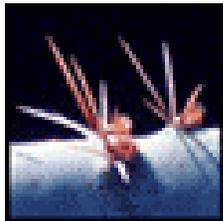
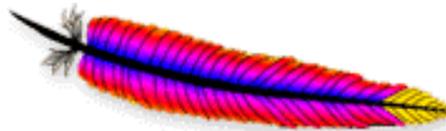


Testes de Integração

em **J2EE** com

Jakarta



C A C T U S

1. Discutir as dificuldades relativas a **testes em J2EE**
2. Apresentar o framework **Cactus** e demonstrar como utilizá-lo para testar aplicações J2EE
3. Mostrar como usar o Cactus com extensões como **HttpUnit** e **JUnitEE** para testar servlets e EJBs

Público-alvo:

Programadores J2EE familiarizados com o JUnit



Por que outro framework para testes J2EE?

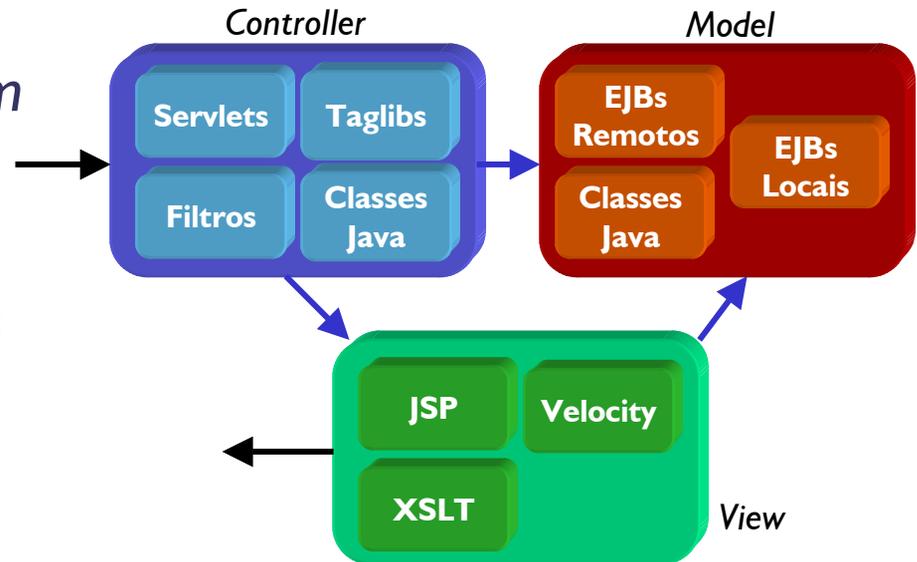
- *JUnit não é suficiente?*
 - *Sim, para testes de unidade (lógica de EJBs, por exemplo)*
 - *Aplicações J2EE possuem muitas dependências (é preciso simular a rede, o servidor, o EJB container, etc.)*
 - *Testes de unidade não garantem o funcionamento de toda uma aplicação J2EE*
- *Mock objects não servem?*
 - *Sim, mock objects ajudam a testar com dependências e permitem testar toda a aplicação*
 - *Mock objects podem ficar muito complicados*
 - *Aplicações geralmente rodam em um servidor de aplicação ou Web - testar a integração da aplicação com o servidor é mais útil e mais fácil*

O que é Cactus?

- Um framework que oferece facilidades para testar componentes J2EE
 - Componentes Web (Camada de **C**ontrole)
 - Camada EJB (**M**odel) e cliente (**V**iew): indiretamente
- Produto Open Source do projeto Jakarta
 - Metas de curto prazo: testar servlets, JSPs, filtros e EJB
 - Metas de longo prazo: oferecer facilidades para testar todos os componentes J2EE; ser o framework de referência para testes in-container.
- Cactus estende o JUnit framework
 - Execução dos testes é realizada de forma idêntica
 - TestCases são construídos sobre uma subclasse de `junit.framework.TestCase`

Para que serve?

