



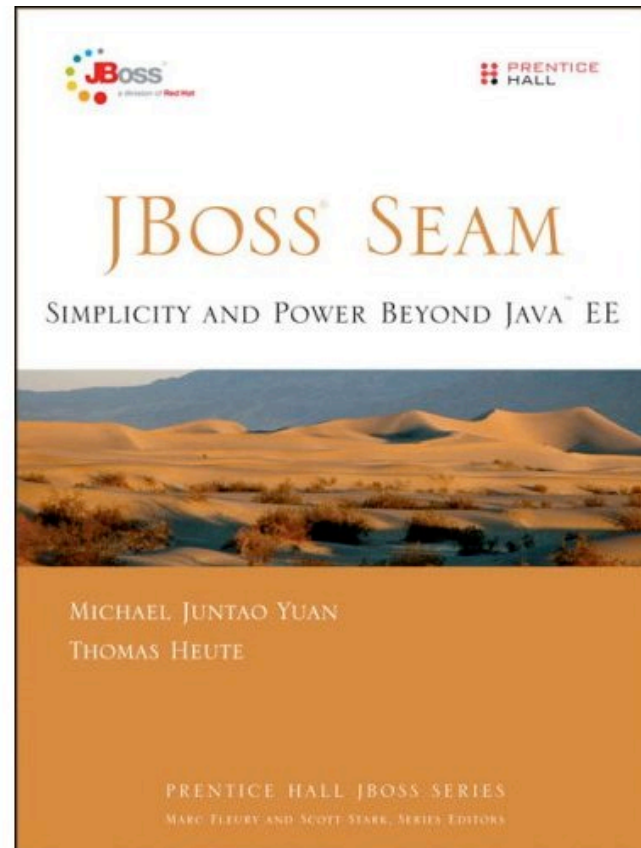
**JBoss Seam**

**Michael Yuan, PhD**  
**Product Manager, Red Hat Inc.**

<http://www.michaelyuan.com/>

# Who am I

- Seam core dev team member
  - Asynchronous method and Quartz integration
  - Performance analysis
  - Support for non-JBoss servers
  - Tools integration
  - Examples and demos
- Product Manager at Red Hat
- Author of “JBoss Seam” from Prentice Hall
- Mobile technology enthusiast
- PhD in Astrophysics



# Agenda



- What is Seam and who should use it?
- 0 to 60
- Seam is made for JSF
- Conversations and business processes
- Security made easy
- The future

A horizontal row of five circles. The first, third, and fifth circles are solid light purple. The second and fourth circles are white with a light purple outline.

Seam in one sentence ...

**Hibernate for the entire enterprise web stack**

# Paradigm shift in programming model

- **Hibernate**

- Use Java objects to access relational databases
- Enable easy unit testing

- **Seam**

- Annotated POJO instead of “XML hell” for all business logic, persistence logic, and UI backing code
- XHTML page for all view needs (dynamic web pages, Ajax, Email, PDF, etc.)
- EL used everywhere (view, pageflow, business process, tests etc.)

# Innovative concepts



- **Hibernate**

- Generate / deploy DB schema from object model
- Filters, validators, search etc. etc.

- **Seam**

- Conversations
  - Supports multi-tab browsers
  - Web transactions
- Rapid CRUD / Ajax app generator
- The rich domain model
- Dependency bijection
- Natural business process integration
- A rule-based security framework that finally works!
- Built-in TestNG framework

# Transparency and choice



- **Hibernate**

- Auto-detects and supports multiple database dialects
- Supports multiple transaction manager lookup strategies
- Supports multiple cache implementations

- **Seam**

- Supports multiple view technologies (AJAX, PDF, Email etc.)
- Supports multiple timers with @Asynchronous
- Access EJB3 beans, Seam POJOs, Groovy beans the same way
- jBPM pageflow and JSF pageflow
- Runs on all popular application servers as well as Tomcat

# The same team

- Gavin King
  - Invented Hibernate out of frustration of EJB
  - Invented Seam out of frustration of
    - the stateless Spring templates
    - the lack of business process integration in web applications.
- Developers are from both inside and outside of JBoss / Red Hat
- Seam is downloaded 10,000+ times / month
  - More than Hibernate in the similar stage
  - The most active forum in all JBoss products



