Curso de Especialização em Desenvolvimento Java

Desenvolvimento baseado em Webservices

Prof. Emílio Dias
emiliodias@gmail.com
http://www.github.com/emiliodias

Conteúdo da disciplina

- Introdução a comunicação inter-processos
- Discussões gerais sobre a Web
- Introdução a Webservices
- Webservices “RESTful”
- Segurança de Webservices com OAuth2
- Webservices SOAP
- Utilização do framework Spring

Avaliação

- Presença e participação em sala de aula.

  - **Aula 1:** Apresentação da disciplina, e introduções, os alunos não serão avaliados.
  - **Aula 2 à 5:** 20 pontos por presença + participação

    Participação = Interesse no conteúdo apresentado, questionamentos, execução de exercícios quando propostos.

Comunicação inter-processos

Motivação para o estudo, entendimento de comunicação inter-processos e relação com Webservices.

Comunicação inter-processos

- Passagem de Mensagem pode ser suportada por duas operações de comunicação (send e receive);
- A comunicação se dá pelo envio da mensagem (seqüência de bytes) do emissor para o receptor;
- Essa troca de mensagem pode envolver a sincronização dos processos envolvidos.

Comunicação inter-processos

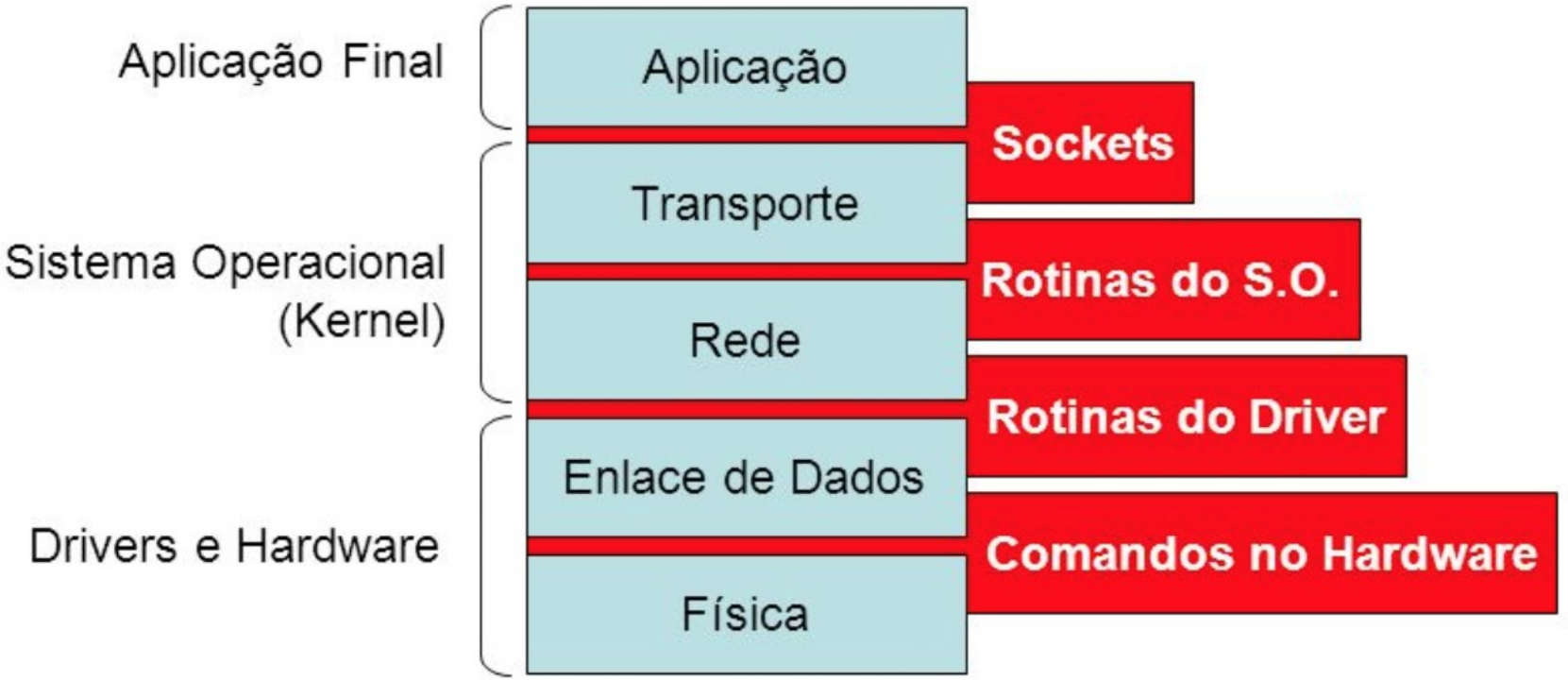
Comunicação síncrona

Ambos, emissor e receptor sincronizam-se a cada mensagem. Assim tanto o send como o receive são operações bloqueantes.

Comunicação assíncrona

Após o send o processo pode continuar executando. A operação receive pode ser bloqueante ou não.

Comunicação inter-processos



Arquitetura em camadas TCP/IP

Comunicação inter-processos

Sockets

- Abstração que disponibiliza um ponto final para comunicação.
- A comunicação inter-processos se dá através da transmissão de mensagem entre um socket em um processo e outro socket em outro processo.
- Para o processo receptor o socket deve estar ligado ao endereço internet local e a uma porta local, no computador onde ele executa.
- Cada socket está associado com um protocolo particular (TCP e UDP)

Comunicação inter-processos

Modelos de comunicação

- Request/Response
- RMI
- **RPC → Modelo utilizado pelo protocolo SOAP**
- **REST → Modelo utilizado na criação de Webservices RESTful**
- Outros...

Discuções gerais sobre a Web

O que é a Web?

Discussões gerais sobre a Web

O que é a Web?

- Algo que tem uma URL...
- Plataforma Web...
- Tudo que funciona no Browser...
- Quem sabe???

Hi, I'm Mark Nottingham. I currently co-chair the IETF HTTP and QUIC Working Groups, and am a member of the Internet Architecture Board. I usually write here about the Web, protocol design, HTTP, and caching.

Fonte: https://www.mnot.net/blog/2014/12/04/what_is_the_web

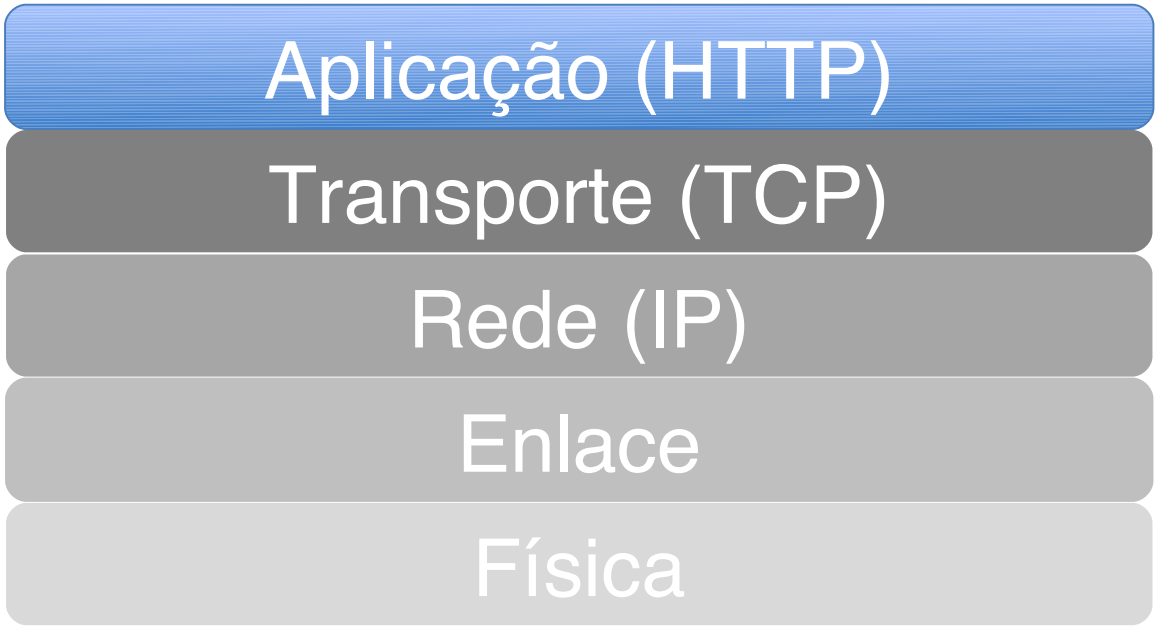
Tradução: https://www.infoq.com/br/news/2015/03/o-que-e-web