- Para testar aplicações que utilizam componentes J2EE
- Arquitetura MVC
 - Servlets, filtros e custom tags (**C**ontroladores)
 - JSPs (camada de apresentação: **V**iew, através de controladores)
 - EJB (**M**odelo de dados/ lógica de negócios)
- Cactus testa a integração desses componentes com seus containers
 - **não usa stubs** - usa o próprio container como servidor e usa JUnit como cliente
 - comunicação é intermediada por um **proxy**



- *Cactus utiliza os test cases simultaneamente no cliente e no servidor: **duas cópias***
 - *Uma cópia é instanciada pelo servlet container*
 - *Outra cópia é instanciada pelo JUnit*
- *Comunicação com o servlet container é feita através de um proxy (XXXRedirector)*
 - *JUnit envia requisições via HTTP para proxy*
 - *Proxy devolve resultado via HTTP e JUnit os mostra*
- *Há, atualmente (Cactus 1.3) três tipos de proxies:*
 - ***ServletRedirector**: para testar servlets*
 - ***JSPRedirector**: para testar JSP custom tags*
 - ***FilterRedirector**: para testar filtros de servlets*

- Parte da mesma classe (*ServletTestCase*) é executada no cliente, parte no servidor

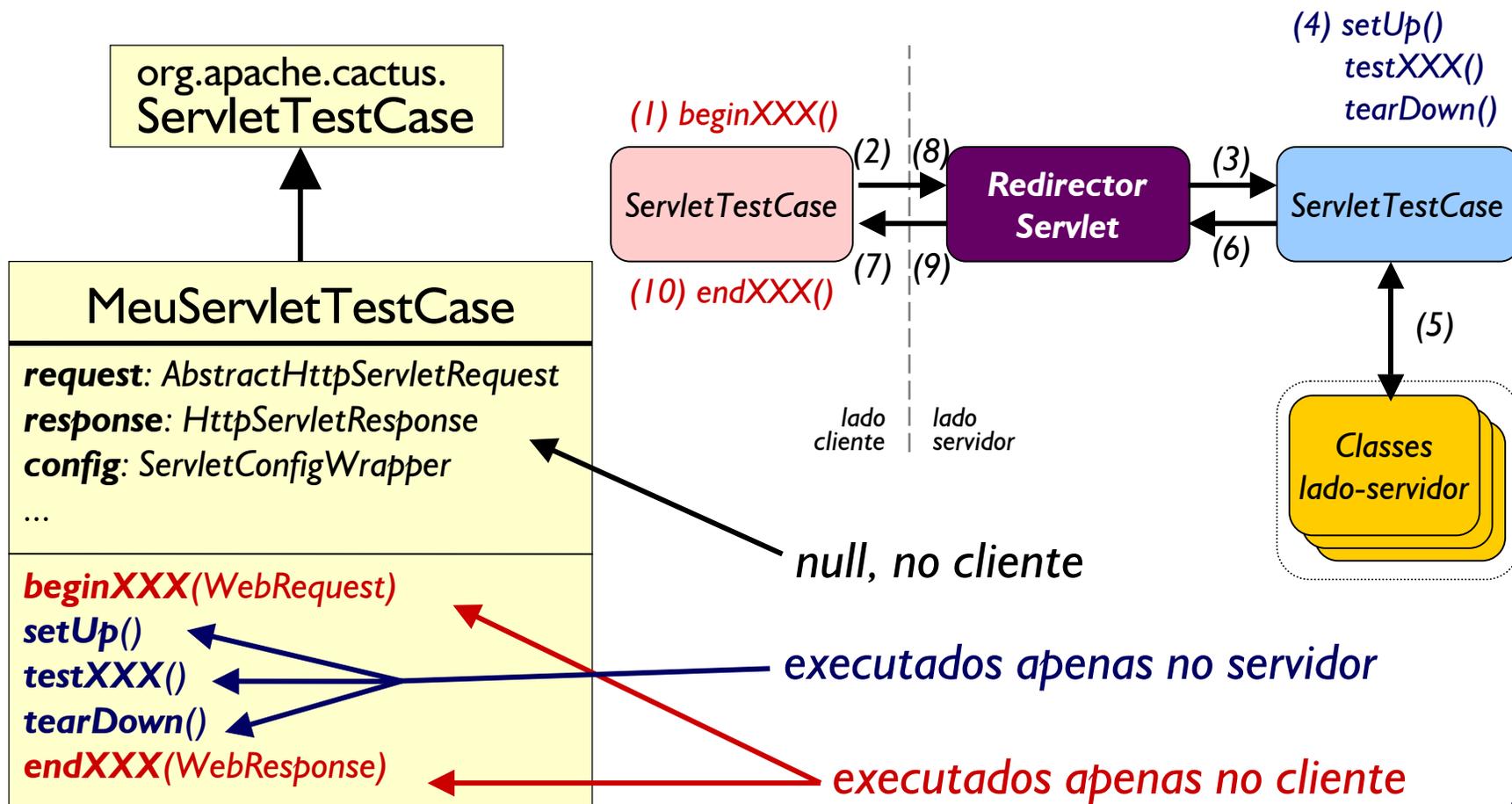
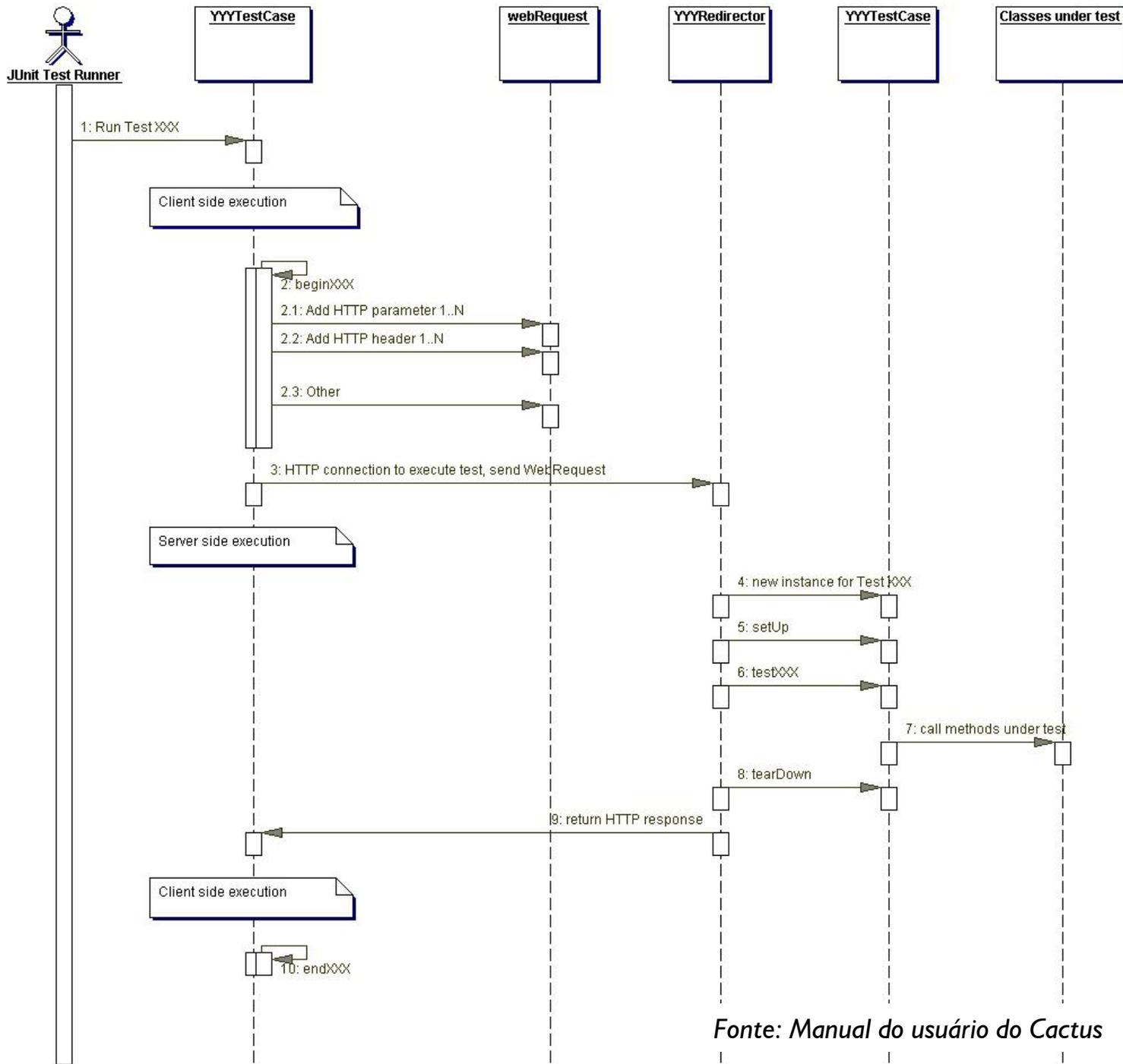


