浅谈对spring框架的理解：

spring框架是一整个生态链工程，其中衍生出来很多具体型的框架，但是纠其核心内容，一个就是Di（依赖注入），一个是AOP（面向切面）。而我的理解，这两个核心又来自于Java语言自身的特性，由于Java支撑接口等解耦式的编程模型，这为spring的强大功能提供了保障。

对于spring的学习，首先应该有良好的Java基础，在学习spring的DI内容时，应该对Java的接口和继承等知识了解透彻，而学习AOP的时候，除了DI的基础知识，Java的代理也是一个需要掌握的重点，我认为，将这两点把握住，对学习spring框架有着极其重要的影响。

前言：经过学习之后，在spring的诸多配置中，我更加喜欢基于Java配置类的配置，和spring自动装配，这两着简洁明了。让我彻底不再想用原来的XML配置文件配置，所以，本笔记中暂时对基于XML的配置不进行记录，如果后面时间多，会补充上来。

spring容器

spring容器简单理解就是一个大的盆，或者一个大缸，里面装满了Java对象，并且建立维护好了这些对象之间的关系，而在我们需要使用这些对象的时候，就不用再用new关键字来创建了，而实直接从这个大缸里面取，然后直接用就行了。

spring的容器有很多，并不止一个，spring框架有很多的容器实现，但是就分类而言，可以归纳为两种不同的类型，bean工厂就是最简单的容器，提供了基本的依赖注入（DI）功能，**应用上下文**是基于bean工厂而来的，提供了框架级别的服务。

所以简单来讲，spring的容器有两种

1. bean工厂
2. 基于bean工厂的应用上下文（更加常用和高级）

spring中的应用上下文又有多种，下面是可能遇见的

上下文	解释
AnnotationConfigApplicationContext	从Java配置类中加载spring应用上下文（容器）
AnnotationConfigWebApplicationContext	从Java配置类中加载springWeb应用上下文（容器）
ClassPathXmlApplicationContext	从类路径下的xml配置文件中加载应用上下文（容器）
FileSystemXmlApplicationContext	从文件系统下的xml配置文件中加载应用上下文（容器）
XmlWebApplicationContext	从web应用下的xml配置文件中加载应用上下文（容器）

不管是从哪里加载spring上下文（容器）将对象bean加载到bean工厂的过程基本都是一致的。

```java
public class ContextTest {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("application.xml");
        System.out.println(applicationContext);
    }
}
```

如上代码，通过在类路径下加载spring容器，其他方式的都是类似的。

在成功加载了容器之后就可以使用getBean()方法获取容器中的对象了

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
    <beans>
        <bean id="person" class="com.ctbu.spring_demo1.Person">
            <property name="name" value="李白"/>
        </bean>
    </beans>
</beans>
```

```java
public class ContextTest {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("application.xml");
        Person person = (Person) applicationContext.getBean("person");
        System.out.println(person.name);
    }
}
```

如上代码，就是在xml中配置了类型之后，在Java中通过ClassPathXmlApplicationContext获取容器，并通过getBean()获取容器中的对象

spring模块

spring框架发展至今，已经形成了多个模块，开发者可以自由选择，同时spring可以很友好的支撑集成第三方框架等。

就spring模块而言基本分为下面6个模块

模块	说明
数据访问与集成	用于持久层的操作，如访问数据库，关系映射等
Web与远程调用	主要用于web应用的开发
面向切面编程	切面
instrumentation	为jvm添加代理
spring核心容器	依赖注入、控制反转
测试	spring测试

如何装配Bean

要装配bean，首先应该知道bean应该要装到哪里去，在spring中容纳bean（也就是对象实例）的东西叫容器。容器是spring的核心，容器里面装着各种各样的对象实例Bean。

spring的容器总体来讲有两个类型，一个是bean工厂，另一个是应用上下文（或者叫应用环境）ApplicationContext。我们常用的是应用上下文这个容器。

装配Bean的三种方式

1. 在XML配置文件中进行显示配置（最次选择）
2. 在Java配置类中进行显示配置（次优选择）
3. 使用注解的spring隐式自动发现机制和自动装配(最优选择)

自动装配

第一步：创建基本显示配置类，开启组件扫描

实现自动装配仍然需要一定的显示配置，首先我们仍然需要一个配置类，这个配置类中什么都不需要写，但是需要加上必要的注解。