# Who should use Seam?

- JSF developers
  - Seam solves a lot of nagging issues with JSF
  - Many Seam JSF features will be in JSF 2.0 and WebBeans
  - It would be crazy if you use JSF w/o Seam
- Spring Hibernate users
  - Seam eliminates the Lazy loading errors
  - Seam provides deep integration into other frameworks as opposed to Spring's "thin wrapper" + XML approach
- Developers for business process driven web apps
- Anyone who has RoR envy but want to leverage the powerful Java platform

# Does Seam lock you into JBoss?

- Seam deeply integrates JBoss and non-JBoss OSS frameworks
  - JSF 1.2, EJB3, JPA, Web services, JMS
  - Facelets, iText, GWT, Dojo
  - Ajax4jsf, RichFaces, IceFaces
  - jBPM, Drools, Quartz, Groovy, , TestNG, Spring
- Seam runs on a variety of application servers
  - Glassfish, Oracle, WebLogic, WebSphere, plain Tomcat
- Seam developers are from inside and outside of JBoss / Red Hat

# Seam versus Spring?

- They are both integration frameworks
- They compete for the same developers
- You can use Spring in Seam applications
  - Seam can inject Spring beans
  - Manage stateless Spring beans in stateful Seam contexts
  - The other way around is more difficult
- However, as the overall architecture of your app, choose Spring versus Seam can have profound differences

# Spring



- “Thin wrapper” integration
  - It integrates \*anything\*
  - Spring provides object factories
  - Spring manages dependency injection
  - Spring provides AOP infrastructure
- The “non-invasive” integration
  - The API and config files of each integrated framework are used
  - Spring itself does not provide much API beyond the thin template wrappers
- XML heavy approach
- Stateless design

# Seam



- Its a “stack”

- Seam selects what frameworks it integrates with and then invest in the time to do it well

- Deep integration

- Seam provides APIs to access frameworks via a consistent programming model
- Seam makes frameworks work together without explicit glue code (e.g., Hibernate lazy loading)
- Seam eliminates the repetitive API calls and XML files in integration work

# Deep integration: jBPM

- Use jBPM to manage JSF pageflow
- Start / end jBPM process with annotated POJO methods
- Use JSF EL in jBPM process conditions
- Use JSF EL to access jBPM built-in objects

```
<pageflow-definition ...>
```

```
<start-page name="selectCategory" view-id="/sell.xhtml">
```

```
<redirect/>
```

```
<transition name="next" to="enterDetails"/>
```

```
</start-page>
```

```
... ..
```

```
<page name="confirm" view-id="/confirm.xhtml">
```

```
<redirect/>
```

```
<transition name="finish" to="summary">
```

```
<action expression="#{auctionAction.confirm}"/>
```

```
</transition>
```

```
<transition name="previous" to="enterDetails"/>
```

```
</page>
```

```
</pageflow-definition>
```

```
@Begin(pageflow="createAuction",  
        join=true)
```

```
public void createAuction() {
```

```
    // ... ..
```

```
}
```

```
... ..
```

```
@End
```

```
public void confirm () {
```

```
    // ... ..
```

```
}
```

# Deep integration: iText

- Write PDF page in declarative XHTML
- No need to fiddle with iText Java API

```
<p:document ... title="Why Seam" keywords="mykeyword"
  subject="seam" author="Seam Team" creator="Seam PDF example app">

  <p:image alignment="right" wrap="true" value="/jboss.jpg" />
  <p:font size="24"><p:paragraph spacingBefore="16" spacingAfter="40">
    Order #{currentOrder.orderId}
  </p:paragraph></p:font>

  <p:paragraph>Dear #{currentOrder.customerName},</p:paragraph>

  <p:paragraph>... </p:paragraph>

  <p:barCode type="code128" code="My BarCode" />

  <p:signature field="My Signature" size="200 200 400 400" />
</p:document>
```

# Deep integration: Quartz

```
@Asynchronous
public QuartzTriggerHandle schedulePayment(
    @Expiration Date when,
    @Cron String cron,
    ... any other call parameters ...) {
    // do the repeating or long running task
}
```

```
QuartzTriggerHandle handle =
    processor.schedulePayment(payment.getPaymentDate(),
        "0 10,44 14 ? 3 WED",
        payment);
payment.setQuartzTriggerHandle( handle );
// Save payment to DB

// later ...

// Retrieve payment from DB
// Cancel the remaining scheduled tasks
payment.getQuartzTriggerHandle().cancel();
```



