Testování aplikací, jednotkové testy, JUnit

Obsah

Testování jednotek nástrojem junit 1
- Proč testování? 1
- V čem spočívá testování? 2
- Co je testování jednotek? 2
- Jak probíhá v Javě 2
- Struktura testu v junit 3
- Postup při práci s JUnit 3

Motivace 4
- Pozice testování v soudobém vývoji SW 4
- Testování Java EE aplikací 4
- Motivace 4
- Specifikace vs. implementace – zdroj chyb 4
- Zásady (Scott Ambler) 4

Java Enterprise Edition 5
- Architektura 5
- Typická Java EE aplikace 5
- Jak a co testovat? 5

Techniky a nástroje testování 5
- Obecně použitelné - junit 5
- Integrace testování do vývoje 6
- Testování a správa (verzí) zdrojů 6
- Řízení testů 6
- Protokolování testů 7

Testování vrstev aplikace 8
- Testy datové vrstvy 8
- Testy aplikační vrstvy 8
- Integrační testy 8
- Testy prezentační vrstvy – webové rozhraní 9
- Testy prezentační vrstvy – desktopové aplikace/GUI 10

Mock-objekty 10
- Motivace 10
- Mockrunner 10

Zátěžové testování 11
- Apache JMeter 11
- JMeter - prostředí 12

Testování jednotek nástrojem junit

Proč testování?

- Teprve **otestovaný a uživatelem akceptovaný** program je v pořádku.

- Testování softwaru je **klíčová součást vývoje** - v podstatě ta **nejdůležitější**.
- Některé metodiky vývoje se o testování zásadně **opírají** (agilní metodiky [http://agilemanifesto.org/principles.html] obecně, konkrétně pak typicky Extreme Programming [http://www.extremeprogramming.org/])
- Některé jsou jím doslova **řízené** - Test-Driven Development [http://www.testdriven.com/] (vývoj řízený testy)

V čem spočívá testování?

- Testování softwaru je mnohaúrovňový, kontinuálně probíhající proces (v podstatě po celou dobu životního cyklu).
- Nemyslíme tím obvykle *formální verifikaci* (dokazování správnosti) programů, ale "pouze" testování **vybraných konkrétních případů** běhu (*case-based testing*)
- Podle úrovně pohledu rozlišujeme testy:

jednotek	elementů programu, někdy i vytržených z kontextu - např. jednotlivých tříd a jejich souhry
integrační	určené k otestování právě integrované části kódu (např. po jeho změnách) do většího celku
akceptační	ověřují, zda je SW přijatelný pro uživatele/klienta

Co je testování jednotek?

Testování jednotek (**unit testing**) je první fází každého testovacího procesu.

- testuje jednotlivé elementární části kódu - v případě objektového světa třídy;
- často probíhá formou "black-box" testování (tzn. jen pohled z venku, tedy na veřejné vlastnosti - typicky metody), ale
- může se kombinovat s nahlížením dovnitř objektů - např. na "package-local" metody a atributy, může testovat "package-local" třídy a rozhraní

Jak probíhá v Javě

Pro testování jednotek obvykle používáme vhodný **testovací rámec** (i když by šlo dělat i bez něj, vlastními ad-hoc vytvořenými nástroji)

- v Javě se nejčastěji používá volně dostupný balík junit [http://junit.org]
- existují jeho porty pro jiné jazyky/prostředí (C, C#, PHP, Perl,...)
- pro Javu existují také koncepčně velmi podobné alternativy (
- starší verze (junit 3.x -- ten používáme) nepracovaly s tzv. anotacemi, novější již ano (junit 4.x)
- novější mají též "učesanější" pojmenovávání balíků a tříd - např. `org.junit.Assert`

Struktura testu v junit

Jednotkové testy v junit mají v zásadě tuto strukturu:

- elementárním testem je testovací metoda - její název začíná předponou *test* nebo je opatřena příslušnou anotací (junit 4.x) nebo obojí
- pojmem "vykonání tohoto elementárního testu" se vlastně rozumí:
  1. konstrukce objektu testu - **instance testovací třídy**
  2. inicializace testu metodou **setUp** - ta obvykle vytvoří tzv. testovací **přípravek** (fixture)
  3. proběhne příslušná (jedna z) testovací **metoda**
  4. test je "uklizen" voláním metody **tearDown**
- Inicializační a uklizecí metody, stejně jako metody testu mají napevno předepsané názvy nebo přípony jen pro junit do verzí 3.x, od 4.x lze označit anotacemi - ale pro přehlednost se jmenné konvence drží nadále.