```
@Configuration
@ComponentScan(value = {"aop"})
public class AopConfig {
}
```

@Configuration

@Configuration注解会告诉spring这是一个配置类，这是必须的注解

@ComponentScan

@ComponentScan注解指明需要扫描哪些包下的类，并为他们创建实例放入spring容器中，也就是开启组件扫描。

@ComponentScan的基本使用：

如果直接在配置类上贴上@ComponentScan注解，表示只扫描被贴上注解的配置类所在的包以及所在包的子包。

```
@Configuration
@ComponentScan
public class AopConfig {
}
```

但是我们很多时候需要将配置类单独放到一个包中。这样就他的默认扫描范围就不满足我们的要求。于是我们可以这样修改。可以指定需要扫描的包，值是一个string类型的数组，值的具体元素为包得类路径，所以可以指定多个扫描的范围。

```
@Configuration
@ComponentScan(value = {"aop"})
public class AopConfig {
}
```

明确的指明所设置的包是基础包

```
@Configuration
@ComponentScan(basePackages = {"com.ctbu.spring_demo1.config"})
public class AppConfig {
}
```

扫描的基础包可以有多个，支持接收集合

```
@Configuration
@ComponentScan(basePackages = {"com.ctbu.spring_demo1.config",
"com.ctbu.spring_demo1.controller"})
public class AppConfig {
}
```

支持类作为扫描标记，使用这种方式的话对重构代码很友好，spring会自动去寻找这个类，并把这个类所在的包作为扫描的基础包。如果需要的时候，可以创建一个空的接口为组件扫描提供标记。

```
@Configuration
@ComponentScan(basePackageClasses = Person.class)
public class AppConfig {
}
```

第二步：标注需要加入容器的类

```
@Component
public class Dance implements Performance {
}
```

@Component

@Component注解会将贴上注解的类作为spring的组件扫描进spring容器中。

@Component详解

spring应用上下文会给所有的bean都分配一个id，作为这个对象实例的唯一标识，当我们没有明确的指定这个id的时候，spring会自动为其分配一个id

情况	措施
在xml配置文件中没有明确指定beanID	spring会以全限定类名写作为BeanID
在组件扫描中没有明确指定BeanID	spring会以类名的首字母小写作为BeanID
在Java配置类中没有明确指定BeanID	spring会以方法名作为BeanID

如果想自定义beanID

情况	措施
在组件扫描中	指定@Component的值：@Component("person")
在xml配置文件中	指定bean的id属性： id="person"
在Java配置类中	指定@Bean注解的值：@Bean（name=“person”）

spring还支持使用@Name来为bean（对象实例）命名，用法：@Name（“person”），但是此方法基本不使用。

第三步：使用容器中的组件

```
public class MagicTest {
    @Autowired
    private Performance magic;

    @Test
    public void perform() {
        magic.perform(20);
        SpecialService service = (SpecialService) magic;
        service.specialService();
    }
}
```

@Autowired

@Autowired注解表示spring会对被标注的变量使用自动装配，spring会从容器中找出适合的bean实例，并将他赋值给这个变量。

@Autowired注解的使用

@Autowired注解不仅可以使用在一般的变量上，还可以使用在构造器方法，和属性的setter方法上。实际上@Autowired注解可以用在类中的任何方法上，spring会尽力去满足方法上所声明的依赖。

在自动注入的过程中，如果spring没有在spring容器中找到合适的bean进行注入，spring就会抛出异常，为了避免这个异常的发生，我们以及将@Autowired注解的required属性设置为false

```
@Autowired(required = false)
```

但此时也要注意空指针异常的情况。

同样@Autowired注解也有替代方案@Inject，但是我们基本不使用这个注解，@Autowired更加适合spring，也更加让人容易接收。

使用Java配置类进行显示配置

为什么要使用Java配置类的方式？尽管自动装配已经很强大了，对于我们自己写的bean，自动装配完全可以满足要求，但是，对于第三方的bean，比如datesource等，我们无法在其源码上贴上@Component注解，因此必须通过new其对象来创建对应的bean，而基于xml配置文件的方式也可以达到这样的效果，但是xml实在是太繁琐了。

第一步：创建配置类