# Agenda

- What is Seam and who should use it?
- 0 to 60
- Seam is made for JSF
- Conversations and business processes
- Security made easy
- The future

# Gavin's dilemma



- If I talk about the really unique stuff in Seam, everyone thinks it must be complicated and difficult to use
- If I show how easy it is to use, people think its just another action/CRUD framework

# Setup a project

**New Project**

**Steps**

1. Choose Project
2. **Name and Location**
3. Packages
4. Database Connection

**Name and Location**

Project Name:

Project Location:

Project Folder:

Seam Location:

JBoss Server:

☐ Create as Java Web Application (with no EJB components)

☒ Create as Java Enterprise Application (with EJB components)

# What was generated?

- A basic project skeleton
- With an Ant build script
- That deploys to a WAR or exploded directory
- With support for persistence via JPA
  - with test, dev and prod database profiles
- Basic login / logout
- A Facelets template
  - What if we don't like it? (Change the CSS!)
- And a welcome page
  - What if we don't like the welcome message?

# Add a web page

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    template="layout/template.xhtml">

    <ui:define name="body">
        Hello!
        The time is #{currentTime} .
    </ui:define>

</ui:composition>
```

# Add some interaction

```
<ui:composition ...>

<ui:define name="body">
<h:form>
  Hello! #{hello.name} <br/> The time is #{currentTime}.<br/>

  <h:inputText value="#{hello.name}"/><br/>
  <h:commandButton type="submit" value="Say Hello"
    action="#{hello.sayHello}"/>
</h:form>
</ui:define>

</ui:composition>
```

Each EL symbol corresponds to a Java object / method

# The action class

@Name("hello")

```
public class HelloAction {  
    String name;  
    public String getName() {return name;}  
    public void setName(String name) {this.name = name;}  
  
    public void sayHello () { name = name.toUpperCase();}  
}
```

- How it works:
  - Seam creates a new HelloAction instance and put it under name "hello" for each request
  - The object is destroyed after the response is completely rendered
  - There are other scopes available
  - You do not create new objects by hand

# What we learned so far



- Look ma, no XML!
- Bi-directional Java binding to web page
- The action class is a fully fledged Seam component
  - Business logic, business process, rules are encapsulated in the same class
  - Fully object oriented and easily tested
  - The name “action” is just for Struts and RoR developers



# Support persistence

```
<ui:composition ...>
```

```
<ui:define name="body">
```

```
<h:form>
```

```
<h:inputText value="#{person.name}"/><br/>
```

```
<h:commandButton type="submit" value="Say Hello"  
    action="#{hello.sayHello}"/><br/>
```

```
The follow people said hello! <br/>
```

```
<h:dataTable value="#{fans}" var="fan">
```

```
<h:column><h:outputText value="#{fan.name}"/></h:column>
```

```
</h:dataTable>
```

```
</h:form>
```

```
</ui:define>
```

```
</ui:composition>
```

# The entity and action beans

```
@Entity
@Name("person")
public class Person implements Serializable {

    private long id;
    private String name;

    @Id @GeneratedValue
    public long getId() { return id;}
    public void setId(long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
@Name("hello")
public class HelloAction {

    @In @Out private Person person;

    @Out private List <Person> fans;

    @In (create=true) private EntityManager em;

