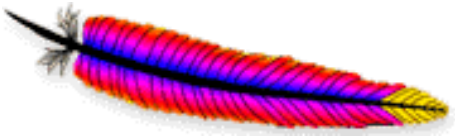
J820

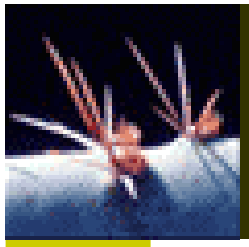
Testes em J2EE com

Jakarta



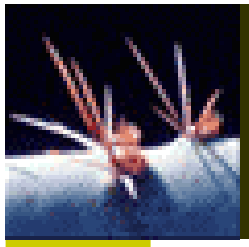
CACTUS

Helder da Rocha (helder@acm.org)



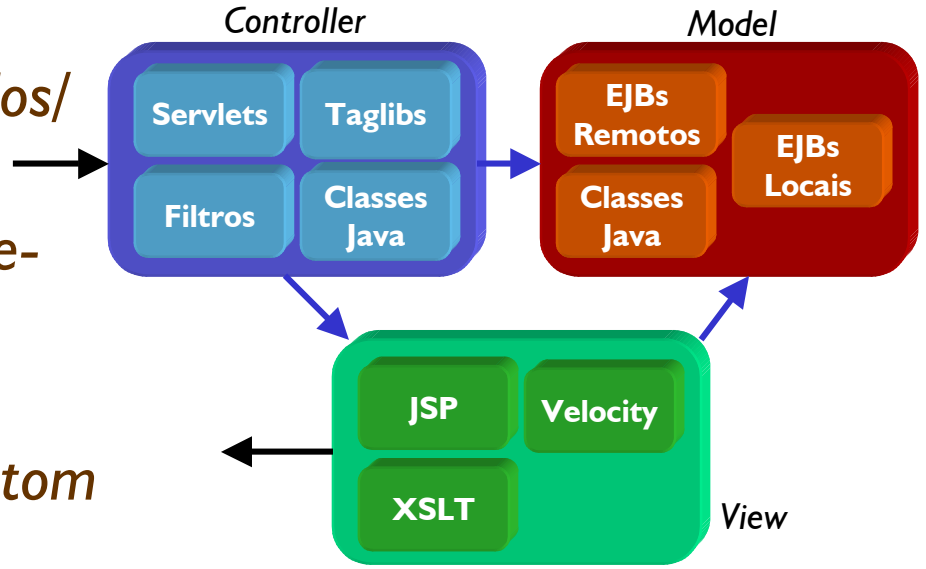
Cactus: framework para J2EE

- Testa componentes J2EE no próprio container
  - Componentes Web (Camada de Controle)
  - Camada EJB (Model) e cliente (View): indiretamente
- Cactus estende o JUnit framework
  - Execução dos testes é realizada de forma idêntica
  - TestCases são construídos sobre uma subclasse de junit.framework.TestCase
- Por que usar?
  - Para testes de integração (JUnit testa lógica local)
  - Aplicações J2EE possuem muitas dependências (é preciso simular a rede, o servidor, o EJB container, etc.)
  - Mock objects podem ficar muito complicados
  - Testar a integração com o servidor pode ser mais fácil



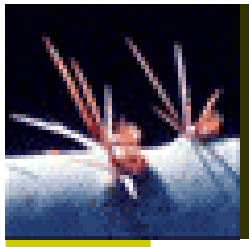
Para que serve?