Introdução a Webservices

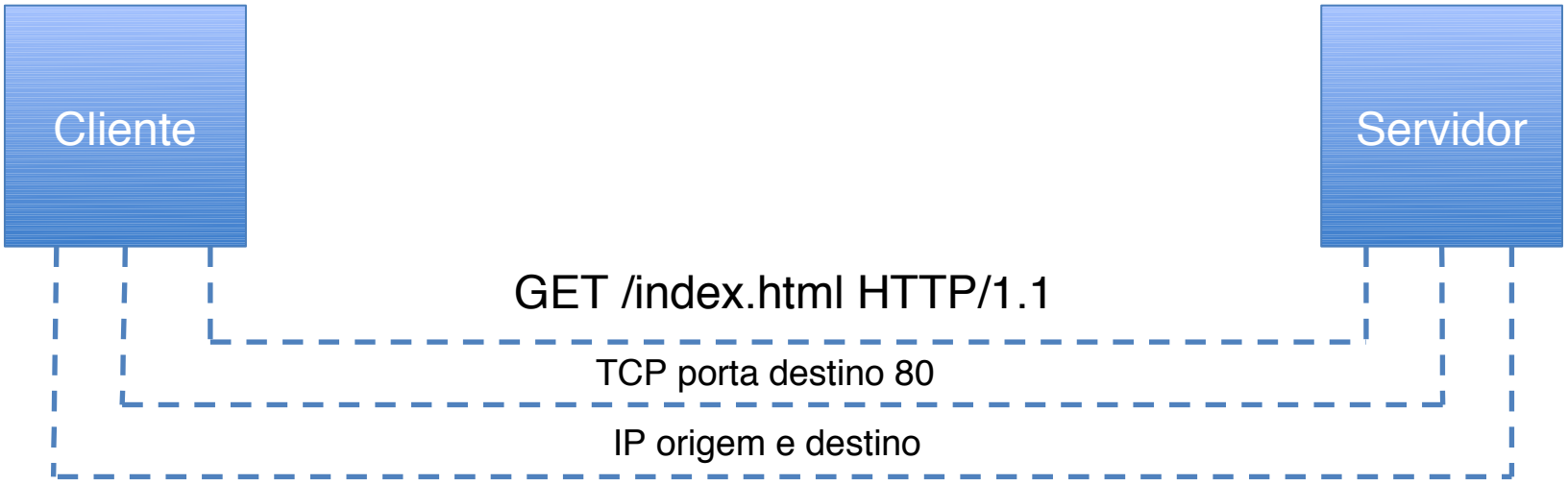
O que são?

Tecnologia de comunicação inter-processos que utilizam como mecanismo básico de comunicação as tecnologias presentes na Web, sendo a principal o protocolo HTTP.

O protocolo HTTP



O protocolo HTTP



O protocolo HTTP: URI

URI – Universal Resource Identifier

O protocolo HTTP: Métodos

- **GET**
  - Utilizado quando existe a necessidade de se obter um recurso.
- **POST**
  - Utilizamos o método POST quando desejamos criar um recurso
- **PUT**
  - Semelhante ao método POST, esse método nos permite a atualização de um recurso já existente
- **DELETE**
  - Como o próprio nome sugere, o DELETE é utilizado quando existe a necessidade de remoção de um recurso

O protocolo HTTP: Respostas

- **1xx**
  - **Informações gerais**
- **2xx**
  - **Sucesso**
- **3xx**
  - **Redirecionamento**
- **4xx**
  - **Erro no cliente**
- **5xx**
  - **Erro no servidor**

O protocolo HTTP

Outras importantes características do protocolo HTTP

- **Media-type (Content-type)**
- **Caching**

REST

UNIVERSITY OF CALIFORNIA,
IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Roy Thomas Fielding

2000

Roy Fielding

American computer scientist

Roy Thomas Fielding is an American computer scientist, one of the principal authors of the HTTP specification and the originator of the Representational State Transfer architectural style. Wikipedia

Born: 1965 (age 53 years), Laguna Beach, California, United States

Education: University of California, Irvine (2000)

Field: Computer Science

Books: Dean and the Dark Angels, Dyslexia

Organization founded: Apache Software Foundation

REST

Constraints

- Cliente/Servidor
- Stateless
- Cache
- Interface Uniforme
- Sistema em camadas
- Código sob demanda

Webservices RESTful

O que são?

Webservices que utilizam como base para sua arquitetura as constraints definidas pelo modelo arquitetural REST.

Spring REST Web Services

Spring REST Web Services

@RestController

Cria um controlador REST, deve ser colocado em sua classe.

Spring REST Web Services

@RequestMapping

Faz o mapeamento de uma requisição, pode ser colocado em sua classe, bem como em seus métodos. Principais atributos:

- Value
  - Pode ser utilizado para definir o nome do seu recurso
- Method
  - Utilizado para informar os métodos HTTP suportados. Por default, aceita todos os métodos.
- Consumes
  - Define qual o tipo de conteúdo é aceito.
- Produces
  - Define qual o formato da representação o cliente deseja

Spring REST Web Services

Mapeando métodos

- **@RequestMapping(method={RequestMethod.GET})**
  - Mapeia método GET
- **@RequestMapping(method={RequestMethod.POST})**
  - Mapeia método POST
- **@RequestMapping(method={RequestMethod.PUT})**
  - Mapeia método PUT
- **@RequestMapping(method={RequestMethod.DELETE})**
  - Mapeia método DELETE

Spring REST Web Services

Mapeando métodos

- **@RequestMapping(method={RequestMethod.GET, RequestMethod.POST, RequestMethod.PUT, RequestMethod.DELETE})**
  - Você pode fazer o mapeamento para mais de um tipo de método, basta informá-los conforme exemplo.

Spring REST Web Services

Mapeando métodos

Você também pode utilizar uma anotação específica de cada um dos métodos:

- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping

Spring REST Web Services

@RequestParam

Utilizado para mapear parâmetros do tipo QueryString para o seu recurso.

Exemplo: /alunos?id=10

@RequestParam(“id”)

Spring REST Web Services

@PathVariable

Utilizado para mapear templates baseados na URI.

Exemplo: /alunos/10

/alunos/{id}

@PathVariable(“id”)

Spring REST Web Services

HttpEntity

Representação de uma resposta ou requisição HTTP.

```
public HttpEntity get(String name) ...
```

Spring REST Web Services

Construindo uma resposta com HttpEntity

Exemplos:

- `ResponseEntity.ok(rep);`
- `ResponseEntity.notFound().build();`
- `ResponseEntity.accepted().build();`
- `ResponseEntity.status(HttpStatus.CREATED).build();`

Spring REST Web Services

@ResponseStatus

Mapeia um status HTTP para uma determinada exception.

```
1  package br.com.unitri.posjava.restful.webservicesrestful.resources;
2
3  import org.springframework.http.HttpStatus;
4  import org.springframework.web.bind.annotation.ResponseStatus;
5
6  @ResponseStatus(HttpStatus.NOT_FOUND)
7  public class ResourceNotFoundException extends RuntimeException {
8     //
9  }
```

Spring REST Web Services

Media-type

Utilizando o Header Accept, o cliente pode escolher qual o formato mais adequado para representação ele deseja aceitar.

```
@RequestMapping(method = RequestMethod.GET, produces = {
                MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE
})
```

Spring REST Web Services

Criando recursos:

Ao criar um recurso, o servidor deve informar ao cliente o identificador do mesmo. Isto pode ser feito de diversas formas, sendo a adição do header “Localization” o mais adequado.

