1.创建工程

2.改pom,引入jar包

```
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.2.3.RELEASE</version>
 </dependency>
```

3.简单demo

3.1

```
public class HelloWorld {
    private String name;

    public void getName() {
        System.out.println("YOU MESSAGE"+"\n"+name);

    }

    public void setName(String name) {
        this.name = name;
    }
}
```

3.2

```
public class MainApp {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        HelloWorld obj =(HelloWorld) context.getBean("helloWorld");
        obj.getName();

    }
}
```

3.3

```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-3.0.xsd">

    <bean id="helloWorld" class="cn.allweing.HelloWorld">
        <property name="name" value="Hello World!"/>
    </bean>

</beans>
```

4.IOC

4.1bean的作用域

```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-3.0.xsd">

    <!--<bean id="helloWorld" class="cn.allweing.Helloworld" scope="singleton">
    </bean>-->
    <bean id="helloWorld" class="cn.allweing.Helloworld" scope="prototype">
    </bean>

</beans>
```

4.2Spring bean 的生命周期

Bean的生命周期可以表达为：Bean的定义——Bean的初始化——Bean的使用——Bean的销毁

5.依赖注入(基于注解)

5.1Spring 基于注解的配置，修改配置文件

```
<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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>
   <!-- bean definitions go here -->

</beans>
```

5.2四个主要注解

序号	注解 & 描述
1	@Required @Required 注解应用于 bean 属性的 setter 方法。
2	@Autowired @Autowired 注解可以应用到 bean 属性的 setter 方法，非 setter 方法，构造函数和属性。
3	@Qualifier 通过指定确切的将被连线的 bean，@Autowired 和 @Qualifier 注解可以用来删除混乱。
4	JSR-250 Annotations Spring 支持 JSR-250 的基础的注解，其中包括了 @Resource，@PostConstruct 和 @PreDestroy 注解。

5.3@Autowired注解

```
<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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:annotation-config/>
    <!-- bean definitions go here -->

    <bean id="student" class="cn.allweing.Student"/>
    <bean id="say" class="cn.allweing.Say">
        <property name="name" value="zhuyu"/>
        <property name="age" value="12"/>
    </bean>

</beans>
```

```
public class Student {

    @Autowired
    private Say say;

    public void speak() {
        say.getName();
    }
}
```

```
public class Say {
    private String name;

    public String getName() {
        System.out.println(name);
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    private int age;
}
```

```
public class MainApp {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
        Student bean = context.getBean("student", Student.class);
        bean.speak();
    }
}
```

```
zhuyu
```

5.4@Qualifier 注释

Spring @Qualifier 注释

可能会有这样一种情况，当你创建多个具有相同类型的 bean 时，并且想要用一个属性只为它们其中的一个进行装配，在这种情况下，你可以使用 **@Qualifier** 注释和 **@Autowired** 注释通过指定哪一个真正的 bean 将会被装配来消除混乱。下面显示的是使用 @Qualifier 注释的一个示例。

```
<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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:annotation-config/>
    <!-- bean definitions go here -->

    <bean id="student" class="cn.allweing.Student"/>
    <bean id="say" class="cn.allweing.Say">
        <property name="name" value="zhuyu"/>
        <property name="age" value="12"/>
    </bean>
     <bean id="say2" class="cn.allweing.Say">
        <property name="name" value="zhuyuguang"/>
        <property name="age" value="12"/>
    </bean>
</beans>
```

```
public class Student {

    @Autowired
    @Qualifier("say2")
    private Say say;

    public void speak() {
        say.getName();
    }
}
```

5.5Spring JSR-250 注释

Spring JSR-250 注释

Spring还使用基于 JSR-250 注释，它包括 @PostConstruct， @PreDestroy 和 @Resource 注释。因为你已经有了其他的选择，尽管这些注释并不是真正所需要的，但是关于它们仍然让我给出一个简短的介绍。

@PostConstruct 和 @PreDestroy 注释：

为了定义一个 bean 的安装和卸载，我们使用 **init-method** 和/或 **destroy-method** 参数简单的声明一下 。init-method 属性指定了一个方法，该方法在 bean 的实例化阶段会立即被调用。同样地，destroy-method 指定了一个方法，该方法只在一个 bean 从容器中删除之前被调用。

你可以使用 **@PostConstruct** 注释作为初始化回调函数的一个替代，**@PreDestroy** 注释作为销毁回调函数的一个替代，其解释如下示例所示。

