14 Testes em Aplicações Web com Cactus

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Sobre este módulo

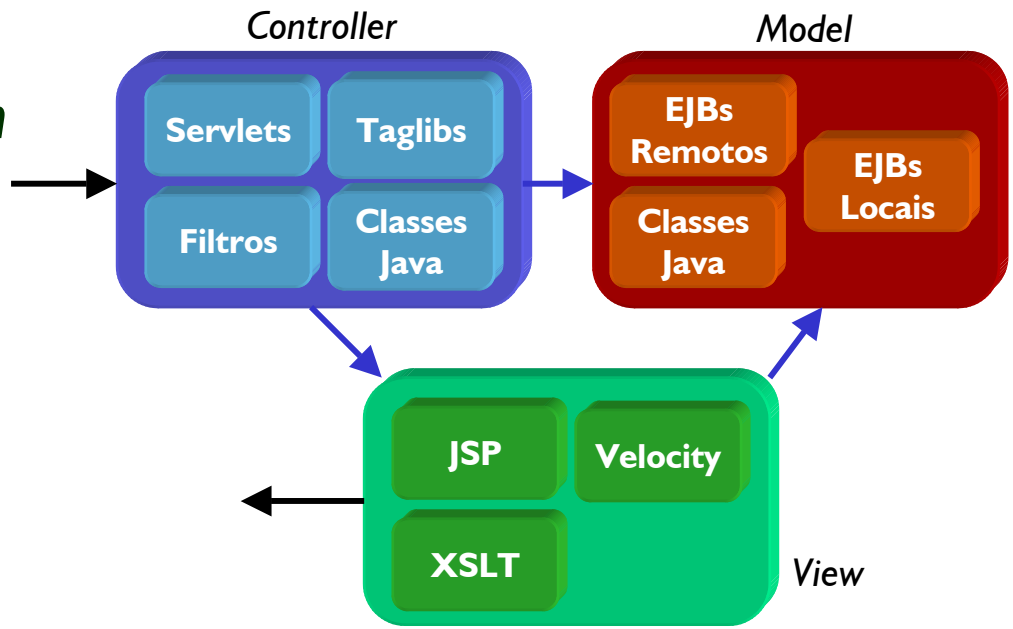
- Este módulo descreve um framework - o Jakarta Cactus - que pode ser utilizado para testar aplicações Web em Java
  - É uma extensão do JUnit
  - Testa os componentes dentro do container
- O assunto é extenso e implementar os primeiros testes pode ser trabalhoso, portanto, o assunto será apresentado superficialmente através de demonstrações
  - Execute as demonstrações usando o build.xml
  - Leia o README.txt para instruções mais detalhadas

O que é Cactus?

- É um framework que oferece facilidades para testar componentes J2EE
  - Componentes Web (Camada de Controle)
  - Camada EJB (Model) e cliente (View): indiretamente
- Produto Open Source do projeto Jakarta
  - Metas de curto prazo: testar componentes acima + 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 (Controladores)
  - JSPs (camada de apresentação: View, através de controladores)
  - EJB (Modelo 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

Ilustração: Manual do Cactus

Como funciona?

