## Spring专题

1. **Spring**

Spring让人们把对象间的依赖关系转而用配置文件来管理，也就是依赖注入机制。

* 1. **Spring三大核心**

1. Bean：核心中的核心，Bean包装的是对象
2. Context：是Bean关系的集合，Bean的生存环境
3. Core：是发现、建立、维护每个Bean之间关系所需的工具
   1. **Bean组件在org.sf.beans包下包括**
4. Bean的定义
5. Bean的创建：使用者只关注Bean的创建
6. Bean的解析
   1. **Spring Bean的创建是典型的工厂模式**，它的顶级接口是BeanFactory
   2. **ApplicationContext**
7. 标识一个应用环境
8. 利用BeanFactory创建Bean对象
9. 保存对象关系
10. 能够捕获各种事件
    1. **主要工作**
11. 构建BeanFactory，以便产生所需的演员
12. 注册可能感兴趣的事件
13. 创建Bean实例对象
14. 触发被监听的事件
    1. **Bean的生命周期**
15. 解析XML文件，创建一个BeanFactory，里面包含所有要实例化的Bean和Bean实例化需要的属性和配置
16. 通过BeanFactory创建Bean实例，并进行基本的属性注入、依赖注入。如果有依赖的Bean，则递归创建依赖Bean
17. 如果Bean实现了BeanNameAware，调用setBeanName(ID)方法，此处传递的是spring配置文件中Bean的ID
18. 如果Bean实现了ApplicationContextAware方法，会调用SetApplicationContext(ApplicationContext)方法传入spring上下文，使该Bean能获得其他Bean实例，实现对Bean的管理，同理SetBeanFactoryAware
19. 如果Bean关联了BeanPostProcessor接口，会调用PostProcessBeforeInitialization(bean,beanName)方法，来对Bean进行一些设置、修改
20. 若有inti-method，则执行
21. 执行postAfterInitialization(bean,beanName)方法，如AOP
22. 当Bean不需要时，如果实现了DisposableBean接口，调用destroy方法
23. 如果有destroy-method，则调用
    1. **Bean实例化过程**
24. Constructor
25. PostConstruct
26. InitializingBean
27. Init-method
28. preDestroy
29. DisposableBean
30. Destroy-mehtod
    1. **Spring设计模式**
31. 工厂模式：spring的核心之一就是一个Bean工厂
32. 策略模式

① AOPProxy是接口，cglib和动态jdk是具体实现

② 各种各层的接口，选择哪个具体实现，如XML解析，Beanfactory类型，加载文件方式，spring事务管理接口

3）观察者模式：当bean依赖变化时，通知所有被依赖者，Listener

4）代理模式：AOP，其他一些beanPostProcess

5）包装器模式：BeanWrap，给对象添加功能

6）适配器模式

7）单例模式

8）模板模式：refresh

9）责任链模式：过滤器

10）原型模式

* 1. **Spring上下文**

Spring具有子类上下文和父类上下文的概念，子类上下文可以读取父类上下文中的信息，但父类上下文不能读取子类上下文中的信息。

1. SpringAOP

**2.1 AOP的组成部分**

① 通知（Advice）: 自定义的需要增强的功能，比如捕获日志，那么我需要定义一个日志捕获的方法，也就是方法。

② 连接点（joinpoint）：所有类的方法的前后，都是连接点

③ 切入点（pointcut）：具体的某个类的某个方法的前后，即我们需要增强的方法

④ 切面（aspect）：就是我自定义的增强方法所在的类，叫做切面。它有通知和切入点组成。

⑤ 目标（target）：就是被通知的对象。

⑥ 代理（proxy）：我们就是通过用代理模式，将目标对象，代理成新的增强后的对象，将目标对象中需要增强的方法，引入我们的通知，实现增强功能。

**2.2 AOP实现的方式**

如果bean关联了beanPostProcess接口，就会调用postProcessAfterInition，对bean进行一些设置。其中AOP就实现了beanPostProcess接口，在postProcessAferinition方法中对bean进行AOP代理。

**2.3 代理实现**

① 如果类实现了接口，就用JDK动态代理来创建代理类

② 如果类没有实现接口，就用CGLIB继承原生类来实现代理类

③ 在代理类中需要增强的方法中，织入增强方法，再用反射调用原始类的该方法。

④ 这样用户调用该类的方法，其实执行的是代理类的方法。

**2.4 SpringAOP描述：**

用代理类包裹切面，把它们织入到spring管理的bean中，也就是说伪装成目标类，通过接口伪装成与目标类有相同方法的类，在相同方法中反射调用目标类的方法和织入增强的方法，即形成了代理类。如果没有接口则使用CGLIB创建一个目标类的子类。

1. SpringMVC

**3.1 SpringMVC流程**

1）前端控制器接收用户请求

2）DispatchServelt请求处理器映射器查找对应的Handler

3）HandlerMapping向DispatchServelt返回Handler

4）DispatchServelt调用处理器适配器执行handler

5）HandlerAdapter执行Handler（controller）

6）Handler执行完向HandlerAdapter返回ModelAndView

7）HandlerAdapter向前端控制器返回ModelAndView

8）前端控制器请求视图解析器解析视图

9）视图解析器解析完向前端控制器返回view

10）前端控制器进行视图渲染，将数据填充到request域中

11）前端控制器向用户返回结果

**3.2 异常处理**

1）配置springMVC自带的simpleMappingExceptionResolver类，再里面配置不同种类异常所返回的错误页面，这里需要自定义一些不同的异常类，如BussinessException，parmaterException。系统将会根据所抛出的异常类型返回特点的错误页面

2）定义一个全局异常处理类实现HandlerExceptionResolver接口。这样抛出的异常最后会到自定义的全局异常处理类中，我们在这个类里，对不同的异常做处理。

3）使用@ExceptionHandler和@controllerAdvice注解。定义一个全局异常处理类，用@controllerAdvice注释，使抛出的异常会到此类中来，在用@ExceptionHandler来注释不同的方法，每个方法处理不同的异常信息。和第二种差不多。

第一种的缺点是，只能简单的返回特点的错误页面，不能对数据进行处理。

1. Spring事务

**4.1 事务隔离级别**

1. 默认：使用底层数据库的默认隔离级别
2. Read\_uncommitted：读未提交
3. Read\_committed：读已提交
4. Repeatable\_read：可重复读，一个事物对某个数据进行操作，其他事务不能修改
5. Serializable：串行读，所有事务依次执行

4.2 事务的传播行为

1. required：如果当前存在事务，则加入该事务，若不存在，创建新事务
2. Required\_new：首先创建一个事务，如果当前存在事务，则挂起当前事务
3. Supports：如果当前存在事务，则加入该事务，若没有，则以非事务运行
4. Not\_supported：以非事务方式运行，如果存在事务，则把当前事务挂起
5. Never：以非事务方式运行，如果存在事务，则抛出异常
6. Mandatory：如果存在事务，则加入该事务，如果没有则抛出异常
7. Nested：嵌套事务，事务中的事务。