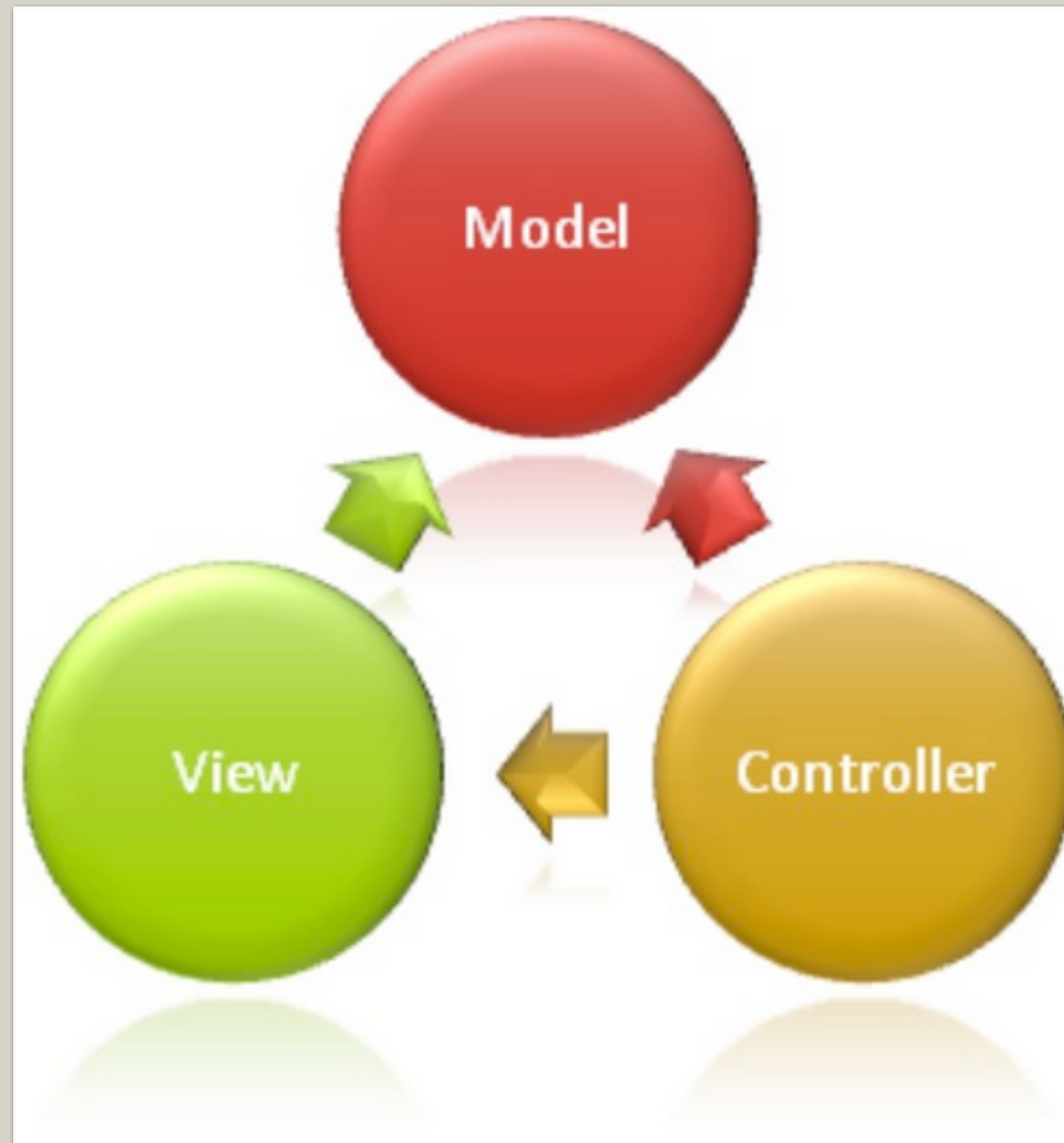
