

Seam - jak wystartować

Contributed by Administrator
poniedziałek, 31 sierpień 2009

Przykład prostej aplikacji stworzonej w technologii JBoss Seam. Artykuł opisuje podstawowe kroki, które należy wykonać aby stworzyć projekt, skompilować i uruchomić na serwerze Tomcat lub JBoss. Wyjaśnia działanie framework'a Seam na przykładzie systemu aukcyjnego. Odnośniki do dokumentacji i przykładów, które można znaleźć w sieci.

```
.code {  
border: solid 1px black; padding: 5px;  
background: rgb(230,230,230);  
width: 480px;  
overflow-x: auto;  
display: block;  
margin-right: 4px;  
}  
.fileName {  
border: 1px solid gray;  
#background: #ffffff;  
width: 98%;  
margin-bottom: 5px;  
color : #003f7d;  
font-family : Arial, Verdana, Helvetica, sans-serif;  
font-size : 15px; font-weight: bold;  
}
```

Instalacja i infrastruktura

Do realizacji przykładowego projektu potrzebujemy Javy 1.5+, Seam oraz serwer Tomcat lub JBoss. (Oczywiście można też użyć innych serwerów zgodnych z JEE). Użyjemy też środowiska IDE by nie pisać kodu w notatniku :)

Skąd pobrać:

<http://kasper.eobjects.dk/2009/04/seam-ejbs-and-ear-packaging-in-maven.html>

Seam 2.2.0 <http://seamframework.org/>

Tomcat <http://tomcat.apache.org/>

JBoss <http://jboss.org/jbossas/>

Java <http://java.sun.com/javase/downloads/index.jsp>

Netbeans IDE <http://www.netbeans.org/>

Eclipse IDE <http://www.eclipse.org/>

Przykładowa dziedzina projektu

Na potrzeby przykładu będziemy używać dziedziny porad eksperckich. System ma umożliwiać zadawanie pytań do ekspertów, przeglądanie pytań i udzielanie odpowiedzi. Wszystko oczywiście w bardzo uproszczonej formie.

Lista kroków do realizacji projektu

- Przygotowanie Tomcat'a
- Przygotowanie struktury projektu pod kątem użycia Maven'a
- Napisanie klas encyjnnych, biznesowych
- Stworzenie interfejsu użytkownika
- Kompilacja i zbudowanie projektu
- Instalacja zbudowanej aplikacji w środowisku Tomcat'a
- Wywołania stron aplikacji w przeglądarce WWW

Przygotowanie Tomcat'a

Wystarczy zainstalować Tomcat'a. Jeśli planujemy używać bazy danych, to sterownik do niej wgrujemy do katalogu TOMCAT_HOME/lib. W naszym przypadku użyjemy PostgreSQL, ale zmiana w jednym miejscu umożliwi użycie praktycznie dowolnej bazy danych. Mechanizmowi zależności Maven'a pozostawimy zajęcie się pozostałymi bibliotekami, z których nasza aplikacja ma korzystać. Doda on wymagane przez Seam'a biblioteki do paczki WAR. Zależności te opiszemy w deskryptorach POM.

Przygotowanie struktury projektu pod kątem użycia Maven'a

Przy tworzeniu projektu można się posłużyć Seam'owym narzędziem seamgen, który wygeneruje nam cały szkielet aplikacji, a później można z jego pomocą generować poszczególne komponenty. To podejście daje nam szybki start i przygotowuje strukturę projektu do stosowania hotdeploymentu na JBoss. Niestety wadą tego podejścia jest brak możliwości łatwego wykorzystania maven'a do zarządzania zależnościami, procesem kompilacji, testów i deploy'mentu. Jeśli zależy nam na maven'ie, to możemy użyć istniejących plugin'ów, które wygenerują nam szkielet aplikacji Seam zgodnej z filozofią maven'a, albo tworzymy taką strukturę ręcznie. Ze względu na pewne wady podejścia 1 i 2 wybraliśmy wariant 3 i samodzielnie stworzyliśmy odpowiednią strukturę. Poniżej struktura oraz deskryptory opisujące poszczególne moduły.