Diagrama UML



ServletTestCase (ou similar)

- Para cada método `XXX()` a ser testado, pode haver:
 - Um `beginXXX()`, para inicializar a requisição do cliente
 - encapsulada em um objeto `WebRequest` a ser enviado ao servidor
 - Um `testXXX()`, para testar o funcionamento do método no servidor (deve haver ao menos um)
 - Um `endXXX()`, para verificar a resposta do servidor
 - devolvida em um objeto `WebResponse` retornada pelo servidor
- Além desses três métodos, cada `TestCase` pode conter
 - `setUp()`, opcional, para inicializar objetos no servidor
 - `tearDown()`, opcional, para liberar recursos no servidor
- Os métodos do lado do servidor têm acesso aos mesmos objetos implícitos disponíveis em um servlet ou página JSP: `request`, `response`, etc.

Cactus: exemplo

- Veja **cactusdemo.zip** (distribuído com esta palestra)
 - Usa duas classes: um servlet (**MapperServlet**) e uma classe (**SessionMapper**) que guarda cada parâmetro como atributo da sessão e em um `HashMap` - veja fontes em **src/xptoolkit/cactus**
- Para rodar, configure o seu ambiente:
 - **build.properties** - localização dos JARs usados pelo servidor Web (`CLASSPATH` do servidor)
 - **runtests.bat** (para Windows) e **runtests.sh** (para Unix) - localização dos JARs usados pelo JUnit (`CLASSPATH` do cliente)
 - **lib/client.properties** (se desejar rodar cliente e servidor em máquinas separadas, troque as ocorrências de `localhost` pelo nome do servidor)
- Para montar, execute:
 - 1. **ant test-deploy** instala `cactus-tests.war` no tomcat
 - 2. o servidor (Tomcat 4.0 startup)
 - 3. **runtests.bat** roda os testes no JUnit

veja demonstração

cactusdemo.zip

- O objetivo deste servlet é
 - 1) gravar qualquer parâmetro que receber na sessão (objeto session)
 - 2) devolver uma página contendo os pares nome/valor em uma tabela
 - 3) imprimir resposta em caixa-alta se `<init-param> ALL_CAPS` definido no `web.xml` contiver o valor `true`

```
public void doGet(...) throws IOException {
    SessionMapper.mapRequestToSession(request); ← (1) Grava request
    writer.println("<html><body><table border='1'>");
    // (... loop for each parameter ...)
    if (useAllCaps()) { ← (3) Retorna true se <init-param>
        key = key.toUpperCase();
        val = val.toUpperCase();
        "ALL_CAPS" contiver "true"
    }
    str = "<tr><td><b>"+key+"</b></td><td>"+val+"</td></tr>";
    writer.println(str); ← (2)
    // (...)
    writer.println("</table></body></html>");
}