Além disso, para a serialização do body recebido, podemos utilizar a annotation **@RequestBody**.

Spring REST Web Services

Criando recursos, exemplo:

```
@RequestMapping(method = RequestMethod.POST)
public ResponseEntity<Void> salvar(@RequestBody Livro livro) {
        livro = livrosService.salvar(livro);

        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().
                        path("/{id}").buildAndExpand(livro.getId()).toUri();

        return ResponseEntity.created(uri).build();
}
```

Spring REST Web Services

Serialização, exemplo:

```
@Entity
public class Autor {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @NotEmpty(message = "O campo nome não pode ser vazio.")
        private String nome;

        @JsonFormat(pattern = "dd/MM/yyyy")
        @JsonInclude(Include.NON_NULL)
        @NotNull(message = "Campo nascimento é de preenchimento obrigatório.")
        private Date nascimento;

        @JsonInclude(Include.NON_NULL)
        @NotNull(message = "Campo nacionalidade é de preencimento obrigatório.")
        private String nacionalidade;

        @OneToMany(mappedBy = "autor")
        @JsonIgnore
        private List<Livro> livros;
```

Spring REST Web Services

Serialização

O mecanismo de serialização XML/JSON normalmente utilizado pelo Spring é o Jackson. Sendo algumas de suas importantes anotações:

- **@NotEmpty**
- **@JsonFormat**
- **@JsonInclude**
- **@NotNull**
- **@JsonIgnore**

Spring REST Web Services

Dependências adicionais

Serializador XML.

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

Spring REST Web Services

Mapeamento Representação Externa vs Representação Interna

http://modelmapper.org

Spring REST Web Services

Mapeamento Representação Externa vs Representação Interna

Exemplo:

TargetObject = modelMapper.map(sourceObject, **TargetObject.class**);

ClienteRepresentation cr = modelMapper.map(cliente, ClienteRepresentation.class);

Spring REST Web Services

Cache

Você pode adicionar um controle de cache para as respostas que você envia ao cliente, para diminuir a quantidade de requisições que são enviadas ao servidor.

```
@RequestMapping(value = "/{id}", method = RequestMethod.GET)
public ResponseEntity<?> buscar(@PathVariable("id") Long id) {
        Livro livro = livrosService.buscar(id);

        CacheControl cacheControl = CacheControl.maxAge(20, TimeUnit.SECONDS);

        return ResponseEntity.status(HttpStatus.OK).cacheControl(cacheControl).body(livro);
}
```

Spring REST Web Services

Criando recursos:

Ao criar um recurso, o servidor deve informar ao cliente o identificador do mesmo. Isto pode ser feito de diversas formas, sendo a adição do header “Localization” o mais adequado.

```
@RequestMapping(method = RequestMethod.POST)
public ResponseEntity<Void> salvar(@RequestBody Livro livro) {
        livro = livrosService.salvar(livro);

        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().
                        path("/{id}").buildAndExpand(livro.getId()).toUri();

        return ResponseEntity.created(uri).build();
}
```

Spring REST Web Services

Hateoas

```java
package br.com.unitri.posjava.restful.webservicesrestful.representations;

import org.springframework.hateoas.ResourceSupport;

public class Aluno extends ResourceSupport{

    private String nome;

    private String endereco;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getEndereco() {
        return endereco;
    }

    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }

}
```

Spring REST Web Services

Hateoas

```
@RequestMapping(value="/testeHipermedia", method={RequestMethod.POST})
public Aluno testeHipermedia() {

    Aluno aluno = new Aluno();
    aluno.setNome("Nome do aluno");
    aluno.setEndereco("Endereço do aluno");

    aluno.add(linkTo(methodOn(TesteResource.class).testeHipermedia()).withSelfRel());


    return aluno;

}
```

Spring REST Web Services

Dependências adicionais

Hateoas

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

Autenticação / Autorização com OAuth 2

OAuth 2

[Docs] [txt|pdf] [draft-ietf-oaut...] [Tracker] [Diff1] [Diff2] [IPR] [Errata]

Updated by: 8252 PROPOSED STANDARD
Errata Exist

Internet Engineering Task Force (IETF) D. Hardt, Ed.
Request for Comments: 6749 Microsoft
Obsoletes: 5849 October 2012
Category: Standards Track
ISSN: 2070-1721

The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

OAuth 2

Analogamente, segundo a própria especificação:

“Alguns carros de luxo vem com uma chave extra chamada valet key. Essa chave pode ser utilizada quando queremos que um manobrista estacione o carro. Ao utilizar essa chave para ligar o carro, o manobrista não será capaz de dirigir o carro por mais de 2 km. Independente da restrição que essa chave especial impõe a seus utilizadores, a ideia é clara: você dá a alguém acesso limitado ao seu carro utilizando uma chave especial, e quando quiser acesso ilimitado ao carro, utiliza a chave normal.”

OAuth 2

Papéis

Resource Owner: A entidade capaz de controlar o acesso aos recursos protegidos. Como o nome diz, é o “dono do recurso”.

Resource Server: Servidor que hospeda os recursos a serem acessados. É quem recebe as requisições. É quem expõe a API que queremos acessar.

Client: A aplicação que solicita acesso aos recursos protegidos do Resource Owner.

Authorization Server: Servidor que gera tokens de acesso, permite que o Client acesse os recursos, que o Resource Owner permitiu, com o nível de acesso que o Resource Owner especificou.

