JBoss Seam:一个深度集成框架（一）

作者: Michael Yuan **翻译:** Richard **来源:** TSS

作者简介: Michael Yuan，技术专家，《JBoss Seam: Simplicity and Power Beyond Java EE》、《Lightweight Java Web Application Development》等书的作者，软件顾问，目前就职于 JBoss。

摘要: 本文介绍了 JBoss Seam 的产生背景以及集成框架的概念，并且演示了在 JBoss Seam 中如何使用 POJOs 处理 JSF 事件以及实现双向依赖注入。

本文是《JBoss Seam：一个深度集成框架》一文的第一部分，**JBoss Seam中文站**将陆续发布该文的其他部分。

1. JBoss Seam 的产生背景

软件框架对于企业级 Java 开发者来说，是非常有用的工具。它们被广泛地用来组装可重用的软件组件和服务。

每个框架都提供了一系列的设计模式、APIs 和组件模型——用于构建在框架之上的应用程序。许多受欢迎的 Java EE 框架的实例，既包括 Hibernate、Spring 和 Struts 这样的开源项目，又包括基于 Servlet/JSP、JSF、EJB、 JMS、Web Services 等标准实现的产品。

一个典型的企业级 Java 应用程序，可以同时使用多个框架。所以，Java EE 开发者们的主要竞争能力之一，就体现在使用这些框架的能力上。

然而，存在太多框架的缺点在于，每一种框架都提供了不同的编程模型（或组件模型）。而要在一个 Web 应用中使用不同的组件模型，开发人员一般需要写很多“胶水”代码（例如数据传输对象、组件管理等）以及“胶水”配置文件，这样将会大大地降低开发效率。

在这种背景下，作为“集成框架”的 Seam 应运而生。Seam 的目标就是减少这些“胶水”代码，整合现有的各种框架，为企业级应用提供一个一致的编程模型。

2. 什么是集成框架？

Java EE 本身就可以被认为是一个集成框架。它定义了不同的框架，如 Servlet/JSP、JSF、EJB、JMS、JTA，、JCA 等，并使它们在企业级应用中携手合作。

但是作为标准规范的 Java EE 由于自身的设计问题，导致发展非常缓慢。新的理念和前沿技术在成为 Java EE 标准之前，通常都已经作为开源项目出现了。

Spring 框架是另一个被广泛应用的集成框架。Spring 对许多现有的框架提供了轻量级的封装，开发人员可以通过配置 XML 文件来管理应用程序中的组件。但是开发人员仍需要自己混合和匹配多个不同的组件编程模型；另外，Spring “大量 XML 配置”的方式也因为包含太多的“XML 代码”而变得繁琐。

Seam 是一个开源的“深度集成”框架，它试图吸收 Java EE 和 Spring 世界中的精华为己所用。

Seam 牢牢地扎根于 Java EE 标准：JSF 和 EJB3。

Seam 的最初目标是为了解决 JSF 和 EJB3 中的一些设计缺陷。随后，Seam 的许多核心特性被采纳为将来的官方 Java EE 标准，例如 JSF 2.0 和 WebBeans。随着越来越多的用户开始采用 Seam，它已经远远超过 Java EE 的范畴。

Seam 走了一条与早期的 Spring 框架不同的路。Seam 为被它集成的所有框架提供了一个统一的组件模型，开发人员可以使用统一的 Seam 组件，而不需要学习每个独立框架的组件管理 APIs。

对于开发人员来说，Seam API 与被 Seam 集成的框架相比，有非常大的改进。Seam 的注释（annotation）/API 设计和 Java EE5 非常相似。

Seam 集成的框架可能成为“幕后工作者”。在某些情况下，你在使用 Seam 的同时，却要“奔波”在其它相互竞争的框架之中——与 Java EE 允许相同的 API 有多种实现是同样的方式。