Postup při práci s `JUnit [http://www.google.com/search?q=JUnit]` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=JUnit]

Uvědomit si, že žádný nástroj za nás nevymyslí, JAK máme své třídy testovat. Pouze nám napomůže ke snadnějšímu sestavení a spuštění testu.

Postup:

1. Stáhnout si z http://junit.org poslední (stačí binární) distribuci testovacího prostředí.
2. Nainstalovat `JUnit [http://www.google.com/search?q=JUnit]` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=JUnit] (stačí rozbalit do samostatného adresáře).
3. Napsat testovací třídu/třídy - buďto implementují rozhraní `junit.framework.Test [http://www.google.com/search?q=junit.framework.Test]` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=junit.framework.Test] nebo obvykleji rovnou rozšiřují třídu `junit.framework.TestCase [http://www.google.com/search?q=junit.framework.TestCase]` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=junit.framework.TestCase]
4. Testovací třída obsahuje metodu na nastavení testu (setUp), testovací metody (test*Neco*) a úklidovou metodu (tearDown).
5. Testovací třídu spustit v prostředí (řádkovém nebo GUI) - `junit.textui.TestRunner [http://www.google.com/search?q=junit.textui.TestRunner]` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=junit.textui.TestRunner],

`junit.swingui.TestRunner` [http://www.google.com/search?q=junit.swingui.TestRunner] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=junit.swingui.TestRunner]...

6. Testovač zobrazí, které testovací metody případně selhaly.

Motivace

Pozice testování v soudobém vývoji SW

Role testování narůstá, testování podstatnou standardní součástí vývoje:

- Agilní metodiky (např. Extreme Programming), Test-driven development…

Testování Java EE aplikací

- Mocné, ale složité aplikační prostředí
- Aplikace typicky vícevláknové, klient-server
- Soudobé metodiky jsou přitom často založeny na testování…

Motivace

- Jasná motivace: SW chyby mohou mít a mají katastrofální následky…
- a přinejmenším zdražují vývoj – čím později je chyba odhalena, tím jsou náklady větší
- Testování patří k „dobrému tónu“ i u open-source vývoje (nebo hlavně tam?)

Specifikace vs. implementace – zdroj chyb

- Nesoulad mezi specifikací a implementací
- Nedostatečná/neúplná/chybná specifikace
- …systém pak specifikaci splňuje a přesto nevyhovuje…

Zásady (Scott Ambler)

- testovat během celého životního cyklu projektu,
- vyvíjet testy ještě před samotným kódem,
- (kontinuálně) testovat všechny artefakty programu,
- testováním odhalovat příčiny chyb a nepřekrývat je,
- při testování používat přitom jednoduché a efektivní nástroje a
- testování zahrnout po všech stránkách do vývoje

Java Enterprise Edition

Architektura

- Platforma pro rozsáhlé, komponentně orientované programové systémy schopné běhu aplikačních serverů.
- Prostředí = middlewarové zázemí pro vývoj, nasazení, konfiguraci, běh, vzájemnou komunikaci a sledování.
- Systémy mají řadu vrstev a jsou často distribuované mezi více fyzických počítačů.

Typická Java EE aplikace

Typická Java EE aplikace s Enterprise JavaBeans (EJB):

- klientské webové rozhraní,
- to komunikuje se servlety a stránkami JSP, JSF...
- aplikační logika realizovaná Session Beans (mezi tím příp. fasáda)
- komponenty Entity Beans s perzistencí zajištěnou relační databází.
- Namísto EJB se dnes často používá objektově-relační mapování běžných (POJO) objektů,
- pro řízení na vyšších vrstvách se využívají webové rámce.

Jak a co testovat?

Type 1: code logic unit testing	Probably the best strategy for these tests is to use a Mock Objects type framework.
Type 2: integration unit testing	These tests will exercise the interactions with the container. (Cactus)
Type 3: functional unit testing	These unit tests will let you test the returned values from your server code. This is for example HttpUnit (Note that HttpUnit also performs standard functional testing - as opposed to functional unit testing -, which let you test full use cases - a login use case for example, which is comprised of several requests/responses).