OAuth 2

Fluxo abstrato

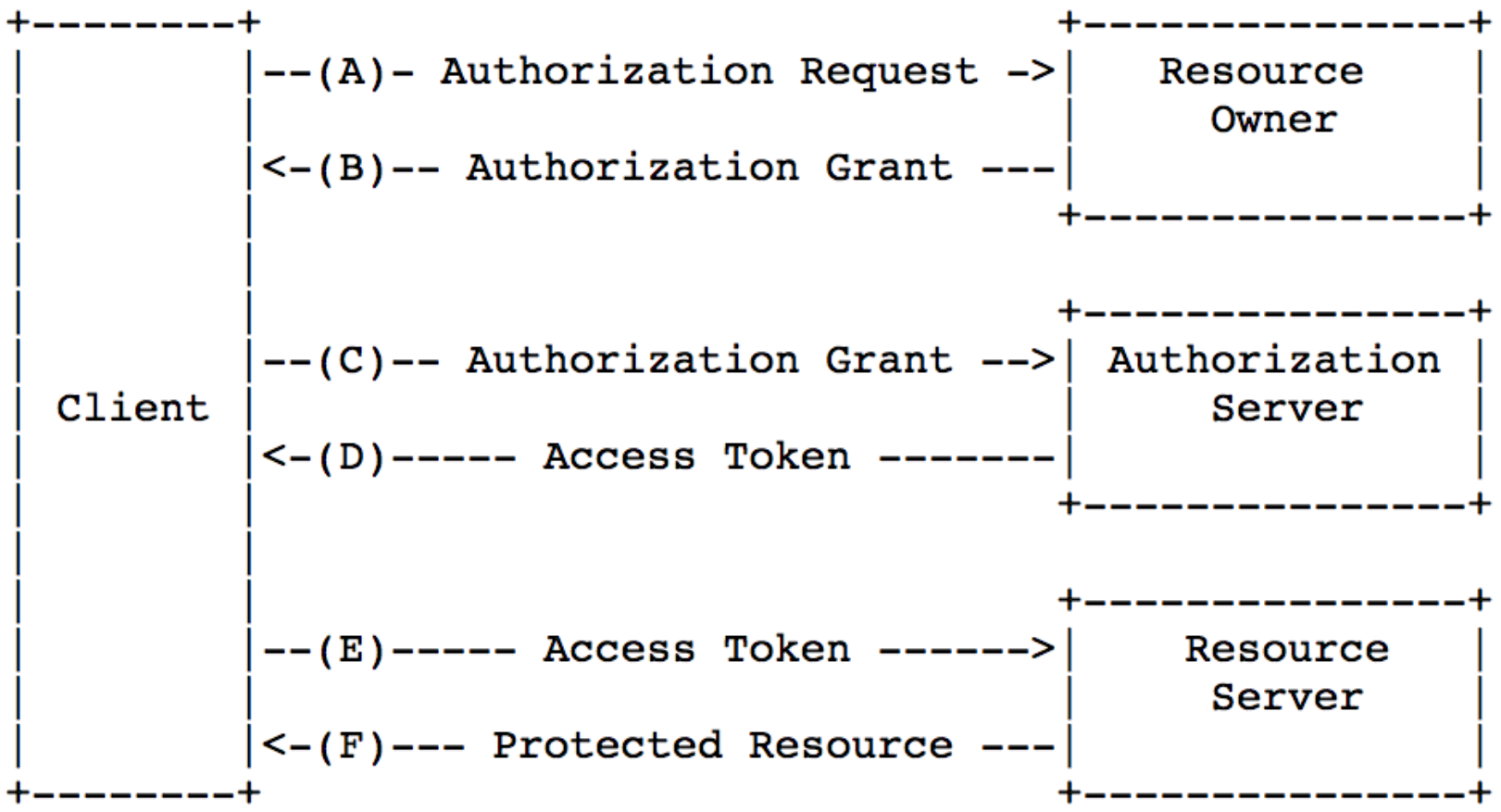


Figure 1: Abstract Protocol Flow

OAuth 2

Authorization Grant Types

Authorization Code: Utilizados por aplicações web que executam em servidores. Esse grant type é o mais comum. Utilizado principalmente quando queremos obter recursos de usuários de aplicações de terceiros, login com Facebook por exemplo.

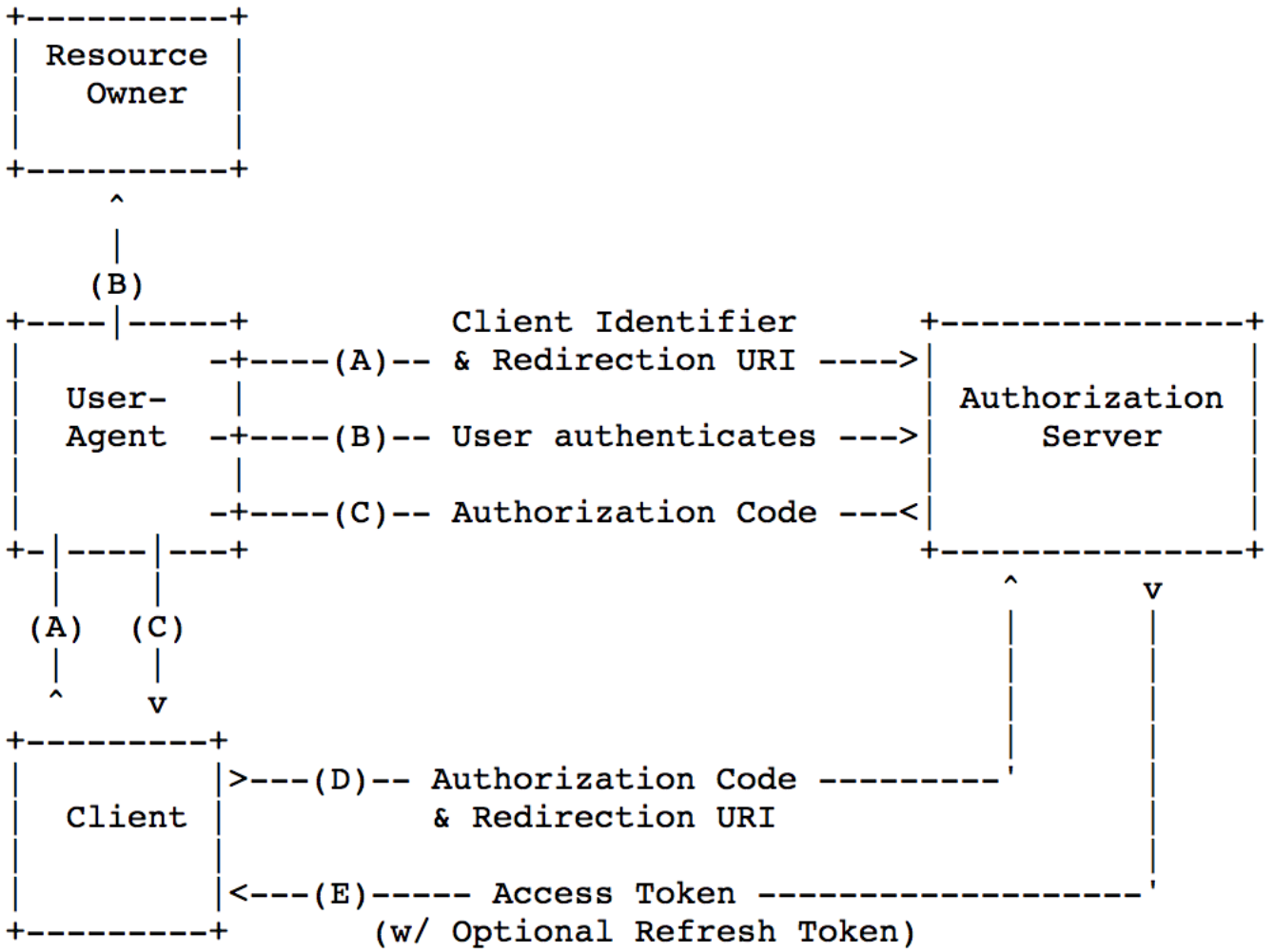
Implicit: Utilizado por SPAs (single page applications) que executam em browsers.

Resource Owner Password Credentials: Utilizado por trusted apps (aplicativos confiáveis). Aplicativos podem ser considerados trusted apps por quem define as restrições de acesso do Resource Owner.

Client Credentials: Utilizado em comunicações do tipo “machine-to-machine”.

