**SOA**

**Começando um projeto SOA:**

**HTTP:**

O protocolo HTTP possibilita a integração de sistemas de diferentes linguagens através de links de envio e recebimento de arquivos de determinados tipos.

**Transformando uma classe java em uma web service:**

Para transformar uma classe em web service, basta anotá-la com **javax.jws.@WebService;** e para expôr este web service, ou seja, deixá-lo pronto para uso em outras aplicações, basta utilizar a classe **javax.xml.ws.Endpoint**. Não se preocupe, tudo isso está incluso na JDK, a partir do Java 6.

Exemplo:

package com.knight.estoque.servicos;

// imports omitidos

@WebService

public class ListagemLivros {

public List listarLivros() {

LivroDAO livroDAO = obterDAO();

return livroDAO.listarLivros();

}

// Outros métodos...

public static void main(String[] args) { Endpoint.publish("http://localhost:8080/livros",

new ListagemLivros());

System.out.println("Serviço inicializado!");

} }

Devemos também identificar as classes que irão trafegar pela rede perante o sistema de parsing de dados, acrescentando a classe **jaxb.index** onde essa classe ira referenciar as outras classes presentes no mesmo package.

**Fluxo de dados**

**Estrutura de namespaces do XML:**

Basicamente, o XML possui uma estrutura de identificação bastante semelhante à estrutura de nomenclatura em Java. Para evitar confusão de conceitos, é utilizada uma estrutura de namespaces, que são utilizados para diferenciar uma tag de outra.

<ns2:listarLivrosResponse

xmlns:ns2="http://servicos.estoque.knight.com/">

<return>

<!-- Corpo da resposta.. -->

</return>

</ns2:listarLivrosResponse>

Neste XML, o namespace é http://servicos.estoque.knight.com/. Ele pode ser referenciado em qualquer ponto do XML em questão, através do prefixo ns2. Desta maneira, quando o elemento XML é ns2:listarLivrosResponse, sabemos automaticamente que é o elemento listarLivrosResponse definido no namespace <http://servicos.estoque.knight.com/>.

**Funcionamento do SOAP:**

SOAP é uma sigla que significa Simple Object Access Protocol (http://www.w3.org/ TR/soap/). Ele foi criado e é mantido pelo World Wide Web Consortium, ou simplesmente W3C. Ele também é conhecido como Envelope SOAP, já que seu elemento raiz é o Envelope. Ele obedece ao seguinte formato geral:

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:ser="http://servicos.estoque.knight.com/"/>

<!-- Aqui pode ter, ou não, um elemento soapenv:Header -->

<soapenv:Body>

<ser:listarLivros />

</soapenv:Body>

</soapenv:Envelope>

O elemento Envelope é puramente um container para os elementos Header e Body. O elemento Body contém o corpo da requisição, propriamente dito: o nome da operação, parâmetros etc. Já o elemento Header contém metadados pertinentes à requisição, como informações de autenticação, endereço de retorno da mensagem etc.

**Entenda WSDL:**

No capítulo 1, ao acessarmos o endereço http://localhost:8080/livros?wsdl para visualizar o WSDL, nos deparamos com um enorme XML, que para o desenvolvedor que nunca o viu, pode ser um código grande e assustador:

<?xml version="1.0" encoding="UTF-8"?>

<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:tns="http://servicos.estoque.knight.com/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns="http://schemas.xmlsoap.org/wsdl/"

targetNamespace="http://servicos.estoque.knight.com/"

name="ListagemLivrosService">

<types>

<xsd:schema><xsd:import

namespace="http://servicos.estoque.knight.com/"

schemaLocation="http://localhost:8080/livros?xsd=1">

</xsd:import>

</xsd:schema>

</types>

<message name="listarLivros">

<part name="parameters" element="tns:listarLivros">

</part>

</message>

<message name="listarLivrosResponse">

<part name="parameters" element="tns:listarLivrosResponse">

</part>

</message>

<portType name="ListagemLivros">

<operation name="listarLivros">

<input message="tns:listarLivros"></input>

<output message="tns:listarLivrosResponse"></output>

</operation>

</portType>

<binding name="ListagemLivrosPortBinding"

type="tns:ListagemLivros">

<soap:binding transport="http://schemas.xmlsoap.org/soap/http"

style="document">

</soap:binding>

<operation name="listarLivros">

<soap:operation soapAction=""></soap:operation>

<input>

<soap:body use="literal"></soap:body>

</input>

<output>

<soap:body use="literal"></soap:body>

</output>

</operation>

</binding>

<service name="ListagemLivrosService">

<port name="ListagemLivrosPort"

binding="tns:ListagemLivrosPortBinding">

<soap:address location="http://localhost:8080/livros">

</soap:address>

</port>

</service>

</definitions>

Nota-se que o WSDL possui cinco seções bem definidas: types, message (onde pode existir mais de um elemento), portType, binding e service.

**A definição dos tipos e a seção types:**

O JAXB (Java Architecture for XML Binding) é o responsável, em Java, por gerar tipos em XML Schemas a partir de classes Java e vice-versa. Por exemplo, tomando como parâmetro a seguinte classe:

package com.knight.estoque.modelos;

// imports omitidos

@XmlAccessorType(XmlAccessType.FIELD)

public class Livro {

private Integer anoDePublicacao;

private List autores;

private String editora;

private String nome;

private String resumo;

//Getters e setters...

}

O JAXB vai mapear esta classe para o seguinte conteúdo num XML Schema:

<xs:complexType name="livro">

<xs:sequence>

<xs:element name="anoDePublicacao" type="xs:int" minOccurs="0">

</xs:element>

<xs:element name="autores" type="xs:string" nillable="true"

minOccurs="0" maxOccurs="unbounded">

</xs:element>

<xs:element name="editora" type="xs:string" minOccurs="0">

</xs:element>

<xs:element name="nome" type="xs:string" minOccurs="0">

</xs:element>

<xs:element name="resumo" type="xs:string" minOccurs="0">

</xs:element>

</xs:sequence>

</xs:complexType>

Isto mostra que as classes são, preferencialmente, mapeadas para complexType’s ao invés de elementos. Você pode notar que os campos da classe Java foram todos mapeados com seus equivalentes em uma linguagem chamada XML Schema. O atributo type, assim como numa classe Java, determina a tipagem do elemento em questão.

ComplexType’s é uma forma de instanciar uma classe java em xml , o minOccurs e maxOccurs determinam, respectivamente, o número mínimo de ocorrências e o número máximo.

**A necessidade da seção messages:**

O objetivo da seção messages é “amarrar” diversos elementos, presentes ou importados na seção types, para compôr os dados que trafegarão efetivamente de/para o serviço.