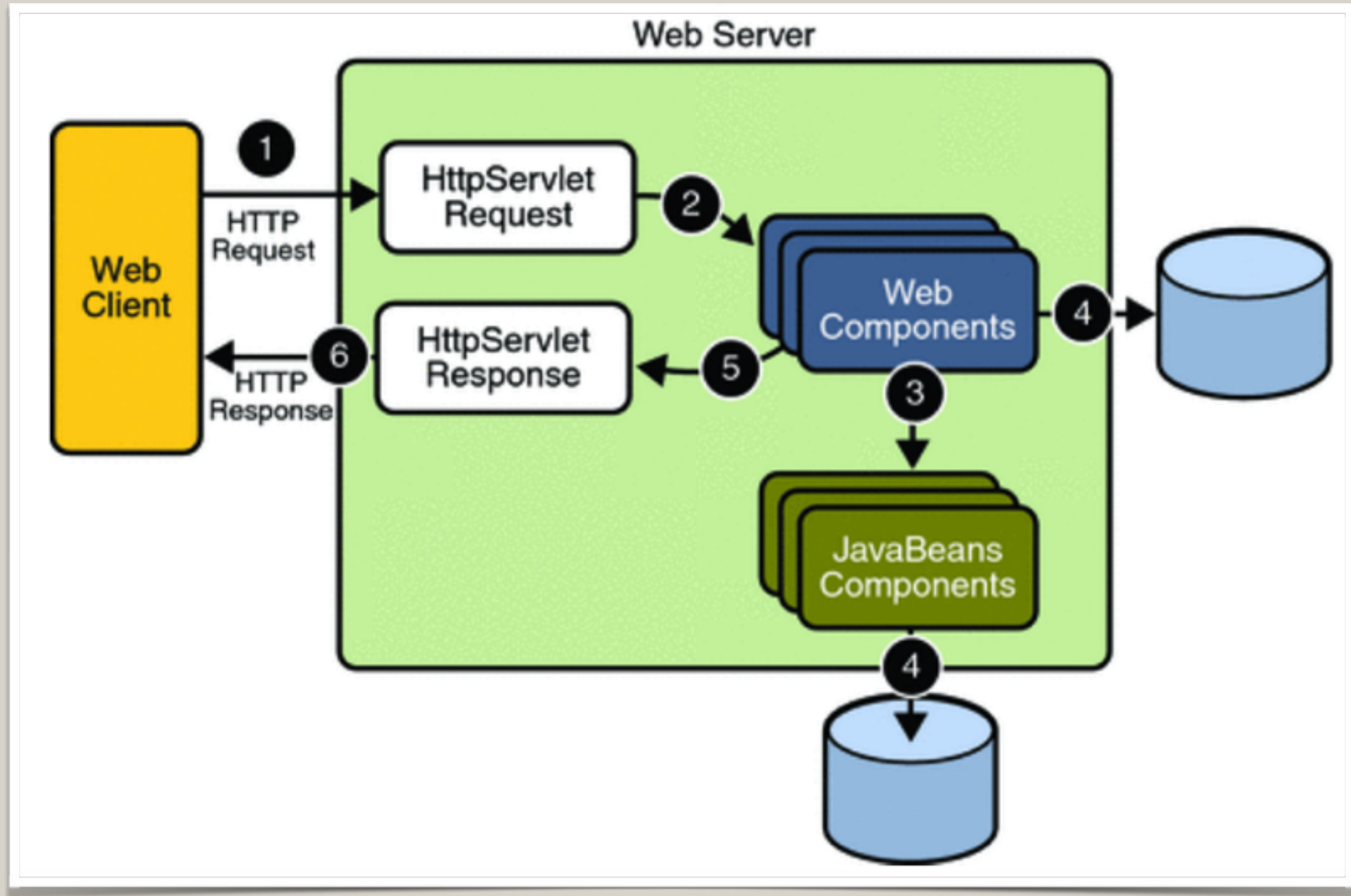