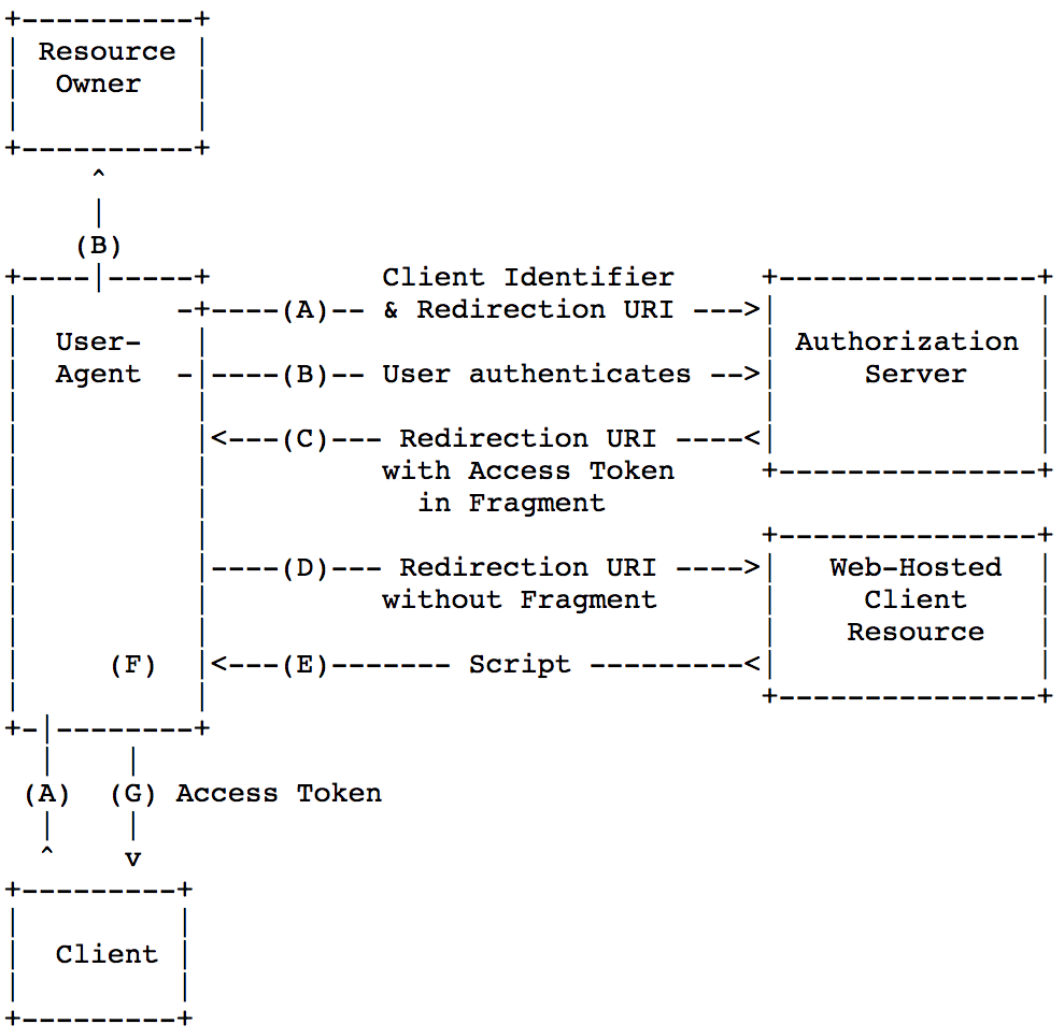
OAuth 2

Authorization code



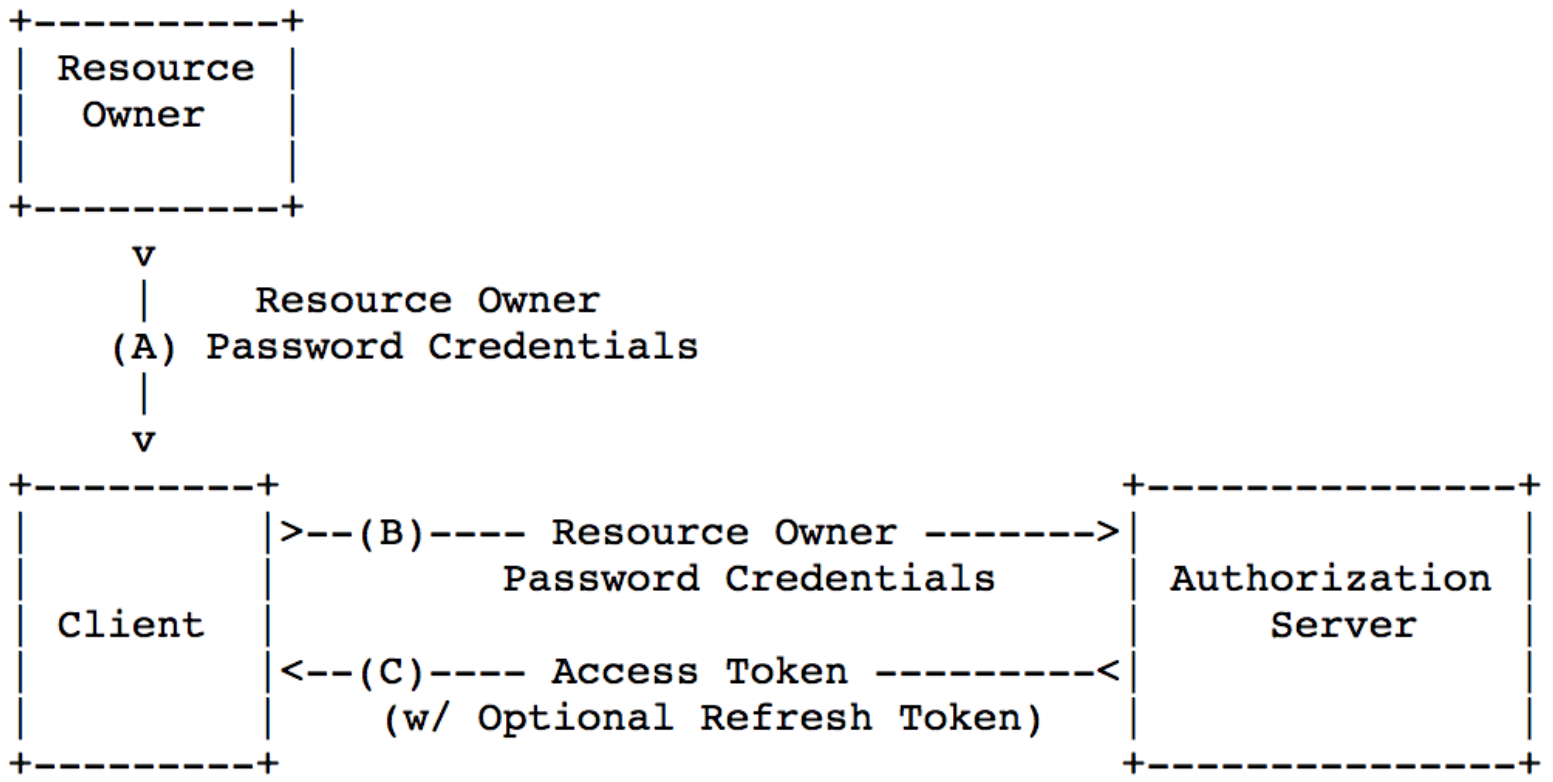
OAuth 2

Implicit



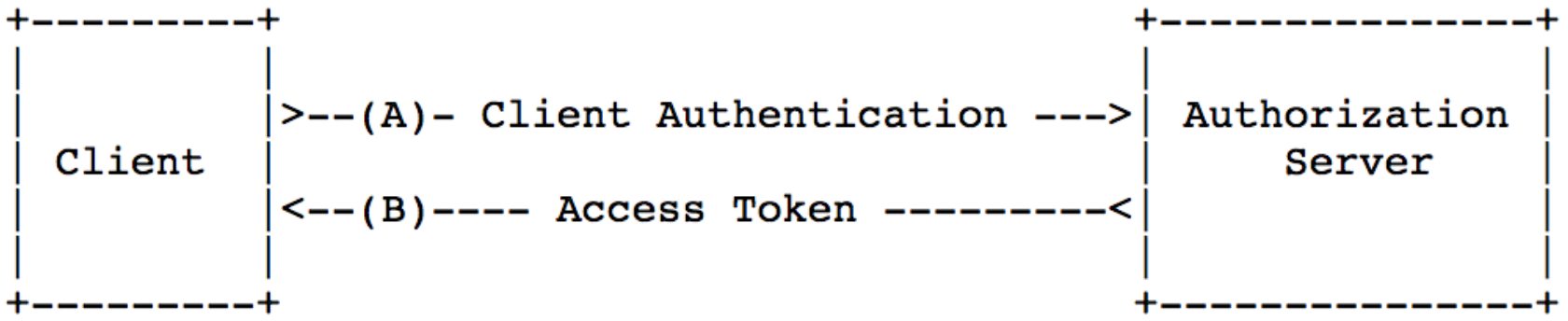
OAuth 2

Resource Owner Password Credentials



OAuth 2

Client Credentials



OAuth 2

Exemplo

Authorization server: oauth2-authorizationserver