```

Trecho de MapperServlet.java

- Escreveremos os testes para avaliar esses objetivos

CactusDemo: testes

MapperServletTest.java

```
public class MapperServletTest extends ServletTestCase { (...)  
    private MapperServlet servlet;  
    public void beginDoGet(WebRequest cSideReq) {  
        cSideReq.addParameter("user", "Jabberwock");  
    }  
    public void setUp() throws ServletException {  
        this.config.setInitParameter("ALL_CAPS", "true");  
        servlet = new MapperServlet();  
        servlet.init(this.config);  
    }  
    public void testDoGet() throws IOException {  
        servlet.doGet(this.request, this.response);  
        String value = (String) session.getAttribute("user");  
        assertEquals("Jabberwock", value);  
    }  
    public void tearDown() { /* ... */ }  
    public void endDoGet(WebResponse cSideResponse) {  
        String str = cSideResponse.getText();  
        assertTrue(str.indexOf("USER</b></td><td>JABBERWOCK") > -1);  
    }  
}
```

Simula DD
<init-param>

Simula servlet
container

Verifica se parâmetro foi
mapeado à sessão

Verifica se parâmetro aparece na tabela HTML

Exemplo: funcionamento

Cliente (JUnit)

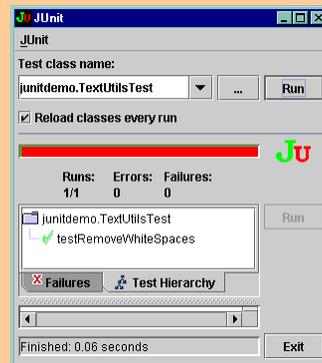
beginDoGet (WebRequest req)

- Grava parâmetro:
nome = **user**
value = **Jabberwock**

SUCCESS!!



FAIL!



endDoGet (WebResponse res)

- Verifica se resposta contém
USER</td><td>**JABBERWOCK**

Servidor (Tomcat)

ReqInfo

setUp ()

- Define init-params
no objeto config
- Roda init (config)

2 conexões HTTP:

- Uma p/ rodar os testes e obter saída do servlet
- Outra para esperar resultados de testes (info sobre exceções)

testDoGet ()

- Roda doGet ()
- Verifica se parâmetro
(no response) foi
mapeado à sessão

TestInfo

tearDown ()

Output

falha local

falha remota

&

- Onde encontrar
 - `http://httpunit.sourceforge.net`
- Framework para testes funcionais de interface (teste tipo "caixa-preta")
 - Verifica a resposta de uma aplicação Web ou página HTML
 - É teste funcional caixa-preta (não é "unit")
 - Oferece métodos para "navegar" na resposta
 - links, tabelas, imagens
 - objetos DOM (Node, Element, Attribute)
- Pode ser combinado com Cactus no `endXXX()`
 - Argumento `com.meterware.httpunit.WebResponse`
- Acompanha **ServletUnit**
 - stub que simula o servlet container

veja também

[httpunitdemo.zip](#)

Resumo da API do HttpUnit

- **WebConversation**
 - *Representa uma sessão de cliente Web (usa cookies)*

```
WebConversation wc = new WebConversation();  
WebResponse resp = wc.getResponse("http://xyz.com/t.html");
```
- **WebRequest**
 - *Representa uma requisição*
- **WebResponse**
 - *Representa uma resposta. A partir deste objeto pode-se obter objetos **WebLink**, **WebTable** e **WebForm***
- **WebLink**
 - *Possui métodos para extrair dados de links de hipertexto*
- **WebTable**
 - *Possui métodos para navegar na estrutura de tabelas*
- **WebForm**
 - *Possui métodos para analisar a estrutura de formulários*