现在，所有关于集成方面的论述都显示乏味和抽象。下面，我们将通过一些简单的例子让你深刻地体会到 Seam 是如何使开发变得轻松容易的。

3. 运用 EJB3 Beans 或 POJOs 处理 JSF 事件

Seam 最初引人注目的特性就是直接使用 EJB3 作为 JSF 的 Backing Beans。

JSF 和 EJB3 都是 Java EE 的关键技术，然而，Java EE 却没有很好地整合这两个框架。它们具有不同的组件模型：JSF 使用基于 POJOs 的“Backing Beans”处理 UI 事件，并且需要进行大量的 XML 配置；EJB3 使用注释型 POJOs 表示持久层和业务逻辑，只需要简单的 XML 配置。

一般情况下，如果要在 JSF 页面中进行数据库操作，需要写 JSF Backing Beans 去处理 UI 事件，然后调用 EJB3 会话 Bean 中的方法。

因此，如果开发人员需要在 Backing Beans 和 EJB3 Beans 之间传递复杂的数据，通常需要在这些框架中创建 DTOs(数据传输对象)。这样，整合 JSF 和 EJB3 框架将成为繁琐的事情。

在 Seam 中，你可以直接使用 EJB3 Beans 作为 JSF 的 Backing Beans。举个例子，下面的 JSF 页面引用一个名为“echo”的 Seam 组件。当用户点击 Echo 按钮时，页面将会把输入变成大写形式，并显示在页面上。

```
My name is: <h:inputText value="#{echo.name}"/><br/>
Hello #{echo.name} <br/>
<h:commandButton type="submit" value="Echo"
                 action="#{echo.toUpperCase}"/>
```

如果你还不是很熟悉 JSF，下面将介绍一下页面代码是如何工作的：

当服务器生成页面的时候，#{echo.name}被替换成 echo 对象中 getName()方法的返回值。当你点击 Echo 按钮时，服务器首先调用 setName()方法将输入框的值填入#{echo.name}，然后调用 toUpperCase()方法。

#{…}格式的表达式语言(EL)是 Seam 编程模型中的关键组件，因为它不仅可以在网页中使用，也可以在任何的文本环境中使用，比如 XML 配置文件、流程/规则定义、测试脚本等，我们将在后面介绍这些。

在单纯的 JSF 中，echo 组件将被作为 JSF 的 Backing Beans。而在 Seam 中，我们可以直接使用 EJB3 bean（或简单的 POJO）来实现该组件。你只需要通过@Name 将组件注释成相应的名字。当存在 Web 请求时，Seam 用“echo”名字创建一个 EchoBean 实例，并将所有输入数据传递给它，调用相应的方法；最后当响应返回时销毁这个组件。在这个过程中，你不需要写 XML 代码或有关对象生命周期管理的代码。

```
@Stateful
@Name("echo")
public class EchoBean implements EchoInt {
  private String name;
  public String getName () { return name; }
  public void setName (String name) { this.name = name; }
  public void toUpperCase () { name = name.toUpperCase (); }
  @Remove destroy () {}
}
```

当然，对于这个简单的应用，EJB 显得有些太臃肿。EJB 需要接口、方法和注释。最简单的方法是写一个 POJO 类，然后在这个类上面加上@Name 注释就可以了。

```
@Name("echo")
```

```
public class EchoPojo {

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

在 Seam 中，注释 POJOs 可以完全取代 EJB3 中的会话 Beans，我们将在后面的内容中使用 Seam POJOs。

4. 双向依赖注入

我们将通过一个多组件交互的例子，来领略一下 Seam 的组件管理能力。

页面由与实体 Bean 相联的输入框、与 POJO 相联的按钮和由 List 对象构成的数据表格组成：

```
Your name: <h:inputText value="#{person.name}"/><br/>
<h:commandButton type="submit" value="Say Hello"
              action="#{manager.sayHello}"/>
<h:dataTable value="#{fans}" var="fan">
  <h:column>
    <h:outputText value="#{fan.name}"/>
  </h:column>
</h:dataTable>
```

下面是 Person 实体 Bean 和管理类 POJO：