Resource server: oauth2-resourceserver

Client: ooauth2-client

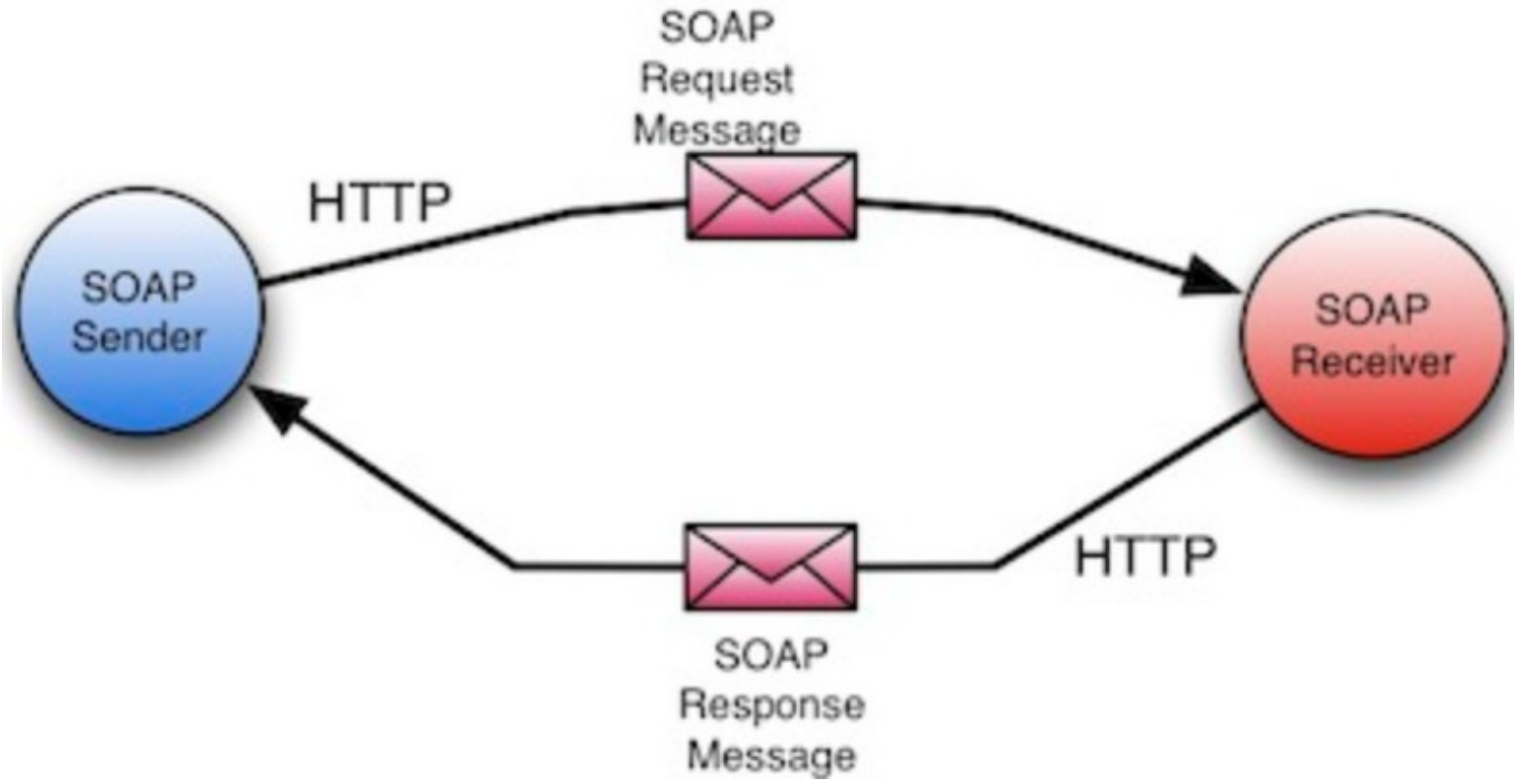
SOAP

SOAP

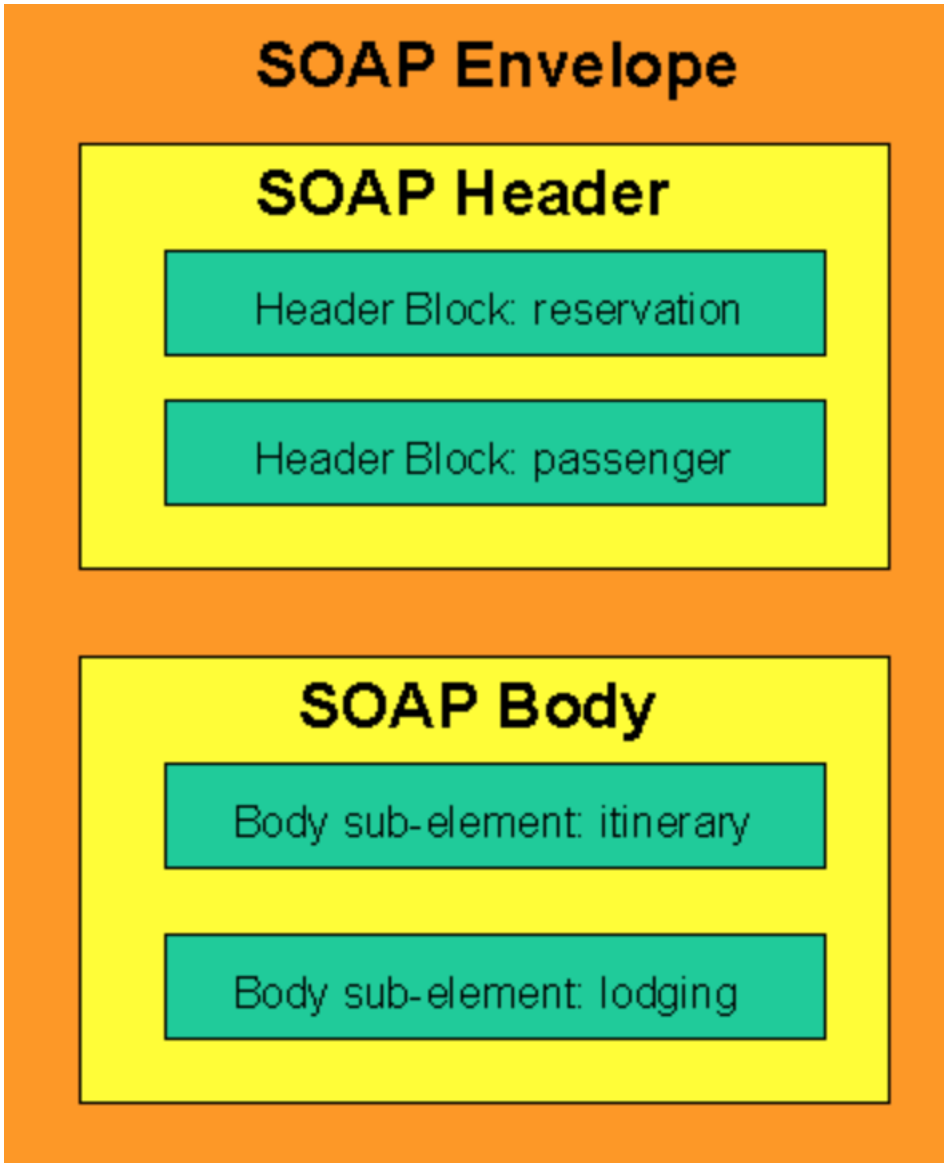
Características

- Baseado em XML;
- Baseado no modelo RPC;
- Independente do protocolo de transporte, sendo o principal HTTP;
- Independente de plataforma;
- É um padrão W3C.