Techniky a nástroje testování

Obecně použitelné - junit

Nástroj JUnit, http://junit.org

Příklad testu:

```
public class StackTest extends JUnit.framework.TestCase {
```

```
    private Stack st;
    public StackTest(String testCaseName) {
        super(testCaseName);
    }
    // vytvoří přípravek, nastaví prostředí každého testu
    public void setUp() {
        st = new Stack(10);
    }
    // testuje prázdnost právě vytvořeného zásobníku
    public void testEmptyAfterCreation() {
      assertTrue("Stack should be empty after creation.",
                 st.isEmpty());
    }
    // testuje neprázdnost zásobníku po vložení prvku
    public void testPushPopEquals() {
      st.push("something");
      assertEquals("What was pushed, must be popped...",
                 "something", st.pop());
    }
    ...
    // uklidí po testu, je-li třeba
    public void tearDown() {  }
}
```

Integrace testování do vývoje

Integrovaná do všech reálně používaných vývojových nástrojů:

- Maven, Ant,
- IDE (NetBeans, Eclipse, IDEA)
- a dokonce i výuková prostředí – BlueJ.

Dnes se vždy předpokládá, že ke KAŽDÉMU netriviálnímu projektu existují testy.

Testování a správa (verzí) zdrojů

- Systémy správy zdrojových kódů zajišťují centralizovanou evt. distribuovanou správu s podporou verzování (CVS, Subversion)
- Při neúspěšnosti integračních testů se tak můžeme v repozitáři vrátit ke starší verzi určitého modulu a pokusit se chybu najít.
- Systémy evidence chyb (např. BugZilla)

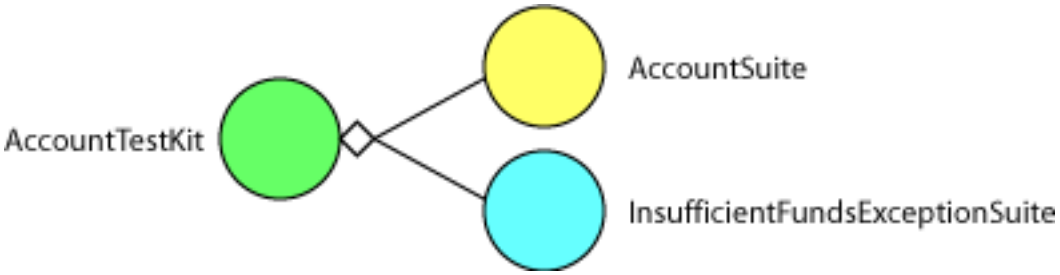
Řízení testů

V netriviálních projektech, tam, kde se integruje např. jednou za den, není možné pokaždé spouštět všechny testy.

Pokročilé nástroje je umějí

- uskupovat
- parametrizovat
- rozběhnout jen testy, které předtím neprošly
- pohodlně spustit např. všechny testy v jednom balíku

Např. *Artima SuiteRunner*:



Protokolování testů

Např. Maven pracuje hojně s protokoly (reports) vč. testovacích.

Většina testovacích nástrojů umí kromě "transientního" zobrazení průběhu výsledky testu zaznamenat.

Elementárně např. ve dvojici nástrojů - Ant tasks - `junit` (otestuje), `junitreport` [http://www.google.com/search?q=] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=] (zaformátuje protokoly):

Otestování:

```
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <pathelement location="${build.tests}"/>
    <pathelement path="${java.class.path}"/>
  </classpath>

  <formatter type="plain"/>

  <test name="my.test.TestCase" haltonfailure="no" outfile="result">
    <formatter type="xml"/>
  </test>

  <batchtest fork="yes" todir="${reports.tests}">
    <fileset dir="${src.tests}">
      <include name="**/*Test*.java"/>
      <exclude name="**/AllTests.java"/>
    </fileset>
  </batchtest>
</junit>
```

Formátování reportu:

```
<junitreport todir="./reports">
```

```
  <fileset dir="./reports">
    <include name="TEST-*.xml"/>
  </fileset>
  <report format="frames" todir="./report/html"/>
</junitreport>
```

Testování vrstev aplikace

Testy datové vrstvy

Co je u DB aplikací třeba?