Desenvolvimento Web com VRaptor 4

MVC

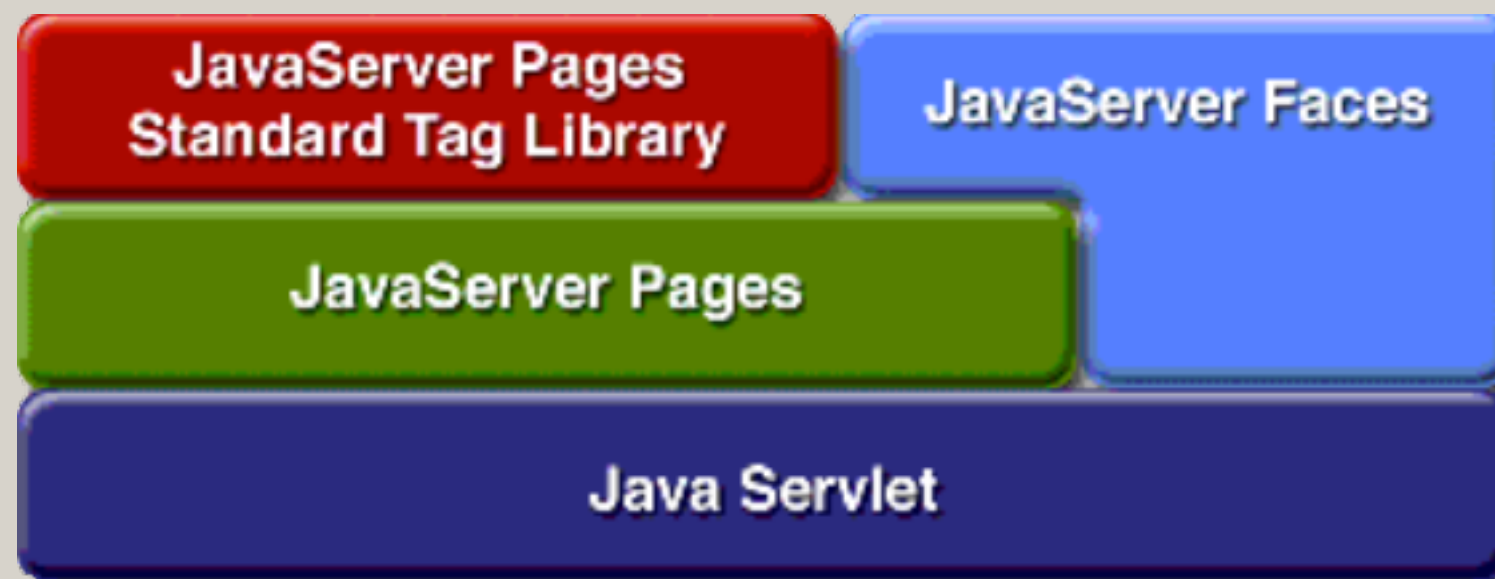


Atendimento de requisições HTTP



Desenvolvimento Web com Java

- Servlet e JSPs



JSTL: JSP Standard Tag Library

- Conjunto de controles comuns encapsulados

Taglib	Funcionalidades	Prefixo
Core	Suporte a variáveis Controle de fluxo Manipulação de URLs Entre outras..	c
I18N	Localização Formatação de mensagens Formatação de números e datas	fmt
Functions	Tamanho de coleções Manipulação de Strings	fn

JSTL: JSP Standard Tag Library

- Declaração de início

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- Exemplo:

```
<c:forEach var="item" items="{itensLista}">
```

```
    <li> ${item.descricao} </li>
```

```
</c:forEach>
```

EL: Expression Language

`${objeto.atributo}`

`${objeto['atributo']}`

`${objeto.lista[1]}`

`${objeto.array[0]}`

`${objeto.mapa.chave}`

`${objeto.mapa['chave']}`

`${objeto.mapa[valor]}`

EL: Expression Language

- Operadores:

Aritméticos	+, -, *, / e div, % e mod (binários) - (unário)
Lógicos	and, &&, or, , not, !
Relacionais	==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le
Condicionais	A ? B : C
empty	

EL: Expression Language

- Palavras reservadas:

and	or	not	eq
ne	lt	gt	le
ge	true	false	null
instanceof	empty	div	mod

vraptor

Facilidades

- Framework WEB Open Source focado em produtividade
- **Brasileiro**, mantido pela Caelum e comunidade
- Encapsula a complexidade da API `javax.servlet`
- Boas práticas adotadas:
 - Convenção sobre configuração
 - Injeção de dependências

Benefícios



ALTA PRODUTIVIDADE

Usar o VRaptor 3 é simples e intuitivo. Você atingirá níveis altíssimos de produtividade com Java para Web.