    public void sayHello () {
        em.persist (person);
        person = new Person ();
        fans = em.createQuery(
            "select p from Person p")
            .getResultList();
        return null;
    }
}
```

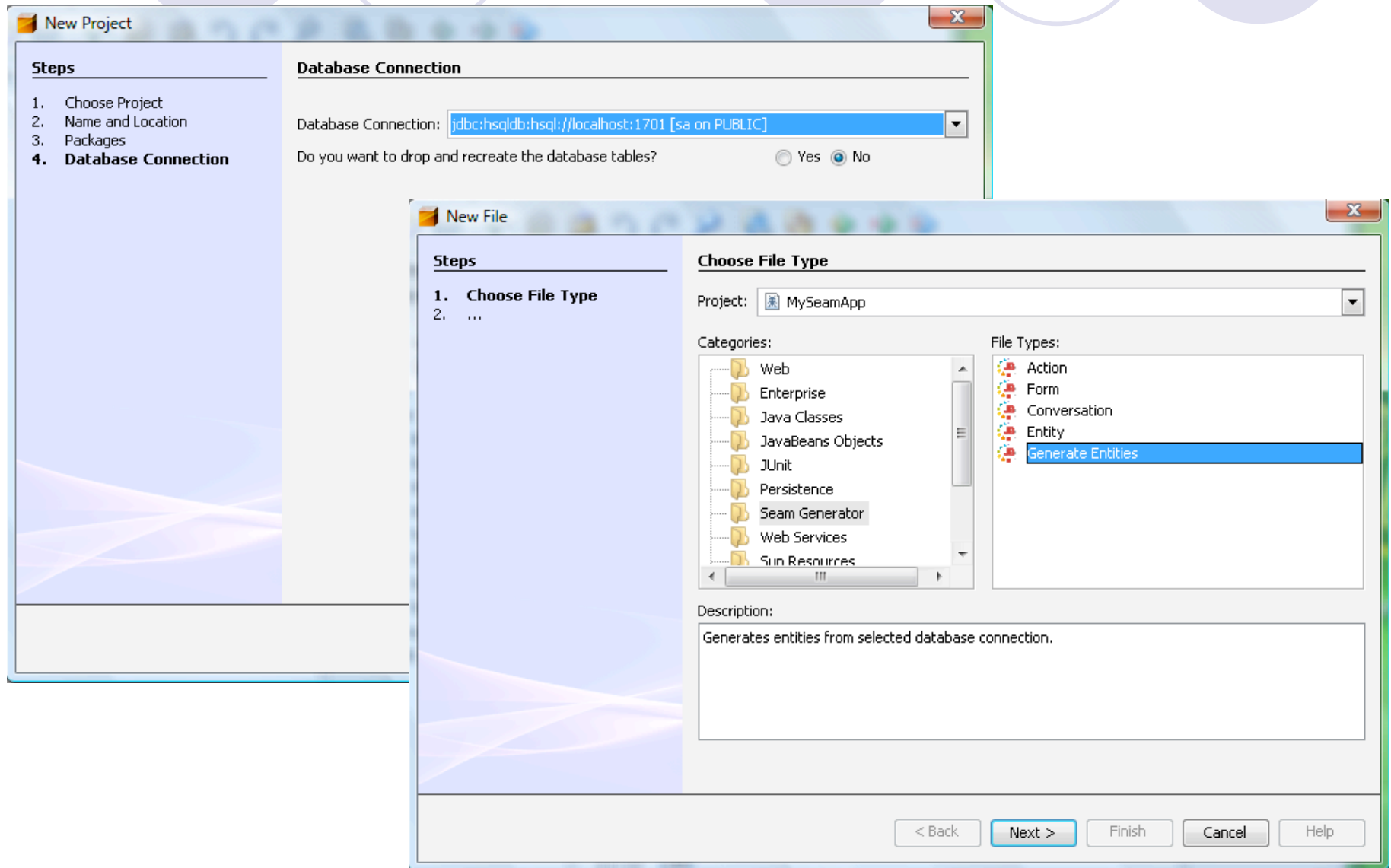
Notice how bi-jection works in the action class

# The rich domain model



- The “entity bean” is now a fully fledged Seam component
  - It can back UI or business logic
  - Just like any other POJOs in Seam
- You can add transient logic to the entity objects
  - The methods can be invoked from web pages (via EL) or from other Seam components (via bijection)
  - Encapsulate data with behavior!
  - True OO programming model!

# Now, something more interesting



# The generated app in action

[Home](#) | [Payments List](#) | [Offices List](#) | [Orders List](#) | [Orderdetails List](#) | [Customers List](#) | [Productlines List](#) | [Employees List](#) | [Products List](#)

Welcome, demo | [Logout](#) |


## MySeamApp

### Payments

Generated edit page

**paymentnumber** \*  value is required

**checknumber** \*

**paymentdate** \*   value is required

**amount** \*

\* required fields

**customers** \*

No customers

April 2007						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
13 April 2007						

Select customers

# Key features



- Reverse engineer interlinked database tables to Hibernate / JPA POJOs
  - Foreign keys, composite keys are supported
- Generate CRUD pages for each table and its associated tables
- Fully Ajax-based validation and data tables
- RESTful URLs are supported for each table
- Security is already built in

# Agenda

- What is Seam and who should use it?
- 0 to 60
- Seam is made for JSF
- Conversations and business processes
- Security made easy
- The future

# JSF: what's hot



- Fully component-based web framework
  - Event-based UI programming model
  - Great for visual tools support
  - Free and commercial component libraries
- Rich interaction model
  - Many places for external frameworks to plugin
- Integrated input validation / conversion
- Unified Expression Language (EL)
- Multiple rendering output from same pages
- Non-JSP template engines available
- Official Java EE standard



# JSF: what's not



- Does not follow Java EE 5 programming model
  - No special integration with EJB3, WS, JMS
  - Use verbose and repetitive XML instead of annotations
- Not “web friendly”
  - HTTP GET and RESTful URLs are hard
  - Hard to integrate custom JavaScript libraries
- Poor exception handling and error page redirection
- Hard to test out of the container

# Key Seam enhancements



- Reduces boilerplate code
- No XML hell
- Integrated ORM support
- RESTful URLs and page actions
- Extended use of JSF EL
- Direct JavaScript integration for AJAX
- Elegant input validation
- Graceful exception handling
- Full support for “Redirect after Post”
- Very easy to test



# Reduces boilerplate code

- No JSF backing beans
  - Use transactional components directly on web pages or in other components
  - There is only one kind of Seam component!
- Bi-jection of named components
- Simplified navigation rules
  - Navigation based on state not returned strings
- Configuration by exception

# No XML hell



- Do not use XML for code
- XML is used in the following scenarios
  - Configure the framework components
  - Configure web page parameters and actions
  - Configure business processes and pageflows

# Integrated ORM support



- Lazy loading just works
  - Eliminate the dreaded lazy loading exception
  - Built-in support for “open session in view” pattern
  - No need for complex DTOs between layers
- Automatic transaction rollback
  - Based on method return value or exceptions
- Can keep database session open across multiple requests (conversation and web transaction)

# RESTful web applications

- Inject HTTP GET parameters directly to Seam components
- Invoke arbitrary Seam methods when loading the page
- Seam's JSF tags (e.g., <s:link>) renders into direct HTTP GET links

