JSON podem ser acessados por qualquer linguagem de programação, através de API's específicas (API javascript por exemplo).



JSON x XML

```
{"employees": [  
    { "firstName": "John", "lastName": "Doe" },  
    { "firstName": "Anna", "lastName": "Smith" },  
    { "firstName": "Peter", "lastName": "Jones" }  
]
```

```
<employees>  
  <employee>  
    <firstName>John</firstName> <lastName>Doe</lastName>  
  </employee>  
  <employee>  
    <firstName>Anna</firstName> <lastName>Smith</lastName>  
  </employee>  
  <employee>  
    <firstName>Peter</firstName> <lastName>Jones</lastName>  
  </employee>  
</employees>
```



Criando nossa primeira API com Java – Principais Tecnologias Envolvidas

- ✓ JAX-RS é uma especificação que permite criar RESTful Web services (análoga a JAX-WS para SOAP).
- ✓ Jersey é a principal implementação da especificação JAX-RS.
- ✓ Wildfly é o servidor Web utilizado para executar as aplicações Java.



Instalação das ferramentas

Netbeans

JDK

Wildfly

Postman (Documentação Collection, Requests)

Console do Navegador – Chrome/Firefox/Edge/Opera.

GIT



Instalação das ferramentas

Wildfly 31.0.1

Para adicionar um usuário admin ao wildfly precisamos executar o seguinte processo:

Acesse o seguinte path onde a instalação do wildfly foi descompactada: [~/wildfly-26.1.3.Final/bin](#)

Execute o script add-user.

Linux/Mac

`./add-user.sh`

Windows

`add-user.bat`



Instalação das ferramentas

```
Last login: Sun Feb 26 08:45:46 on console
~/wildfly-26.1.3.Final/bin
> ./add-user.sh
What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): a
Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : unipar
Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.
- The password should be different from the username
- The password should not be one of the following restricted values {root, admin, administrator}
- The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password :
WFLYDM0102: Password should have at least 1 non-alphanumeric symbol.
Are you sure you want to use the password entered yes/no? yes
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]:
About to add user 'unipar' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'unipar' to file '/Users/andersonbosing/wildfly-26.1.3.Final/standalone/configuration/mgmt-users.properties'
Added user 'unipar' to file '/Users/andersonbosing/wildfly-26.1.3.Final/domain/configuration/mgmt-users.properties'
Added user 'unipar' with groups to file '/Users/andersonbosing/wildfly-26.1.3.Final/standalone/configuration/mgmt-groups.properties'
Added user 'unipar' with groups to file '/Users/andersonbosing/wildfly-26.1.3.Final/domain/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server Jakarta Enterprise Beans calls.
yes/no? yes
To represent the user add the following to the server-identities definition <secret value="dW5pcGFyMTIz" />
~/wildfly-26.1.3.Final/bin
> [Parameter: artifactId, Value: minha-primeira-api]
BUILD SUCCESS
Total time: 1.716 s
52s | 09:32:10
```



Por que o Standalone ?

https://docs.wildfly.org/19.1/Getting Started_Guide.html#wildfly-10-configurations



Criando conexão com o Banco de Dados Postgres no Wildfly.

