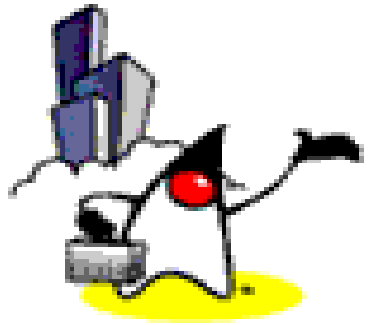
Az Apache Wicket keretrendszer

JUM – 2007. július 11.

Cserép János

cserepj@metaprime.hu

Tartalom



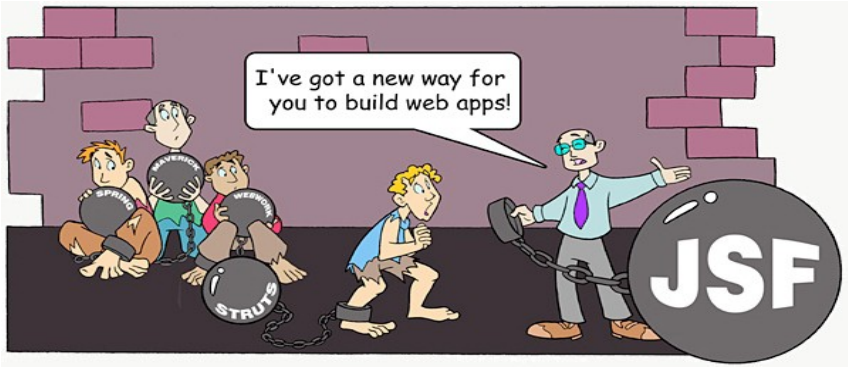
- Bemutatkozás
- Java Web keretrendszerek
- Wicket
  - Bevezetés, alapkoncepciók
  - Application, Session, Component, és társaik
  - Model típusok
  - AJAX
  - Unit Testing
- Kérdések, válaszok

Bemutatkozás

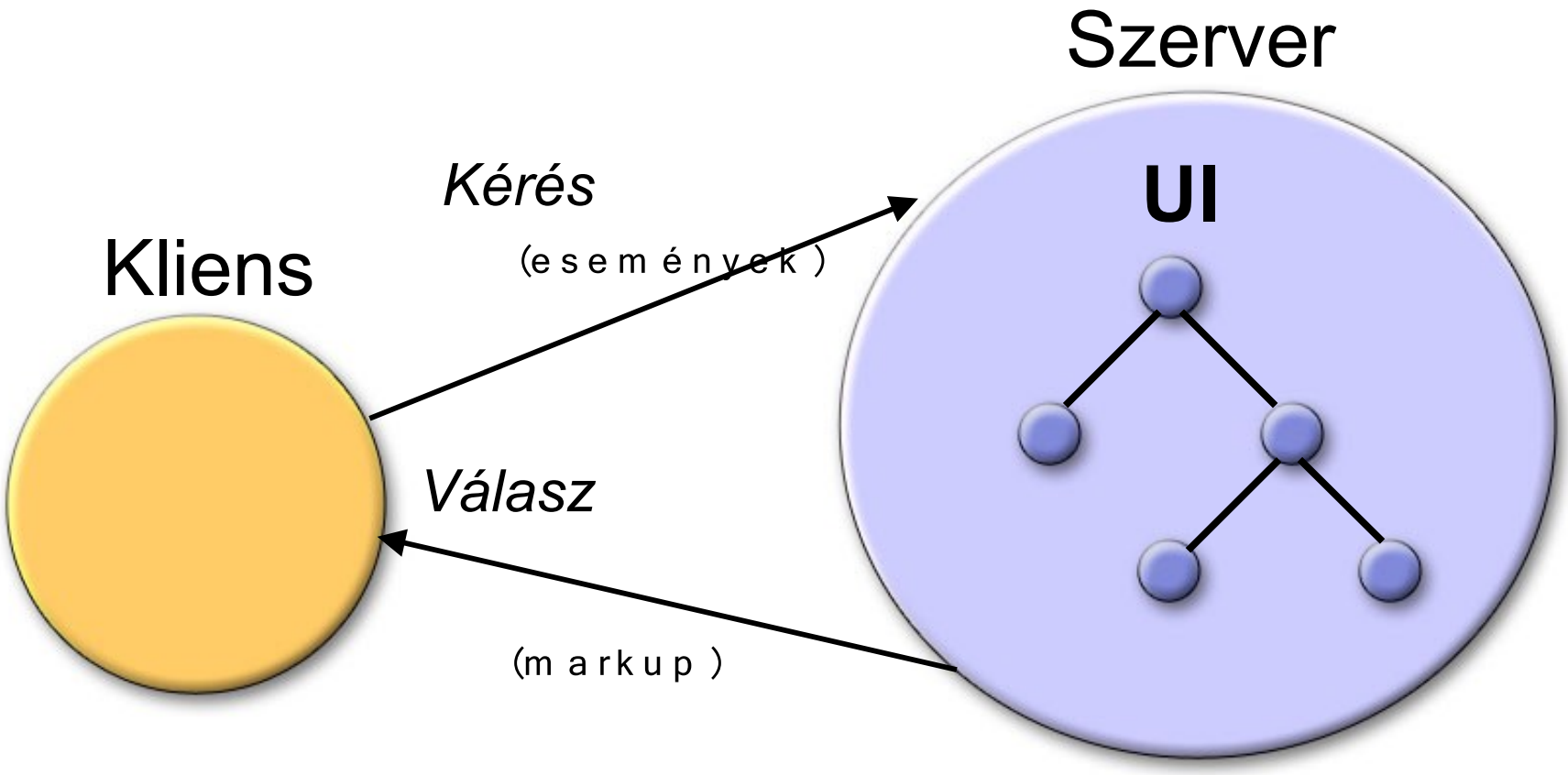
- 9 éve foglalkozom webprogramozással
- 2002 óta Java-val és EE-vel
- 2005-ben találtam rá a Wicket-re
- 3 + 1 Wicket projekt az elmúlt másfél évben
  - Vezető fejlesztőként:
    - Egy 15 képernyőből álló Struts alkalmazás komplett újraírása 5 hét alatt 2 – a projekt közben betanuló – emberrel
  - Tudásátadás wicket-tel ismerkedő fejlesztőknek
  - “Pet project”:
    - Web alapú CMS fejlesztése Java-ban, Wicket-tel, JPA-val: http://www.szeretgom.hu

