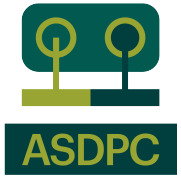
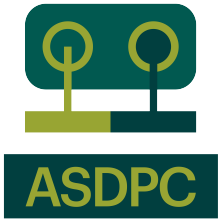
ASDPC 08

Estudo de Caso SOA: WS SOAP usando JWS



Arquitetura de Sistemas Distribuídos, Paralelos e Concorrentes
Prof. Luiz Lima Jr.
Escola Politécnica - PUCPR



Suporte Java para WS

WS-SOAP

APIs Java para Web Services

- Orientadas a **documentos**:
  - *Java API for XML Processing* (**JAXP**)
    - *Parsers para documentos XML*
  - *Java Architecture for XML Binding* (**JAXB**)
    - *Processa documentos XML – (un)marshalling (esquemas XML è classes)*
  - *SOAP with Attachments API for Java* (**SAAJ**)
    - *Envia mensagens SOAP sobre a Internet*

APIs Java para Web Services

- Orientadas a **RPC**:
  - *Java API for XML-based Web Services* (**JWS**)*
    - *Envia chamadas de métodos em SOAP para objetos remotos sobre a Internet + resposta.*
  - *Java API for XML-RESTful Services* (**JAX-RS**)
    - *Suporte para chamadas no estilo REST*
  - *Java API for XML Registries* (**JAXR**)
    - *Meio padrão para acessar registros de negócios e compartilhar informações (UDDI – Java)*

Java Web Services (JWS)

- Tecnologia para **construção de serviços web** e clientes:
  - Comunicação via XML (**SOAP**)
- Serviços web:
  - Orientados a **RPC**
- **API** JWS:
  - Esconde detalhes dos programadores

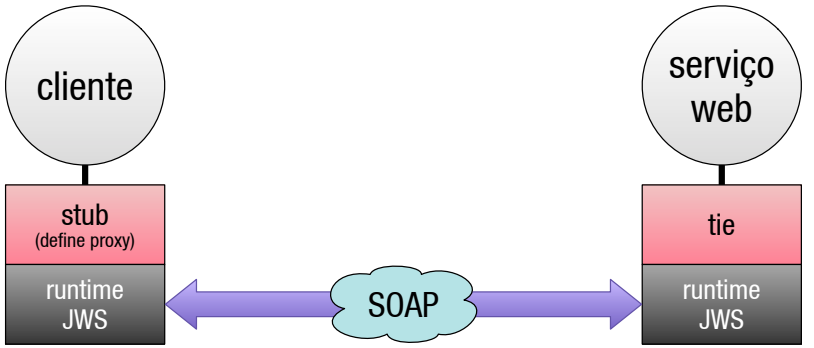
Desenvolvimento JWS

WS-SOAP

JWS

- Servidor:
  - Interface (opcional)
  - Classe de implementação
- Cliente:
  - ***Proxy*** (objeto local representante do serviço)
  - Invoca métodos do ***proxy***
- **Não** há necessidade de **analisar/gerar XML** (SOAP, WSDL, etc.) diretamente.

Comunicação no JWS



WS Baseados em SOAP

- **Interface** que declara os métodos que o cliente poderá invocar (opcional)
  - **SEI** = *Service Endpoint Interface*
- **Implementação** que define os métodos
  - **SIB**: *Service Implementation Bean*
    - *Instâncias de classes Java regulares*
    - *Um SIB define implicitamente o* endpoint*, se SEI não for definido.*

JWS

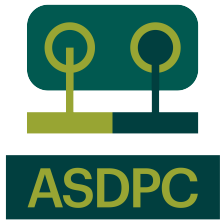
- `javax.jws.WebService` ***annotation***:
  - Define a **classe** como sendo um ***endpoint*** de um serviço web.