- Use Cactus para testar a integração de aplicações Web (servlets, JSP, filtros) e aplicações EJB com o container
- Arquitetura do Cactus
  - **M**: EJB (Modelo de dados/ lógica de negócios)
  - **V**: JSPs (camada de apresentação: View, através de controladores)
  - **C**: Servlets, filtros e custom tags (Controladores)



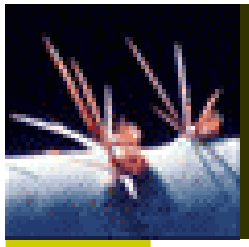
- Se não precisar testar a integração, use JUnit
  - Pode-se usar ambos em conjunto, uma vez que Cactus é extensão do JUnit e usa a mesma interface

Ilustração: Manual do Cactus



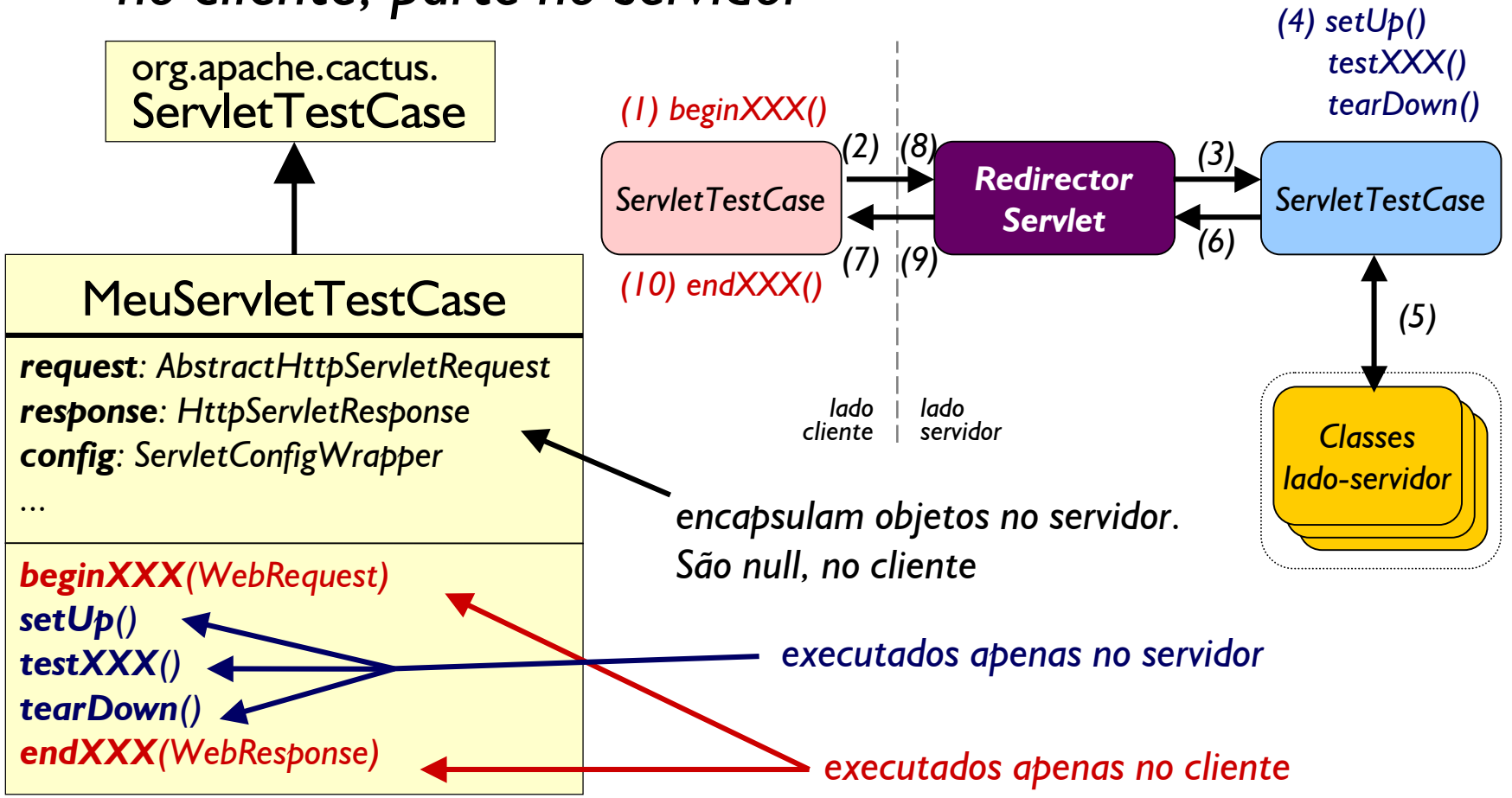
Como funciona?