Java Web keretrendszerek

- Servlet
- Servlet + Template (velocity, webmacro)
- JSP Model 1
- Model 2 (MVC) – Kérés alapú
  - Struts 1, Webworks, Struts 2, shale
- Komponens alapú keretrendszerek
  - JSF, Tapestry, RIFE, Echo2, **Wicket**



Komponens alapú keretrendszer



Wicket

- Nyílt forrású (Apache 2 licence)
- 2004-ben indult a fejlesztése (1.0)
- Június óta teljes jogú Apache projekt (1.3)
- Komponens alapú keretrendszer
- A Markup és a kód teljes szétválasztása
  - Nincs üzleti logika a markupban. Valid X(HT)ML.
- Beépített AJAX támogatás, amihez javascript tudás nem szükséges
- Nincsen semmilyen XML konfigurációs állomány:)
  - Csupán a web.xml, egyetlen Servlet (1.2-ig) vagy Filter (1.3-tól) definiálása

Wicket célok

- Objektumorientált fejlesztés a web rétegben
  - Struktúra vs. műveletek
- Komponensek könnyű újrahasznosíthatósága
- Feladatok szétválasztása
  - HTML a prezentáció
  - Java az állapot, modell, felületi logika
- Produktivitás, karbantarthatóság (tesztelhetőség)

Állapotot tárolni?

- A Wicket állapotot tároló (stateful) komponenseket definiál
- REST vs Állapottárolás
  - Állapotmentes szolgáltatási réteg: “vissza a procedurális programozáshoz”
  - Lassabb implementáció
  - “CPU és sávszélesség” vs “memória”
  - Tényleg minden oldal csak egyvalamit csinál?

Állapottal rendelkező komponensek

- Minden renderelt oldal mögött a szerveren egy komponens fa “él” a felhasználó session-jében
  - A komponensek verziózottak, korábbi komponensfák elérhetőek (1.3: nem feltétlenül a memóriából)
  - Elegáns programozási modell
- Wicket-ben szinte minden komponens
  - A markupban wicket:id attribútum jelöli a helyüket
  - Java kódban id property a konstruktorban

HTML

```
<span wicket:id=”name”></span>
```

Java

```
add(new Label(“name”));
```

Újrahasznosítás

- A komponensek bepakolhatók egy .jar-ba
- A szükséges resource-ok szintén
  - HTML, CSS, képek, javascript
- .jar a classpath-ba
- A “gyári” komponenseken túl több tucat alprojekt
  - wicket-gmap
  - wicket-dojo
  - wicket-scriptacolus
- Wicket-ben alapvetően nem oldalakat fejlesztünk
  - Hanem komponenseket...