- uvedení prostředí před testem do vhodného stavu
- po testu “uklidit”

DBUnit – rozšíření JUnit (http://dbunit.sf.net)

- dokáže podle definičního souboru vytvořit tabulky,
- z XML či jiných zdrojů do nich načíst data
- a spustit testy.

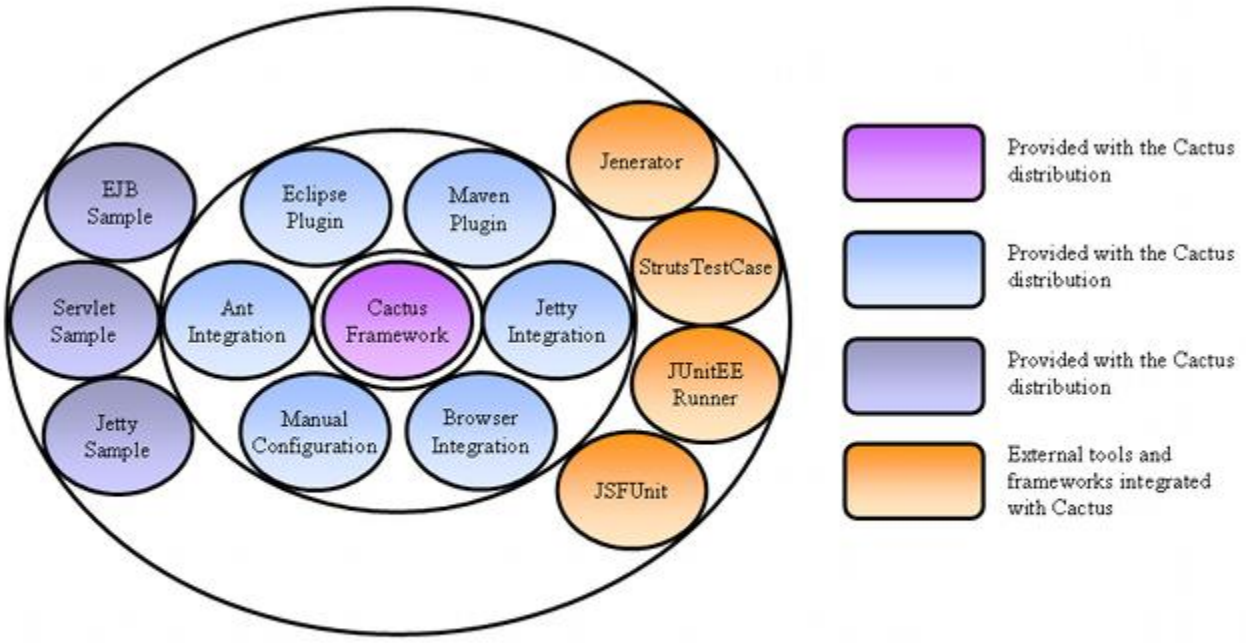
Testy aplikační vrstvy

Používá se JUnit a nadstavby, resp. obdobné nástroje:

TestNG	http://testng.org
	moderní, na anotacích založený systém, lepší než junit díky jemnější organizovatelnosti testů (skupiny testů, parametry, protokolování...)
djunit	http://www.fi.muni.cz/~tomp/djunit
	rozšíření junit s možností "krmit" testy daty ze souborů, proudů, zdrojů (resources)
junit v4	nová verze JUnit s podporou anotacemi
Artima SuiteRunner	http://www.artima.com/suiterunner/index.html
	Reporter: Collect and present test results in a highly customizable way to the user. Examples are text, graphics, web pages, database output, CSV files, XML, and email alerts. Runpath: Load classes for your tests from anywhere with this easy to configure list filenames, directory paths, and/or URLs. Recipe File: Capture and save the properties of a run of a particular suite of tests in a file for easy reuse.

Integrační testy

Systém Apache Cactus, http://jakarta.apache.org/cactus/



The Cactus Ecosystem

Architektura Cactus:

The Cactus Framework	This is the heart of Cactus. It is the engine that provides the API to write Cactus tests.
The Cactus Integration Modules	They are front ends and frameworks that provide easy ways of using the Cactus Framework (Ant scripts, Eclipse plugin, Maven plugin, ...).

