详细介绍JSF框架技术中使用的设计模式 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022__E8_AF_A6_E7_BB_86_E4_BB_8B_E7_c104_252524.htm 设计模式可以帮助用户在更高层次上抽象细节，更好地理解体系结构。如果比较熟悉 GoF 设计模式和 JavaServer Faces (JSF) 框架，本文可以帮助您洞察 JSF 框架中使用的设计模式，深入理解其工作原理。 本文探讨了 JSF 框架中使用的设计模式。详细讨论的设计模式包括 Singleton、Model-View-Controller、Factory Method、State、Composite、Decorator、Strategy、Template Method 和 Observer 模式。 设计模式和 JavaServer Faces (JSF) 技术 首先简要地介绍一下模式和 JSF 框架。 模式，设计模式是对问题和解决方案进行抽象的普遍适用的方法。因为模式是所有开发人员和架构师公认的，所以模式可以节约时间和资源。用外行话来说，模式就是关于某个人所共知的问题的经过验证的解决方案。模式可以重用，重用使得解决方案更健壮。 Java Server Faces， JSF 体系结构是一种 Web 应用程序框架。它是 Java Community Process (JCP) 推动的，有望成为 Web 应用程序开发的标准框架。目前用于开发 Web 应用程序的框架有 50 多个，这说明迫切需要实现框架的标准化，这正是 JSF 框架的目标！ 深入剖析 JSF 模式 现在我们来讨论 JSF 体系结构中的各种设计模式。本文将详细讨论 Singleton、Model-View-Controller、Factory Method、State、Composite、Decorator、Strategy、Template Method 和 Observer 设计模式。我将分析每种模式的用途及其在 JSF 框架中的作用。 Singleton 模式 Singleton 模式的目的是保证类只有一个实例被

加载，该实例提供一个全局访问点。当启动具有 JSF 支持的 Web 应用程序时，Web 容器初始化一个 FacesServlet 实例。在这个阶段，FacesServlet 对每个 Web 应用程序实例化 Application 和 LifeCycle 实例一次。这些实例就采用众所周知的 Singleton 模式，通常只需要该类型的一个实例。 使用 JSF 的 Web 应用程序只需要 Application 和 LifeCycle 类的一个实例。LifeCycle 管理多个 JSF 请求的整个生命期。因为其状态和行为在所有请求之间共享，这些对象采用 Singleton 模式合情合理。LifeCycle 维护的 PhaseListeners 也是 Singleton 模式的。PhaseListeners 由所有 JSF 请求共享。在 JSF 框架中可以广泛使用 Singleton 模式，以减少内存占用和提供对象的全局访问。NavigationHandler（用于确定请求的逻辑结果）和 ViewHandler（用于创建视图）也是使用 Singleton 模式的例子。 Model-View-Controller (MVC) MVC 模式的目的是从数据表示（View）中将数据（即 Model）分离出来。如果应用程序有多种表示，可以仅替换视图层而重用控制器和模型代码。类似的，如果需要改变模型，可以在很大程度上不改变视图层。控 制器处理用户动作，用户动作可能造成模型改变和视图更新。当用户请求一个 JSF 页面时，请求发送到 FacesServlet。FacesServlet 是 JSF 使用的前端控制器 servlet。和其他很多 Web 应用程序框架一样，JSF 使用 MVS 模式消除视图和模型之间的耦合。为了集中处理用户请求，控制器 servlet 改变模型并将用户导航到视图。 FacesServlet 是 JSF 框架中所有用户请求都要经过的控制器元素。FacesServlet 分析用户请求，使用托管 bean 对模型调用各种动作。后台（backing）或托管（managed）bean 就是该模型的例子。JSF 用户界面（UI）组

件是视图层的例子。MVC 模式把任务分解给具有不同技能的开发人员，使这些任务能够同时进行，这样 GUI 设计人员就可以使用丰富的 UI 组件创建 JSF 页面，同时后端开发人员可以创建托管 bean 来编写专门的业务逻辑代码。 Factory Method 模式 Factory Method 模式的目的是定义一个用于创建对象的接口，但是把对象实例化推迟到子类中。在 JSF 体系结构中，Factory Method 模式被用于创建对象
。LifeCycleFactory 是一个创建和返回 LifeCycle 实例的工厂对象。LifeCycleFactory 的 getLifeCycle (String LifeCycleId) 方法采用 Factory Method 模式，根据 LifeCycleId 创建（如果需要）并返回 LifeCycle 实例。自定义的 JSF 实现可以重新定义 getLifeCycle 抽象方法来创建自定义的 LifeCycle 实例。默认的 JSF 实现提供默认的 LifeCycle 实例。此外，对于每个 JSF 请求，FacesServlet 都从 FacesContextFactory 得到 FacesContext
。FacesContextFactory 是一个抽象类，公开了 getFacesContext API，JSF 实现提供了 FacesContextFactory 和 getFacesContext API 的具体实现。这是另外一个使用 Factory Method 模式的例子，具体的 FacesContextFactory 实现创建 FacesContext 对象。State 模式 State 模式的目的是在表示状态的不同类之间分配与状态有关的逻辑。FacesServlet 对 LifCycle 实例调用 execute 和 render 方法。LifeCycle 协调不同的 Phrase 以便执行 JSF 请求。在这里 JSF 实现就遵循了 State 模式。如果没有使用这种模式，LifeCycle 实现就会被大量的条件（即“ if ”语句）搅得一塌糊涂。JSF 实现为每个状态（或阶段）创建单独的类并调用 step。phase 是一个抽象类，定了每个 step 的公共接口。在 JSF 框架中定义了六个 phrase（即 step）：RestoreViewPhase

、ApplyRequestValues、ProcessValidationsPhase、UpdateModelValuesPhase、InvokeApplicationPhase 和 RenderResponsePhase。 在 State 模式中，LifeCycle 把 FacesContext 对象传递给 phase。每个阶段或状态改变传递给它的上下文信息，然后设置 FacesContext 本身中的标志表明下一个可能的步骤。JSF 实现在每个步骤中改变其行为。每个阶段都可以作为下一个阶段的起因。FacesContext 有两种标志 renderResponse 和 responseComplete 可以改变执行的顺序。每个步骤执行完成后，LifeCycle 检查上一阶段是否设置了这些标志。如果设置了 responseComplete，LifeCycle 则完全放弃请求的执行。如果经过某个阶段后设置了 renderResponse 标志，JSF 就会跳过剩下的阶段而直接进入 Render Response 阶段。如果这两个标志都没有设置，LifeCycle 就会按顺序继续执行下一步。 