Warszawa JUG

Struts 2 – rusztowanie dla Łebu

czyli majster i malarz - ściemniacz na budowie (magazynier ma wolne)

Tfórca:
Łukasz Lenart
lukasz.lenart@gmail.com

Agenda

- Struts - co, gdzie, kiedy?
- Pierwszy projekt – brzydula maven2 wchodzi na scenę
- Konfiguracja – ble, ble, …
- Akcje, rezultaty, interceptory, … - a gdzie to całe HTTP?
- OGNL – a to co za potworek?
- Walidacja – czyli „Ordnung must sein!”
- Struts Tags – jakoś to musi wyglądać…
- Parlez-vous anglais? – e …. ktoś zna francuski?
- Spring – a ten tu czego?
- Pluginy – kiedy się nudzisz…
- Czy coś jeszcze? – pytania są tendencyjne!!!

Struts^2 – co, gdzie, kiedy?

- A na początku był servlet …
- Pierwszy był Model Pierwszy – JSP (czyli skrypty jako servlety)
- Drugi przyszedł Model Drugi – servlet do JSP
- A potem był zespół Dwa plus Jeden (MVC), czyli złoty przebój - Struts1
  - Trochę skomplikowany, dużo XMLa, … (dodać własne wady)
  - Jednak stał się standardem i wzorcem
- Nadchodzi nowe – Struts2, kolejne złote dziecko? ;-)
- Action Base Framework czyli „rusztowanie bazujące na działaniu”

Struts^2 – co, gdzie, kiedy?

- Strona główna projektu http://struts.apache.org
- Grupa wsparcia dla użytkowników
  http://struts.apache.org/mail.html
- Inne pomocne sznurki
  http://struts.sourceforge.net/
  http://www.planetstruts.org/
  http://www.vitarara.org/cms/struts_2_cookbook
  http://www.roseindia.net/.../struts2-tutorials
  http://www.infoq.com/struts
  http://www.infoq.com/minibooks/starting-struts2

Struts^2 – co, gdzie, kiedy?

- Struts1 – release 1.0 to rok 2001 (sam pomysł narodził się gdzieś w 1999)
  - najnowsza wersja to 1.3.8
- Struts2 – tak naprawdę to mezalians komuny Struts1 i ŁebRoboli (WebWork 2), a i tak cała robotę odwala Xwork, pomysł narodził się gdzieś koło 2005/2006 roku
  - najnowsze wersja to 2.0.11.1
  - na wersję 2.1 wciąż nie można się doczekać
- Struts3 – prima aprilis!!!
  - chodź nigdy nie wiadomo … ;-)

Brzydula maven2(.0.8)

Tworzymy pierwszy projekt w Struts2:

1. mvn archetype:create
2. 19 [enter]
3. pl.org.lenart.wjug.example1 [enter]
4. example1 [enter]
5. 1.0-SNAPSHOT [enter]
6. pl.org.lenart.wjug.example1[enter]
7. [enter]
8. [enter]
9. Voilà – możemy iść do domu…

Minimalne wymagania

- Servlet API 2.4
- JSP API 2.0
- Java 5 (dostępny port dla 1.4, ale nie polecam ;-)
- Min wymagane biblioteki (2.0.11):
  - commons-logging-1.0.4.jar
  - freemarker-2.3.8.jar
  - ognl-2.6.11.jar
  - struts2-core-2.0.11.jar
  - xwork-2.0.4.jar
- Można rozpocząć pracę na bazie struts2-blank.war