- Troque o **WebResponse** em cada `endXXX()` por **`com.meterware.httpunit.WebResponse`**

```
public void endDoGet(com.meterware.httpunit.WebResponse resp)
                    throws org.xml.sax.SAXException {
    WebTable[] tables = resp.getTables();
    assertNotNull(tables);
    assertEquals(tables.length, 1); // só há uma tabela
    WebTable table = tables[0];
    int rows = table.getRowCount();
    boolean keyDefined = false;
    for (int i = 0; i < rows; i++) {
        String key    = table.getCellAsText(i, 0); // col 1
        String value  = table.getCellAsText(i, 1); // col 2
        if (key.equals("USER")) {
            keyDefined = true;
            assertEquals("JABBERWOCK", value);
        }
    }
    if (!keyDefined) {
        fail("No key named USER was found!");
    }
}
```

Outros testes com Cactus

- **Testes em taglibs (JspRedirector)**
 - Veja exemplos em `cactusdemo/taglib/src`
- **Testes em filtros (FilterRedirector)**
 - Usa proxy `FilterRedirector`
 - Teste básico é verificar se método `doFilter()` foi chamado
 - Veja exemplos em `cactusdemo/src/xptoolkit/AuthFilter`
- **Testes indiretos em páginas JSP (camada **V**iew)**
 - Ideal é JSP não ter código Java
 - Principais testes são sobre a interface: `HttpUnit!`
- **Testes indiretos em EJB (camada **M**odel)**
 - Indireto, através dos redirectors + `JUnitEE`
 - Redirectors permitem testar EJBs com interface local ou remota chamados por código no servidor

