Spring是容器框架，用于配置管理bean（java中的任何一个对象）和维护bean之间关系的框架，配置管理项目的各个组件

spring的核心配置文件applicationContext.xml一般放在src目录下，配置时需要引入指定格式的xsd文件，applicationContext.xml文件的一般格式：

<?xml version="1.0" encoding="utf-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<context:annotation-config></context:annotation-config>（启动注解）

<context:property-placeholder location="classpath: 文件名（包含包名）[,classpath: 文件名（包含包名）]"/>（分散配置引入分散文件，通过${属性名}取出文件中对应的属相值）

[<bean id="实例化对象名" class="管理的类名（包含包名）" parent="继承的父类的实例化对象名" scope="singleton/prototype/request/session/global-session（作用域，默认singleton，单例模式）" init-metho="指定初始化方法名" destroy-metho="指定销毁方法名" autowire="byname/byType/constructor/autodetect/no（bean对应类的关联关系未设置会自动装配，默认值由beans标签中的default-autowire属性的值决定，default-autowire表示所有bean的默认自动装配方式，默认值为no）">（指定bean在实例化对象以及销毁时调用指定的初始化函数和销毁函数进行处理，函数是bean对应类的成员函数）

[(<property name="类的属性名">

<value>属性值</value>

</property>)/(<property name="类的属性名" value="属性值"></property>)]

[<property name="类属性的名称" ref="引用的类的spring实例化对象名"></property>]（类属性的配置，将引用的类的spring实例化对象名对应类设置为类的属性，在类中的名称设置为类属性的名称）

</bean>]（配置spring管理的bean，当spring加载时会自动创建实例化管理的类的对象名称为实例对象名，并将类的属性名的值设置为指定的的属性值）

<bean class="PropertyPlaceholderConfigurer">

<property name="locations">

<list>

[<value>文件名（包含包名）</value>]

</list>

</property>

</bean>（另一分散配置引入分散文件的方式）

</beans>

ApplicationContext 名称=new ClassPathXmlApplicationContext("spring核心配置文件（默认路径是src）"); 获得applicationContext对象（容器对象），从类路径中加载spring配置文件并实例化配置的所有类对象

ApplicationContext 名称=new FileSystemXmlApplicationContext("spring核心配置文件（绝对路径即硬盘中的路径）"); 获得applicationContext对象（容器对象），从文件系统中加载spring配置文件并实例化配置的所有类对象

ApplicationContext 名称=new XmlWebApplicationContext("spring核心配置文件"); 获得applicationContext对象（容器对象），从web系统中加载spring配置文件并实例化配置的所有类对象

类名 对象名=(类名) ApplicationContext.getBean("spring配置文件中的对应类的实例化对象名"); 通过spring的applicationContext对象获得对应类的实例对象，创建实例化类对象的任务由spring完成和管理

ioc(反向控制 inverse of control)：控制权反转，创建对象（bean）和维护对象的关系的权利从程序中转移到spring的容器（applicationContext.xml）

di( dependency injection 依赖注入)：di与ioc是同一个概念，spring设计者认为di更准确表示spring的核心

BeanFactory 工厂对象名=new XmlBeanFactory(new ClassPathResource("spring配置文件名")); 加载spring配置文件不实例化配置类对象，获得BeanFactory对象（工厂对象）

工厂对象名.getBean("spring配置文件中的类名"); 实例化spring配置文件中指定类名的对象

Bean的作用域

Singleton：单例，一个Spring容器中指定bean的所有对象是同一个，加载配置文件时对象会被实例化

Prototype：bean定义对应多个对象实例，加载配置文件时对象不会被实例化

Request：指定bean对象为单例对象，作用域是http的一次请求

Session：指定bean对象为单例对象，作用域是http的会话Session

global-session：指定bean对象为单例对象，作用域是web的Spring ApplicationContext

Spring配置文件配置的实现BeanPostProcessor接口的类的对应bean：

<bean id="实例化对象名" class="实现BeanPostProcessor接口的类（包含包名）" />（实现BeanPostProcessor接口的类对所有Spring配置文件配置的bean对应的类在实例化时进行后置处理）

Spring配置文件配置的bean使用应用环境（ClassPathXmlApplicationContext()函数）调用的生命周期（从开始到结束的顺序）：

1.bean实例化函数（对应类的构造函数）

2.根据配置文件设置bean对应类实例化对象的属性值

3.调用BeanNameAware接口的setBeanName()函数：Spring配置文件配置的bean对应的类实现BeanNameAware接口，调用setBeanName(String 名称)函数通过名称获得类在加载配置文件实例化对象的名称

4.调用BeanFactoryAware接口的setBeanFactory ()函数：Spring配置文件配置的bean对应的类实现BeanFactoryAware接口，调用setBeanFactory (BeanFactory 名称)函数通过名称获得工厂信息

5.调用ApplicationContextAware接口的setApplicationContext()函数：Spring配置文件配置的bean对应的类实现ApplicationContextAware接口，调用setApplicationContext(ApplicationContext 名称)函数通过名称获得ApplicationContext信息

aop（面向切面编程，对所有对象编程）

6.调用BeanPostProcessor接口的postProcessorBeforeInitialization()函数：Spring配置文件配置的bean对应的类实现BeanPostProcessor接口，调用postProcessorBeforeInitialization()函数对Spring配置文件配置的bean对应的类实例化时做前置处理，调用postProcessorAfterInitialization()函数对Spring配置文件配置的bean对应的类实例化时做后置处理

7.调用InitializingBean接口的afterrPropertiesSet()函数：Spring配置文件配置的bean对应的类实现InitializingBean接口，调用afterrPropertiesSet()函数对Spring配置文件配置的bean对应的类实例化的对象做处理