```
@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("manager")
public class Manager {
  @In @Out
  private Person person;
  @Out
  private List <Person> fans;
  @In
  private EntityManager em;
  public String sayHello () {
    em.persist (person);
```

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

在管理类 POJO 中，当 sayHello()方法运行时，person 组件被注入（inject）；当方法退出时，fans 组件被注出（outject）。

同样地，JPA 实体管理类也可以被注入，提供对数据库的支持。注入和注出（双向依赖注入）可以在当前页面请求范围内同时发生。所以在 sayHello()方法使用用户输入的#{person.name}值的同时，fans 的值也将在结果页面中显示。

双向依赖注入让我们可以利用简单的注释来管理组件间的复杂关系。

从集成的角度来看，这个例子过于简单。Person 实体 Bean 直接联系 UI 元素。实际上，实体 Bean 仅仅是一个 Seam 组件，你也可以用它来连接 UI 行为（例如点击按钮）。

这样，不同框架的组件之间的“独裁统治”将被打破；更远点说，如果在实体 Bean 中加入对业务逻辑的支持，就能够建立包括数据和行为的富领域模型。

富领域模型在 OOP 中已经被实现，但是在 Seam 之前，还没有被 Web 框架使用。

审校：骆驼

JBoss Seam：一个深度集成框架（二）

作者: Michael Yuan **翻译:** Richard **来源:** TSS

作者简介: Michael Yuan，技术专家，《JBoss Seam: Simplicity and Power Beyond Java EE》、《Lightweight Java Web Application Development》等书的作者，软件顾问，目前就职于 JBoss。

摘要: 本文介绍了 JBoss Seam 如何在 JSF 中进行 JPA 延迟加载和 Hibernate 验证以及 JBoss Seam 对 Ajax 的支持。

本文是《JBoss Seam：一个深度集成框架》一文的第二部分，JBoss Seam中文站将在近期发布该文的第三部分。

5. 支持在 JSF 中进行 JPA 延迟加载

ORM 框架重要特征之一就是支持相关对象的“延迟加载”。没有延迟加载，即使简单的对象查询也有可能顺带出大量数据到结果集中，因为在映射对象图中，可能所有的表都是潜在关联的。

例如，在显示订单列表的 Web 应用程序中，你可能需要在控制器中查询订单对象。然后，当页面被加载后，你可能需要显示列表中相应的条目。

在页面加载的时候，我们将“延迟加载”订单条目到订单对象中，而不会在控制器中查询订单对象时，就将所有与订单关联的对象都加载到订单对象中。

然而，在传统的 MVC 框架中应用“延迟加载”可不是一件容易的事情。在 Seam 之前的 MVC 框架中，控制器会在持久化环境中通过会话在事务中运行查询，然后在事务提交时关闭会话。所以，当控制器退出、网页显示完成后，持久化环境将变成不可用状态。

这样，在以前的 MVC 框架中，如果想在显示页面中试图“延迟加载”数据，持久化引擎将抛出“lazy initialization exception”异常。

你可以通过“Open Session In View”模式实现“延迟加载”——在页面生成阶段保持持久会话。但是应用这种模式，需要在 Web 框架和持久层框架之间编写大量的整合代码。

Seam 默认支持“Open Session In View”。所有的 Seam 组件，除了 EJB3 中的无状态会话 Beans 以外，都是有状态的。在开始从 UI 事件调用数据库操作一直到响应页面生成期间，Seam 将维持一个打开的持久会话。在 Seam 应用中，开发人员不需要编写额外的代码来实现“延迟加载”。

6. 在 JSF 输入页面中支持 Hibernate 验证

在多层企业级应用中，Web 框架和 ORM 持久层框架通常会有不同的数据验证机制。Web 框架在 Web 表单被提交时验证用户输入，而持久层框架在保存数据进数据库之前验证数据。在大多数情况下，它们显得多余。

Seam 允许你在实体 Beans 中直接注释数据验证约束，这样使用与实体 Beans 相联系的 JSF 输入框时，同样的验证约束将应用于输入数据。

下面的例子中，Person 对象的名字必须由两个单词组成，并且年龄必须在 3 到 100 岁之间。