Add Datasource X

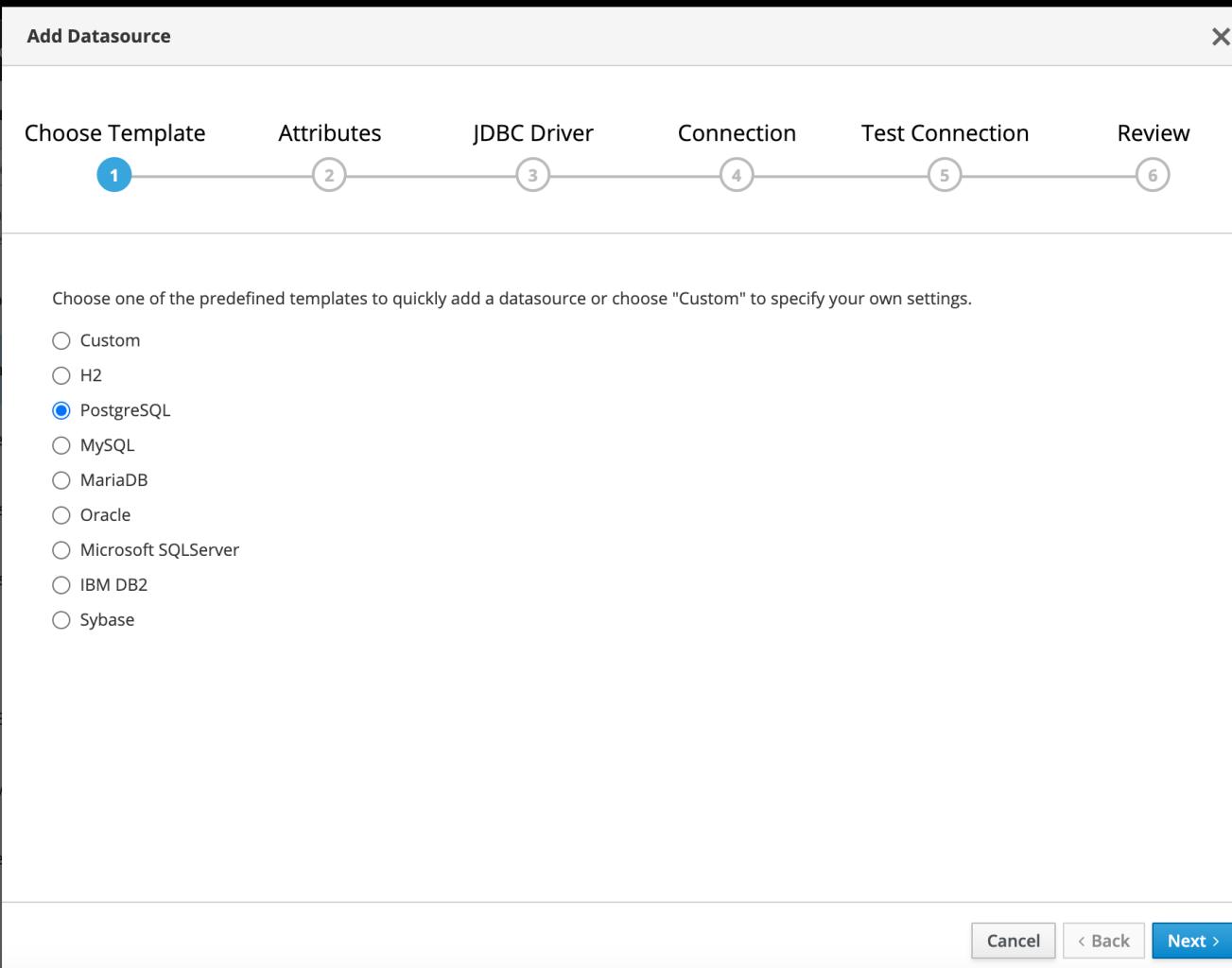
Choose Template Attributes JDBC Driver Connection Test Connection Review

1 2 3 4 5 6

Choose one of the predefined templates to quickly add a datasource or choose "Custom" to specify your own settings.

Custom
 H2
 PostgreSQL
 MySQL
 MariaDB
 Oracle
 Microsoft SQLServer
 IBM DB2
 Sybase

Cancel < Back Next >



Add Datasource

X

Choose Template Attributes JDBC Driver Connection Test Connection Review

① Help

Name: Petdogui

JNDI Name *: java:/Petdogui

Required fields are marked with *

Cancel < Back Next >



Add Datasource

X

Choose Template Attributes JDBC Driver Connection Test Connection Review

1 2 3 4 5 6

② Help

Driver Name * ▾

Driver Module Name

Driver Class Name

Required fields are marked with *

Cancel < Back Next >



Add Datasource

Choose Template Attributes JDBC Driver Connection Test Connection Review

① ② ③ ④ ⑤ ⑥

⑤ Help

Connection URL: `jdbc:postgresql://localhost:5432/app`

User Name: unipar

Password: unipar123

Security Domain:

Credential Reference Store:

Credential Reference Alias:

Credential Reference Clear T...

