# Spring

原始：class + Factory + xml（通过字符串）+反射可以解决对象依赖关系

Spring通过bean + xml+反射进行管理

默认单例，也就是每个class只生成一个实例，多个请求共用。

# IOC/DI

## 控制什么

IOC容器控制对象的创建、实例化。

IOC容器控制对象之间的依赖

## 为什么叫反转（对应于正向）

因为应用程序不能主动去获取外部资源或者对象，而是被动的等待容器给他注入资源或者对象。

## 哪些方面反转了？

1. 创建对象
2. 程序获取资源（比如：依赖其他对象）的方式

## 为什么要反转？

1. 引入容器后，体系松散，管理更有序
2. 类之间真正实现松散耦合，使得开发、测试、修改、升级等都变得容易了。

IOC并没有帮我们实现任何业务功能，原本该有应用实现的功能还是由应用自身完成。

## ApplicationContext就是IOC容器

ApplicationContext context = new ClassPathApplicationContext(new String[“applicationContext”])

ApplicationContext context = new ClassPathApplicationContext(new String[“applicationContext”])

ApplicationContext context = new ClassPathApplicationContext(new String[“applicationContext”])

表示创建了多个IOC容器，每个容器都有一份自己的bean实例。

## 代理方式区别

Test test = (Test)content.getBean(“test”);

根据Test判断使用什么代理。如果Test是类：则使用CGLIB 如果Test是接口：使用JDK代理。

一般采用面向接口编程，所以这个Test应该是接口

## 如何初始化BEAN

构造器

静态工厂方法

实例化工厂方法

## 依赖注入（DI）方式选择

1. 优先选择set注入
2. 构造器注入，如果要保证对象之间有依赖或者先后顺序，比如要构造A前，需要先构造B（一般情况少）

## 延迟初始化bean

ApplicationContext默认是启动时将所有singleton bean（默认配置）提前进行实例化。

如果要延迟加载可加lazy-init=”true”后者全局beans中加入属性default-lazy-init=”true”

默认单例，也就是每个class只生成一个实例，多个请求共用

## singleton

在每一个Spring IOC容器中一个bean定义对应一个对象实例，在读取配置文件时自动初始化。

此处的单例跟设计模式单例不同，设计模式是对于每一个ClassLoader中指定class创建的实例有且只有一个。针对虚拟机级别

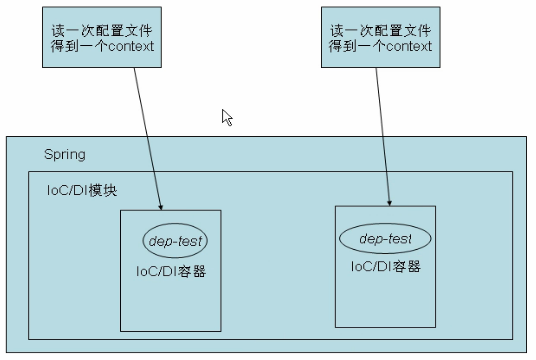
Spring中单例只针对一个IOC容器，因为可以定义多个容器，针对IOC容器级别的。

其他作用域如下

prototype：每次请求时就创建一个实例。

request：基本不用

session：基本不用



## 有状态bean和无状态bean

有实例变量叫有状态bean

无实例变量叫无状态bean

Spring IOC默认是单例，所有尽量使用无状态bean，这样就不存在同步问题。

## Lookup-method（用的不都）

如果一个单例bean依赖一个原型bean，实际情况单例bean只实例化一次，这样依赖的原型也就是同一个了，如何保证是不同的依赖对象呢。

解决办法：

将原型bean改成抽象类，增加一个抽象方法获取依赖的原型bean，然后在xml中ref方法改成loopup-method方式。原理就是动态创建子类覆盖父类方法，使用了cglib代理。

适用于：向一个singleton中注入非singleon对象。



## 生命周期（这一次理解深了）

指对象实例的生命周期

我们不能控制生命周期

学的生命周期的回调方法，应用这些方法，我们可以在生命周期发生时额外添加一些操作。

比如：servlet中init、service、destroy触发

通过两个方式完成对象实例化后执行动作

第一种：public class TestB implements InitializingBean

@Override

public void afterPropertiesSet() throws Exception {

System.out.println("afterPropertiesSet...");