CURVA DE APRENDIZADO

Em pouco tempo você conseguirá aprender tudo o que é necessário para desenvolver suas aplicações com o VRaptor.



TESTABILIDADE

Escreva código modularizado e desacoplado do VRaptor. Sua aplicação fica altamente testável e de fácil manutenção.



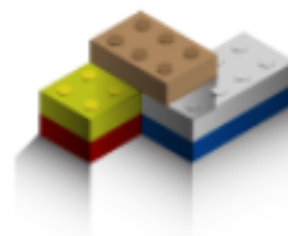
ECONOMIA

Economize muitas horas de trabalho com a alta produtividade do VRaptor, a facilidade em treinar a sua equipe e a qualidade final do seu projeto.



FLEXIBILIDADE

Integre o seu projeto com qualquer framework de sua preferência. Você não estará preso a nenhuma tecnologia específica.



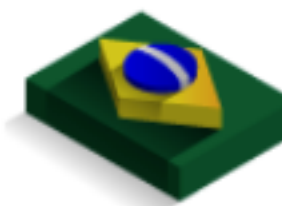
SOA E REST - READY

Faça aplicações RESTful ou orientadas a serviço sem complicações, como se estivesse fazendo aplicações Web normais.



MELHORES PRÁTICAS DE DESENVOLVIMENTO

Utilizando os conceitos de Injeção de Dependência, Inversão de Controle e POJOs, seu código fica simples e testável.



DOCUMENTAÇÃO EM PORTUGUÊS

Aprenda tudo sobre VRaptor 3 contando com uma ampla documentação, fóruns e listas de discussão em português.

Desvantagens

- Não possui componentes próprios
- Maior dependência de HTML e Javascript

Action-based x Component-based frameworks

CONTROLLERS

Controllers

- Objetos que irão controlar o fluxo da
- Aplicação web Classes anotadas com **@Controller**

Controllers

@Controller

```
public class VeiculoController {  
    ...  
    public List<Veiculo> listar() {  
        return veiculos;  
    }  
}
```