```
<page view-id="/person.xhtml">
  <param name="pid" value="#{manager.pid}"
    converterId="javax.faces.Long"/>
</page>
```

```
@Name("manager")
public class ManagerAction {

    public void setPid (Long pid) {
        this.pid = pid;
        if (pid != null) {
            person = (Person) em.find(Person.class, pid);
        } else {
            person = new Person ();
        }
    }
}
```

# Expand JSF EL

- Expand the EL syntax

```
<h:inputText value="{person.name}"/><br/>  
<h:commandButton type="submit" value="Save"  
    action="{hello.sayHello(person)}/>
```

- Expand the EL usage

- web pages
- annotation parameters
- XML configuration files
- TestNG test scripts

# Seam and Ajax

- Use Ajax enabled JSF components (e.g., RichFaces and IceFaces)
- Use Ajax4jsf to wrap regular JSF components
- Use Seam remoting to directly access Seam component from JavaScript
  - Very much like DWR
  - All popular JavaScript libraries supported
- Use the Seam GWT integration servlet



# Elegant input validation

- Use Hibernate validators on data model
- Decorate the invalid fields
- Ajax validator supported out of the box

```
@Entity
@Name("person")
public class Person implements Serializable {

    private String name;
    @NotNull
    @Pattern(regex="^[a-zA-Z.-]+ [a-zA-Z.-]+$",
        message="Need a firstname and a lastname")
    public String getName() { return name; }
    public void setName(String name) {this.name = name;}
}
```

```
<s:decorate>
    <h:inputText value="#{person.name}"/>
</s:decorate>
```

# Failing Gracefully

- Redirect any exception to any page
- Configure error pages on a per-page basis
- Redirect to previous page upon login exception

```
<exception class="java.lang.RuntimeException">  
  <redirect view-id="/generalError.xhtml">  
    <message>Unexpected failure</message>  
  </redirect>  
</exception>
```