JWS

- `wsgen`:
  - Ferramenta para gerar ***ties*** a partir da implementação do *endpoint*.
    - *(chamada implicitamente)*
- `wsimport`:
  - Ferramenta para gerar ***stubs*** a partir da definição da interface (**WSDL**).
    - *Precisa ser executada explicitamente.*

Passos Típicos

1. Escreva a interface (**SEI**)
2. Escreva classe de implementação (**SIB**)
3. Compile a classe de implementação (gerando arquivo ***.war**)
4. Lance o arquivo **war**
   - as classes **TIE** são geradas automaticamente durante o lançamento
5. Escreva a **classe cliente**
6. Gere o ***stub*** a partir da interface **WSDL** do servidor
7. **Compile e execute** o cliente



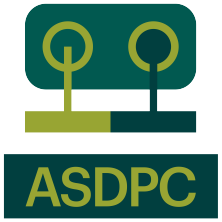
EcoWS (espec)

Endpoint JWS (servidor)

- Requisitos da classe de implementação:
  - Anotação: `@WebService`
  - Pode referenciar uma SEI explicitamente:
    - *endpointInterface (da anotação WebService)*
  - Se não especificada, uma SEI é implicitamente definida para a classe de implementação.
  - Métodos remotos devem ser `public` (não podem ser `static` ou `final`)
  - Métodos expostos a clientes: `@WebMethod`
  - Métodos expostos: parâmetros compatíveis c/ JAX-B

Endpoint JWS (servidor)

- Requisitos da classe de implementação (cont.):
  - A classe não pode ser declarada `final` ou `abstract`.
  - Um **construtor público** precisa ser definido (sem parâmetros).
  - O método `finalize` não deve ser definido.
  - Anotações para métodos de ciclo de vida (opcional):
    - `@PostConstruct`*: chamado pelo container antes que a classe comece a responder a requisições.*
    - `@PreDestroy`*: chamado antes de tirar o serviço do ar.*



O Serviço

SEI: Exemplo

```
package eco.servidor;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
@WebService
public interface EcoIF {
  @WebMethod public String diga(String oque);
  @WebMethod public int contador();
}
```


SIB: Exemplo

liga este SIB ao SEI

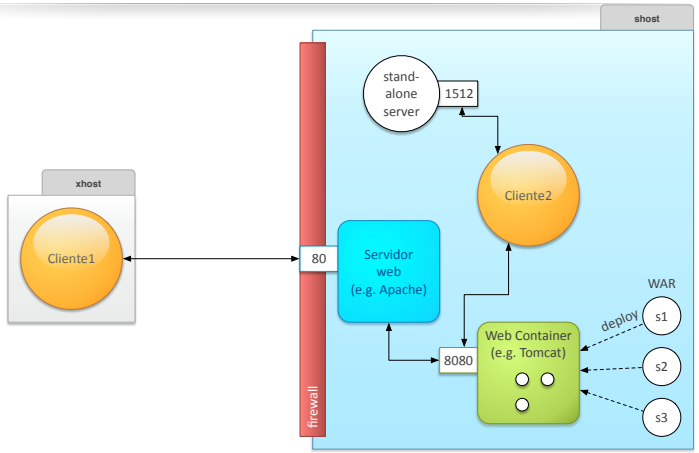
```
package eco.servidor;
import javax.jws.WebService;

@WebService(endpointInterface = "eco.servidor.EcoIF")
public class Eco implements EcoIF {
    private int cont = 0;
    public String diga(String oque)  {
        cont++;
        return oque + "_ECO";
    }
    public int contador() { return cont; }
}
```


Modalidades de Servidor

- Servidor "***stand-alone***":
  - Servidor roda independentemente escutando em porta disponível.
  - Problema: *firewall*?
- Servidor gerenciado por um ***web container*** (e.g. Tomcat) :
  - Necessidade de "empacotar" servidor (arquivo "**.war**") e publicá-lo no *web container*.

