**Spring Boot依赖注入的三种方式**

官方文档5.3.3：[https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-dependencies](https://links.jianshu.com/go?to=https%3A%2F%2Fdocs.spring.io%2Fspring-framework%2Fdocs%2Fcurrent%2Freference%2Fhtml%2Fcore.html%23beans-dependencies" \t "_blank)

**准备工作**

* 初始化一个SpringBoot项目
* 创建Weslie、Wolffy两个实体类
* 创建Main类

**[java]**

@Component  
public class Weslie {  
  
 public String getName(){  
 return "Weslie";  
 }  
  
 public String getRace(){  
 return "Caprinae";  
 }  
}

**[java]**

@Component  
public class Wolffy {  
  
 public String getName() {  
 return "Wolffy";  
 }  
  
 public String getRace() {  
 return "Lupo";  
 }  
}

可以看到Weslie、Wolffy两个类上都打上了@Component注解，该注解将某个类声明为一个Spring的bean, 然后将其加入到Spring容器中，这是实现注入的前提。（Service、Controller等注解实现注入同样依赖于Component注解）

**注入方式**

Bean的注入通常使用@Autowired注解，该注解用于bean的field、setter方法以及构造方法上，显式地声明依赖。  
在最新的文档中注入方式有两大类：

* 基于构造函数的依赖注入（推荐使用）
* 基于setter的依赖注入

但是通常认为还有一种是基于成员变量的依赖注入（spring framerwork 4.0后不推荐使用）

**成员变量注入**

**[java]**

@SpringBootApplication  
public class Main implements CommandLineRunner {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Main.class, args);  
 }  
  
 @Autowired  
 private Weslie weslie;  
 @Autowired  
 private Wolffy wolffy;  
@Override  
 public void run(String... args) throws Exception {  
 System.***out***.println("name:"+weslie.getName()+"\trace:"+weslie.getRace());  
 System.***out***.println("name:"+wolffy.getName()+"\trace:"+wolffy.getRace());  
 }  
}

输出：

name:Weslie race:Caprinae

name:Wolffy race:Lupo

**构造函数注入**

* 方式一

**[java]**

@SpringBootApplication  
public class Main implements CommandLineRunner {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Main.class, args);  
 }private Weslie weslie;  
 private Wolffy wolffy;

@Autowired  
 public Main(Weslie weslie,Wolffy wolffy){  
 this.weslie = weslie;  
 this.wolffy = wolffy;  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 System.***out***.println("name:"+weslie.getName()+"\trace:"+weslie.getRace());  
 System.***out***.println("name:"+wolffy.getName()+"\trace:"+wolffy.getRace());  
 }  
}

* 方式二  
  该方法利用lombok的注解**@RequiredArgsConstructor**实现构造器注入，需要注意的是要注入的属性需要加上final修饰

首先要在pom.xml中添加依赖：

**[pom.xml]**

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
</dependency>

**[java]**

@SpringBootApplication  
@RequiredArgsConstructor  
public class Main implements CommandLineRunner {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Main.class, args);  
 }  
private final Weslie weslie;  
 private final Wolffy wolffy;  
  
 @Override  
 public void run(String... args) throws Exception {  
 System.***out***.println("name:"+weslie.getName()+"\trace:"+weslie.getRace());  
 System.***out***.println("name:"+wolffy.getName()+"\trace:"+wolffy.getRace());  
 }  
}

**setter注入**

**[java]**

@SpringBootApplicationpublic class Main implements CommandLineRunner {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Main.class, args);  
 }private Weslie weslie;  
 private Wolffy wolffy;  
   
 @Autowired  
 public void setWeslie(Weslie weslie) {  
 this.weslie = weslie;  
 }  
   
 @Autowired  
 public void setWolffy(Wolffy wolffy) {  
 this.wolffy = wolffy;  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 System.***out***.println("name:"+weslie.getName()+"\trace:"+weslie.getRace());  
 System.***out***.println("name:"+wolffy.getName()+"\trace:"+wolffy.getRace());  
 }  
}

**拓展 autowired注入形式**

首先改造一下我们的项目，将Weslie和Wolffy两个类继承于User接口（需要新建），看看运行效果

**[java]**

public interface User {  
 String getName();  
 String getRace();  
}

**[java]**

@Component  
public class Weslie implements User{  
  
 @Override  
 public String getName(){  
 return "Weslie";  
 }  
  
 @Override  
 public String getRace(){  
 return "Caprinae";  
 }  
}

**[java]**

@Component  
public class Wolffy implements User {  
  
 @Override  
 public String getName() {  
 return "Wolffy";  
 }  
  
 @Override  
 public String getRace() {  
 return "Lupo";  
 }  
}

**[java]**

@SpringBootApplication  
@RequiredArgsConstructor  
public class Main implements CommandLineRunner {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Main.class, args);  
 }private final User weslie;  
 private final User wolffy;  
  
 @Override  
 public void run(String... args) throws Exception {  
 System.***out***.println("name:"+weslie.getName()+"\trace:"+weslie.getRace());  
 System.***out***.println("name:"+wolffy.getName()+"\trace:"+wolffy.getRace());  
 }  
}

* 情景一，变量声明为User user，且只有weslie实现了User接口，则控制台输出name:Weslie race:Caprinae
* 情景二，变量声明为User user，weslie和wolffy同时实现了User接口，则运行程序报错constructor in com.example.controller.HelloController required a single bean, but 2 were found:
* 情景三，变量声明为Person weslie，控制台输出name:Weslie race:Caprinae
* 情景四，变量声明为Person wolffy，控制台输出name:Wolffy race:Lupo

上面四个情景展现了Spring中Autowired的两种方式

* byType，默认的注入方式，与Bean的属性具有相同类型的其他Bean自动装配到Bean的对应属性中。
* byName，与Bean的属性具有相同名字的其他Bean自动装配到Bean的对应属性中

装配方式总结：

* 找不到任何一个bean报错
* 一个会直接注入
* 找到多个，不一定会报错，会按照字段名注入，如果没有同名字的bean则报错

**技巧：**使用@Qualifier  
@Qualifier注解是和@Autowired一起使用的。使用此注解可以让你对注入的过程有更多的控制。@Qualifier可以被用在单个构造器或者方法的参数上。当上下文有几个相同类型的bean, 使用@Autowired则无法区分要绑定的bean，此时可以使用@Qualifier来指定名称。

**[java]**

@SpringBootApplicationpublic class Main implements CommandLineRunner {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Main.class, args);  
 } *//使用接口* @Qualifier("weslie")  
 @Autowired  
 private User user1;  
 @Qualifier("wolffy")  
 @Autowired  
 private User user2;  
  
 @Override  
 public void run(String... args) throws Exception {  
 System.***out***.println("name:"+user1.getName()+"\trace:"+user1.getRace());  
 System.***out***.println("name:"+user2.getName()+"\trace:"+user2.getRace());  
 }  
}

输出：

name:Weslie race:Caprinae

name:Wolffy race:Lupo

如果你想同时使用RequiredArgsConstructor和Qualifier，仅仅写成下面这样还是不行的，需要在项目根目录下新建 [lombok.config](https://links.jianshu.com/go?to=https%3A%2F%2Fprojectlombok.org%2Ffeatures%2Fconfiguration" \t "_blank)并写入

lombok.copyableAnnotations += org.springframework.beans.factory.annotation.Qualifier