```
@Entity
@Name("person")
@Table(name="extperson")
public class Person implements Serializable {
  private long id;
```

```
    private String name;
    private int age;
    @Id @GeneratedValue
    public long getId() { return id;}
    public void setId(long id) { this.id = id; }
    @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;}
    @NotNull
    @Range(min=3,  max=100,
       message="Age must be between 3 and 100")
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
  }
```

下面的 JSF 页面将自动“包装”一些验证逻辑。如果用户提交无效值，将重新显示页面，同时高亮显示无效的字段。

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

你可以通过简单的 JSF facets 和 CSS 样式配置这些高亮显示的错误信息。也可以在无效字段前加入有 CSS 样式修饰的错误提示图片。如下所示，当验证失败时，<s:message/>将显示验证注释中的信息属性。

```
  <f:facet name="beforeInvalidField">
    <h:graphicImage styleClass="errorImg" value="error.png"/>
  </f:facet>
  <f:facet name="afterInvalidField">
    <s:message styleClass="errorMsg" />
  </f:facet>
  <f:facet name="aroundInvalidField">
    <s:div styleClass="error"/>
  </f:facet>
```

使用基于 Ajax 的 JSF 控件，你无需提交任何表单，就可以进行输入字段的验证。

7. 多种方式使用 Ajax

作为一个现代的应用程序框架，Seam 为 Ajax 应用提供了最好的支持。

在 Seam 中，你可以通过很多方式使用 Ajax。初学者可以使用 Seam 集成的 Ajax JSF 组件包，比如 Ajax4jsf、RichFaces 和 IceFaces。它们提供了一系列 Ajax 功能的 JSF 控件，包括输入框、数据表格、交互面板、拖放面板等，你可以直接将它们用在你的页面上。这些 Ajax 控件允许你开发 Ajax Web 应用程序，而无需写一行 JavaScript 代码。Seam 已经帮你做好了所有和 Ajax 整合的工作，使得建立 Ajax 应用比在单纯的 JSF 环境中更容易。下面的例子显示了如何使用 Ajax 数据输入框，当你离开输入框时，数据将被验证，同时无效的字段将高亮显示。

```
<s:validateAll>
  Your name:<br/>
  <a4j:outputPanel id="nameInput">
    <s:decorate>
      <h:inputText value="#{person.name}">
        <a4j:support event="onblur" reRender="nameInput"/>
      </h:inputText>
    </s:decorate>
  </a4j:outputPanel>
  Your age:<br/>
  <a4j:outputPanel id="ageInput">
    <s:decorate>
      <h:inputText value="#{person.age}">
        <a4j:support event="onblur" reRender="ageInput"/>
      </h:inputText>
    </s:decorate>
  </a4j:outputPanel>
</s:validateAll>
```

考虑到开发人员想直接使用 JavaScript 取代 JSF 控件，Seam 同样也整合了 JavaScript 和服务器端组件。你可以像调用本地 JavaScript 方法一样调用 Seam 组件的方法并且在本地使用返回值。开发人员也可以通过 Seam 服务整合其他流行的 JavaScript 库。例如，下面的代码显示了如何通过 Seam 组件整合 Dojo 的“内嵌文本编辑器”，当你双击“Hello World”文本时，将出现内嵌的文本编辑器；编辑完成后，输入值将被传回服务器处理。

```
<script language="javascript">
  // Seam.Remoting.setDebug(true);
  // don't display the loading indicator
  Seam.Remoting.displayLoadingMessage = function() {};
  Seam.Remoting.hideLoadingMessage = function() {};
  // Get the "manager" Seam component
  var manager = Seam.Component.getInstance("manager");
  function init () {
    var commentEditor = dojo.widget.byId("comment");
    commentEditor.onSave = submitComment;
  }
  function submitComment (newValue, oldValue) {
    manager.setComment (newValue);
  }
  dojo.addOnLoad(init);
</script>
<div id="comment" dojoType="inlineEditBox">Hello Seam</div>
```

下面代码显示了通过 JavaScript 远程调用访问 Manager 组件。