```
@ApplicationException(rollback=true)  
@Redirect(viewId="/inventoryError.xhtml")  
public class InventoryException extends Exception {  
  
  public InventoryException () { }  
  
}
```

# Redirect after Post

- A very popular design pattern to avoid “double submit”
- JSF partially supports it via the <redirect/> navigation rule
  - However, JSF messages are not passed to the redirected page
  - Making it hard to use in current JSF
- Seam stateful context maintains JSF messages across re-directs

# Seam testing framework

```
@Test
public void testSayHello() throws Exception {
    new FacesRequest("/hello.jsp") {

        @Override
        protected void updateModelValues() throws Exception {
            setValue("#{person.name}", "Michael Yuan");
        }

        @Override
        protected void invokeApplication() {
            assert getValue("#{person.name}").equals("Michael Yuan");
            assert invokeMethod("#{manager.sayHello}") == null;
            assert getValue("#{person.name}") == null;
        }

        @Override
        protected void renderResponse() {
            List<Person> fans =
                (List<Person>) getValue("#{fans}");
            assert fans!=null;
            assert fans.get(fans.size()-1)
                .getName().equals("Michael Yuan");
        }
    }.run();
}
```

TestNG-based

The entire JSF lifecycle  
as well as database session  
are mocked

# Agenda



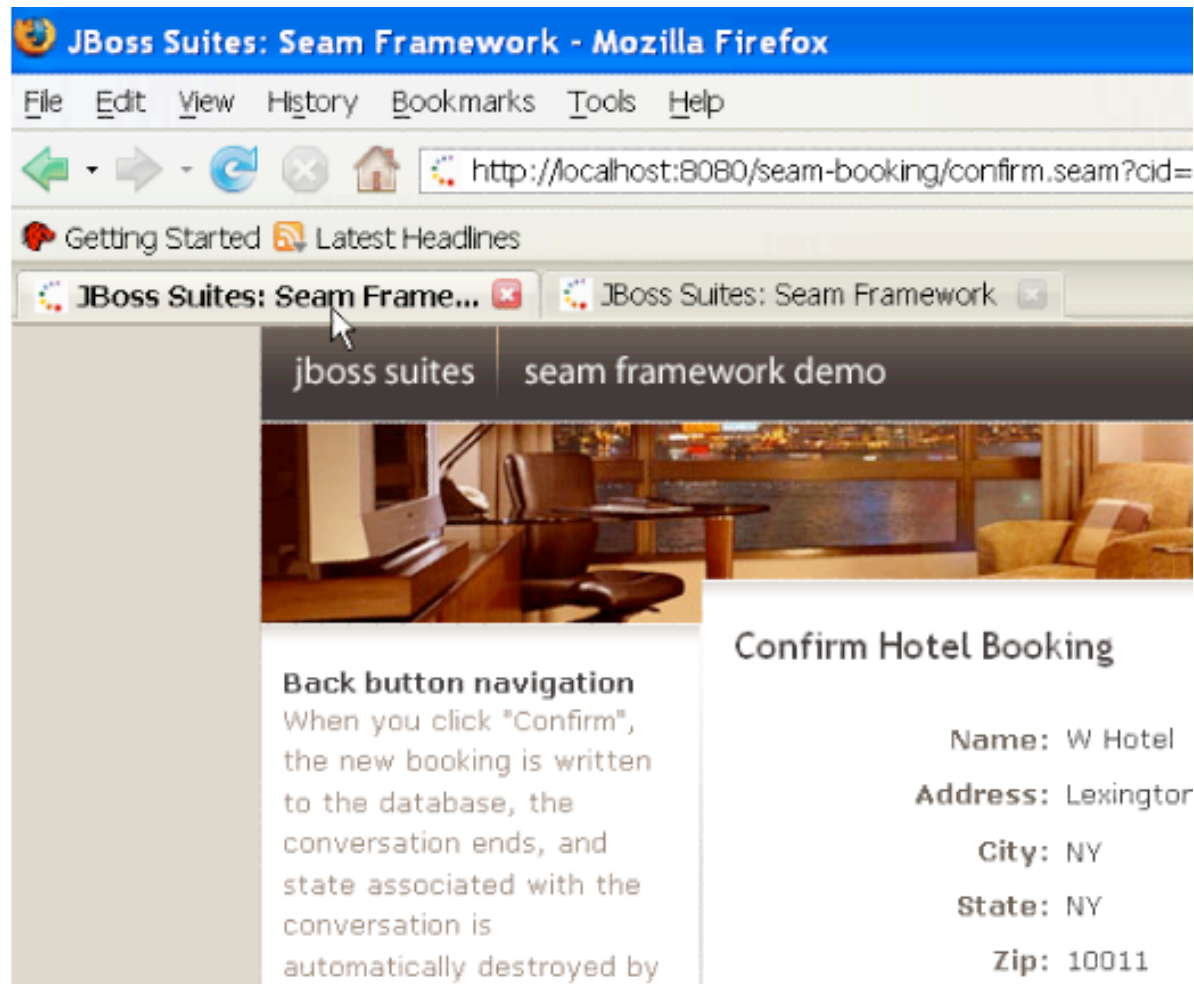
- What is Seam and who should use it?
- 0 to 60
- Seam is made for JSF
- Conversations and business processes
- Security made easy
- Conclusions

# Conversations



- More finely grained than HTTP session
- Support “web transaction”
- Support multiple browser windows / tabs
- Support BACK buttons
- Great for wizards, shopping carts etc.

# Conversation in action



**JBoss Suites: Seam Framework - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

http://localhost:8080/seam-bookings/confirm.seam?cid=...

Getting Started Latest Headlines

JBoss Suites: Seam Frame... JBoss Suites: Seam Framework

jboss suites seam framework demo

**Back button navigation**  
When you click "Confirm", the new booking is written to the database, the conversation ends, and state associated with the conversation is automatically destroyed by

**Confirm Hotel Booking**

**Name:** W Hotel  
**Address:** Lexington  
**City:** NY  
**State:** NY  
**Zip:** 10011



# Business process

- Manage long-lived objects in the “process scope”
- Map business process actors and actions to web actions via JSF EL
- Integrate with JBoss Rules (a.k.a Drools)



# Agenda



- What is Seam and who should use it?
- 0 to 60
- Seam is made for JSF
- Conversations and business processes
- Security made easy
- The future



# Finely-grained security

- Username/password authentication
  - Authenticate against any backend
  - Authentication logic in Java code
- Finely-grained access control
  - Web pages
  - Elements on a web page
  - Java methods
- Rule-based instance-level access control

# Protect a web page