Credential Reference Type:

Cancel < Back Next >



Add Datasource X

Choose Template Attributes JDBC Driver Connection Test Connection Review

1 2 3 4 5 6



Test Connection Successful

Successfully tested connection for datasource **PetdoguiDS**.

Cancel < Back Next >



JAX-RS Annotations

Annotation

[@GET](#)

[@Produces](#)

[@Path](#)

[@PathParam](#)

[@QueryParam](#)

[@POST](#)

[@Consumes](#)

[@PUT](#)

[@DELETE](#)

[@ApplicationPath](#)

Pacote/Import

`import javax.ws.rs.GET;`

`import javax.ws.rs.Produces;`

`import javax.ws.rs.Path;`

`import javax.ws.rsPathParam;`

`import javax.ws.rsQueryParam;`

`import javax.ws.rs.POST;`

`import javax.ws.rs.Consumes;`

`import javax.ws.rs.PUT;`

`import javax.ws.rs.DELETE;`

`import javax.ws.rs.ApplicationPath`



Anotações – JAX-RS

@Path("/cliente") - Transforma uma classe Java em um recurso REST disponível via HTTP.

Devem ser aplicadas na declaração das classes criando um path relativo de URL padrão de acesso.



Anotações – JAX-RS

`@Path("/abertos")`

Quando aplicados nos métodos Java (expostos como métodos HTTP) definem um path específico de execução, resultando na concatenação como URL de acesso.

Modelo = path relativo + path recurso + path método.

Exe: <http://www.sistema.com/cliente/abertos>



Anotações – JAX-RS

`@Path("/abertos/{nomeusuario}")`

Podem ser usados para expressar parâmetros variáveis com “{}” que serão preenchidas na URL em tempo de execução, nas invocações do serviços.

Estas variáveis são automaticamente disponíveis no métodos de serviços via DI pelo JAX-RS usando a anotação `@PathParam`.



Anotações – JAX-RS

@GET, @POST, @PUT, @DELETE

Mapeia os métodos da classe Java para responder aos correspondentes métodos do protocolo HTTP no qual serão automaticamente invocados pelo JAX-RS.

Devem ser aplicadas somente na declaração dos métodos da classe recurso.



Anotações – JAX-RS

@QueryParam

Utilizado para extrair informações enviados dentro da URL de uma requisição HTTP.

@DefaultValue

Utilizado para definir valores padrões para informações que não foram informados dentro da URL de uma requisição HTTP.



Anotações – JAX-RS

@Produces - mapeia o(s) tipo(s) MIME que será gerado como resposta de um método HTTP REST.

Quando aplicado na classe é automaticamente propagado para todos os métodos expostos como REST.

Quando aplicado no método, sobrepõe a definição da classe, especificando o MIME da resposta do método.



Anotações – JAX-RS

@Produces

1. Converte automaticamente e RETORNA os tipos:

**String, java.io.byte[], java.io.InputStream, Reader e
java.io.File na resposta HTTP.**

**2. Usa a API JAXB para conversão automática de
objetos Java para XML e JSON.**



Anotações – JAX-RS

@Consumes

1. Converte automaticamente os tipos RECEBIDOS :

`String, java.io.byte[], java.io.InputStream, Reader e
java.io.File` na entrada de métodos vindo na
requisição HTTP.

2. Usa a API JAXB para conversão automática de
objetos Java para XML e JSON.



Respostas com JAX-RS

- 1. Métodos declarados como void são mapeados automaticamente para gerar uma resposta HTTP 204 No Content (sem MIME).**
- 2. Métodos declarados como String são mapeados automaticamente para gerar uma resposta HTTP 200 OK, convertendo o conteúdo da String no MIME declarado.**



Respostas com JAX-RS

3. Métodos declarados como Objetos Java JAXB são mapeados automaticamente para gerar uma resposta HTTP 200 OK, usando MIME XML ou JSON.

4. JAX-RS fornece classes chamadas de Response e ResponseBuilder utilizadas para customizar variações de possíveis retornos, tanto relacionado com o conteúdo retornada, bem como o status HTTP.