}

第二种：<bean id="test" class="com.nnngu.web.Test" init-method="init"></bean>

等对象构造完成后才会调用init方法，不是创建对象前。

## 国际化配置

.properties文件，通过messageResource，自己详细了解

## ApplicationListener接口

实现此接口，可以监听IOC容器初始化事件。

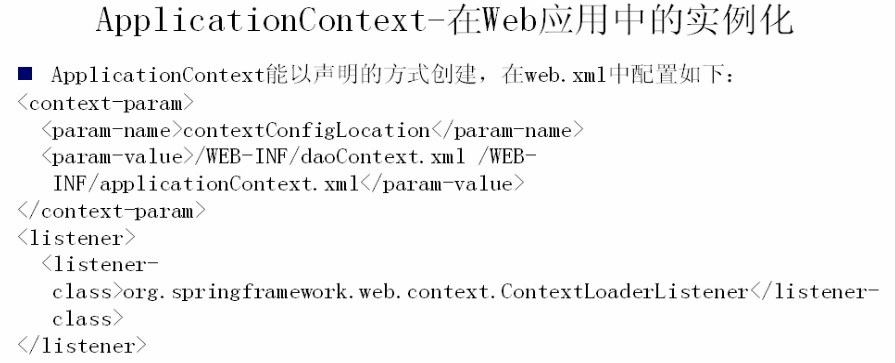
老师没有详细展开说明

## Web中如何启动ApplicatoinContet呢

我们main方法测试spring可以通过如下：

ApplicationContext context = new ClassPathXmlApplicationContext(new String[]{"spring/applicationContext-test.xml"});  
  
Test test = (Test)context.getBean("test");

那么Web项目中通过在web.xml声明式listener方式启动

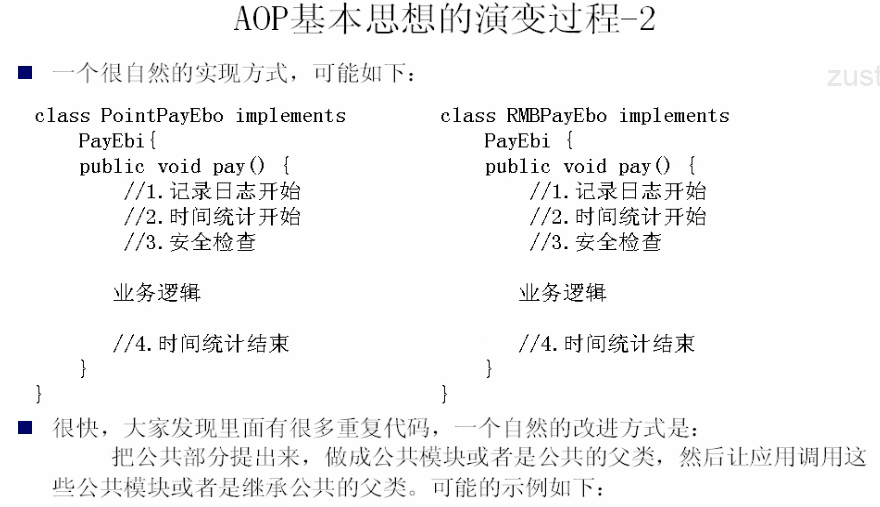


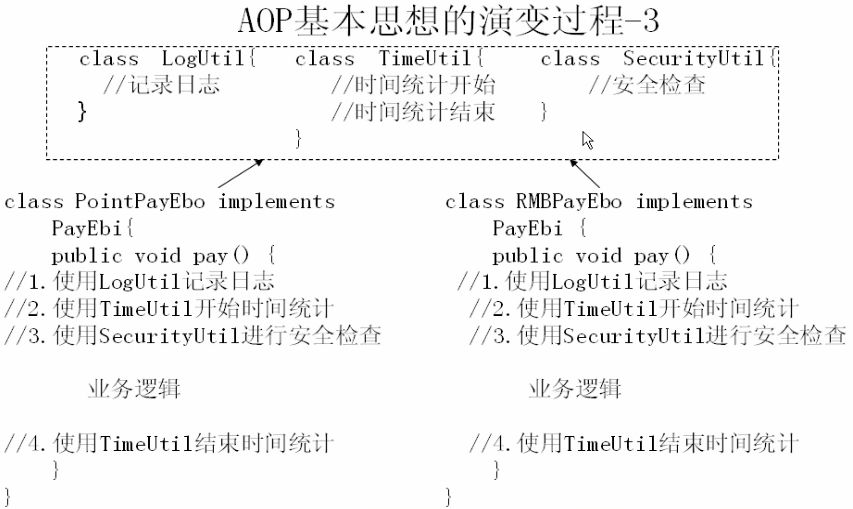
# AOP

OOP设计是从上下思考方式，比如视图、模型、控制。

AOP是横切关注点

技术思考





我不主动调用，而是有日志、统计等工具类注入进来且自动执行。

最终实现：主动调用方法变被动注入调用，采用装饰模式或者代理模式

代理模式：控制对象的访问控制，一般通过接口实现，jdk动态代理只能是接口，cglib可代理类和接口

装饰模式：增加额外功能，一般通过抽象类实现