Projekt został podzielony na następujące moduły i opiasny deskryptorami POM:

- ExpertAdvice-businessapi - zawiera interfejsy logiki biznesowej (w tym DAO)
- ExpertAdvice-businessimpl - klasy implementujące logikę biznesową
- ExpertAdvice-domain - klasy reprezentujące obiekty dziedziczne, reprezentujące dane trwałe
- ExpertAdvice-war - moduł grupujący warstwę prezentacji (widoki)
- ExpertAdvice - moduł rodzic, grupujący pozostałe moduły

Widok katalogów:

```
ExpertAdvice
|- businessapi
|- businessimpl
|- domain
|- war
```

```
pom.xml
pom.xml
pom.xml
pom.xml
```

Napisanie klas encyjnych, biznesowych

Aдекватnie do podziału na moduły piszemy najpierw klasy dziedziczne (encyjne), później interfejsy biznesowe oraz DAO. W kolejnym kroku tworzymy implementacje i kontroler do obsługi interfejsu użytkownika.

QuestionUCInterfejs fasady do logiki biznesowej

QuestionDAO, UserDaoInterfejsy DAO

QuestionUCBeanImplementacja fasady do logiki biznesowej

QuestionDAOBean, UserDaoBeanImplementacja DAO

Question, UserKlasy dziedziczne (encyjne)

QuestionBrowserKontroler do obsługi interfejsu użytkownika

Do dzieła. Na początek klasa encyjna Question - zawiera pytanie, udzieloną odpowiedź oraz relacje do osoby zadającej pytanie i udzielającej odpowiedzi. W podstawowej wersji aplikacji nie będziemy obsługiwać tych relacji. Używając JPA mapujemy ją do relacyjnej tabeli QUESTION.

Do klasy dodajemy dwie metody public String getAnswerPreview() i public String getQuestionPreview(), które zwracają jedynie zająwkę pytania i odpowiedzi (pierwszych 20 znaków, z wyłączeniem znaczników HTML). Zająwki te są nam potrzebne by przy wyświetlaniu listy pytań widzieć w każdym wierszu tylko skrót tekstów i to bez formatowania HTML.

```
pl\expert\domain\Question.java
```

```
package pl.expert.domain;
```

```
import java.io.Serializable;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.ManyToOne;
```

```
import org.apache.commons.lang.StringUtils;
```

```
@Entity
```

```
public class Question implements Serializable {  
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

```
    @GeneratedValue
```

```
private long      id;
private String    question;
private String    answer;
@ManyToOne
private User      answerAuthor;
@ManyToOne
private User      questionAuthor;

public String getQuestion() {
    return question;
}

public void setQuestion(String question) {
    this.question = question;
}

public String getQuestionPreview() {
    String t = question;
    if (t != null) {
        t = t.replaceAll("\\\\", "");
        t = StringUtils.abbreviate(t, 20);
    }
    return t;
}

public String getAnswer() {
    return answer;
}

public void setAnswer(String answer) {
    this.answer = answer;
}

public String getAnswerPreview() {
    String t = answer;
    if (t != null) {
        t = t.replaceAll("\\\\", "");
        t = StringUtils.abbreviate(t, 20);
    }
    return t;
}

public User getAnswerAuthor() {
    return answerAuthor;
}

public void setAnswerAuthor(User answerAuthor) {
    this.answerAuthor = answerAuthor;
}

public User getQuestionAuthor() {
    return questionAuthor;
}

public void setQuestionAuthor(User questionAuthor) {
    this.questionAuthor = questionAuthor;
}

public long getId() {
    return id;
}
```

```
@Override
public String toString() {
    return "Question [answer=" + answer + ", id=" + id + ", question="
        + question + "];"
}
}
```

W drugim kroku tworzymy interfejsy: QuestionDAO i bazowy DAO.