Wicket alaptulajdonságok

- Markup inheritance
- Ahol az esemény kiváltódik – ott kezeljük le

```
add(new Button("print",
  new StringResourceModel("activityreport.print", this, null)) {
    @Override
    public void onSubmit() { ... }
```

- Beépített lokalizációs lehetőségek
  - .properties és .html külön-külön lokalizálható

```
<wicket:message key=”person.name”>
```

- Annotációkkal komponens szinten (és nem feltétlenül URL-enként) leírhatóak az authorizációs igények, pluggable az authorizáció implementáció

Egy Wicket alkalmazás

```
package wicket.examples.helloworld;
import wicket.protocol.http.WebApplication;
public class HelloWorldApplication extends WebApplication
{
    public HelloWorldApplication()
    {
    }
    public Class getHomePage()
    {
        return HelloWorld.class;
    }
}
```

HelloWorld.java

```
package wicket.examples.helloworld;


import wicket.markup.html.WebPage;
import wicket.markup.html.basic.Label;


public class HelloWorld extends WebPage
{
    public HelloWorld()
    {
        add(new Label("message", "Hello World!"));
    }
}
```

HelloWorld.html

```
<html>
<body>
    <span wicket:id="message">Message goes here</span>
</body>
</html>
```

- Ugyanabban a könyvtárban (package-ben) mint a HelloWorld.java
- Ez általánosságban is igaz minden markuppal rendelkező komponensre (Page, Panel, Border)
- Ha egy komponenshez nincs .html, akkor a superclass-ét fogja használni

Application (WebApplication)

- 1 db példányosodik belőle, nem thread safe
- Application.get()
- Érdekesebb metódusok:
  - init()
  - getSessionFactory(), newSession() (WebApplication)
  - getHomePage()
  - get****Settings()
  - mountBookmarkablePage()
  - mountSharedResource()

Session (WebSession)

- 1 felhasználó, 1 Session objektum, thread safe
  - Saját céljainkra általánan létrehozunk egy subclass-t a WebSession-ből, és erősen típusosan használjuk a benne tárolt objektumokat (nincs getAttributes()/setAttributes())
- Page-et érintő kérésfeldolgozás a Session-re synchronized() blokkban fut
- Érdekes metódusok:
  - dirty()
  - error(), warn()
  - getLocale(), setLocale()

Component

- Érdekes metódusok
  - getParent()
  - getModel()/setModel()/initModel()
  - isVisible()/setVisible()
  - getPage(), getSession(), getApplication()
  - findPage(), findParent()
  - getLocale(), getLocalizer()
  - getStyle()
  - add(IBehaviour m)
  - debug(String), info(String), warn(String), error(String) és fatal(String)
  - setResponsePage()

MarkupContainer extends Component

- Gyermek komponenseket tartalmazó komponensek őse
- Metódusok:
  - getChildren()
  - add(Component c), replace(Component c), remove(Component c)
  - visitChildren(Component.IVisitor v)

Page extends MarkupContainer

- WebPage-ből származtatunk
  - Konstruktor:
    - () – ha public, akkor bookmarkolható a Page
    - (PageParameters param) – GET paraméterek átvétele
- Page-ek állnak a felépített komponensobjektum hierarchia tetején
- A Wicket framework “dolga” egy bejövő URL-hez a megfelelő Page-et, azon belül a megfelelő akciót, eseménykezelőt meghívni és a kérést lekezelni

WebMarkupContainerWithAssociatedMarkup

- Saját .html template fájlal rendelkező komponensek
- Border

```
<wicket:border>
  First <wicket:body/> Last
</wicket:border>
```

- Panel

```
<wicket:panel>
  <span wicket:id="mylabel">label</span>
</wicket:panel>
```

FormComponent

- Leszármazottai a megszokott HTML form komponensek, a nevek AWT-ből ismerősek lehetnek
  - Érdekes metódusok:
    - setRequired()/isRequired()
    - add(IValidator v)
    - getLabel()/setLabel()
  - Button:
    - onSubmit()