Konfiguracja – web.xml

- W web.xml musimy zdefiniować filtr i mapowanie

```
<filter>
   <filter-name>action2</filter-name>
   <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
   </filter-class>
</filter>
<filter-mapping>
   <filter-name>action2</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Konfiguracja – struts.xml

- <**package**> - podstawowa jednostka
  - <action> - nasze akcje (ale nie fundusz)
  - <interceptors> - przechwytywacze wykonania akcji
  - <result-types> - co ma z tego wyniknąć
  - <default-*> - różne domyślne rzeczy
  - <global-*> - różne globalne rzeczy
- <**include**> – pozwala rozbić konfigurację na moduły
- <**bean**> – nasze implementacje podstawowych klas S^2
- <**constant**> – definiowane wartości stałych, w tym naszych własnych

Konfiguracja – struts.xml, cd…

- <**action**>
  - name – jak będzie widział ją świat
  - class – gdzie faktycznie jest nasz kod
  - method – co ma zostać wywołane, domyślnie execute()
  - <result name=? type=?> - jaki będzie wynik
- <**interceptors**> - przechwytywacze tylko dla tej akcji
  - <interceptor name=? class=?>
  - <interceptor-stack name=?>

Konfiguracja – inne pliki

- struts.properties – kolejne miejsce na definiowanie stałych
- I18N - zasoby językowe dla naszych akcji, plików JSP
  - jakiś/pakiet/<ActionClassName>.properties
  - jakiś/pakiet/package.properties
  - pakiet/super/klasy/<SuperClassName>.properties
- <***>-validation.xml – nasze zasady walidacji formularzy i wartości
- validators.xml – jeśli napisaliśmy nasze własne walidatory, to umieszczamy je tutaj; nadpisuje również domyślny plik szkieletu (do wersji 2.0.8)

Akcje

- W najprostszym przypadku akcja musi posiadać metodę execute(), która zwraca String, jako nazwę rezultatu, zdefiniowanego w **struts.xml**
- W rzeczywistości, akcje rozszerza się z klasy **ActionSupport**, żeby nie odkrywać koła na nowo
- Tworzona jest osobna instancja dla każdego wywołania

```
public class SimpleAction {
    public String execute() { return „hello”;}
}
```

Akcje

- Dodatkową funkcjonalność „wstrzykuje” się do akcji za pomocą przechwytywaczy (interceptors) i powiązanych z nimi interfejsów SessionAware, RequestAware, etc…
- 99% akcji będzie niezależnych od HttpRequest, HttpSession, HttpResponse
- Na bazie powyższych interfejsów, wartości są wstrzykiwane jako obiekty **java.lang.Map**

Rezultaty

- Rezultaty definiuje się w ramach konfiguracji akcji

```
<action name="list„ class=„SomeAction" method="doList">
  <result>/jsp/employees/list.jsp</result>
</action>
```

- Domyślny rezultat to Dispatcher a domyślna wartość to „success” – stąd powyższy przykład działa
- Nowe typy rezultatów można dołączać za pomocą pluginów i rozszerzyć dany pakiet (np. tiles-default)
- Można również zdefiniować je dla pakietu w sekcji