Convenção para Controllers

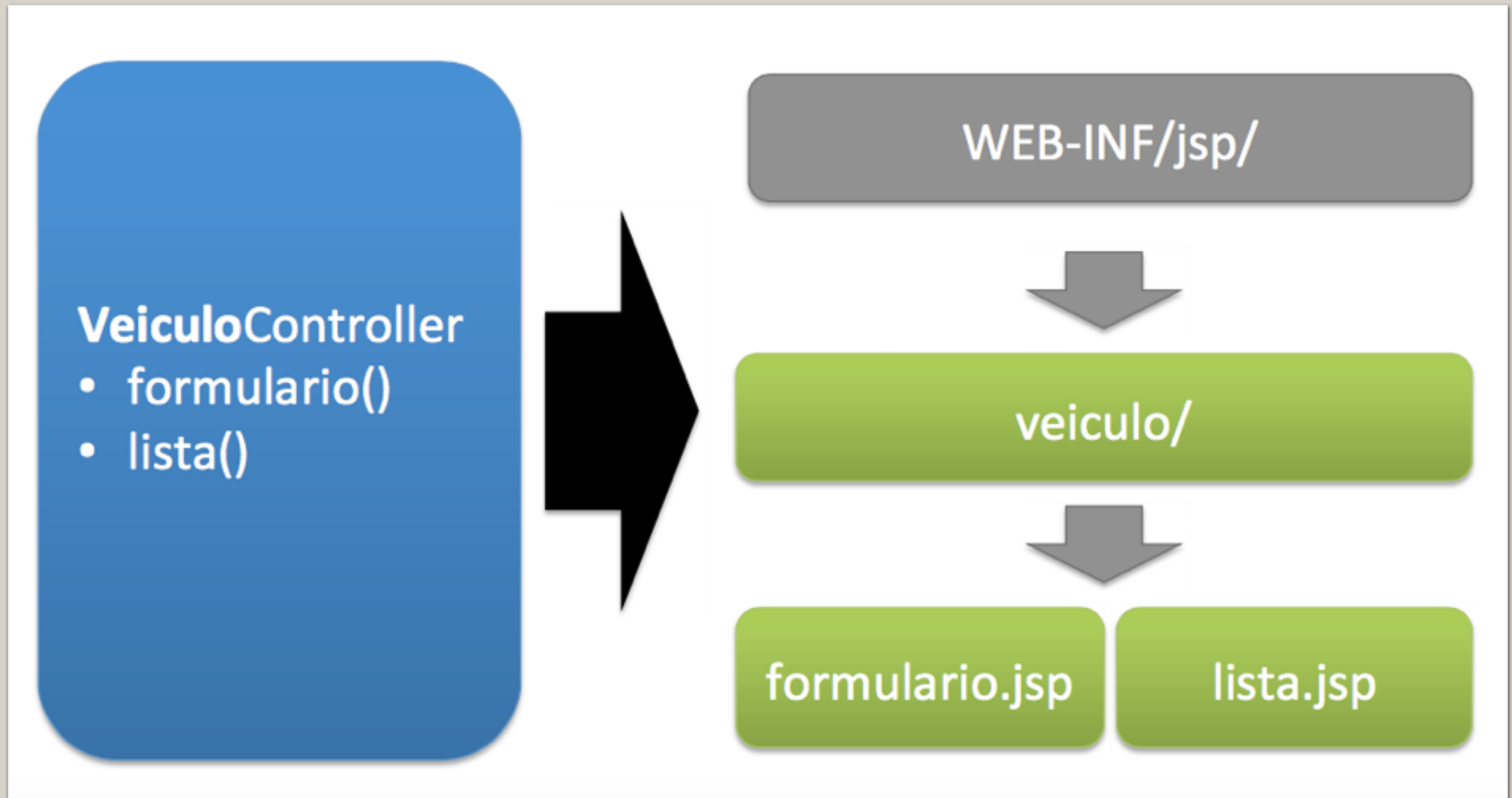
- URL da requisição

`/<nome_do_controller>/<nome_do_metodo>`

- Redirecionamento

`/WEB-INF/jsp/<nome_do_controller>/<nome_do_metodo>.jsp`

Convenção para Controllers



Retornando coleções

```
public List<Veiculo> lista() { ... } <VeiculoController>
```

```
<ul>  
  <c:forEach items="{veiculoList}" var="veiculo">  
    <li> {veiculo.placa} </li>  
  </c:forEach>  
</ul>
```

<lista.jsp>

Convenção para nomes de coleções

<tipoDaCollection>List

Submetendo dados

@Controller

```
public class VeiculoController {  
    ...  
    public void salvar() {  
    }  
}
```

Submetendo dados

```
<form action="<c:url value="/veiculo/salvar"/>">
```

Placa:

```
<input type="text" name="veiculo.placa" /><br/>
```

Modelo:

```
<input type="text" name="veiculo.modelo" /><br/> <input  
type="submit" value="Salvar" />
```

```
</form>
```

Submetendo dados

@Controller

```
public class VeiculoController {  
    ...  
    public void salvar(Veiculo veiculo) {  
        dao.salvar(veiculo) ;  
        result.redirectTo(this).lista();  
    }  
}
```

Escopos de Componentes

@RequestScoped

O componente é o mesmo durante uma requisição

@SessionScoped

O componente é o mesmo durante uma HTTP Session

@ApplicationScoped

O componente é um singleton, apenas um por aplicação

@ConversationScoped

A instância do componente é mantida durante uma conversation.