```
@Name("manager")
public class Manager {
  @In @Out
  private Person person;
  public void setComment (String comment) {
    person.setComment (comment);
  }
  ... ...
}
```

审校：骆驼

原创文章如转载，请注明：转载自JBoss Seam中文站
[http://www.jbossseam.com/]
本文链接地址：http://www.jbossseam.com/2007/11/03/introduction-to-jboss-seam-part2/

JBoss Seam：一个深度集成框架（三）

作者: Michael Yuan **翻译:** Richard **来源:** TSS

作者简介: Michael Yuan，技术专家，《JBoss Seam: Simplicity and Power Beyond Java EE》、《Lightweight Java Web Application Development》等书的作者，软件顾问，目前就职于 JBoss。

摘要: 本文介绍了 JBoss Seam 如何集成业务流程、使用 iText 和任务调度，并且总结了 Seam 编程模型中的关键要素。

本文是《JBoss Seam：一个深度集成框架》一文的最后一部分。

8. 在 Web 应用中集成业务流程

大部分企业级应用存在许多业务流程和规则。例如，在一个简单的电子商务网站（以“在线购物”为例）中，客户登录后进行购物流程，商店管理人员登录后进行审批流程，仓库职员登录后进行发货流程。不同的人员站在不同的角度，去执行不同的任务，然而，他们又同时合作完成同一个业务场景。

在企业级应用中，业务分析人员通常定义业务流程和规则。他们使用专业的业务流程软件绘制这些流程和规则，然后应用开发人员实现这些设计。

然而，由于大部分 Web 应用框架没有集成流行的业务流程和规则引擎，开发人员只能通过自己的方式进行业务流程的整合。这样势必会造成开发人员和业务分析人员工作的脱节，使得业务分析人员很难去审核和验证。

Seam 通过 jBPM 和 JBoss Rules（以前的 Drools）对业务流程和规则的整合提供了极好的支持。

在 Seam 应用中，你可以指定 UI 动作（例如按钮点击）来触发业务流程。你只需要通过@CreateProcess 注释来标记 UI 事件处理方法。业务流程是不同的用户以相应顺序来完成的一系列任务。你可以用@BeginTask 和@EndTask 标注任务的开始和结束。当前任务结束时，jBPM 引擎将自动把进程前移，进行下一个任务。

```
@Name("ticketSystem")
public class TicketSystemAction {
  @CreateProcess(definition="TicketProcess")
  public String newTicket() {
     return "home";
  }
  @BeginTask
  public String reply() {
    return "reply";
  }
  @EndTask
  public String sendAnswer() {
     System.out.println("Answered");
    return "home";
  }
}
```

Seam 让每个用户可以查看他/她的当前任务列表以及完成任务的下一个动作。这些任务列表是基于当前登录的用户角色生成的，并且通过用户认证和授权的方式紧密地整合到 Seam 安全框架中。

```
<h1>Assigned Tickets - #{login.user.username}</h1>
```

```
<h:dataTable value="#{taskInstanceList}" var="task">
  <h:column>#{task.description}</h:column>
  <h:column>Title: #{ticket.title}</h:column>
  <h:column>
    <h:commandLink action="#{ticketSystem.reply}">
      <h:commandButton value="Reply"/>
        <f:param name="taskId" value="#{task.id}"/>
    </h:commandLink>
  </h:column>
</h:dataTable>
```

在整合 jBPM/JBoss Rules 的 Seam 应用中，开发人员可以直接使用 Seam 注释和组件驱动业务流程和规则引擎，而不需要单独掌握特定的 jBPM 和 JBoss Rules 的 Java APIs。

9. 使用 iText 生成不同的视图

iText 库是一套被广泛用于生成 PDF 文档的开源 Java 库。然而，使用 iText API 创建 PDF 文档是十分耗时的（想想用 DOM 创建 XML 文档或者用 Swing 写 UI 的经历）。

Seam 整洁地整合了 iText、JSF 和 Facelets，开发人员可以通过和生成 JSF 页面一样简单的方式，将动态的内容生成 PDF 页面，你甚至可以在 PDF 页面中使用模板。

Seam 为 PDF 元素创建了特殊的 XHTML 标记库，然后在生成页面的时候透明地调用 iText。下面的示例显示了如何在 Seam 应用中，生成有数字签名支持的 PDF 页面。