Respostas com JAX-RS

Utilizadas para oferecer implementações de respostas dinâmicas para serviços REST. Dessa forma não precisa mais declarar `@Produces`.



Desenvolvimento de Aplicações para WEB

**Aula: Criação de APIs's
com Spring Framework.**



Prof. Anderson Augusto Bosing

Tecnologias Utilizadas

- ✓ Spring Boot
- ✓ IntelliJ
- ✓ Java 21
- ✓ Lombok
- ✓ Postgres
- ✓ JPA/Spring Data
- ✓ Maven
- ✓ Postman



Spring

Spring é um framework para desenvolvimento de aplicações em Java, criado em meados de 2002 por Rod Johnson, que se tornou bastante popular e adotado ao redor do mundo devido a sua simplicidade e facilidade de integração com outras tecnologia.

O framework foi desenvolvido de maneira modular, na qual cada recurso que ele disponibiliza é representado por um módulo, que pode ser adicionado em uma aplicação conforme as necessidades.



Spring – Conceitos Inicias

Injeção de Dependências

É um padrão de projeto que ajuda muito a deixar o código desacoplado, melhora a legibilidade e interpretação do código, melhora a distribuição de responsabilidades entre as classes e facilita a manutenção do código.

Padrão de projetos já implementado no spring framework, este padrão possui alguns princípios e destaco dois deles:

Diminuir o acoplamento entre as classes(Trabalhar com Interfaces, se baseando em abstrações).

Alta coesão(Conceito de single responsibility).



Spring – Conceitos Inicias

Uma dependência é simplesmente um objeto que a sua classe precisa para funcionar.

Inversão de Controle, outro padrão de projeto onde o desenvolvedor passa a responsabilidade da criação da instancia das suas classes para o próprio software ou container que implementa o padrão de injeção de dependência.

No caso do spring ele já possui a implementação do container com essa funcionalidade.



Spring – Vamos entender como utilizar esses conceitos.

Bom, agora que já conhecemos os conceitos vamos aprender como utilizar esses recursos junto ao spring framework. No spring para trabalharmos com os conceitos de DI/IOC iremos utilizar os beans.

Os beans são as classes as quais passamos a responsabilidade do gerenciamento dessas classes para o spring, e para que possamos passar a responsabilidade para ele utilizamos as anotações.



Spring – Anotações

@Component

Anotação mais básica para criar qualquer tipo de bean gerenciado pelo spring framework.

Normalmente usada quando não se define um bean como @Repository ou @Service.

```
1 5 @Component
2 6 public class ComputadorUtil {
3 7
4 8     //metodos e atributos omitidos
5 9
6 0 }
```



Spring – Anotações @Repository

Define um bean como sendo do tipo persistente para uso de classes de acesso a banco de dados. A partir desta anotação o Spring pode usar recursos referentes a persistência, como tratar as exceções específicas para este fim.

```
5
6 @Repository
7 public class ComputadorDAO {
8
9     //metodos e atributos omitidos
0
1 }
```



Spring – Anotações

@Service

Usado para definir classes do tipo serviço(Service Layer), que possuem, por exemplo, regras de negócios.

```
6  
7 @Service  
8 public class ComputadorService {  
9  
10    //métodos e atributos omitidos  
11  
12 }
```



Spring – Anotações

@Autowired

Anotação usada para informar ao spring que ele deve injetar a variável anotada na classe em que está declarada.

```
@Autowired  
private ComputadorService computadorService;
```



Spring – Anotações

@RestController

Usado para definir classes do tipo controller/resource.

```
@RestController
@RequestMapping("/hello")
public class HelloController {
```



Spring – Anotações

@PathVariable

Tem o objetivo de extrair da URL um parâmetro que foi incluído como path da URL.

```
@GetMapping(path = "/listagem/{dsMarca}")
public String helloWorld(@PathVariable("dsMarca") String dsMarca) {
    return "OLÁ MUNDO "+dsMarca;
}
```



Spring – Anotações

@RequestParam

Tem o objetivo de capturar um parâmetro de consulta(Query Param) enviado por uma solicitação.

```
@GetMapping  
public String helloWorld(@RequestParam("nome") String nome) {  
    return "OLÁ MUNDO "+nome;  
}
```



Spring – Anotações

@GetMapping, @PostMapping, @PutMapping, @DeleteMapping

Usada para mapear URLs de acesso a um controller e aos métodos contidos nele definindo os verbos HTTP de acesso aos métodos.