- 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

Arquitetura

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

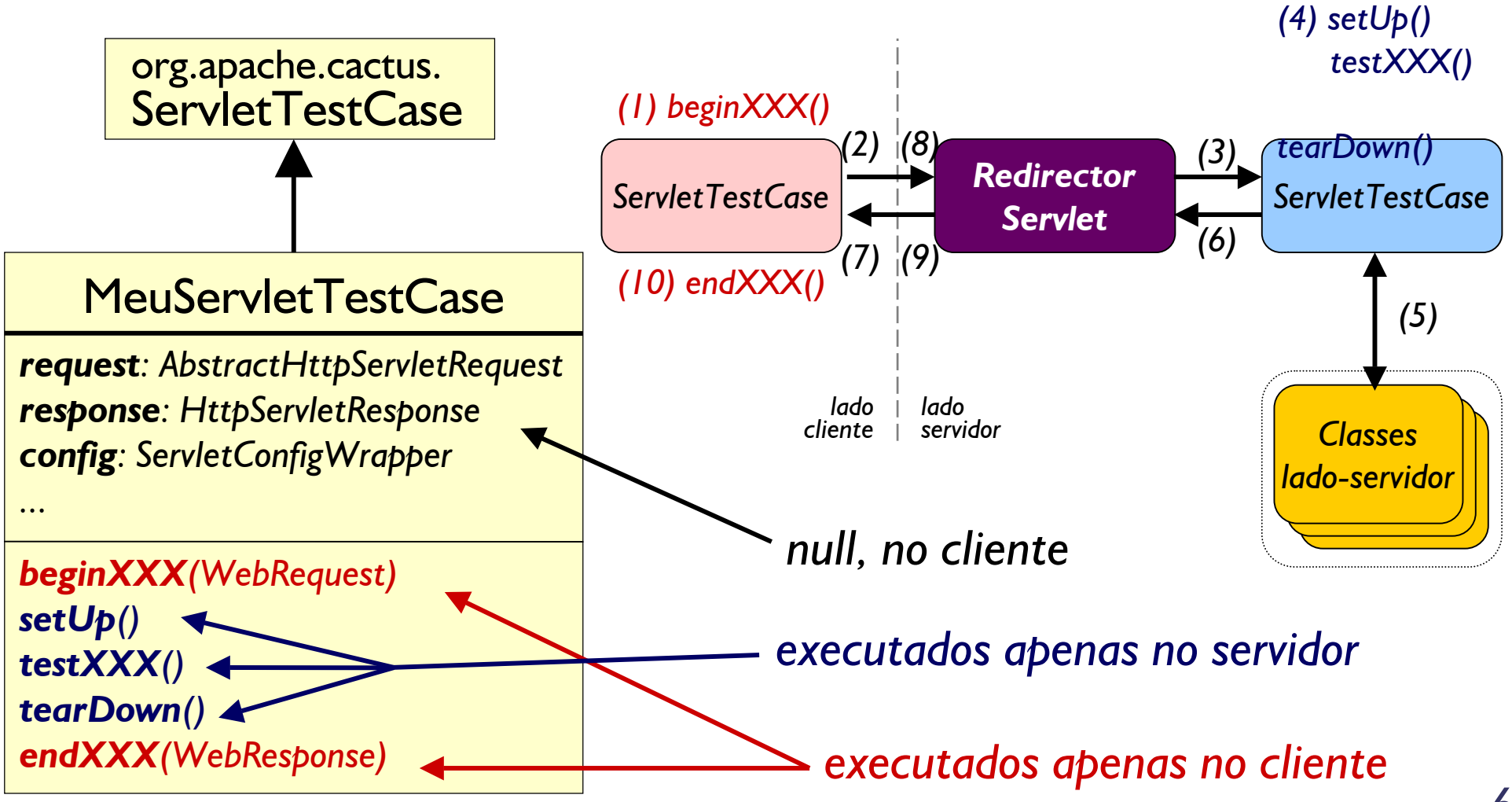
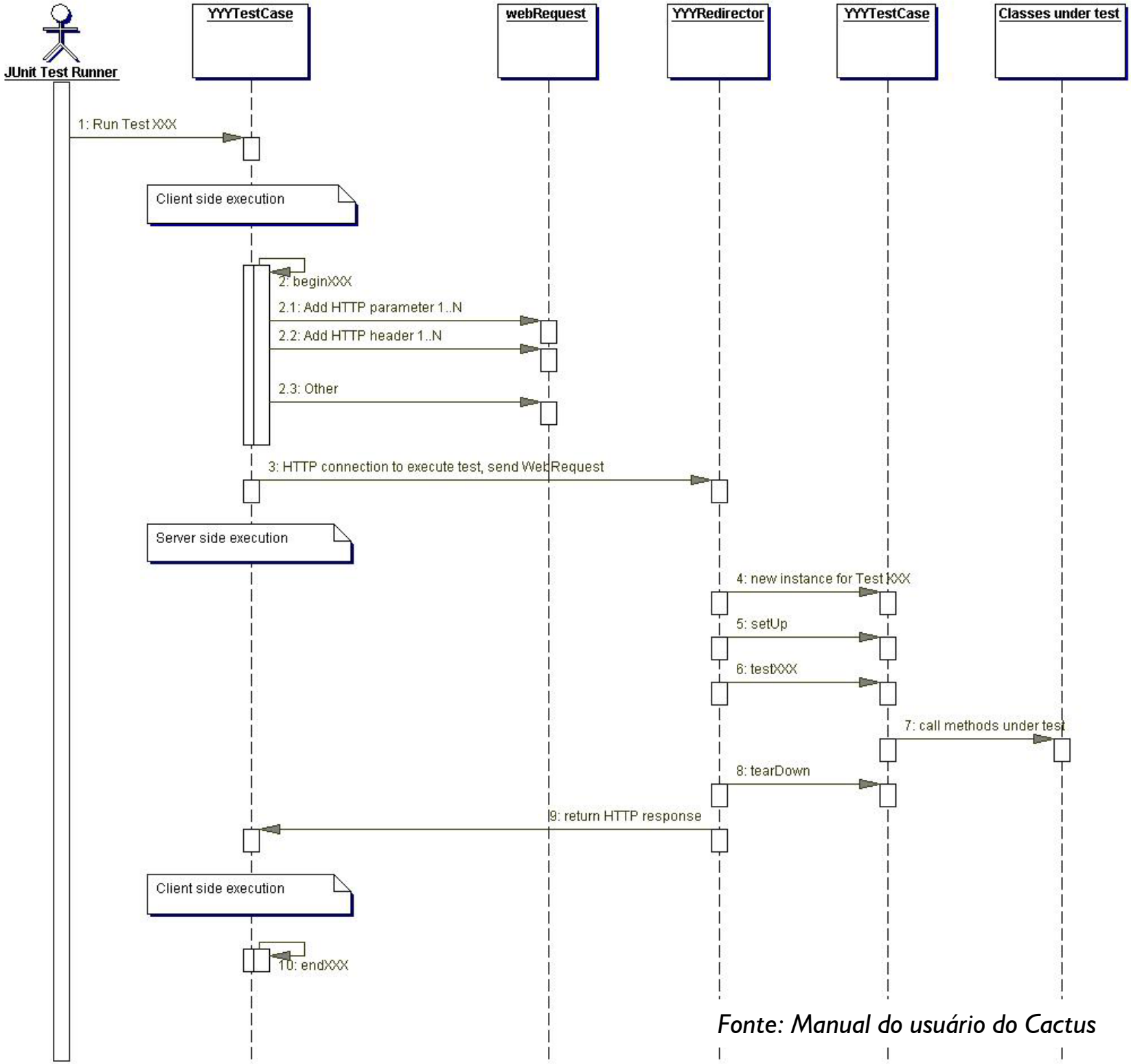