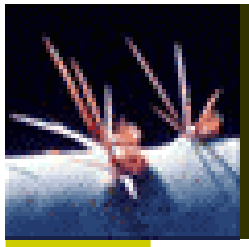
- Cactus utiliza os test cases simultaneamente no cliente e no servidor: duas cópias
  - Uma cópia é instanciada pelo servlet container
  - Outra cópia é instanciada pelo JUnit
- Comunicação com o servlet container é feita através de um proxy (Redirector)
  - JUnit envia requisições via HTTP para proxy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



Arquitetura

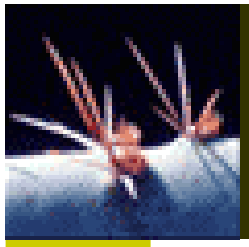
- *Parte da mesma classe (ServletTestCase) é executada no cliente, parte no servidor*





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.



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

```
public void doGet(...) throws IOException {
    Enumeration e = request.getParameterNames();
    HttpSession s = request.getSession();
    while (e.hasMoreElements()) {
       String param = (String)e.nextElement();
       s.setAttribute(param, request.getParameter(param));
    }
    writer.println("<html><body><table border='1'>");
    String str = "";
    while(e.hasMoreElements()) {
        String param = (String)e.nextElement();
        key = param.toUpperCase();
        val = request.getParameter(param).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) *Transforma nome e valor para caixa-alta*

(2) *Devolve tabela*

argonavis.com.br

Testes com o Cactus

MapperServletTest.java

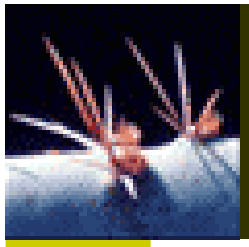
```
public class MeuServletTest extends ServletTestCase { (...)
  private MeuServlet servlet;
  public void beginDoGet(WebRequest cSideReq) {
     cSideReq.addParameter("user", "Jabberwock");
  }

  public void setUp() throws ServletException {
    servlet = new MeuServlet();
    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 endDoGet(WebResponse cSideResponse) {
    String str = cSideResponse.getText();
    assertTrue(str.indexOf("USER</b></td><td>JABBERWOCK") > -1);
  }
}
```

Simula servlet container

Verifica se parâmetro foi mapeado à sessão

Verifica se parâmetro aparece na tabela HTML e em maiúsculas

argonavis.com.br



Exemplo: funcionamento

argonavis.com.br

Cliente (JUnit)

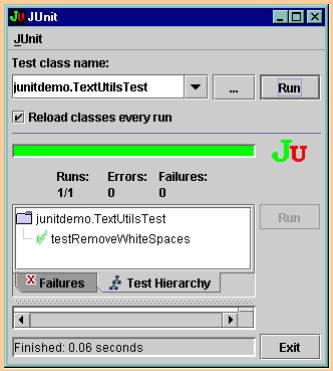
beginDoGet(WebRequest req)

- *Grava parâmetro:*

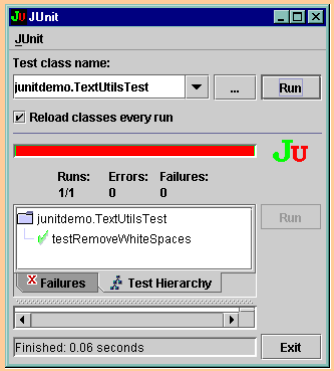
nome=`user`

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)

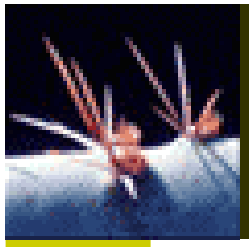
`setUp()`

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

`testDoGet()`

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

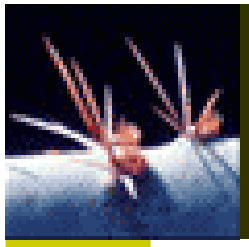
`tearDown()`



HttpUnit com Cactus

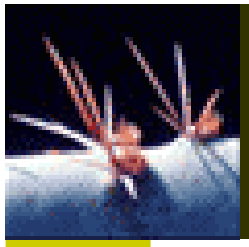
- *Troque o* **WebResponse** *em cada endXXX() por* **com.meterware.httpunit.WebResponse**

