v 1.1
Web Framework

Piotr Gaertig / 2009

Grails - treść prezentacji

- Historia
- Ogólnie o Grails
- Groovy
- Starter
- Model GORM
- Scaffolding
- Kontroler
- Widok GSP
- URL Mapping
- Development & deployment
- Pozostałe możliwości

Historia Grails

- **Ruby on Rails** aka **RoR** (2004) - framework webowy oparty o język Ruby. Stosuje nowatorskie rozwiązania m.in. Convention over Configuration, AJAX, URL mapping wyrażeniami, wzorzec Active Record, REST.
- **Groovy on Rails** (2005) - G2One próbóje odtworzenia funkcjonalności Ruby on Rails w języku Groovy na JVM. W 2006 następuje zmiana nazwy na:
- **Grails** - obecnie udostępnia analogiczną funkcjonalność jak RoR, a w większości przypadków wyprzedza go.
- W 2008 **SpringSource** przejmuje G2One – popularność rośnie.

Grails: Czym jest?

- Framework zaimplementowany w języku Groovy
- **Grails = (MVC + CoC) * (Groovy + Hibernate + Spring + GSP + ...)**
- MVC oraz
- Convention over Configuration (CoC)
  - Konwencja narzucana przez: szkielety, ustawienia domyślne oraz szczególne umiejscowienie plików w strukturze katalogowej projektu.
  - Łatwiej zacząć projekt i prototypować
  - Zawsze można rozwinąć szkielet i ustawienia by uzyskać zaawansowaną funkcjonalność.
- Produktem końcowym jest WAR do osadzenia na dowolnym serwerze J2EE

Groovy

- Dynamiczny język programowania podobny składniowo do Javy. Kompilowalny do bytecodu (.class) > pracuje na każdym JVM.
- Możliwości znane z Ruby, Python, Perl, Smalltalk:
  - dynamiczne typowanie (def)
  - domknięcia – closures
  - przeładowywanie operatorów
  - składnia do obsługi list, map oraz wyrażeń regularnych
  - iteratory polimorficzne
  - wyrażenia w literałach znakowych
  - operator ?. - bezpieczny null (level?.room[0]?.number)
  - dynamiczne rozszerzanie istniejących obiektów - także Java.
  - wspiera składnię XML i HTML

Groovy: przykład

- Java

```
for (String it : new String[] {"Groovy", "Java", "C#"})
     if (it.length() <= 4)
         System.out.println("Hi " + it);
```

- Groovy

```
["Groovy", "Java", "C#"].findAll{it.size() <= 4}.each{println "Hi ${it}"}
```

Starter Grails

- Na skróty – co warto wiedzieć:
  - Ściągamy i instalujemy z http://www.grails.org/
  - `$ grails create-app NazwaAplikacji`
  - Applikacja gotowa do uruchomienia/zdeployowania.
  - Domysnie skonfigurowana baza pamięciowa HSQL - restarty niszczą dane.
  - `$ grails run-app`
  - http://localhost:8080/NazwaAplikacji/
  - Większość modyfikacji kodu jest natychmiast widoczna w przeglądarce.
  - `$ grails shell` – interaktywne wykonywanie instrukcji Groovy w Grails

Starter: struktura projektu

%PROJECT_HOME%

- + **grails-app**
  - + **conf** ---> location of configuration artifacts
    - + **hibernate** ---> optional hibernate config
    - + **spring** ---> optional spring config
  - + **controllers** ---> location of controller artifacts
  - + **domain** ---> location of domain classes
  - + **i18n** ---> location of message bundles for i18n
  - + **services** ---> location of services
  - + **taglib** ---> location of tag libraries
  - + **util** ---> location of special utility classes
  - + **views** ---> location of views
    - + **layouts** ---> location of layouts
- + **lib**
- + **scripts** ---> scripts
- + **src**
  - + **groovy** ---> optional; location for Groovy source files (of types other than those in grails-app/*)
  - + **java** ---> optional; location for Java source files
- + **test** ---> generated test classes
- + **web-app**
  - + **WEB-INF**

Model: GORM

- Model zwany domeną
- Groovy Object-Relational Mapping
- Rozszerzenie/nakładka na Hibernate, pozwala:
  - modelować encje i relacje,
  - automatycznie tworzyć schemat w bazie,
  - ograniczać/walidować dane w polach,
  - przechwytywać zdarzenia (przed/po zapisie, odczycie, aktualizacji, usunięciu)
  - korzystać z dynamicznych metod do wyszukiwania (finders) używających nazw pól i typów relacji,
  - wykorzystać już istniejące POJO z adnotacjami Hibernate i dopiąć do nich dynamiczne metody.
  - Wszystko to w przystępnej składni Groovy.