8.调用配置文件中bean标签中配置的初始化函数

9.调用BeanPostProcessor接口的postProcessorAfterInitialization ()函数

10.调用Bean对应类的成员函数

11.Spring容器框架关闭

12.调用DisposableBean接口的destroy()函数：Spring配置文件配置的bean对应的类实现DisposableBean接口，调用destroy()函数对Spring配置文件配置的bean对应的类实例化的对象销毁时做相应处理

13.调用配置文件中bean标签中配置的销毁函数

Spring配置文件配置的bean使用工厂方法（XmlBeanFactory ()函数）调用的生命周期（从开始到结束的顺序）：1,2,3,4 ,7,8,10,11,12,13

Spring配置文件bean对应类的方法可以通过注解方式进行配置对应函数上一行为（@对应注解函数名称）表示对应函数是注解函数

一般在配置文件中进行配置不使用接口

Spring配置bean对应类的（list或者数组）/set/map/properties集合类型的属性的属相值：

<property name="类的属性名">

<list/set/map/props>

[<value>属性值</value>]/[<ref bean="引用的类的spring实例化对象名"></ref>（类中有类的list/set关联关系，对象名类的对应bean在配置文件中需要配置）]/[<entry key="键值" value-ref="属性值"></entry>]/[<prop key=""></prop>]

</list/set/map/props>

</property>

Map集合的取出值方法：

for (Entry<String, 值类型> 名称 :对象的map集合.entrySet()) {

名称.getValue(); 获得值

}

迭代器方法

Map<String, 值类型> 名称=对象的map集合

Iterator iterator=名称.keySet().iterator();

while (iterator.hasNext()) {

String key = (String) iterator.next();

类型名 对象名=名称.get(key);

}

Properties集合的取出值方法：

Enumeration 枚举名=Properties对象名.keys();

while (枚举名.hasMoreElements()) {

String key = (String) 枚举名.nextElement();

类型名 对象名= Properties对象名.get Property(key);

}

内部bean：

<bean id="实例化对象名" class="管理的类名（包含包名）">

<property name="实例化对象名称对应的类在实例化对象名对应的类中的属性名">

<bean id="实例化对象名称" class="管理的类名（包含包名）"></bean>

</property>

</bean>

Spring配置bean对应类的集合类型的属性的属相值设置为空（null）：

<property name="类的属性名">

<null/>

</property>

Spring配置bean对应类的属性的属相值通过类的构造函数进行设置：

<bean>

[<constructor-arg index="属性序列号（从0开始）" type="属性类型" value="属性值"></constructor-arg>]

</bean>

aop（aspect oriented programming）面向切面（方面）编程

汇编语言面向机器

C语言面向过程，一般系统软件的语言（操作系统，数据库，杀毒软件，防火墙，驱动）

Java语言面向对象

Spring面向方面（切面）

把实现的接口的引入关联的源码文件后生成的相应的需要实现的函数的参数名称会相对标准容易理解

Spring配置文件中进行aop配置

<bean id="实例化通知的对象名" class="通知的类名（包含包名）"></bean>（配置通知即实现通知接口的类，通知接口有前置/后置/环绕/异常通知MethodBeforeAdvice/AfterReturningAdvice/ MethodInterceptor/ ThrowsAdvice）

<bean id="实例化的目标对象的名" class="目标对象的类名（包含包名）">

<property name="属性名" value="属性值"></property>

</bean>（配置目标对象即实现相应接口的类）

<bean id="实例化切入的通知的对象名" class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">

<property name="advice" ref="需要切入的通知对象名"></property>

<property name="mappedNames">

<list>

[<value>切入通知的函数名（名称支持使用正则表达式表示）</value>]

</list>

</property>

</bean>

<bean id="proxyFactoryBean" class="org.springframework.aop.framework.ProxyFactoryBean">（代理对象，Spring的类）

<property name="proxyInterfaces">（代理的接口集合）

<list>

[<value>目标对象的类实现的接口名（包含包名）</value>]

</list>

</property>

<property name="interceptorNames">（通知集合）

<list>

[<value>通知的对象名/切入的通知的对象名</value>]

</list>

</property>

<property name="target" ref="代理的目标对象的名称"></property>（代理的目标对象）

</bean>

实现多个接口的类，定义对象的类型为接口，所有接口类型之间可以互相转换

Spring配置文件中bean标签中的property标签配置的属性名以及属性值只会根据对应类的set方法进行设置，与对应类中有无配置的属性无关

实现MethodInterceptor环绕通知接口的类需要实现的方法：

public Object invoke(MethodInvocation arg0) throws Throwable {

目标对象方法函数内执行前的通知

Object object=arg0.proceed(); 调用目标对象方法体返回的对象需要返回

目标对象方法函数内执行后的通知

return object; 返回调用目标对象方法体返回的对象

}

ThrowsAdvice异常通知接口不提供方法，实现异常通知接口的类需要添加实现的方法：

public void afterThrowing(Throwable throwable) {}或者

public void afterThrowing(Method method,Object[] objects,Object target,Exception exception) {}

ProxyFactoryBean是BeanFactory中显示创建代理对象的中心类，ProxyFactoryBean会根据实现的接口，代理的对象，植入的通知使用jdk动态代理技术创建新的代理目标对象类型的子类，代理目标对象没有实现接口，ProxyFactoryBean使用CGLIB库生产代理目标对象类型的子类，所以代理目标类以及类中方法是final类型的不能被代理，Spring只支持方法连接点，不提供属性接入点，Spring认为属性拦截破坏了类的封装