```
<result-types>
<result-type name="tiles" class="org...TilesResult" />
```

Rezultaty

- Dostępne typy rezultaty
  - Dispatcher – domyślny
  - Freemarker
  - Velocity
  - XSL
  - Tiles
  - Redirect i RedirectAction
  - Stream (np. do wysyłania plików lub dynamicznych obrazków)
  - JasperReports
  - Chain

Interceptory - używanie

- Klasa wywoływana przed lub po akcji / rezultacie
- Interceptory definiuje się pojedynczo lub jako stos wywołań i czasami kolejność gra rolę %-P
- Stosy interceptorów są zdefiniowane w struts-default.xml
- Jeśli nasz pakiet rozszerza np. struts-default to automatycznie dziedziczy też stos zdefiniowany dla tego pakietu
- W celu zwiększenia wydajności aplikacji, powinno się zdefiniować własny stos tylko z tymi interceptorami, które używamy w aplikacji

Interceptory - tworzenie

- Chcąc stworzyć własny interceptor, należy zaimplementować interfejs **com.opensymphony.xwork.Interceptor**
- Zawiera on tylko jedną metodę **String intercept(InvocationContext invocation)**
- Obiekt **InvocationContext** zawiera referencję do akcji i jej otoczenia
- Przetwarzanie należy zakończyć przez **return invocation.invoke();** w celu wywołania kolejnych interceptorów

OGNL – nowy na podwórku

- Object Graph Navigation Language
- Wszystko jest obiektem a nie Stringiem!
- Jeśli chcemy ewaluować wyrażenie to musimy umieścić je w **%{ }**, literały w **%{'sometext'}**
- Wartości są umieszczane i pobierane z **ValueStack** - mapa obiektów, gdzie obiekt 0 to root czyli akcja
- Oznacza to, że wywołanie **<s:property name=„employee.firstName”>** będzie się odnosić do **action.getEmployee().getFirstName();**
- Dostęp do innych zakresów jest możliwy za pomocą **#**

Struts Tags

- Największa bolączka przy migracji!
- Tagi S^2 nie są kompatybilne z S^1 (ze względu na OGNL)
- Podział na Tagi UI i generyczne (kontrolne i danych)
  - <s:textfield name=„employee.firstName”/>
  - <s:if test=„%{employee.firstName != null}”>hurrra!</s:if>
- Wygląd taga zależy od zastosowanego tematu, domyślnie xhtml
- Można definiować własne tematy, tworząc wygląd taga za pomocą Freemarkera lub Velocity
- Te same tagi można użyć w szablonach Freemarker i Velocity

Walidacja – xml, adnotacje

- Walidacje można zdefiniować w pliku xml o odpowiedniej nazwie
  - jakiś/pakiet/<ActionClassName>-validation.xml
  - jakiś/pakiet/<ActionClassName>-<alias>-validation.xml
- Walidacja można zdefiniować za pomocą adnotacji przy setterach dla pól albo globalnie dla klasy
  - @RequiredStringValidator(message = "Field is required!")
  - @SkipValidation – pomija walidacje przy wywołaniu metody (nie chodzi o settera!)
  - @Validations() – definiuje listę wszystkich walidacji dla danej klasy

Walidacja – programowo

- Walidację można również przeprowadzić programowo, definiując odpowiednie metody
  - validate() – wywoływana za każdym razem, niezależnie od metody
  - validate<MethodName>() – wywoływana tylko dla danej metody

Parlez-vous anglais - wielojęzyczność

- Wszystkie Tagi S2 są przygotowane do pracy z wieloma językami
  - <s:property value="getText('some.key')" />
  - <s:text name="some.key" />
  - <s:textfield key="some.key" name="firstName"/>
- Podobnie w validation.xml
  - <message>${getText("validation.failednotice")}</message>
- Adnotacje wykorzystywane do walidacji
  - @RequiredFieldValidator(message="Enter data", key="required.data")

Spring - upraszczamy

- Do tworzenia obiektów można użyć Springa albo Google Guice jako ostatnio pokazał szimano
- Wystarczy zdefiniować stałą w struts.properties struts.objectFactory = spring
- Domyślnie obiekty wiązane są po nazwie
- Jeśli tworzymy akcje, to należy ustawić, że nie są singletonami (singleton=false lub scope=prototype)
- Możemy użyć Springa do tworzenia interceptorów, jednak, scope zależy jak używamy danego interceptora (stos czy pojedynczo dla akcji)

Pluginy - rozszerzamy

- Szkielet można łatwo rozszerzyć pisząc własne pluginy
- Plugin to może być jedna klasa lub zbiór klas, które dodają nową funkcjonalność lub zmieniają istniejącą
- **struts-plugin.xml** umieszczamy w naszym pliku jar i w nim definiujemy np. nowy typ rezultatu albo nadpisujemy implementację wewnętrznej klasy szkieletu (np. Tiles – nowy typ rezultatu, Spring – nadpisuje domyślną fabrykę obiektów)
- Przykład
  - struts2-spring-plugin
  - struts2-tiles-plugin

Dostępne pluginy

- Spring – nowa fabryka obiektów
- SiteMesh – inne podejście do szablonów
- Tiles – system dekompozycji strony
- Struts 1 – ułatwia migrację
- JSF – pozwala używać JSF, jednak z ograniczeniami
- JasperReports - raporty
- … i parę innych, sprawdź -> http://cwiki.apache.org/S2PLUGINS/home.html

Czy coś jeszcze?