Diagrama UML



Fonte: Manual do usuário do Cactus

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

CactusDemo: servlet

- *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);
     writer.println("<html><body><table border='1'>");
    // (... loop for each parameter ...)
        if (useAllCaps()) {
          key = key.toUpperCase();
          val = val.toUpperCase();
        }
        str = "<tr><td><b>"+key+"</b></td><td>"+val+"</td></tr>";
        writer.println(str);
    // (...)
     writer.println("</table></body></html>");
}
```

(1) *Grava request em session*

(3) *Retorna* true *se <init-param> "ALL_CAPS" contiver "true"*

(2)

Trecho de MapperServlet.java

- *Escreveremos os testes para avaliar esses objetivos*

Fonte do exemplo: [1]

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)

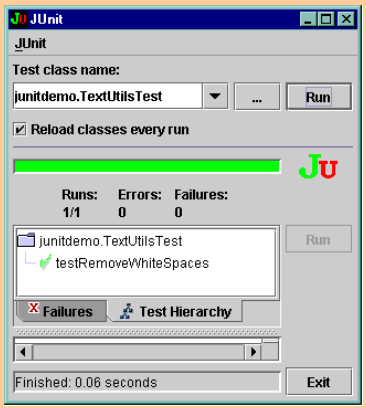
`beginDoGet(WebRequest req)`

- Grava parâmetro:

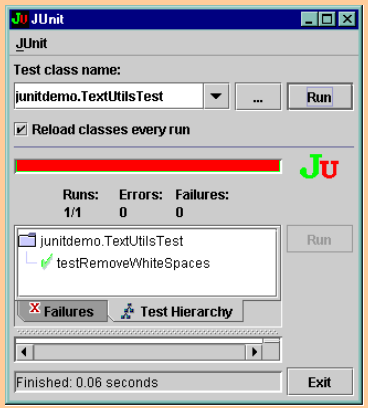
nome=`user`

value=`Jabberwock`

SUCCESS!!



FAIL!



falha local

falha remota

&

`endDoGet(WebResponse res)`

- Verifica se resposta contém

`USER</b></td><td>JABBERWOCK`

ReqInfo

2 conexões HTTP:

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

TestInfo

Output

Servidor (Tomcat)

`setUp()`

- Define init-params no objeto `config`

- Roda `init(config)`

`testDoGet()`

- Roda `doGet()`

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

`tearDown()`

HttpUnit

- 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

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*

HttpUnit com Cactus

Veja MapperServletTest2.java

- *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

hellojsp_2

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

Exercícios

- *1. Escreva testes JUnit (não Cactus) para a MessageBeanDAO.*
- *2. Escreva testes Cactus para testar um dos servlets que você desenvolveu neste curso (comece com uma simples)*

helder@acm.org

argonavis.com.br