Testy prezentační vrstvy – webové rozhraní

Nástroj JWebUnit umožní virtuálně "klikat" po webové aplikaci a srovnávat výstupy od serveru s očekávanými.

```
public class JWebUnitSearchExample extends WebTestCase {
   ...
   public void setUp() {
      getTestContext().setBaseUrl("http://www.google.com");
   }
   public void testSearch() {
      beginAt("/"); // na kterém URL začít
      setFormElement("q", "httpunit"); // co na stránce vybrat
      submit("btnG"); // jaké tlačítko stisknout
      clickLinkWithText("HttpUnit"); // kam kliknout
      assertTitleEquals("HttpUnit"); // co má od serveru přijít
      assertLinkPresentWithText("User's Manual");
   }
```

```
}
```

Dalšími příklady jsou:

- HttpUnit
- příp. komplexnější, které webovou vrstvu otestují také - Cactus nebo MockRunner

Testy prezentační vrstvy – desktopové aplikace/GUI

Nástroj Marathon umožní virtuálně "klikat" po GUI aplikaci a srovnávat výstupy od serveru s očekávanými.

- obecně se testuje těžko…
- většinou na principu zachycení či naskriptování uživatelských akcí nad GUI
- na jednodušší v Javě lze použít "robota" pro javové GUI

Příklad v JWebUnit:

```
public class JWebUnitSearchExample extends WebTestCase {
   ...
   public void setUp() {
      getTestContext().setBaseUrl("http://www.google.com");
   }
   public void testSearch() {
      beginAt("/"); // na kterém URL začít
      setFormElement("q", "httpunit"); // co na stránce vybrat
      submit("btnG"); // jaké tlačítko stisknout
      clickLinkWithText("HttpUnit"); // kam kliknout
      assertTitleEquals("HttpUnit"); // co má od serveru přijít
      assertLinkPresentWithText("User's Manual");
   }
}
```

Mock-objekty

Motivace

Testování Java EE aplikací je náročné, protože:

- reálné prostředí běhu JavaEE aplikací je příliš složité, dlouho startuje…
- Často stačí testovat nad prostředím nasimulovaným pomocí mock-objektů (nepravých, zástupných objektů)
- Je-li vše OK, následně se testuje v "ostrém" prostředí jinými nástroji (např. Cactus, JWebUnit, DBUnit...)

Mockrunner

Mockrunner umí pomocí mock-objektů testovat JDBC, webové (servletové) aplikace i EJB.

Příklad:

```
public class RedirectServletTest extends BasicServletTestCaseAdapter {
    protected void setUp() throws Exception {
        super.setUp();
        createServlet(RedirectServlet.class);
    }
    public void testServletOutputAsXML() throws Exception {
        addRequestParameter("redirecturl", "http://www.mockrunner.com");
        doPost();
        Element root = getOutputAsJDOMDocument().getRootElement();
        assertEquals("html", root.getName());
        Element head = root.getChild("head");
        Element meta = head.getChild("meta");
        assertEquals("refresh", meta.getAttributeValue("http-equiv"));
        assertEquals("0;URL=http://www.mockrunner.com",
                     meta.getAttributeValue("content"));
    }
}
```

Zátěžové testování

Apache JMeter

Apache JMeter [http://jakarta.apache.org/jmeter/] 100% pure Java desktop application designed to load test functional behavior and measure performance.

Nejen pro webové aplikace, ale i např. databázové (SQL) dotazy, JMS komunikaci, FTP přenosy...

- Can load and performance test HTTP and FTP servers as well as arbitrary database queries (via JDBC)
- Complete portability and 100% Java purity
- Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by seperate thread groups.
- Caching and offline analysis/replaying of test results.

Highly Extensible: o Pluggable Samplers allow unlimited testing capabilities. o Several load statistics may be choosen with pluggable timers . o Data analysis and visualization plugins allow great extendibility as well as personalization. o Functions (which include JavaScript) can be used to provide dynamic input to a test o Scriptable Samplers (BeanShell is supported in version 1.9.2 and above)

JMeter - prostředí