@Resource 注释：

你可以在字段中或者 setter 方法中使用 **@Resource** 注释，它和在 Java EE 5 中的运作是一样的。@Resource 注释使用一个 ‘name’ 属性，该属性以一个 bean 名称的形式被注入。你可以说，它遵循 **by-name** 自动连接语义，如下面的示例所示：

```
package com.tutorialspoint;
import javax.annotation.Resource;
public class TextEditor {
   private SpellChecker spellChecker;
   @Resource(name= "spellChecker")
   public void setSpellChecker( SpellChecker spellChecker ){
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker(){
      return spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

如果没有明确地指定一个 ‘name’，默认名称源于字段名或者 setter 方法。在字段的情况下，它使用的是字段名；在一个 setter 方法情况下，它使用的是 bean 属性名称。

5.6Spring 基于 Java 的配置（不需要beans.xml）

@Configuration 和 @Bean 注解

带有 **@Configuration** 的注解类表示这个类可以使用 Spring IoC 容器作为 bean 定义的来源。**@Bean** 注解告诉 Spring，一个带有 @Bean 的注解方法将返回一个对象，该对象应该被注册为在 Spring 应用程序上下文中的 bean。最简单可行的 @Configuration 类如下所示：

```
package com.tutorialspoint;
import org.springframework.context.annotation.*;
@Configuration
public class HelloWorldConfig {
   @Bean
   public HelloWorld helloWorld(){
      return new HelloWorld();
   }
}
```

上面的代码将等同于下面的 XML 配置：

```
<beans>
   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld" />
</beans>
```

在这里，带有 @Bean 注解的方法名称作为 bean 的 ID，它创建并返回实际的 bean。你的配置类可以声明多个 @Bean。一旦定义了配置类，你就可以使用 *AnnotationConfigApplicationContext* 来加载并把他们提供给 Spring 容器，如下所示：

```
public static void main(String[] args) {
   ApplicationContext ctx =
   new AnnotationConfigApplicationContext(HelloWorldConfig.class);
   HelloWorld helloWorld = ctx.getBean(HelloWorld.class);
   helloWorld.setMessage("Hello World!");
   helloWorld.getMessage();
}
```

你可以加载各种配置类，如下所示：

```
public static void main(String[] args) {
   AnnotationConfigApplicationContext ctx =
   new AnnotationConfigApplicationContext();
   ctx.register(AppConfig.class, OtherConfig.class);
   ctx.register(AdditionalConfig.class);
   ctx.refresh();
   MyService myService = ctx.getBean(MyService.class);
   myService.doStuff();
}
```

@Import 注解:

@import 注解允许从另一个配置类中加载 @Bean 定义。考虑 ConfigA 类，如下所示：

```
@Configuration
public class ConfigA {
   @Bean
   public A a() {
      return new A();
   }
}
```

你可以在另一个 Bean 声明中导入上述 Bean 声明，如下所示：

```
@Configuration
@Import(ConfigA.class)
public class ConfigB {
   @Bean
   public B a() {
      return new A();
   }
}
```

现在，当实例化上下文时，不需要同时指定 ConfigA.class 和 ConfigB.class，只有 ConfigB 类需要提供，如下所示：

```
public static void main(String[] args) {
   ApplicationContext ctx =
   new AnnotationConfigApplicationContext(ConfigB.class);
   // now both beans A and B will be available...
   A a = ctx.getBean(A.class);
   B b = ctx.getBean(B.class);
}
```

指定 Bean 的范围：

默认范围是单实例，但是你可以重写带有 @Scope 注解的该方法，如下所示：