Wicket modellek

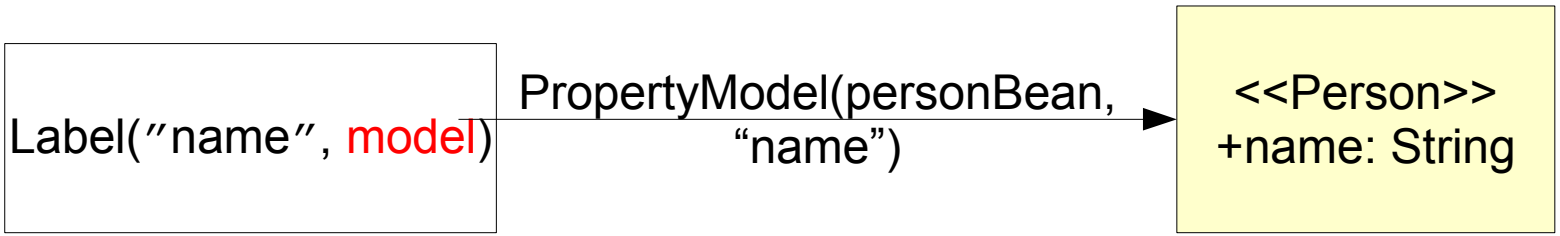
- A komponensek az adatokat tartalmazó POJO-kat model adaptereken keresztül érik el

```
add(new Label(”name”, new Model(”Gipsz Jakab”));

add(new Label(”name”, new PropertyModel(personBean, ”name”);

setModel(new CompoundPropertyModel(personBean);
   add(new Label(”name”));

setModel(new CompoundPropertyModel(new
   DetachablePersonModel(personBean)));
   add(new Label(”name”));
```



Modellek

- Detachable modellek
  - Csak egy azonosítót tárol, nem magát a teljes objektumot
  - On demand be tudja tölteni az objektumot
  - Amikor már nem kell, eldobja a referenciát
- Compound modellek
  - Szülő komponensen haszálandó
  - Modell nélküli komponenshez az id alapján keres hozzá objektumot

Wicket AJAX – először egy példa

- Normál link

```
final Label label = ...;

add(new Link("link") {

public void onClick() {

label.setVisible(!
   label.isVisible());

}

});
```

HTML:

```
<a href="#" wicket:id="link">
   Toggle
</a>
```

- Ajax link

```
final Label label = ...;

add(new AjaxLink("link") {

public void
   onClick(AjaxRequestTarget t)
   {

    label.setVisible(!
      label.isVisible());

    t.addComponent(label);

}

});
```

Wicket AJAX

- HTML-ben semmi változás
- AJAX-szal frissítendő komponenseken:
  - `setOutputMarkupId(true);`
- Normál komponensek – AJAX-osított komponensek
  - Link – AjaxLink, Button – AjaxButton
- AJAX Behaviour-ök, amiket normál komponensekhez lehet add()-olni:
  - `usersPresentPanel.add(new AjaxSelfUpdatingTimerBehavior(Duration.ONE_MINUTE);`
  - ```
    DropDownChoice choice=new DropDownChoice(...);
      choice.add(new AjaxEventBehavior("onchange") {
        protected void onEvent(AjaxRequestTarget target) {
           target.addComponent(componentToRefresh);
        }
      }
    ```
    ```

WicketTester

- Mock javax.servlet osztályok segítségével container nélkül teszi lehetővé a komponensek unit tesztelését
- Érdemes a leválasztott Service/Dao réteget Mock osztályokkal emulálni a tesztekhez

```
private WicketTester tester;

    public void setUp() {
      tester = new WicketTester();
    }

    public void testRenderMyPage() {
          //start and render the test page
          tester.startPage(MyPage.class);
          //assert rendered page class
          tester.assertRenderedPage(MyPage.class);
          //assert rendered label component
          tester.assertLabel("myMessage", "Hello!");
   }
```

Resource

- Az alkalmazáshoz tartozó statikus vagy dinamikus erőforrások kezelésére
  - Pl: a .jar-ba csomagolt .css vagy .jpg file, vagy egy adatbázisból visszaadott blob, esetleg egy RSS feed
  - Egy resource Mount-olható egy URL-re
  - Kiszolgálás közben nem szinkronizál a Session-re
  - “mini servlet”, 1 példány él belőle az adott alkalmazásban

Összefoglalva

- A Wicket egy sokoldalú, komponens alapú open source keretrendszer
- 100% Java, 100% OO – jobb programozóvá tesz
  - Tipikusan az első alkalmazását fél év után feleannyi LoC-ból újra tudja írni az ember
- Jelenleg “hivatalosan” kissé aluldokumentált
  - 1 db angol nyelvű könyv az 1.2-ről, 1.3-ról készül egy új
  - De egész jók az example-ök és a javadoc, a forrás is olvasmányos
  - További információk:
    - http://wicket.sourceforge.net
    - http://incubator.apache.org/wicket - remélhetőleg hamarosan http://wicket.apache.org

Kérdések, válaszok

?