```
@Configuration
public class AopConfig {
}
```

如上代码是一个完整的基于Java配置类的组件装配设置。

如果完全使用Java配置类来为spring容器中添加bean，那么就可以不用开启组件扫描。

@Configuration

@Configuration注解表示被标注的类是一个配置类，spring容器对象可以从这个配置类中加载需要装入容器的bean实例。

第二步：声明Bean

```
@Configuration
public class AopConfig {
    @Bean
    public Dance dance(){
        return new Dance();
    }
}
```

如上代码，当spring加载这个配置类的时候就会把创建的对象实例，作为组件，放入到spring容器中。默认情况下，这里创建的bean的名称ID为方法名，但是我们也可以自己指定名称ID。 **（在上面的表格中有详细归纳）**

```
@Bean(name = "dance")
```

@Bean

@Bean注解会告诉spring这个被标注的方法会返回一个对象，而这个返回的对象要注册为spring容器中的bean

第三步：使用组件

这一步和自动装配是一样的，同样可以使用@Autowired进行注入。

Java配置类装配存在引用的bean

很多时候，我们在装配一个bean的时候，这个bean可能引用了其他的bean实例，这样的配置在Java配置类中同样可以实现。下面的代码展示如何装配存在对象引用的bean

```
public class Man {

    Food food;

    public Man(Food food){
        this.food = food;
    }

    public void eat() {
        System.out.println("吃: " + this. food);
    }
}
```

在上面的代码中，可以知道，如果要通过Java配置类的方式将Man这个bean放入spring容器中，那么newMan对象的时候需要有一个Food对象的引用。可以这样做。

```
@Configuration
@ComponentScan(basePackageClasses = Person.class)
public class AppConfig {

    @Bean
    public Food food(){
        return new Food();
    }

    @Bean
    public Man man(){
        return new Man(food());
    }
}
```

直接在man这个方法中调用food()方法，以为food方法一定会产生一个Food类型的实例bean。

还有一种更加简单的当时来实现有引用的装配。

```
@Configuration
@ComponentScan(basePackageClasses = Person.class)
public class AppConfig {

    @Bean
    public Food food(){
        return new Food();
    }

    @Bean
    public Man man(Food food){
        return new Man(food);
    }
}
```

如上Java配置类，直接将要引用的对象作为方法的形参传入，在创建对应bean的时候可以直接使用这个形参，spring会自动满足这个依赖。而且，要强调的是，这个要传入的形参food并不一定要求和Man在同一个配置类中，它可以是通过自动装配实现的，也可以在其他Java配置类中，甚至可以是在xml配置文件中，只要spring能在容器中发现他的存在。

使用XML配置文件进行装配

使用XML配置文件进行bean的装配之前，首先需要创建一个xml配置文件，这一步相当于Java配置类中的@Configuration注解的作用。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

</beans>
```

声明一个简单的Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="com.ctbu.xmlconfigtest.domain.Person"/>

</beans>
```

如上就声明了一个最简单的，以XML配置文件形式装配的bean实例

这里没有明确指定bean的id，所以默认情况下，会以类的全限定名称，也就是：com.ctbu.xmlconfigtest.domain.Person 作为bean的默认id，但是这太麻烦了，每一次引用难道都要写这么大一串？

因此，可以使用id属性指定bean的id；

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person"/>

</beans>
```

XML中构造器参数的注入

XML配置文件中也会存在要配置对象引用的时候，这个使用有两种方式

1. 元素
2. c-命名空间：spring3.0后引入的功能

使用<constructor-arg / >元素

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person"/>

    <bean id="work" class="com.ctbu.xmlconfigtest.domain.Work">
        <constructor-arg ref="person"/>
    </bean>

</beans>
```

可以看到，创建work对象的时候需要注入一个person对象的引用，使用元素实现了这一需求。

使用c-命名空间

这里的c：后面使用了构造器参数的名称

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person"/>

    <bean id="work" class="com.ctbu.xmlconfigtest.domain.Work" c:person-
ref="person"/>

</beans>
```

使用位置标志

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person"/>

    <bean id="work" class="com.ctbu.xmlconfigtest.domain.Work" c:_0-
ref="person"/>

</beans>
```

当构造器只有一个参数的时候，可以直接使用下划线（这种方式在IDEA集成环境下会报错，但是并不影响使用，这是集成环境的检查机制）所以选择一个最合适的即可

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person"/>

    <bean id="work" class="com.ctbu.xmlconfigtest.domain.Work" c:_-ref="person"/>

</beans>
```