```
@Configuration
public class AppConfig {
   @Bean
   @Scope("prototype")
   public Foo foo() {
      return new Foo();
   }
}
```

6.Spring -AOP

6.1Spring 框架的 AOP

AOP 术语

在我们开始使用 AOP 工作之前，让我们熟悉一下 AOP 概念和术语。这些术语并不特定于 Spring，而是与 AOP 有关的。

项	描述
Aspect	一个模块具有一组提供横切需求的 APIs。例如，一个日志模块为了记录日志将被 AOP 方面调用。应用程序可以拥有任意数量的方面，这取决于需求。
Join point	在你的应用程序中它代表一个点，你可以在插件 AOP 方面。你也能说，它是在实际的应用程序中，其中一个操作将使用 Spring AOP 框架。
Advice	这是实际行动之前或之后执行的方法。这是在程序执行期间通过 Spring AOP 框架实际被调用的代码。
Pointcut	这是一组一个或多个连接点，通知应该被执行。你可以使用表达式或模式指定切入点正如我们将在 AOP 的例子中看到的。
Introduction	引用允许你添加新方法或属性到现有的类中。
Target object	被一个或者多个方面所通知的对象，这个对象永远是一个被代理对象。也称为被通知对象。
Weaving	Weaving 把方面连接到其它的应用程序类型或者对象上，并创建一个被通知的对象。这些可以在编译时，类加载时和运行时完成。

通知的类型

Spring 方面可以使用下面提到的五种通知工作:

通知	描述
前置通知	在一个方法执行之前，执行通知。
后置通知	在一个方法执行之后，不考虑其结果，执行通知。
返回后通知	在一个方法执行之后，只有在方法成功完成时，才能执行通知。
抛出异常后通知	在一个方法执行之后，只有在方法退出抛出异常时，才能执行通知。
环绕通知	在建议方法调用之前和之后，执行通知。

6.2Spring 中基于 AOP 的 XML架构

```
<dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.5</version>
        </dependency>
```

```
<?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:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <aop:config>
        <aop:aspect id="log" ref="logging">
            <aop:pointcut id="selectAll"
                          expression="execution(* cn.allweing.*.*(..))"/>
            <aop:before pointcut-ref="selectAll" method="beforeAdvice"/>
            <aop:after pointcut-ref="selectAll" method="afterAdvice"/>
            <aop:after-returning pointcut-ref="selectAll"
                                 returning="retVal"
                                 method="afterReturningAdvice"/>
```

```
            <aop:after-throwing pointcut-ref="selectAll"
                                throwing="ex"
                                method="AfterThrowingAdvice"/>
        </aop:aspect>
    </aop:config>

    <!-- Definition for student bean -->
    <bean id="student" class="cn.allweing.Student">
        <property name="name" value="Zara"/>
        <property name="age" value="11"/>
    </bean>

    <!-- Definition for logging aspect -->
    <bean id="logging" class="cn.allweing.Logging"/>

</beans>
```

```
public class Student {
    private Integer age;
    private String name;

    public void setAge(Integer age) {
        this.age = age;
    }

    public Integer getAge() {
        System.out.println("Age : " + age);
        return age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        System.out.println("Name : " + name);
        return name;
    }

}
```

```
public class Logging {

    public void beforeAdvice(){
        System.out.println("Going to setup student profile.");
    }
    /**
     * This is the method which I would like to execute
     * after a selected method execution.

     */
```

```
    public void afterAdvice(){
        System.out.println("Student profile has been setup.");
    }
    /**
     * This is the method which I would like to execute
     * when any method returns.
     */
    public void afterReturningAdvice(Object retVal){
        System.out.println("Returning:" + retVal.toString() );
    }
    /**
     * This is the method which I would like to execute
     * if there is an exception raised.
     */
    public void AfterThrowingAdvice(IllegalArgumentException ex){
        System.out.println("There has been an exception: " + ex.toString());
    }
}
```

```
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
        Student student = (Student) context.getBean("student");
        student.getName();
        student.getAge();
    }
}
```

6.3Spring 中基于 AOP 的 注解架构

```
<dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.5</version>
        </dependency>
```

```
<?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:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <aop:aspectj-autoproxy/>

    <bean id="logging" class="cn.allweing.Logging"/>

    <bean id="student" class="cn.allweing.Student"/>
```

```
</beans>
```

```
public class Student {

    public void say() {
        System.out.println("cn.allweing.Student**********say******");
    }

    public void speak() {
        System.out.println("cn.allweing.Student**********speak*****");
    }
}
```

```
@Aspect
public class Logging {

    @Pointcut("execution(* cn.allweing.*.*(..))")
    public void all() {

    }
    @Before("all()")
    public void beforAdvice() {
        System.out.println("cn.allweing.Logging************beforAdvice*********");
    }
    @After( "all()")
    public void AfterAdvice() {
        System.out.println("cn.allweing.Logging*************AfterAdvice********");
    }

}
```

```
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
        Student student = context.getBean("student",Student.class);
        student.speak();
        student.say();

    }
}
```