1. 请简单说明Spring 框架的体系结构，如果能画示意图最好。



核心容器：核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转 （IOC） 模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。

Spring 上下文：Spring 上下文是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。

Spring AOP：通过配置管理特性，Spring AOP 模块直接将面向方面的编程功能集成到了 Spring 框架中。所以，可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

Spring DAO：JDBC DAO 抽象层提供了有意义的异常层次结构，可用该结构来管理异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量（例如打开和关闭连接）。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。

Spring ORM：Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具，其中包括 JDO、Hibernate 和 iBatis SQL Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。

Spring Web 模块：Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文。所以，Spring 框架支持与 Jakarta Struts 的集成。Web 模块还简化了处理多部分请求以及将请求参数绑定到域对象的工作。

Spring MVC 框架：MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

1. 说明如何使用Spring做统一的声明式事务处理和控制。

<!-- 事务通知 -->

<tx:advice id="txAdvice" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="save\*Trans" propagation="REQUIRED" />

<tx:method name="update\*Trans" propagation="REQUIRED" />

<tx:method name="del\*Trans" propagation="REQUIRED" />

</tx:attributes>

</tx:advice>

<!-- 支持注解的声明式事务 -->

<tx:annotation-driven transaction-manager="transactionManager" />

<aop:config proxy-target-class="true">

<!-- <aop:pointcut expression="execution(\* com.thunisoft.cardreader.common..\*.\*(..))"

id="basePointCut" />

<aop:advisor advice-ref="txAdvice" pointcut-ref="basePointCut"

/> -->

</aop:config>

1. 说明SpringAOP功能的实现机制或原理。

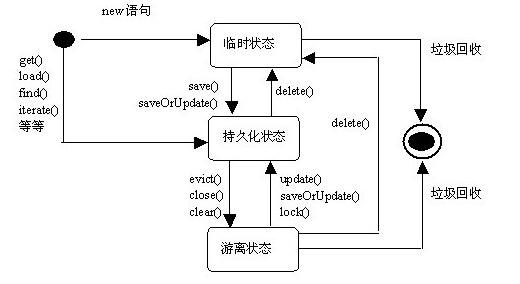
AOP实际上是由目标类的代码类实现的。AOP代码是AOP框架动态生成的一个对象，该对象可以做为目标对象使用。AOP代码在特定切入点添加了处理并回调了对象方法。

1. 请说明Hibernate 中持久化对象的几个状态，以及其含义。

临时态：不在session缓存中，不与任何session实例关联，在数据库中没有对应记录。

持久态：在session缓存中，与session实例关联，在数据库中有对应数据。

脱管态：不在session缓存中，但与session实例关联，数据库上有对应数据。



1. 介绍Hibernate的缓存种类和含义。

Hibernate的缓存包括Session的缓存和SessionFactory的缓存，其中SessionFactory的缓存又可以分为两类：内置缓存和外置缓存。

Session的缓存是内置的，不能被卸载，也被称为Hibernate的第一级缓存。

SessionFactory的内置缓存和Session的缓存在实现方式上比较相似，前者是SessionFactory对象的一些集合属性包含的数据，后者是指Session的一些集合属性包含的数据。SessionFactory的内置缓存中存放了映射元数据和预定义SQL语句，映射元数据是映射文件中数据的拷贝，而预定义SQL语句是在Hibernate初始化阶段根据映射元数据推导出来，SessionFactory的内置缓存是只读的，应用程序不能修改缓存中的映射元数据和预定义SQL语句，因此SessionFactory不需要进行内置缓存与映射文件的同步。

SessionFactory的外置缓存是一个可配置的插件。在默认情况下，SessionFactory不会启用这个插件。外置缓存的数据是数据库数据的拷贝，外置缓存的介质可以是内存或者硬盘。SessionFactory的外置缓存也被称为Hibernate的第二级缓存。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 组件 | Provider类 | 类型 | 集群 | 查询缓存 |
| Hashtable | org.hibernate.cache.HashtableCacheProvider | 内存 | 不支持 | 支持 |
| EHCache | org.hibernate.cache.EhCacheProvider | 内存，硬盘 | 最新支持 | 支持 |
| OSCache | org.hibernate.cache.OSCacheProvider | 内存，硬盘 | 不支持 | 支持 |
| SwarmCache | org.hibernate.cache.SwarmCacheProvider | 集群 | 支持 | 不支持 |
| JBoss TreeCache | org.hibernate.cache.TreeCacheProvider | 集群 | 支持 | 支持 |

1. Spring与struts1整合方式，至少两种，说明优缺点。

方式1:

最原始而易懂的方式:

Action继承spring提供的类

org.springframework.web.struts.MappingDispatchActionSupport

action中可以通过以下方式获取bean

ApplicationContext context = **this**.getWebApplicationContext();

userService=(UserService)context.getBean("userService");

优点：

保持你原先struts的配置.只需要改变一下你相应的Action类继承

缺点：

Action将会和Spring耦合在一起.

方式2：

启动Spring再在相应的Action类中实例化ApplicationContext   
spring有三种启动方式,使用ContextLoaderServlet,ContextLoaderListener和ContextLoaderPlugIn.

ApplicationContext ac=WebApplicationContextUtils.getWebApplicationContext(**this**.getServlet().getServletContext());

优点：

比第一种耦合低。

缺点：

Action不受spring管理，只能通过 ApplicationContext 获取bean.

方式3：

org.springframework.web.struts.DelegatingActionProxy

<action path="/login" parameter="login" type="org.springframework.web.struts.DelegatingActionProxy">

<bean name="/login" class="com.addresslist.action">

<property name="userService" ref="userService"></property>

</bean>

优点：

Action由spring管理，可以注入bean，可以使用spring管理bean的功能。

缺点：

配制文件修改多。

方式4：

使用 org.springframework.web.struts.DelegatingRequestProcessor 类来覆盖 Struts 的 RequestProcessor 处理程序

<controller processorClass="org.springframework.web.struts.DelegatingRequestProcessor" />

Struts中的action配置则无需配置type属性（即使配置了type属性也不起任何作用，除非在spring的配置文件里找不到对应的name属性值）

Struts会将拦截到的用户请求转发到Spring context下的bean，根据bean的name属性来匹配。

配置的时候需要注意的是Action bean不再需要id属性，而要用name替代id属性，这时name属性的值应与struts-config.xml中配置的Action的path属 性的值相同