SOAP – Request / Response



SOAP – Envelope



SOAP – Mensagens

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ws="http://ws.posjava.unitri.com.br/">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:hello>
         <!--Optional:-->
         <arg0>Emílio</arg0>
      </ws:hello>
   </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:helloResponse xmlns:ns2="http://ws.posjava.unitri.com.br/">
         <return>Hello Emílio</return>
      </ns2:helloResponse>
   </soap:Body>
</soap:Envelope>
```

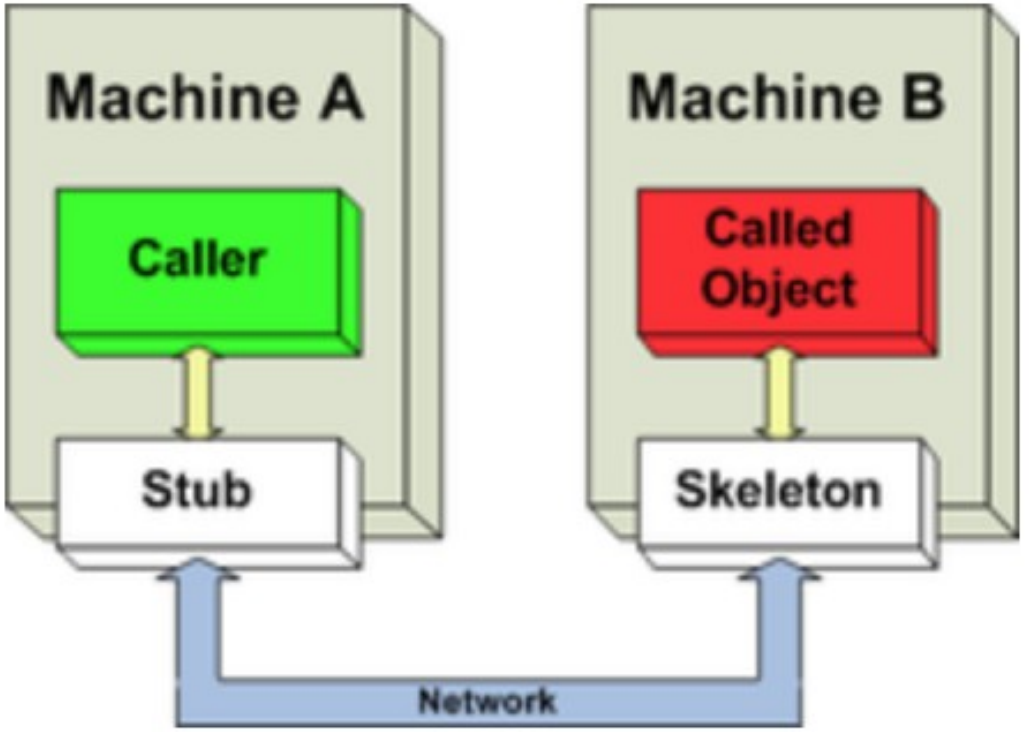
SOAP – Mensagens

WSDL – Web Service Description Language

```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://ws.posjava.unitri.com.br/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="HelloWorldWSService" targetNamespace="http://ws.posjava.unitri.com.br/">
  ▼<wsdl:types>
    ▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://ws.posjava.unitri.com.br/" elementFormDefault="unqualified" targetNamespace="http://ws.posjava.unitri.com.br/" version="1.0">
       <xs:element name="hello" type="tns:hello"/>
       <xs:element name="helloResponse" type="tns:helloResponse"/>
      ▼<xs:complexType name="hello">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="arg0" type="xs:string"/>
         </xs:sequence>
       </xs:complexType>
      ▼<xs:complexType name="helloResponse">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="return" type="xs:string"/>
         </xs:sequence>
       </xs:complexType>
     </xs:schema>
   </wsdl:types>
  ▼<wsdl:message name="helloResponse">
     <wsdl:part element="tns:helloResponse" name="parameters"> </wsdl:part>
   </wsdl:message>
  ▼<wsdl:message name="hello">
     <wsdl:part element="tns:hello" name="parameters"> </wsdl:part>
   </wsdl:message>
  ▼<wsdl:portType name="HelloWorld">
    ▼<wsdl:operation name="hello">
       <wsdl:input message="tns:hello" name="hello"> </wsdl:input>
       <wsdl:output message="tns:helloResponse" name="helloResponse"> </wsdl:output>
     </wsdl:operation>
   </wsdl:portType>
  ▼<wsdl:binding name="HelloWorldWSServiceSoapBinding" type="tns:HelloWorld">
     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    ▼<wsdl:operation name="hello">
       <soap:operation soapAction="" style="document"/>
      ▼<wsdl:input name="hello">
         <soap:body use="literal"/>
       </wsdl:input>
      ▼<wsdl:output name="helloResponse">
         <soap:body use="literal"/>
       </wsdl:output>
     </wsdl:operation>
   </wsdl:binding>
  ▼<wsdl:service name="HelloWorldWSService">
    ▼<wsdl:port binding="tns:HelloWorldWSServiceSoapBinding" name="HelloWorldWSPort">
       <soap:address location="http://localhost:8080/webservices-soap/services/HelloWorld"/>
     </wsdl:port>
   </wsdl:service>
 </wsdl:definitions>
```

SOAP – Mensagens

Stubs e Skeletons - RPC



JAX-WS

SOAP – Mensagens

Exemplo - Servidor

```java
package br.com.unitri.posjava.ws;

import javax.jws.WebService;

@WebService(endpointInterface = "br.com.unitri.posjava.ws.HelloWorld")
public class HelloWorldWS implements HelloWorld {

    public String hello(String name) {

        return "Hello " + name;

    }

}
```

SOAP – Mensagens

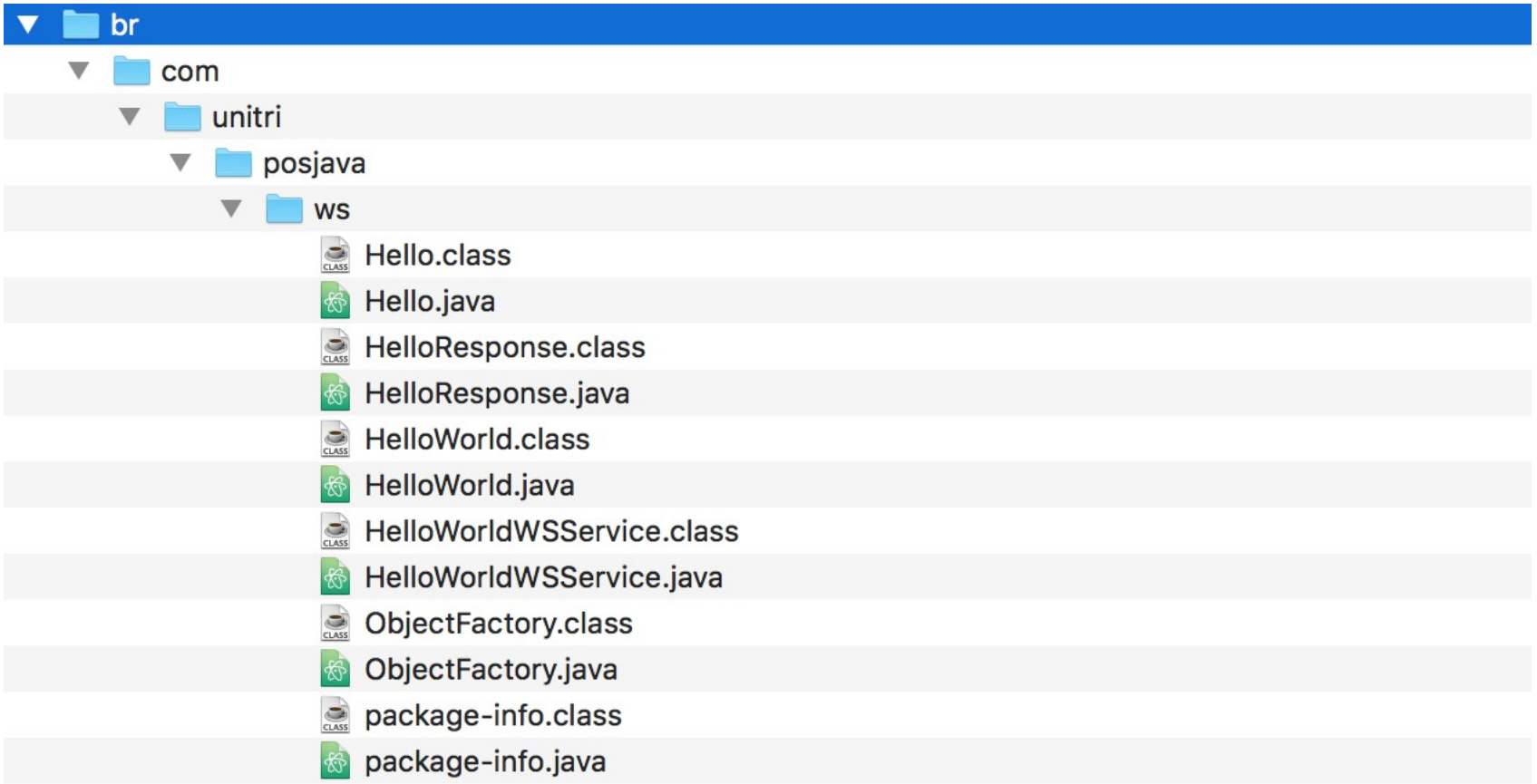
Exemplo - Cliente