veja

`taglibdemo.zip`

veja também

`strutsdemo.zip`

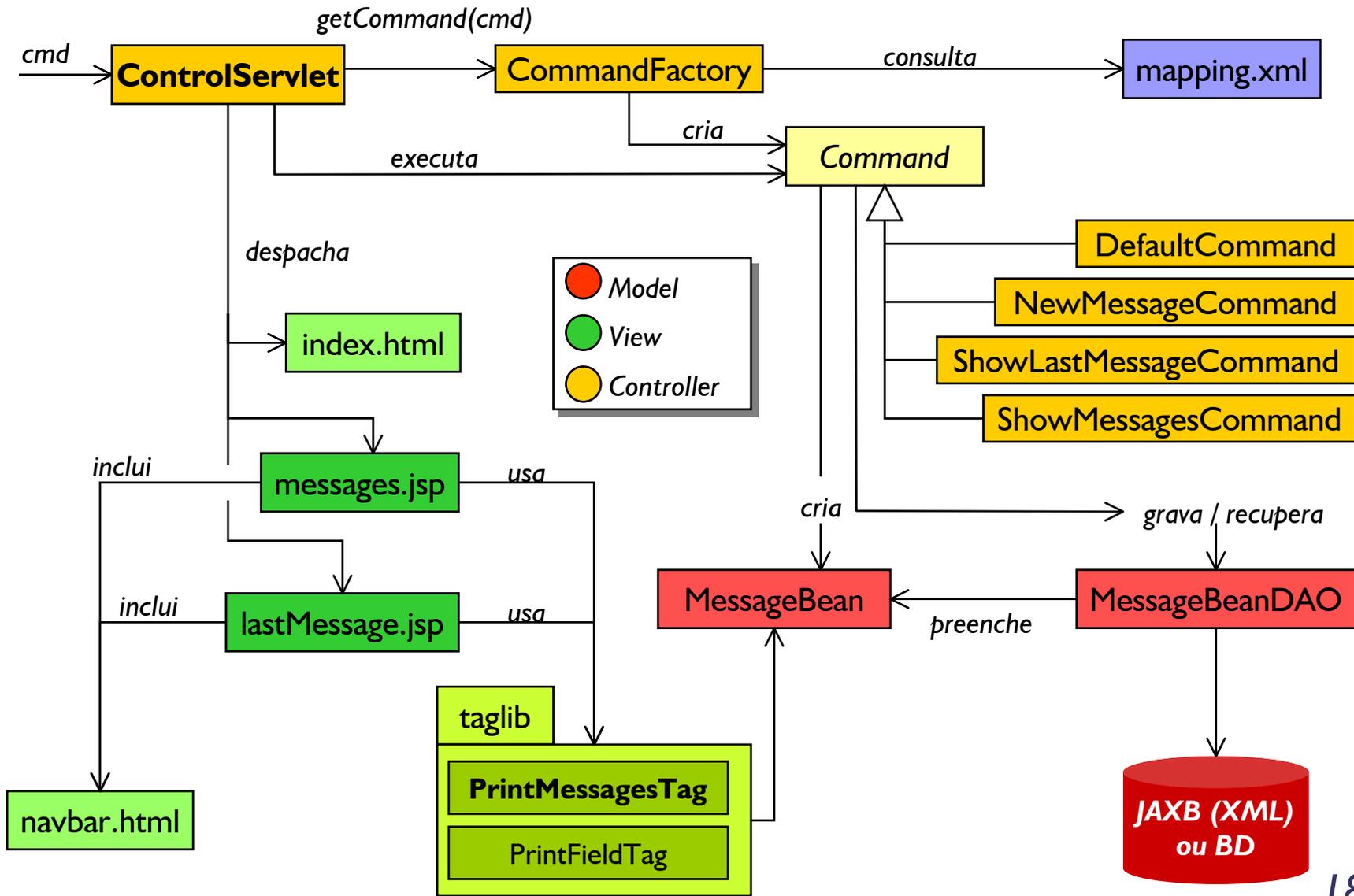
veja

`hellojsp.zip`

veja

`helloejb.zip`

Exemplo de aplicação MVC



ControlServletTest: client set up

```
import org.apache.cactus.*;
import junit.framework.*;

...

public class ControlServletTest extends ServletTestCase {

    private String testMessage = "This is a Web Message.";

    public void beginServiceNewMessage(WebRequest clientRequest) {
        clientRequest.addParameter("cmd", "newMessage");
        clientRequest.addParameter("message", testMessage);
    }

    public void beginServiceShowLastMessage(WebRequest clientRequest) {
        clientRequest.addParameter("cmd", "showLastMessage");
    }

    public void beginServiceShowMessages(WebRequest clientRequest) {
        clientRequest.addParameter("cmd", "showAllMessages");
    }

    public void beginServiceDefaultCommand(WebRequest clientRequest) {
        // nothing or unknown command
    }

    ...
}
```



ControlServletTest: server set up & fixture

...

```
private HttpServlet servlet;  
private String messageDir = "/tmp/test";  
  
public void setUp() throws ServletException {  
    config.setInitParameter("mappings-file",  
                            "/WEB-INF/mapping.xml");  
    config.setInitParameter("message-dir",  
                            messageDir);  
    servlet = new ControlServlet();  
    servlet.init(this.config);  
}
```

...

ControlServletTest: server-side tests

...

```
public void _testService() throws ServletException, IOException {
    System.out.print("Starting servlet...");
    servlet.service(this.request, this.response);
    System.out.println("Server side testing is done.");
}

public void testServiceNewMessage() throws ServletException, IOException {
    assertEquals("newMessage", request.getParameter("cmd"));
    assertEquals(testMessage, request.getParameter("message"));
    _testService();
}

public void testServiceShowLastMessage() throws ServletException, ... {
    assertEquals("showLastMessage", request.getParameter("cmd"));
    _testService();
}

public void testServiceShowMessages() throws ServletException, IOException {
    assertEquals("showAllMessages", request.getParameter("cmd"));
    _testService();
}

public void testServiceDefaultCommand() throws ServletException, ... {
    _testService();
}

...
```