```
pl\expert\server\dao\api\QuestionDAO.java
```

```
package pl.expert.server.dao.api;
import pl.expert.domain.Question;
public interface QuestionDAO extends DAO {
}
```

```
pl\expert\server\dao\api\DAO.java
```

```
package pl.expert.server.dao.api;
import java.util.List;
import pl.expert.server.service.api.ExpertAdviceException;
public interface DAO {
    public E find(long id);
    public List findAll(int start, int howMany);
    public long getAllCount();
    public E insert(E entity);
    public E update(E entity);
    public void remove(long id) throws ExpertAdviceException;
}
```

Teraz interfejs fasadowy QuestionUC. Zawiera wszystkie operacje biznesowe jakie chcemy udostępnić w aplikacji.

```
pl\expert\server\service\api\QuestionUC.java
```

```

package pl.expert.server.service.api;

import java.util.List;

import pl.expert.domain.Question;
import pl.expert.domain.User;

public interface QuestionUC {
    public Question    addQuestion(Question question) throws ExpertAdviceException;
    public void        answerQuestion(long id, String answer, User answerAuthor) throws
ExpertAdviceException;
    public Question    getQuestion(long id) throws ExpertAdviceException;
    public List getQuestions(int start, int howMany) throws ExpertAdviceException;
    public long        getAllQuestionsCount();
    public void        removeQuestion(long id) throws ExpertAdviceException;
    public void        updateQuestion(Question question);
}

```

Kolej na implementacje: QuestionDAOBean, BaseDAO oraz QuestionUCBean. BaseDAO w sposób generyczny realizuje podstawowe operacje CRUD (ang. Create, Retrieve, Update, Delete). Komponenty te są bezstanowe, więc nadajemy im też taki zakres życia we frameworku Seam `@Scope(ScopeType.STATELESS)`. Dostęp do bazy danych (czyli operacje `persist()`, `find()` itd.) dostarcza nam obiekt z interfejsem `EntityManager`. Jest on wstrzyknięty automatycznie przez Seam'a przy uruchomieniu naszego komponentu DAO. Służy to tego adnotacja `@In`

```

@In
private EntityManager em;

```

```

pl\expert\server\dao\impl\QuestionDAOBean.java

```

```

package pl.expert.server.dao.impl;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import pl.expert.domain.Question;
import pl.expert.server.dao.api.QuestionDAO;

//@Stateless @Local
@Name("questionDAO")
@Scope(ScopeType.STATELESS)
public class QuestionDAOBean extends BaseDAO implements QuestionDAO {

    public QuestionDAOBean() {
        super(Question.class);
    }

}

```

```
pl\expert\server\dao\impl\BaseDAO.java

package pl.expert.server.dao.impl;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.Query;

import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.log.Log;

import pl.expert.server.dao.api.DAO;
import pl.expert.server.service.api.ExpertAdviceException;

public abstract class BaseDAO implements DAO {

    @Logger
    Log log;

    private Class<E> entityType;
    // @PersistenceContext
    @In
    private EntityManager em;

    public BaseDAO(Class<E> entityType) {
        this.entityType = entityType;
    }

    public E find(long id) {
        return em.find(entityType, id);
    }

    @SuppressWarnings("unchecked")
    public List<E> findAll(int start, int howMany) {
        Query q = em.createQuery("select o from " + entityType.getSimpleName() + " o");
        q.setFirstResult(start);
        q.setMaxResults(howMany);

        return q.getResultList();
    }

    public long getAllCount() {
        Query q = em.createQuery("select count(o) from " + entityType.getSimpleName() + " o");
        return (Long)q.getSingleResult();
    }

    public E insert(E entity) {
        em.persist(entity);
        return entity;
    }

    public E update(E entity) {
        return em.merge(entity);
    }

    public void remove(long id) throws ExpertAdviceException {
        E entity = em.find(entityType, id);
        if(entity!=null) {
```