```
package br.com.unitri.posjava.ws.main;

import br.com.unitri.posjava.ws.Cliente;
import br.com.unitri.posjava.ws.ClienteWS;
import br.com.unitri.posjava.ws.ClienteWSImplService;



public class Main {

    public static void main(String[] args) {


        ClienteWS clienteWS = new ClienteWSImplService().getClienteWSImplPort();

        Cliente cliente = clienteWS.getCliente(1l);

        System.out.println("Id do cliente: " + cliente.getId());

        System.out.println("Nome do cliente: " + cliente.getNome());

        System.out.println("Endereço do cliente: " + cliente.getEndereco());
    }

}
```

SOAP – Mensagens

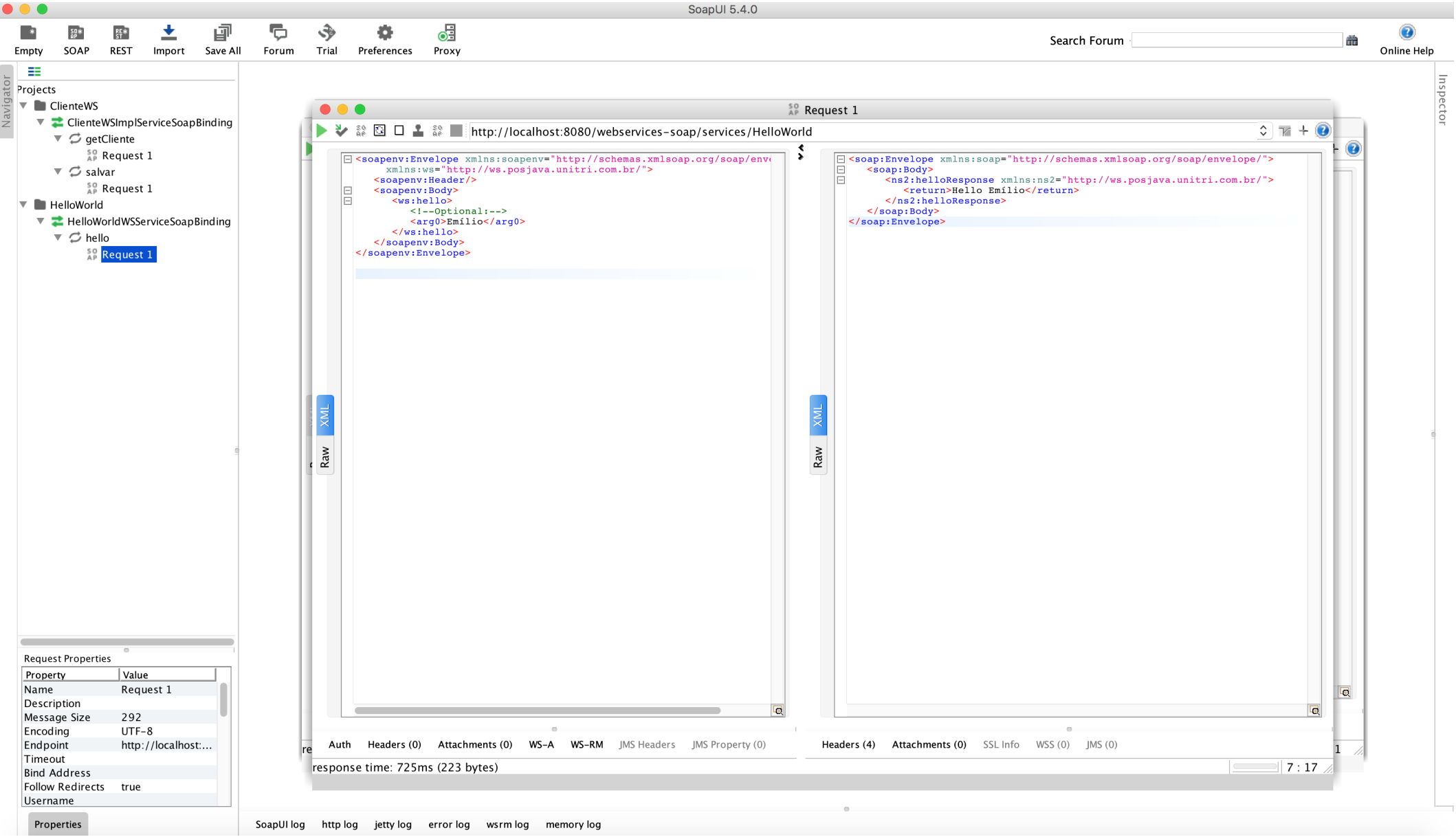
Gerando um Stub JAX-WS

wsimport -s . http://localhost:8080/webservices-soap/services/HelloWorld?wsdl



SOAP – Mensagens

SOAPUI



SOAP – Mensagens

Request/Response

http

No.	Time	Source	Destination	Protocol	Length	Info
87	6.20…	127.0.0.1	127.0.0.1	HTTP/XML	604	POST /webservices-soap/services/HelloWorld HTTP/1.1
89	6.28…	127.0.0.1	127.0.0.1	HTTP/XML	392	HTTP/1.1 200

```
▶ Frame 87: 604 bytes on wire (4832 bits), 604 bytes captured (4832 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 63064, Dst Port: 8080, Seq: 1, Ack: 1, Len: 548
▼ Hypertext Transfer Protocol
  ▶ POST /webservices-soap/services/HelloWorld HTTP/1.1\r\n
    Accept-Encoding: gzip,deflate\r\n
    Content-Type: text/xml;charset=UTF-8\r\n
    SOAPAction: ""\r\n
  ▶ Content-Length: 293\r\n
    Host: localhost:8080\r\n
    Connection: Keep-Alive\r\n
    User-Agent: Apache-HttpClient/4.1.1 (java 1.5)\r\n
    \r\n
    [Full request URI: http://localhost:8080/webservices-soap/services/HelloWorld]
    [HTTP request 1/1]
    [Response in frame: 89]
    File Data: 293 bytes
▼ eXtensible Markup Language
  ▼ <soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ws="http://ws.posjava.unitri.com.br/">
      <soapenv:Header/>
    ▼ <soapenv:Body>
      ▼ <ws:hello>
          <!--Optional:-->
        ▶ <arg0>
          </ws:hello>
      </soapenv:Body>
    </soapenv:Envelope>
```

SOAP – Mensagens

Request/**Response**

http

No.	Time	Source	Destination	Protocol	Length	Info
87	6.20…	127.0.0.1	127.0.0.1	HTTP/XML	604	POST /webservices-soap/services/HelloWorld HTTP/1.1
89	6.28…	127.0.0.1	127.0.0.1	HTTP/XML	392	HTTP/1.1 200

```
▶ Frame 89: 392 bytes on wire (3136 bits), 392 bytes captured (3136 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 8080, Dst Port: 63064, Seq: 1, Ack: 549, Len: 336
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 \r\n
    Content-Type: text/xml;charset=UTF-8\r\n
  ▶ Content-Length: 223\r\n
    Date: Sat, 30 Jun 2018 15:04:18 GMT\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.083975000 seconds]
    [Request in frame: 87]
    File Data: 223 bytes
▼ eXtensible Markup Language
  ▼ <soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    ▼ <soap:Body>
      ▼ <ns2:helloResponse
          xmlns:ns2="http://ws.posjava.unitri.com.br/">
        ▼ <return>
            Hello Em\303\255lio
            </return>
          </ns2:helloResponse>
        </soap:Body>
      </soap:Envelope>
```

Obrigado!