细数JDK里的设计模式 - TinyKing - 博客园

细数JDK里的设计模式

原文出处： javacodegeeks 译文出处：deepinmind

这也是篇老文了，相信很多人也看过。前面那些废话就不翻译了，直接切入正题吧~

结构型模式：

适配器模式：

用来把一个接口转化成另一个接口。

- java.util.Arrays#asList()
- javax.swing.JTable(TableModel)
- java.io.InputStreamReader(InputStream)
- java.io.OutputStreamWriter(OutputStream)
- javax.xml.bind.annotation.adapters.XmlAdapter#marshal()
- javax.xml.bind.annotation.adapters.XmlAdapter#unmarshal()

桥接模式：

这个模式将抽象和抽象操作的实现进行了解耦，这样使得抽象和实现可以独立地变化。

- AWT (It provides an abstraction layer which maps onto the native OS the windowing support.)
-
- JDBC

组合模式

使得客户端看来单个对象和对象的组合是同等的。换句话说，某个类型的方法同时也接受自身类型作为参数。

    - javax.swing.JComponent#add(Component)
    - java.awt.Container#add(Component)
    - java.util.Map#putAll(Map)
    - java.util.List#addAll(Collection)

- java.util.Set#addAll(Collection)

装饰者模式：

动态的给一个对象附加额外的功能，这也是子类的一种替代方式。可以看到，在创建一个类型的时候，同时也传入同一类型的对象。这在JDK里随处可见，你会发现它无处不在，所以下面这个列表只是一小部分。

- java.io.BufferedInputStream(InputStream)
- java.io.DataInputStream(InputStream)
- java.io.BufferedOutputStream(OutputStream)
- java.util.zip.ZipOutputStream(OutputStream)
- java.util.Collections#checkedList|Map|Set|SortedSet|SortedMap

门面模式：

给一组组件，接口，抽象，或者子系统提供一个简单的接口。

- java.lang.Class
- javax.faces.webapp.FacesServlet

享元模式

使用缓存来加速大量小对象的访问时间。

- java.lang.Integer#valueOf(int)
- java.lang.Boolean#valueOf(boolean)
- java.lang.Byte#valueOf(byte)
- java.lang.Character#valueOf(char)

代理模式

代理模式是用一个简单的对象来代替一个复杂的或者创建耗时的对象。

- java.lang.reflect.Proxy
- RMI

创建模式

抽象工厂模式

抽象工厂模式提供了一个协议来生成一系列的相关或者独立的对象，而不用指定具体对象的类型。它使得应用程序能够和使用的框架的具体实现进行解耦。这在JDK或者许多框架比如Spring中都随处可见。它们也很容易识别，一个创建新对象的方法，返回的却是接口或者抽象类的，就是抽象工厂模式了。

- java.util.Calendar#getInstance()
- java.util.Arrays#asList()

- java.util.ResourceBundle#getBundle()
- java.sql.DriverManager#getConnection()
- java.sql.Connection#createStatement()
- java.sql.Statement#executeQuery()
- java.text.NumberFormat#getInstance()
- javax.xml.transform.TransformerFactory#newInstance()

建造模式(Builder)

定义了一个新的类来构建另一个类的实例，以简化复杂对象的创建。建造模式通常也使用方法链接来实现。

- java.lang.StringBuilder#append()
- java.lang.StringBuffer#append()
- java.sql.PreparedStatement
- javax.swing.GroupLayout.Group#addComponent()

工厂方法

就是一个返回具体对象的方法。

- java.lang.Proxy#newProxyInstance()
- java.lang.Object#toString()
- java.lang.Class#newInstance()
- java.lang.reflect.Array#newInstance()
- java.lang.reflect.Constructor#newInstance()
- java.lang.Boolean#valueOf(String)
- java.lang.Class#forName()

原型模式

使得类的实例能够生成自身的拷贝。如果创建一个对象的实例非常复杂且耗时时，就可以使用这种模式，而不重新创建一个新的实例，你可以拷贝一个对象并直接修改它。

- java.lang.Object#clone()
- java.lang.Cloneable

单例模式

用来确保类只有一个实例。Joshua Bloch在Effetive Java中建议到，还有一种方法就是使用枚举。

- java.lang.Runtime#getRuntime()
- java.awt.Toolkit#getDefaultToolkit()
- java.awt.GraphicsEnvironment#getLocalGraphicsEnvironment()

- java.awt.Desktop#getDesktop()

行为模式

责任链模式

通过把请求从一个对象传递到链条中下一个对象的方式，直到请求被处理完毕，以实现对象间的解耦。

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

命令模式

将操作封装到对象内，以便存储，传递和返回。

- java.lang.Runnable
- javax.swing.Action

解释器模式

这个模式通常定义了一个语言的语法，然后解析相应语法的语句。

- java.util.Pattern
- java.text.Normalizer
- java.text.Format

迭代器模式

提供一个一致的方法来顺序访问集合中的对象，这个方法与底层的集合的具体实现无关。

- java.util.Iterator
- java.util.Enumeration

中介者模式

通过使用一个中间对象来进行消息分发以及减少类之间的直接依赖。

- java.util.Timer
- java.util.concurrent.Executor#execute()
- java.util.concurrent.ExecutorService#submit()
- java.lang.reflect.Method#invoke()

备忘录模式

生成对象状态的一个快照，以便对象可以恢复原始状态而不用暴露自身的内容。Date对象通过自身内部的一个long值来实现备忘录模式。

- java.util.Date
- java.io.Serializable

空对象模式

这个模式通过一个无意义的对象来代替没有对象这个状态。它使得你不用额外对空对象进行处理。

- java.util.Collections#emptyList()
- java.util.Collections#emptyMap()
- java.util.Collections#emptySet()

观察者模式

它使得一个对象可以灵活的将消息发送给感兴趣的对象。

- java.util.EventListener
- javax.servlet.http.HttpSessionBindingListener
- javax.servlet.http.HttpSessionAttributeListener
- javax.faces.event.PhaseListener

状态模式

通过改变对象内部的状态，使得你可以在运行时动态改变一个对象的行为。

- java.util.Iterator
- javax.faces.lifecycle.LifeCycle#execute()

策略模式

使用这个模式来将一组算法封装成一系列对象。通过传递这些对象可以灵活的改变程序的功能。

- java.util.Comparator#compare()
- javax.servlet.http.HttpServlet
- javax.servlet.Filter#doFilter()

模板方法模式

让子类可以重写方法的一部分，而不是整个重写，你可以控制子类需要重写那些操作。

- java.util.Collections#sort()
- java.io.InputStream#skip()
- java.io.InputStream#read()
- java.util.AbstractList#indexOf()

访问者模式

提供一个方便的可维护的方式来操作一组对象。它使得你在不改变操作的对象前提下，可以修改或者扩展对象的行为。

- javax.lang.model.element.Element and javax.lang.model.element.ElementVisitor
- javax.lang.model.type.TypeMirror and javax.lang.model.type.TypeVisitor

译者注：很多地方可能会存在争议，是否是某种模式其实并不是特别重要，重要的是它们的设计能为改善我们的代码提供一些经验。

« 上一篇：Tomcat 使用过程中的一些技巧
» 下一篇：上传组件

posted @ 2016-10-08 14:08 TinyKing 阅读(2587) 评论(0) 编辑 收藏