构造器中注入字面量值

使用xml元素注入（此时的constructor-arg后不再用ref进行引用，而直接是value属性）

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <constructor-arg value="李白"/>
    </bean>

</beans>
```

使用c-命名空间

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person" c:name="李白"/>

</beans>
```

构造器装配集合

这种装配在XML中只有属性能办到

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

```

```
    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <constructor-arg>
            <list>
                <value>篮球</value>
                <value>足球</value>
                <value>游戏</value>
            </list>
        </constructor-arg>
    </bean>

</beans>
```

这里person对象的构造器需要一个string数据类型的数据，同样的如果实际上你需要的是一个其他对象类型的引用，你可以把value改成ref即可完成bean引用列表的装配。

对于集合类型的变化也可以自行选择，使用list还是set，他们的差别和Java语法中，list和set集合的差别是一样的。

```
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <constructor-arg>
            <set>
                <value>篮球</value>
                <value>足球</value>
                <value>游戏</value>
            </set>
        </constructor-arg>
    </bean>

</beans>
```

XML中属性值的注入

属性值的注入方式也有2种，**都需要提供属性的setter方法**

- XML元素注入
- p命名空间注入

字面量属性注入——XML元素

```xml
<?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:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <property name="name" value="李白"/>
    </bean>

</beans>
```

p-命名空间的方式

```xml
<?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:c="http://www.springframework.org/schema/c"
xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person"
p:name="李白"/>

</beans>
```

若属性值为对象引用，同样使用ref的方式进行引用

```xml
<?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:c="http://www.springframework.org/schema/c"
xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <property name="work" ref="work"/>
    </bean>

    <bean id="work" class="com.ctbu.xmlconfigtest.domain.Work"/>

</beans>
```

属性中集合的装配

```xml
<?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:c="http://www.springframework.org/schema/c"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
       http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
https://www.springframework.org/schema/util/spring-util.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <property name="hobby">
            <list>
                <value>唱</value>
                <value>鬼畜</value>
            </list>
        </property>
    </bean>

</beans>
```

使用util命名空间完成属性集合的装配，如果使用util命名空间则需要在头部引入util命名空间规范。

```
<?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:c="http://www.springframework.org/schema/c"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
https://www.springframework.org/schema/util/spring-util.xsd">

    <bean id="person" class="com.ctbu.xmlconfigtest.domain.Person">
        <property name="hobby" ref="hobby"/>
    </bean>

    <util:list id="hobby">
        <value>唱</value>
        <value>跳</value>
        <value>rap</value>
    </util:list>

</beans>
```

spring中容器配置的导入和混合使用

在使用spring装配bean的时候，可以有三种方式，这三种方式在实际开发中是可以混用的，spring并不排斥这种做法。

所以在使用Java配置类的时候仍然可以使用XML来做为辅助配置，同理自动装配也能与另外两种混用。

在Java配置类中引入XML配置

如果我现在有一个XML配置文件：application.xml，也有一个Java配置类AppConfig.java，现在我需要把XML中的配置引入到Java配置类中，参数最好是注明类路径。这样，在整个spring工程中，使用AppConfig这个配置类作为总的配置来加载，就能获取所有的bean，包括XML中的。

```java
@Configuration
@ImportResource("classpath:application.xml")
public class AppConfig {

    @Bean
    public Person person(){
        return new Person();
    }

}
```

@ImportResource

@ImportResource注解会告诉spring，去指定的文件中加载bean：如@ImportResource("classpath:application.xml")

在Java配置类中导入Java配置类

```java
@Configuration
@Import(WebConfig.class)
public class AppConfig {

    @Bean
    public Person person(){
        return new Person();
    }

}
```

@Import

@Import注解会告诉spring，去那个配置类中导入配置

在XML中导入其他XML配置文件

同样使用XML的import标签

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <import resource="application2.xml"/>

</beans>
```