```
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!");
    }
}
```



Para testar EJBs

- Não é necessário usar Cactus. EJBs podem ser testados a partir do cliente, usando suas interfaces remotas
  - Teste de unidade simples usando JUnit
  - Possível para beans que tenham interfaces remotas
- Cactus é útil para testar EJBs através de suas interfaces locais, já que roda dentro do container
  - Ideal para testar Entity Beans que geralmente não possuem interfaces remotas
  - Permite também testar objetos que rodam no servidor e são inacessíveis aos clientes (DAOs, Service Locators)
- Cactus testa a integração de EJBs com o Container
  - Aplicação usa referências java:comp/env para ter acesso ao bean (precisam ser declaradas em web.xml)



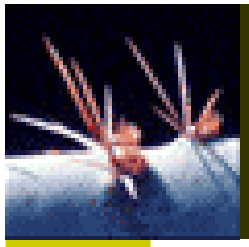
Componentes EJB

- *Servlets e JSPs podem se comunicar com EJBs da aplicação declarando uma referência associada ao bean chamado*
  - *A referência deve informar o tipo do bean (Session, Entity ou MessageDriven e suas interfaces remota e home.*

```
<ejb-local-ref>
   <description>Cruise ship cabin</description>
   <ejb-ref-name>ejb/Cabin</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <local-home>com.titan.cabin.CabinHome</local-home>
   <local>com.titan.cabin.Cabin</local>
</ejb-local-ref>
```

- *Componentes EJB locais não são objetos remotos RMI-IIOP. Para obter sua referência no ServletTestCase use JNDI (javax.naming):*

```
InitialContext initCtx = new InitialContext();
Object ref = initCtx.lookup("java:comp/env/ejb/CabinHome");
CabinHome home = (CabinHome) ref;
```



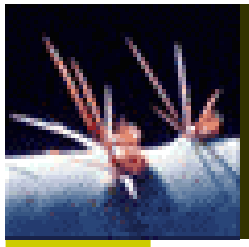
Testes em EJBs locais

- *Uma vez obtida a referência local para o EJB, seus métodos podem ser testados dentro do container da maneira convencional*

```
Cabin cabin = home.findByPrimaryKey("311");
String number = cabin.getNumber();
assertEquals("311", number);
```

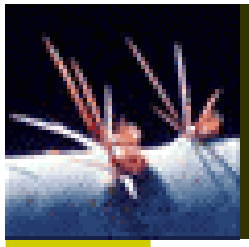
- *Não interessa se a implementação de persistência usada é BMP ou CMP*
- *Uma vantagem do Cactus é a possibilidade de testar relacionamentos CMR, impossíveis remotamente*

```
Collection cabins =
  home.findByShip("Titanic");
...
```



Exercícios

- 1. Execute os exemplos
- 2. Escreva testes para testar a aplicação Web fornecida
  - O ambiente está configurado (use o Ant)
  - Guie-se pelos comentários no código
  - A aplicação recebe parâmetros e grava os dados no contexto. No final retorna uma página HTML
  - Rode a aplicação.
  - Escreva testes para verificar sua interface e para testar se os dados passados como parâmetros foram gravados no contexto.
  - Use Cactus apenas onde for necessário (use test cases normais do JUnit para outros testes)



Fontes

[1] *Richard Hightower e Nicholas Lesiecki. Java Tools for eXtreme Programming. Wiley, 2002.*

[2] *Apache Cactus User's Manual.*

Curso J820
Produtividade e Qualidade em Java: Ferramentas e Metodologias

Revisão 1.1

(helder@acm.org)

argonavis.com.br