```
    em.remove(entity);  
  } else {  
    throw new ExpertAdviceException("Entity with id: " + id + " doesn't exist.");  
  }  
}  
}
```

Dostęp do bazy danych definiujemy w trzech miejscach:

- persistence.xml (deskryptor ten zdefiniowany jest przez specyfikację JPA) - zawiera definicję jednostki trwałości (ang. Persistence Unit)
- context.xml (deskryptor opisujący konteksts naszej aplikacji - używany przez Tomcat'a) - zawiera definicję źródła danych (DataSource), dostęp do bazy danych przez JDBC
- components.xml (deskryptor zdefiniowany przez Seam) - zawiera między innymi definicję EntityManager'a.

ExpertAdvice\war\src\main\resources\META-INF\persistence.xml

```
org.hibernate.ejb.HibernatePersistence  
java:comp/env/jdbc/expertAdviceDB  
pl.expert.domain.Question  
pl.expert.domain.User
```

ExpertAdvice\war\src\main\webapp\META-INF\context.xml

ExpertAdvice\war\src\main\webapp\WEB-INF\components.xml

Wpis `<transaction:entity-transaction entity-manager="{em}"/>` definiuje komponent EntityManager i udostępnia go w aplikacji Seam pod nazwą em, aby mieć do niej dostęp w naszym komponencie wstawiamy pokazaną już wcześniej adnotację @In.

QuestionUCBean w zasadzie tylko deleguje wywołania do QuestionDAO, który dzięki adnotacji @In jest automatycznie wstrzykiwany przez Seam'a podczas uruchamiania naszego komponentu. Również definiujemy ga jako bezstanowy @Scope(ScopeType.STATELESS).

pl\expert\server\service\impl\QuestionUCBean.java

```
package pl.expert.server.service.impl;
```

```
import java.util.List;
```

```
import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;
```

```
import pl.expert.domain.Question;
import pl.expert.domain.User;
import pl.expert.server.dao.api.QuestionDAO;
import pl.expert.server.service.api.ExpertAdviceException;
import pl.expert.server.service.api.QuestionUC;
```

```
//@Stateless @Local
@Name("questionUC")
@Scope(ScopeType.STATELESS)
public class QuestionUCBean implements QuestionUC {
//    @EJB
    @In(value="questionDAO", create=true)
    private QuestionDAO questionDAO;
```

```
public Question addQuestion(Question question) {
    return questionDAO.insert(question);
}

public void answerQuestion(long id, String answer, User answerAuthor) {
    Question question = questionDAO.find(id);
    question.setAnswer(answer);
    question.setAnswerAuthor(answerAuthor);
    questionDAO.update(question);
}

public List getAllQuestions(int start, int howMany) {
    return questionDAO.findAll(start, howMany);
}

public Question getQuestion(long id) {
    return questionDAO.find(id);
}

public long getAllQuestionsCount() {
    return questionDAO.getAllCount();
}

public void removeQuestion(long id) throws ExpertAdviceException {
    questionDAO.remove(id);
}

public void updateQuestion(Question question) {
    questionDAO.update(question);
}
}
```

Na koniec został nam do napisania kontroler odpowiedzialny za obsługę interfejsu użytkownika QuestionBrowser. Zgodnie ze stosowaną już poprzednio techniką nadajemy mu nazwę za pomocą adnotacji @Name("questionBrowser"). W odróżnieniu od poprzednich komponentów (które były bezstanowe) wybieramy dla niego zakres życia @Scope(ScopeType.CONVERSATION). Zakres ten oznacza, że dla każdego otwartego okna przeglądarki bądź karty użytkownik będzie miał do dyspozycji krótkie, niezależne od siebie sesje i w ramach każdej takiej unikalnej "sesji" będzie dostarczona unikalna instancja QuestionBrowser. Dla użytkownika oznacza to, że na każdej karcie może niezależnie przeglądać listę pytań z zapamiętywaniem aktualnego stanu przeglądania (czyli np. która strona z wynikami, jeśli pytań jest tyle, że trzeba je wyświetlać na wielu stronach do przewijania).

Dla pełnego zrozumienia działania zakresu ScopeType.CONVERSATION zachęcamy do sięgnięcia do dokumentacji Seam'a.

Listę pytań do przeglądania trzyma pole private List questionList; oznaczone adnotacją Seam'a @DataModel. Tak oznacza się listę, która może być użyta przez komponent JSF do wyświetlania zawartości tej listy za pomocą mocno rozbudowanej tableki HTML.

Atrybuty page, max, maxPages, pageSize itd. pozwalają na przechowywanie i sterowanie wyświetlaniem listy pytań z podziałem na strony.

Metoda private void prepareQuestionList() za pośrednictwem QUESionUC pobiera fragment listy pytań z bazy danych i inicjuje ustawienia atrybutów odpowiedzialnych za stronicowanie wyników.