```
<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>
```

通过代码，我们可以看到整合是无缝的，页面不依赖于 iText。实际上，将 iText 替换成其他的商业 PDF 库，页面仍然可以工作，这就是 Seam 整合的魅力。

10.高级任务调度程序

在许多企业级应用中，对自动重复任务的支持是相当重要的。在标准的 EJB 中，你可以使用 EJB Timer API 在固定的时间间隔内调度重复的事件。然而，在实际的应用中，我们需要比固定的时间间隔触发更高级的调度服务。

目前流行的开源 Java 调度库是 Quartz 库。但是如果要使用 Quartz 的话，开发人员仍需要自己写“胶水”代码来整合 Quartz 特定的 APIs 和对象模型。

Seam 整合了 Quartz，用于调度异步重复任务。你只需要在重复工作的方法上添加@Asynchronous 注释。

你可以传入任务的开始/结束时间、间隔或者克龙表达式(cron string)的字符串作为参数，也可以在方法定义中注释这些特定目的的参数。指定的方法将返回 QuartzTriggerHandler 对象，你可以稍后使用这个 QuartzTriggerHandler 对象暂停或取消任务，你也可以将这个 QuartzTriggerHandler 对象保存到数据库，以供稍后使用。

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

下面的例子中，schedulePayment()方法设定在下午 2 点 10 分和三月每个星期三的下午 2 点 44 分运行。你可以在 Web UI 事件处理方法中加入这段调用的代码，这样当按钮按下时，重复事件将被安排到调度程序中。

```
QuartzTriggerHandle handle =
   processor.schedulePayment(payment.getPaymentDate(),
                        "0 10,44 14 ? 3 WED”,
                        payment.getPaymentEndDate(),
                        payment);
payment.setQuartzTriggerHandle( handle );
// Save payment to DB
// later ...
// Retrieve payment from DB
// Cancel the remaining scheduled tasks
payment.getQuartzTriggerHandle().cancel();
```

从例子可以看出，开发人员不需要手动启动 Quartz 调度程序、创建 Quartz 触发器和任务，而只需要使用 Seam 注释 POJOs 就可以了。

11.统一的编程模型

目前为止，我们已经介绍了很多 Seam 通过一致的编程模型整合不同框架的例子。除了上面介绍的这些，还有许多其他的框架。但是限于篇幅，我们不可能介绍所有的框架。下面我们将总结一下 Seam 整合这些框架的方法。Seam 编程模型中关键的三要素是：

注释 POJOs：Seam 应用中所有的 Java 组件都是注释的 POJO 类。Seam 通过双向依赖注入管理它们之间的交互。除此之外，Seam 中没有其他的组件模型。

XHTML 显示页面：所有视图(UI)页面都是通过 XHTML 文件显示出来，除去标准 JSF 标签，Seam 还定义了许多自己的 UI 标签，包括 PDF UI 标签等。Seam 同时也加入了 Ajax JSF 库，比如 Ajax4jsf、RichFaces 和 IceFaces。

表达式语言：XHTML 页面通过 JSF 表达式语言(EL)引用 Seam 中的 Java 组件。Seam 增强了标准的 EL 语法，使它支持方法参数等，并且使 EL 可以用于所有的 XML 配置文件和测试脚本。

有了这些 Cool 的特性，Seam 的编程模型将变成异常简便。只要有一些 JSF 基础，你的学习曲线将非常平坦。

下载 Seam，看看实例，快乐地编写 Seam 代码，一切就这样简单！

审校：骆驼