@Depended

O component é instanciado sempre que requisitado.

Componente Result

- Criado automaticamente pelo VRaptor
- **Responsabilidades:**
 - Adicionar atributos na requisição mudar a view a ser carregada
- Deve ser injetado no Controller

Result: Retornando objetos

```
result.include("mensagem" , "Alguma mensagem" );  
result.include("cliente" , new Cliente(id));
```

Result: Exemplo

```
@Get
@Path("/pesquisar")
public void listar(Long codigo, String nome){
    result.include("codigo",codigo);
    result.include("nome",nome);
    List<Cliente> clientes = dao.list(nome);
    result.include("clienteList", clientes);
}
```

```
<form action="<c:url value="/clientes/pesquisar"/>">
    <label for="codigo">Código</label>
    <input name="codigo" id="codigo" value="<u>${codigo}</u>" />
    <label for="nome">Nome</label>
    <input name="nome" id="nome" value="<u>${nome}</u>" />
    <button type="submit">Pesquisar</button>
</form>
<table>
    <c:forEach items="<u>${clienteList}</u>" var="cliente">
```

Result: Definição da View

Exemplo:

```
result.redirectTo(ProdutosController.class ).lista();
```

Result: Outras formas de definição

result.forwardTo("/some/uri")

result.redirectTo("/some/uri")

result.forwardTo(ClientController.class).list()

result.forwardTo(this).list()

result.of(this).list()

Conversores

O VRaptor já possui registrado conversores para todos os tipos definidos na Java Language Specification.

```
@Convert(Pais.class)
@ApplicationScoped
public class PaisConverter implements Converter<Pais> {

    public Pais convert(String value, Class<? extends Pais> type) {
        Pais pais = new Pais();
        if (!isEmpty(value)) {
            pais.setNome(value)
        }

        return pais;
    }
}
```

Interceptadores

Executar alguma tarefa antes e/ou depois de uma lógica de negócios.

Usos mais comuns:

- Validação de dados,
- Controle de conexão e transação do banco,
- Log
- Criptografia/compactação de dados

Interceptadores

Classe deve ser anotada com `@Intercepts` e definido o escopo.

```
@Intercepts
@RequestScoped
public class Log {
    @BeforeCall
    public void before() {
        // código a ser executado antes da lógica
    }
    @AfterCall
    public void after() {
        // código a ser executado depois da lógica
    }
}
```


Interceptadores

@BeforeCall: Executa código antes da lógica.

@AfterCall: Executa código depois da lógica.

```
@Intercepts
@RequestScoped
public class Log {
    @BeforeCall
    public void before() {
        // código a ser executado antes da lógica
    }
    @AfterCall
    public void after() {
        // código a ser executado depois da lógica
    }
}
```


Interceptadores

@AroundCall: Executa código depois da lógica.
Injeta o objeto **SimpleInterceptorStack**

```
@Intercepts
@RequestScoped
public class Log {

    @Inject
    private HttpServletRequest request;

    @AroundCall
    public void intercept(SimpleInterceptorStack stack) {
        System.out.println("Interceptando " + request.getRequestURI());
        // código a ser executado antes da lógica

        stack.next(); // continua a execução
    }
}
```

Interceptadores: Definindo quando interceptar

```
@Accepts  
public boolean accepts(ControllerMethod method) {  
    return method.containsAnnotation(Audit.class);  
}
```

Interceptadores: Definindo a ordem dos interceptors

```
@Intercepts(before=SegundoInterceptor.class)  
public class PrimeiroInterceptor { ... }
```

```
@Intercepts(after=PrimeiroInterceptor.class)  
public class SegundoInterceptor { ... }
```

```
@Intercepts(after={PrimeiroInterceptor.class, SegundoInterceptor.class},  
             before={QuartoInterceptor.class, QuintoInterceptor.class})  
public class TerceiroInterceptor { ... }
```


Dúvidas?

Obrigado