```
Username: <h:inputText value="#{identity.username}"/>
Password: <h:inputSecret value="#{identity.password}"/>
<h:commandButton id="login" action="#{identity.login}" value="Account Login"/>
```

```
public boolean authenticate() {
    List results = em.createQuery("select u from User u where
                                   u.username=#{identity.username} and
                                   u.password=#{identity.password}")
                        .getResultList();
    if ( results.size()==0 ) {
        return false;
    } else {
        user = (User) results.get(0);
        return true;
    }
}
```

```
<page viewId="/book.xhtml"
      login-required="true">
    ... ..
</page>
```

```
<components>
    ... ..
    <security:identity
        authenticate-method="#{authenticator.authenticate}"/>
</components>
```

# Role-based access control

```
public boolean authenticate() {  
    List results = em.createQuery("select u from User u where  
                                u.username=#{identity.username} and  
                                u.password=#{identity.password}")  
                        .getResultList();  
    if ( results.size()==0 ) {  
        return false;  
    } else {  
        user = (User) results.get(0);  
        if (user.getRoles() != null) {  
            for (UserRole mr : user.getRoles())  
                identity.addRole(mr.getName());  
        }  
        return true;  
    }  
}
```

```
@Restrict("#{s:hasRole('admin')}")  
public void someAction () {  
    ... ..  
}
```

```
<s:div rendered="#{s:hasRole('admin')}">  
    ... restricted content ...  
</s:div>
```

```
<page viewId="/audit.xhtml">  
    <restrict>#{s:hasRole('admin')}</restrict>  
    ... ..  
</page>
```

# Agenda

- What is Seam and who should use it?
- 0 to 60
- Seam for JSF web developers
- Conversations and business processes
- Security made easy
- The future

# The future



- Seam is moving extremely fast
  - New release every 2 months
  - Many new features for each release
  - One of the most active Java open source project today
- It is hard to keep up with it
- But the core API and programming model has been stable since 1.2 release
- JBoss EAP 4.2 includes Seam 1.2

# Seam 1.3 features



- Web services and ESB / SOA support
- Integration of new frameworks
  - Groovy, GWT, Quartz