ControlServletTest: clientp-side response

```
public void
    endServiceNewMessage (com.meterware.httpunit.WebResponse clientResponse)
                        throws SAXException, IOException {
    String pageTitle = clientResponse.getTitle();
    assertTrue("Wrong page: " + pageTitle,
                pageTitle.indexOf("Last Message") >= 0);
    String responseString = clientResponse.getText();
    assertTrue("Test Message is not in result page!",
                responseString.indexOf(testMessage) >= 0);
}

public void endServiceShowLastMessage(...) ... {
    String pageTitle = clientResponse.getTitle();
    assertTrue("Wrong page: " + pageTitle,
                pageTitle.indexOf("Last Message") >= 0);
}

public void endServiceShowMessages(...) ... {
    String pageTitle = clientResponse.getTitle();
    assertTrue("Wrong page: " + pageTitle,
                pageTitle.indexOf("All Messages") >= 0);
}
...
}
```

PrintMessagesTagTest

```
public class PrintMessagesTagTest extends JspTestCase {
    ...
    public void testDoAfterBody() throws JspException, java.io.IOException {
        tag.doStartTag();
        bodyContent = pageContext.pushBody();
        tag.setBodyContent(bodyContent); int count = 0;
        do {
            bodyContent.print(
                "<test>" + testBeans[count].getMessage() + "</test>");
            assertEquals("Unexpected contents.",
                "<test>This is Cactus Test Message number "
                + (count + 1) + "</test>",
                tag.getBodyContent().getString());
            count++;
        } while (tag.doAfterBody() == tag.EVAL_BODY_BUFFERED);
        pageContext.popBody();
        assertEquals(3, count);
        assertEquals("Body Content not empty!", "",
            tag.getBodyContent().getString().trim());
    }
    public void endDoAfterBody(WebResponse response) {
        String resultString = "";
        for (int i = 0; i < 3; i++) {
            resultString += "<test>This is Cactus Test Message number "
                + (i + 1) + "</test>";
        }
        assertEquals(resultString, response.getText());
    }
    ...
}
```

Testes em aplicações Web: conclusões

- Aplicações Web são difíceis de testar porque dependem da comunicação com servlet containers
 - Stubs, proxies e APIs, que estendem ou cooperam com o JUnit, tornam o trabalho mais fácil
 - Neste bloco, conhecemos três soluções que facilitam testes de unidade, de integração e de caixa-preta em aplicações Web
- **Stubs** como **ServletUnit** permitem testar as **unidades** de código mesmo que um servidor não esteja presente
- **Proxies** como os "redirectors" do **Cactus** permitem testar a **integração** da aplicação com o container
- Uma **API**, como a fornecida pelo **HttpUnit** ajuda a testar o **funcionamento** da aplicação do ponto de vista do usuário

- [1] *Richard Hightower e Nicholas Lesiecki. Java Tools for eXtreme Programming. Wiley, 2002. Explora as ferramentas Ant, JUnit, Cactus, JUnitPerf, JMeter, HttpUnit usando estudo de caso com processo XP.*
- [2] *Apache Cactus User's Manual. Contém tutorial para instalação passo-a-passo.*
- [3] *Mackinnon, Freeman, Craig. Endo-testing with Mock Objects. <http://mockobjects.sourceforge.net/misc/mockobjects.pdf>. O autor apresenta técnicas para testes usando uma variação da técnica de stubs chamada de "mock objects".*

helder@argonavis.com.br

Selecione o link relativo a esta palestra no endereço

www.argonavis.com.br/xpbrasil2002

Recursos disponíveis no site:

- *Palestra completa em PDF*
- *Todo o código-fonte usado nos exemplos e demonstrações*
- *Instruções detalhadas sobre como rodar e instalar os exemplos*
- *Links para software utilizado e documentação*

Tutorial: Implementando eXtreme Programming em Java

XP Brasil 2002, São Paulo

© 2002, Helder da Rocha