Java核心库中的GoF设计模式示例

问 10年零4个月前　活动 7个月前　浏览了 494k次

672

3214

这个问题的答案是社区的努力。编辑现有答案以改善此职位。它目前不接受新的答案或互动。

我正在学习GoF Java设计模式，我想看看其中的一些实际示例。Java核心库中的这些设计模式有哪些好的示例?

java　oop　设计模式　java-api

15年4月11日于3:40编辑　　社区维基
16转，7位用户56％
unj2

删除/锁定的帖子/评论禁用评论

7个答案

活跃的　最古老的　投票

3186

+250

您可以在Wikipedia中找到许多设计模式的概述。它还提到了GoF提到了哪些模式。我将在这里对其进行总结，并尝试分配尽可能多的模式实现，这些模式实现可在Java SE和Java EE API中找到。

创作模式

抽象工厂 **(通过返回工厂本身的创建方法可识别，该工厂本身又可以用于创建另一个抽象/接口类型)**

- `javax.xml.parsers.DocumentBuilderFactory#newInstance()`
- `javax.xml.transform.TransformerFactory#newInstance()`
- `javax.xml.xpath.XPathFactory#newInstance()`

生成器 **(可通过创建方法返回实例本身来识别)**

- `java.lang.StringBuilder#append()` (未同步)
- `java.lang.StringBuffer#append()` (已同步)
- `java.nio.ByteBuffer#put()` (还 `CharBuffer` , `ShortBuffer` , `IntBuffer` , `LongBuffer` , `FloatBuffer` 和 `DoubleBuffer`)
- `javax.swing.GroupLayout.Group#addComponent()`
- 所有实现 `java.lang.Appendable`
- `java.util.stream.Stream.Builder`

工厂方法 **(可通过返回抽象/接口类型的实现的创建方法识别)**

- `java.util.Calendar#getInstance()`
- `java.util.ResourceBundle#getBundle()`
- `java.text.NumberFormat#getInstance()`
- `java.nio.charset.Charset#forName()`

- javax.xml.bind.JAXBContext#createMarshaller() 和其他类似方法

原型 （可通过创建方法识别，并返回具有相同属性的自身*不同*实例）

- java.lang.Object#clone() （该类必须实现 java.lang.Cloneable ）

单例 （可通过创建方法识别，每次返回*相同的*实例（通常是其自身））

- java.lang.Runtime#getRuntime()
- java.awt.Desktop#getDesktop()
- java.lang.System#getSecurityManager()

结构模式

适配器 （可通过采用*不同*抽象/接口类型的实例的创建方法来识别，并返回自己的/另一个抽象/接口类型的实现，该实现*装饰/覆盖*给定实例）

- java.util.Arrays#asList()
- java.util.Collections#list()
- java.util.Collections#enumeration()
- java.io.InputStreamReader(InputStream) （返回a `Reader` ）
- java.io.OutputStreamWriter(OutputStream) （返回a `Writer` ）
- javax.xml.bind.annotation.adapters.XmlAdapter#marshal() 和 #unmarshal()

桥 （可通过采用*不同*抽象/接口类型的实例的创建方法来识别，并返回*委托/使用*给定实例的自己的抽象/接口类型的实现）

- 还没有人想到。一个虚构的例子是 `new LinkedHashMap(LinkedHashSet<K>, List<V>)` 返回一个不可修改的链接地图，该地图不会克隆项目，而是*使用*它们。但是 java.util.Collections#newSetFromMap() 和 singletonXXX() 方法非常接近。

复合 （通过将*相同*抽象/接口类型的实例放入树结构的行为方法可识别）

- java.awt.Container#add(Component) （因此，实际上遍及整个Swing）
- javax.faces.component.UIComponent#getChildren() （因此实际上遍及整个JSF UI）

装饰器 （通过采用*相同*抽象/接口类型的实例的创建方法可识别，这会增加其他行为）

- 所有子类 java.io.InputStream ， OutputStream ， Reader 并 Writer 有一个构造函数取相同类型的实例。
- java.util.Collections 的 checkedXXX() ， synchronizedXXX() 和 unmodifiableXXX() 方法。
- javax.servlet.http.HttpServletRequestWrapper 和 HttpServletResponseWrapper
- javax.swing.JScrollPane

外观 （可通过内部使用*不同*独立抽象/接口类型的实例的行为方法识别）

- javax.faces.context.FacesContext ，它在内部等使用抽象/接口类型 LifeCycle ， ViewHandler ， NavigationHandler 等等而没有终端用户具有至约它的担心（其然而覆写投放通过注射）。
- javax.faces.context.ExternalContext ，其在内部使用 ServletContext ， HttpSession ， HttpServletRequest ， HttpServletResponse ，等。

Flyweight （通过返回缓存实例的创建方法可以识别，有点“多态”想法）

代理 **（可通过创建方法识别，该方法返回给定抽象/接口类型的实现，然后*委托/使用*给定抽象/接口类型的*不同*实现）**

- java.lang.reflect.Proxy
- java.rmi.*
- javax.ejb.EJB （这里的说明）
- javax.inject.Inject （这里的说明）
- javax.persistence.PersistenceContext