Model: GORM

- Tworzymy prosty model:

```
$ grails create-domain-class Author
$ grails create-domain-class Book
```

- grails-app/domain/Actor.groovy:

```
class Author {
    String name
}
```

- grails-app/domain/Book.groovy:

```
class Book {
    String title
    Author author
}
```

- Powstaly też testy JUnit w grails-app/test/unit/

Model: GORM

- Wkonajmy proste operacje z konsoli

```
// *C*reate
def a = new Author(name: "Prof. Mio")
a.save()

// *R*ead & *U*pdate
def a = Author.findByName("Prof. Mio")
a.name = "Prof. Miodek"
a.save()

// *D*elete
def b = Book.findByAuthor(a)
b.delete()
```

```
class Author {
    String name
}

class Book {
    String title
    Author author
}
```

```
def authors = Author.findAll(); def authors2 = Author.list()
def authors_professors = Author.findAllByNameLike("Prof.%")
def autors_cnt = Author.count()
def first_exists = Author.exists(1)

// HQL!!!
def book = Book.find("from Book b where b.title = ?", [ 'Polszczyzna' ])
```

Model: GORM

- Relacje z kaskadą

```
class Author {
    String name
    static hasMany = [books: Book]
}

class Book {
    String title
    Author author
    static belongsTo = [author: Author]
}
```

```
new Author(name:"Paulo Coelho")
            .addToBooks(new Book(title:"Alhemik"))
            .addToBooks(new Book(title:"Zahir"))
            .save()
```

- Powstaną również CONSTRAINT-y w bazie.

Model: GORM

- Trochę więcej - walidacja i zdarzenia.

```
class Author {
  String name
  int birthYear
  String email
  static hasMany = [books: Book]
  static constraints = {
    name(maxSize:30, nullable: false)
    email(email:true, blank: false)
    birthYear(range: 6..120)
  }
}
```

```
class Book {
  String title
  Author author
  Date dateCreated
  def beforeInsert = {
      dateCreated = new Date()
  }
  static belongsTo = [author: Author]
}
```

- Sprawdzamy:

```
def author = new Author(params)
if(author.validate()) {
    // poprawne - zrob cos z autorem
} else {
    author.errors.allErrors.each { println it }
}
```

Scaffolding

- Scaffolding to generowanie szkieletu CRUD na podstawie modelu.
- Tworzymy model i kontroller:

```
$ grails create-domain-class Author
$ grails create-controller Author
```

- grails-app/domain/Author.groovy - dopisujemy pola.
- grails-app/controllers/AuthorController.groovy, wpisujemy:

```
class AuthorController {
    def scaffold = true
}
```

- Dynamicznie powstaną metody: *list*, *show*, *create*, *new*, *delete*, *edit*, *update*.
- Widok GSP będzie dynamicznie udostępniony z szablonów.
- To wszystko! Uruchamiamy aplikację przez `$ grails run-app`

Kontroler

- *$ grails generate-controller Author* – generuje rzeczywiste pliki ze scaffoldingu – można dostosować operacje do swoich potrzeb, dodawać nowe etc.
- *$ grails create-controller Book*
- grails-app/controllers/BookController.groovy:

```
class BookController {
    def mylist = {
        [ books: Book.findAll{it.owner == params.user} ]
    }
}
```

Widok - przykład GSP

- GSP – podobny do JSP język opisu wzorców stron HTML.
- grails-app/views/book/mylist.gsp:

```
<html>
  <head>
    <title>My books</title>
  </head>
  <body>
    <ul>
      <g:each in="${books}">
        <li>${it.title} (${it.author.name})</li>
      </g:each>
    </ul>
  </body>
</html>
```

- *$ grails generate-views Author* tworzy pliki GSP ze scaffoldingu/modelu

Widok - Tagi GSP

- Bogata biblioteka gotowych tagów w Grails.
- Własne tagi:
  - grails-app/taglib/AuthorTagLib.groovy:

```
def formatDate = { attrs ->
    out << new java.text.SimpleDateFormat(
        attrs.format).format(attrs.date) }
```

  - grails-app/views/author/show.gsp:

```
...
<g:formatDate format="dd.MM.yyyy" date="${author.birthDate}"/>
...
```

- Żadnych importów tagów w GSP.
- Ciekawostka: po pewnych zabiegach mogą działać też w JSP

URL Mapping

- http://localhost:8080/MyApp/book/show/10
  - kontroler **Book**Controller
  - metoda **show**()
  - **10** dostępne przez `params.id` w metodzie `show`
- grails-app/conf/UrlMappings.groovy, domyślnie:

```
class UrlMappings {
    static mappings = {
      "/$controller/$action?/$id?"{
                constraints {
                                // apply constraints here
                    }
            }
      "/"(view:"/index")
           "500"(view:'/error')
         }
}
```

Development i deployment

- Środowiska pracy: development, test, production
- Moża też tworzyć swoje: „UAT”
- `$ grails war` – tworzy WAR, który możemy natychmiast umieścić na serwerze Java EE, domyślnie produkcyjny
- `$ grails run-app` to domyślnie tryb developerski

Pozostałe możliwości Grails

- Filtry/interceptory wywołań akcji
- Web Services, REST
- Warstwa serwisowa z transakcjami
- Web Flow
- AJAX
- Komunikaty flash – pokazywane na bierzącej lub kolejnej (redirect) odsłonie strony WWW, np. po poprawnym wykonaniu operacji.
- Definiowanie Spring beanów w Groovym
- Dependency injection by convention
- Pluginy - kilkaset aktualnie dostępnych
- JSP zamiast GSP
- JDO i JPA zamiast GORM (plug-iny)
- Wsparcie językowe (I18N)
- Wbudowany server Jetty do developmentu.
- Skrypty Gant
- Integracja z Ant i Maven
- Więcej na http://grails.org/

???

```
Sluchacze.findAll{ it.hasPytanie() }.each{
    it.pytanie.say()
}
```

Amen

Dziękuję za uwagę.