Spring Boot

O spring boot é um projeto que chegou para facilitar o processo de configuração e publicação de nossas aplicações.

A intenção é ter o seu projeto rodando o mais rápido possível e sem complicações.

Ele consegue isso favorecendo a convenção sobre a configuração.

Com os módulos informados o spring boot vai reconhece-los e fornecer uma configuração inicial.

Basta que você diga a ele quais módulos deseja utilizar: WEB, template, persistência, segurança, etc...



Criando o projeto no Spring Initializr.

The screenshot shows the Spring Initializr web interface with the following configuration:

- Project:** Maven
- Language:** Java
- Spring Boot:** 2.7.10
- Project Metadata:**
 - Group: br.unipar
 - Artifact: medicalclinic
 - Name: medicalclinic
 - Description: API Rest da Clinica Medica Unipar
 - Package name: br.unipar.medicalclinic
 - Packaging: Jar
 - Java version: 11
- Dependencies:**
 - Spring Boot DevTools** (selected): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
 - Spring Web**: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA**: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - PostgreSQL Driver**: A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.



Configurando o Banco

```
spring.datasource.url=jdbc:postgresql://localhost:5432/app  
spring.datasource.username=unipar  
spring.datasource.password=unipar123
```



Documentando a API com Swagger

O Swagger é, basicamente, um conjunto de ferramentas que nos ajuda a fazer o design, ou seja, fazer a modelagem, a documentar e até gerar código para desenvolvimento de APIs.

```
<dependency>
<groupId>org.springdoc</groupId>
<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
<version>2.0.2</version>
</dependency>
```



Atividade

Pesquisar e utilizar para documentar as classes as annotations do Swagger abaixo:

@Tag

@Operation

@ApiResponses

Lembrem-se de verificar inclusive os parâmetros de entrada das annotations.



Documentação de Referencia

<https://docs.oracle.com/javaee/7/tutorial/bean-validation001.htm>



Documentação de Referencia

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details>



Documentação de Referencia

<https://docs.oracle.com/javaee/7/tutorial/bean-validation001.htm>



Atividade

- ✓ Criar API para Gerenciamento Financeiro
- ✓ Cadastro de Categorias de Receitas Ou Despesas.
- ✓ Cadastrar os registros de Receitas e Despesas do usuário.
- ✓ Na home page da aplicação deve-se trazer um painel com os totalizadores de: Receita, Despesa, Total Líquido, Grid com a listagem dos últimos 10 lançamentos.
- ✓ Tela de consulta de lançamentos com filtros de data, e Receitas ou despesas.





[Dashboard](#) [Realizar Lançamentos](#) [Categorias](#) [Histórico de Transações](#)

Receita

0.0

Despesa

0.0

Total Líquido

0.0

Últimos Lançamentos

Descrição	Categoria	Valor	Data
-----------	-----------	-------	------





Dashboard Realizar Lançamentos Categorias Histórico de Transações

Gerenciamento de Categorias

Adicionar Nova Categoria

Nome da Categoria

Insira o nome da categoria

Tipo da Categoria

Receita

Enviar

Categorias Existentes

ID

Nome

Tipo



 MoneyManager

Dashboard Realizar Lançamentos Categorias Histórico de Transações

dd/mm/aaaa

dd/mm/aaaa

Selezione o tipo

Buscar

Histórico de Transações

Transações Registradas					
ID	Quantia	Data	Categoria	Descrição	Tipo





Dashboard Realizar Lançamentos Categorias Histórico de Transações

Inserir Nova Transação

Dados da Transação

Quantia

Data da Transação

 dd/mm/aaaa

Categoria

 Teste

Descrição

Salvar



Bibliografia Base

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP>

Comer, Douglas E., Interligação de Redes Com Tcp/ip

James F. Kurose, Redes de Computadores e a Internet



Perguntas?



Conclusão