Servidor Gerenciado por WEB CONTAINER



Servidor STAND-ALONE
Publicador

```
package eco.servidor;
import javax.xml.ws.Endpoint;

public class Publicador {
  public static void main(String[] args) {
  Endpoint.publish("http://localhost:XXXX/eco",
                   new Eco());
  }
}
```

URL

XXXX = número da porta

Instância do SIB

Testando Servidor

- `http://localhost:XXXX/eco?wsdl`
  - `portType`:
    - *Agrupamento das operações WS*
    - *Apenas métodos (interface)*
    - *Cada operação em WS: input e output*
    - *Estilo RPC*
  - `service`:
    - *location = http://localhost:XXXX/eco*
    - *Este é o "service endpoint"*

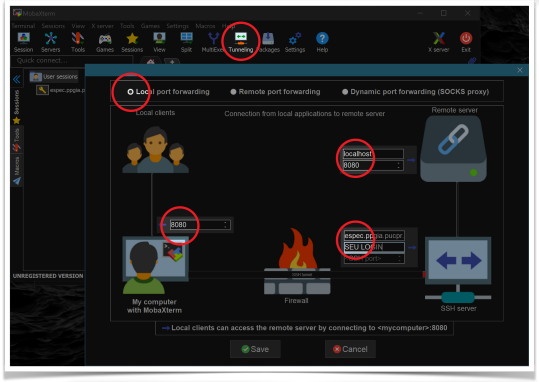
Testando Servidor em Web Container (Tomcat)

- Compilar **SEI** e **SIB** (como no caso *stand-alone*)
- Gerar arquivos de **configuração**:
  - `web.xml`
  - `sun-jaxws.xml`
  - (ver arquivos fornecidos na página web).

Testando Servidor em Web Container (Tomcat)

- Estrutura de **diretórios**:
  - `WEB-INF/`
    - *`classes/`* — *arquivos .class SEI e SIB*
    - *`web.xml`*
    - *`sun-jaxws.xml`*
- Gerar arquivo `.war`:
  - `jar cvf meu-eco.war WEB-INF/`
  - `cp meu-eco.war /tmp/webapps`

Tunelamento para Acesso ao Tomcat Manager a partir do Navegador Local



MobaXTerm

```
ssh -N -L 8080:localhost:8080
LOGIN@espec.ppgia.pucpr.br
```

Linux/macOS

Lançando no Tomcat Manager

http://localhost:8080/manager

Deploy

Deploy directory or WAR file located on server

Context Path (required): /meu-eco

XML Configuration file path:

WAR or Directory path: /tmp/webapps/meu-eco.war

Deploy

WSDL

http://localhost:8080/meu-eco/eco?wsdl

Cliente do Serviço Web

WS-SOAP

Cliente Eco

```
package eco.cliente;

class ClienteEco {
    public static void main(String [] args)
       throws Exception {
       EcoIF eco = new EcoService().getEcoPort();
       System.out.println(eco.diga("ola"));
       System.out.println(eco.contador());
       // …
    }
}
```

WSDL

- Ferramenta para **gerar código cliente** (proxy/stub) a partir de documentos WSDL:
  - `wsimport -p eco.cliente http://localhost:XXXX/eco?wsdl`
- Acesso alternativo ao documento WSDL:
  - `curl http://localhost:XXXX/ts?wsdl`

Compilação e Execução

- Compilação:
  - `javac eco/servidor/*.java`
  - `javac eco/cliente/*.java`
- Execução:
  - `java eco.servidor.Publicador`
  - `java eco.cliente.ClienteEco`

Cliente Python

biblioteca zeep

```
#!/usr/bin/env python3
from zeep import Client

# setup
wsdl_url = 'http://localhost:XXXX/eco?wsdl'
eco = Client(wsdl_url)

# invoke services
res1 = eco.service.diga("ola")
res2 = eco.service.contador()

print("Eco =", res1)
print("Contador =", res2)
```


ASDPC

Exercício: ContaWS (SOAP)

WS-SOAP


ASDPC
Implemente com WS-SOAP - JWS
Conta:
saldo() ➔ float
id() ➔ string
deposito(valor)
saque(valor)
Standalone
Web Container (Tomcat)
© Luiz Lima Jr.
33