```
pl\expert\server\web\QuestionBrowser.java

package pl.expert.server.web;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Begin;
import org.jboss.seam.annotations.Create;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.End;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.faces.FacesMessages;
import org.jboss.seam.log.Log;

import pl.expert.domain.Question;
import pl.expert.server.service.api.ExpertAdviceException;
import pl.expert.server.service.api.QuestionUC;

@Name("questionBrowser")
@Scope(ScopeType.CONVERSATION)
public class QuestionBrowser implements Serializable {

    private static final long serialVersionUID = 1L;

    @In(value = "questionUC", create=true)
    private QuestionUC    questionUC;

    private int          page = 0;
    private boolean      nextPageAvailable;
    private boolean      prevPageAvailable;
    private long         max;
    private int          maxPages;
    private int          pageSize = 10;

    @Logger
    private Log          log;

    @DataModel
    private List questionList;

    @Out(value = "question")
    private Question    question = new Question();
    @In
    private FacesMessages facesMessages;

    public void setQuestionList(List questionList) {
        this.questionList = questionList;
    }

    public List getQuestionList() {
        return questionList;
    }
}
```

```
}

public void insertSampleData() {
    log.debug("starting question insert ...");
    int n = (int)questionUC.getAllQuestionsCount();
    for (int i = n; i < n+10; i++) {
        Question q = new Question();
        q.setQuestion("Pytanie " + i);
        try {
            questionUC.addQuestion(q);
        } catch (ExpertAdviceException e) {
            e.printStackTrace();
            facesMessages.add("Wystąpił problem ze wstawianiem przykładowych danych");
        }
    }
    log.debug("finished question insert");
    prepareQuestionList();
}

@Begin(join=true)
@Create
public void find() {
    page = 0;
    setMax(determineMax());
    maxPages = determineMaxPages();
    prepareQuestionList();
}

@Begin(join=true)
public void refresh() {
    page = 0;
    questionList = null;
    find();
}

private int determineMaxPages() {
    double res = (double) getMax() / pageSize;
    return (int) Math.ceil(res);
}

private long determineMax() {
    return questionUC.getAllQuestionsCount();
}

public void nextPage() {
    page++;
    prepareQuestionList();
}

public void prevPage() {
    page--;
    prepareQuestionList();
}

public void init() {
    prepareQuestionList();
}

private void prepareQuestionList() {
    List results;
    try {
        results = questionUC
```

```
.getAllQuestions(getPage() * pageSize, pageSize);

nextPageAvailable = (getPage() + 1) * pageSize < getMax();
prevPageAvailable = getPage() > 0;

setQuestionList(results);
log.debug("prepareQuestionList(): " + questionList);
} catch (ExpertAdviceException e) {
    e.printStackTrace();
    facesMessages.add("Błąd podczas wczytywania listy pytań: {1}", e);
}
}

public boolean isNextPageAvailable() {
    return nextPageAvailable;
}

public boolean isPrevPageAvailable() {
    return prevPageAvailable;
}

public int getPageSize() {
    return pageSize;
}

public void setPageSize(int pageSize) {
    this.pageSize = pageSize;
}

public void setMax(long max) {
    this.max = max;
}

public long getMax() {
    return max;
}

public int getPage() {
    return page;
}

public void setPage(int page) {
    this.page = page;
}

public void moveToPage(String pageStr) {
    log.info("moveToPage(): #0", pageStr);
    int page = Integer.parseInt(pageStr);
    setPage(page);
    prepareQuestionList();
}

public List getMaxPages() {
    List l = new ArrayList(maxPages);
    for (int i = 1; i
```