在XML中导入Java配置类

在XML中导入Java配置，其实就像是在XML中配置Bean一样。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="com.ctbu.source_merge.config.AppConfig"/>
</beans>
```

建议使用的方式

不管是使用XML配置，还是使用Java配置，实际开发中都建议配置一个总的配置类或者XML，用于导入其他配置。这样会显示的层次分明

spring测试

spring的测试仍然需要junit的支持，所以需要引入依赖

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

@RunWith(SpringJUnit4ClassRunner.class)

此注解会在测试的时候自动创建一个spring的容器

@ContextConfiguration(classes = AppConfig.class)

此注解会告诉spring容器去哪里加载配置，示例就是从AppConfig这个类加载bean和其他配置

完整示例

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = AppConfig.class)
public class PersonTest {

    @Autowired
    private Person person;

    @Test
    public void test1(){
        person.work();
    }

}
```

spring—profile：环境快速迁移

spring-profile简单来说就是spring所提供的能快速切换开发模式的功能。很贴近开发的一个例子，我们在编写代码的时候可能用的本地数据库，自然需要配置一个本地数据库的链接，包括用户名，密码等，而项目完成之后将会使用的是云服务器上的数据库，这个时候怎么让测试时候的数据库一下子就切换到云数据库上去？spring-profile就是干这个事儿的。

当我们使用profile的时候首先需要配置profile，也就是决定哪些Bean应该在哪些环境下才允许被创建。当配置完了之后还不够，需要激活profile，这个时候就是程序真正运行的时候，它需要通过你的配置，加载你指定的环境下的Bean，而非指定环境的Bean将不会被创建。

配置profile

配置profile可以有多种方式

- @Profile注解
- XML配置文件

@Profile注解

贴上了@Profile注解的配置类中的Bean，只有在选择了prod这种环境的时候才会被创建。

```
@Configuration
@Profile("prod")
public class SeverDataSourceConfig {
    @Bean
    public ServeConnect serveConnect(){
        return new ServeConnect();
    }
}
```

现在的@Profile注解不仅支持类上，也支持在方法上进行标注。这样可以把不同环境的配置放入同一个配置类中

```
@Configuration
public class LocalDataSourceConfig {

    @Bean
    @Profile("dev")
    public LocalConnect localConnect() {
        return new LocalConnect();
    }

    @Bean
    @Profile("prod")
    public ServeConnect serveConnect(){
        return new ServeConnect();
    }
}
```

在XML中配置profile

在XML中配置profile也很简单，只需在beans标签中配置profile属性即可，这样，这个配置文件下的所有bean只会在对应环境下才会创建

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd"
profile="dev">

    <bean id="localConnect"
class="com.ctbu.spring.profile.model.LocalConnect"/>

</beans>
```

当然，在XML中也能将多个环境配置在一个XML文件中，只需在beans标签中嵌套beans标签，并定义profile属性。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <beans profile="dev">
        <bean id="localConnect"
class="com.ctbu.spring.profile.model.LocalConnect"/>
    </beans>

    <beans profile="prod">
        <bean id="serveConnect"
class="com.ctbu.spring.profile.model.ServeConnect"/>
    </beans>

</beans>
```

激活profile

当profile配置完成了之后，实际在测试或者运行程序的时候，我们需要指定激活哪一种环境。

激活profile依赖于两个独立参数

```
spring.profiles.active
spring.profiles.default
```

active的优先级高于default，当配置了active的时候，会优先使用，若没有配置则使用default的默认配置，若两个都没有配置，那就没有激活的profile，因此就值会创建没有定义profile的Bean。

有多种方式来设置这两个参数

1. DispatcherServlet的初始化参数
2. Web应用的上下文参数
3. JNDI条目
4. 环境变量
5. JVM的系统属性
6. 测试时，使用@ActiveProfiles注解

在web应用中使用web.xml进行激活

```
<context-param>
    <param-name>spring.profiles.active</param-name>
    <param-value>dev</param-value>
</context-param>
```

或者使用代码配置的方式

```
AnnotationConfigApplicationContext acac = new
AnnotationConfigApplicationContext(AppConfig.class);
acac.getEnvironment().setActiveProfiles("dev");
```

在测试时，使用@ActiveProfiles注解

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = AppConfig.class)
@ActiveProfiles({"dev","prod"})
public class LocalConnectTest {

    @Autowired
    private LocalConnect localConnect;

    @Autowired
    private ServeConnect serveConnect;

    @Test
    public void test(){
        System.out.println(localConnect.connect);
        System.out.println(serveConnect.connect);
    }

}
```