行为模式

责任链 **（可由行为方法识别，该行为方法在队列中具有*相同*抽象/接口类型的*另一个*实现中（间接）调用相同方法）**

- java.util.logging.Logger#log()
- javax.servlet.Filter#doFilter()

命令 **（通过其中调用在的实施方案的方法的抽象/接口类型行为方法recognizeable *不同*已抽象/接口类型*包封*它的创建过程中由命令实现）**

- 所有实现 java.lang.Runnable
- 所有实现 javax.swing.Action

解释器 **（可通过行为方法识别，并返回给定实例/类型的*结构上*不同的实例/类型；请注意，解析/格式化不是模式的一部分，确定模式以及如何应用它）**

- java.util.Pattern
- java.text.Normalizer
- 的所有子类 java.text.Format
- 的所有子类 javax.el.ELResolver

迭代器 **（行为方法可识别，该行为方法可从队列中顺序返回*其他*类型的实例）**

- 的所有实现 java.util.Iterator （因此还有其他实现 java.util.Scanner ！）。
- 所有实现 java.util.Enumeration

介体 **（可通过行为方法来识别，该行为采用委托/使用给定实例的不同抽象/接口类型的实例（通常使用命令模式））**

- java.util.Timer （所有 scheduleXXX() 方法）
- java.util.concurrent.Executor#execute()
- java.util.concurrent.ExecutorService （ invokeXXX() 和 submit() 方法）
- java.util.concurrent.ScheduledExecutorService （所有 scheduleXXX() 方法）
- java.lang.reflect.Method#invoke()

Memento **（可通过内部改变*整个*实例状态的行为方法识别）**

- java.util.Date （setter方法这样做， Date 在内部由一个 long 值表示）
- 所有实现 java.io.Serializable
- 所有实现 javax.faces.component.StateHolder

观察者（或发布/订阅） **（可由行为方法识别，该行为方法将根据自己的状态在*另一种*抽象/接口类型的实例上调用方法）**

- 的所有实现 `java.util.EventListener` （因此实际上遍及整个Swing）
- `javax.servlet.http.HttpSessionBindingListener`
- `javax.servlet.http.HttpSessionAttributeListener`
- `javax.faces.event.PhaseListener`

状态 （通过行为方法可识别，该行为方法可根据实例的状态（可从外部控制）更改其行为）

- `javax.faces.lifecycle.LifeCycle#execute()` （由所控制 `FacesServlet` ，其行为取决于JSF生命周期的当前阶段（状态））

策略 （通过在抽象/接口类型行为方法recognizeable它调用一个方法中的实施方案*不同*，其已被抽象/接口类型*传入的*作为方法参数到策略执行）

- `java.util.Comparator#compare()` 由其他人执行 `Collections#sort()` 。
- `javax.servlet.http.HttpServlet` ， `service()` 和所有 `doXXX()` 方法都采用 `HttpServletRequest` 和 `HttpServletResponse` ，实现者必须对其进行处理（而不是将其作为实例变量！）。
- `javax.servlet.Filter#doFilter()`

模板方法 （可被已经具有抽象类型定义的“默认”行为的行为方法识别）

- 所有非抽象方法 `java.io.InputStream` ， `java.io.OutputStream` ， `java.io.Reader` 和 `java.io.Writer` 。
- 所有非抽象方法 `java.util.AbstractList` ， `java.util.AbstractSet` 和 `java.util.AbstractMap` 。
- `javax.servlet.http.HttpServlet` ， `doXXX()` 默认情况下，所有方法都会向响应发送HTTP 405“不允许使用方法”错误。您可以自由实现任何一个或任何一个。

访客 （可通过两种*不同的*抽象/接口类型识别，其定义的方法彼此采用*另一种*抽象/接口类型；一种方法实际上调用了另一种方法/接口，另一种方法在其上执行了所需的策略）

- `javax.lang.model.element.AnnotationValue` 和 `AnnotationValueVisitor`
- `javax.lang.model.element.Element` 和 `ElementVisitor`
- `javax.lang.model.type.TypeMirror` 和 `TypeVisitor`
- `java.nio.file.FileVisitor` 和 `SimpleFileVisitor`
- `javax.faces.component.visit.VisitContext` 和 `VisitCallback`

18年10月2日于15:08编辑

社区维基
43转，14位用户73%
BalusC

22 令人印象深刻.. :) +1。 `javax.lang.model.element` 定义的游客;）我不是很确定是否 `doXXX` 和 `doFilter` 在“策略”。 – 博若 10年 4月26日在13:14

14 提到的构建器（例如StrinbgBuilder）都不是构建器模式的示例。但是，将他们视为构建者是一个非常常见的错误（因此，您并没有真正责怪^ _ ^） – Angel O'Sphere 2011年 5月25日在13:41

76 @BalusC，我有一个问题要问你。您是否阅读了Java和JSF 的**WHOLE**源代码？ – Tapas Bose 13年1月9日在21:39

20 @塔帕斯：我没有阅读所有内容，只阅读了我需要的部分，或者只是对“他们”是如何做到的很好奇。 – BalusC 2013年1月9日在21:41

6 “工厂方法”下的大多数示例都是不是GoF模式的“静态工厂”的示例。不正确。 – 环形承载者 15年5月4日在11:16

1. 整个摆动过程中的观察者模式（ `Observable` ， `Observer` ）