**Spring**

目录

[概述 1](#_Toc24505)

[Spring组成 1](#_Toc21627)

[IOC容器 2](#_Toc21474)

[依赖注入(DI) 2](#_Toc29146)

[bean的作用域 3](#_Toc25562)

**概述**

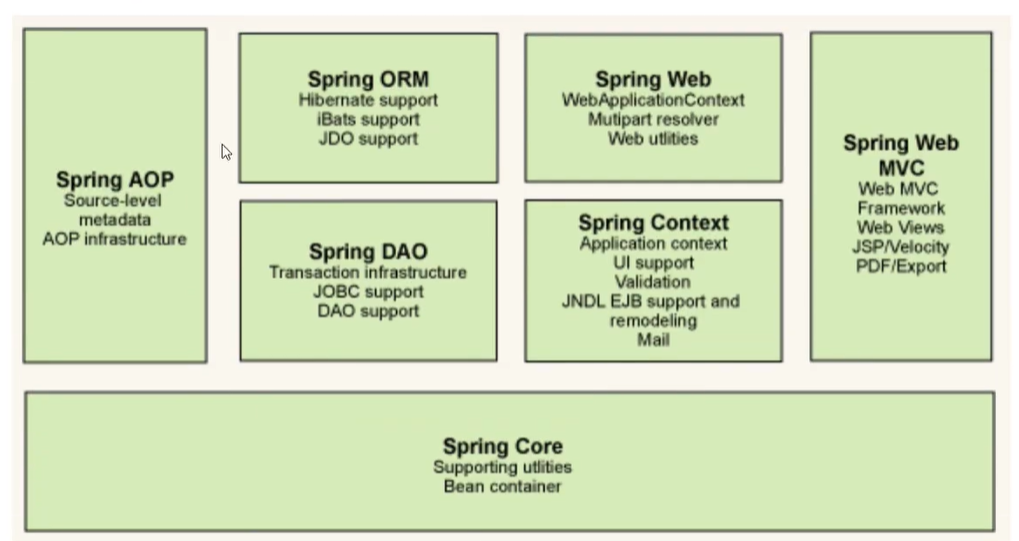
Spring 春天。Spring框架的出现，解决了企业应用开发的复杂性。

Spring 优点

* Spring是一个开源、免费的框架（容器）
* Spring是一个轻量级的，非入侵式的框架
* Spring的核心: 控制反转(IOC)和面向切面编程(AOP)
* 支持事务的处理，对框架整合的支持

总结: Spring是一个轻量级的控制反转和面向切面编程的框架。

**Spring组成**



官网地址: <https://spring.io/projects/spring-framework#overview>

maven中央仓库地址: https://mvnrepository.com/

**IOC容器**

IOC（控制反转）：是一种思想，DI(依赖注入)是实现IOC的一种方式。IOC的核心思想，总结就是所有的java对象都由SpringIoC容器创建、管理和装配。

org.springframework.context.ApplicationContext接口代表 Spring IoC 容器，负责实例化、配置和组装 bean。容器通过读取配置元数据来获取有关要实例化、配置和组装哪些对象的指令。

**依赖注入**

依赖注入有两种方式:

- [基于构造函数的依赖注入](https://www.docs4dev.com/docs/zh/spring-framework/5.1.3.RELEASE/reference/core.html" \l "beans-constructor-injection)（通过bean类的构造器注入）

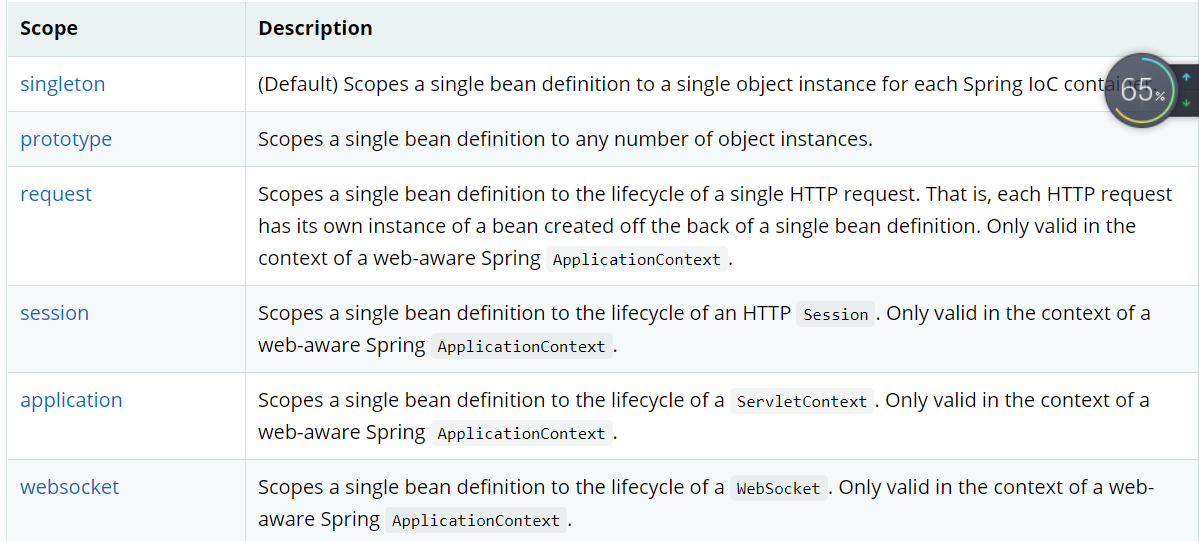
- 基于setter(set方法)的依赖注入（通过属性的set方法注入）(重点)

- 扩展注入

1. p命名空间注入，可以直接给bean的属性赋值。p命名空间注入本质还是setter注入，需要提供set方法。
2. c命名空间注入。c命名空间注入本质还是构造器注入，需要提供带参数的构造方法。

注: p和c命名空间注入，不能直接使用，需要导入约束。

**bean的作用域**



- 单例模式（singleton），每次从容器get的时候，拿到的都是同一个对象

- 原型模式（prototype），每次从容器get的时候，都会产生一个新的对象

- request，session和application就是在Web开发时bean的作用域。

注: 如果没有显示的指明，默认是singleton。

**bean的自动装配**

在容器中注入一个bean，如果这个bean包含其它属性，除了手动的一个个为其装配属性外，还可以自动装配属性。自动装配是spring满足bean依赖的一种方法。有以下3中自动装配属性的方法。

- ByType自动装配

根据bean的属性类型，在IoC容器中查找，如果存在两个类型相同的

bean，则报错。

- ByName自动装配

根据bean的属性名，在IoC容器中查找，如果存在两个名字相同的bean，

则报错。

- 使用注解实现自动装配。

@Autowired注解。该注解默认通过ByType进行装配，如果容器中存在

多个满足类型的实例(bean)，就会再通过ByName进行装配，如果容器中

又存在多个满足Name的实例，则装配失败，抛出异常。所以当容器中存

在多个类型相同的实例时，可以使用@Qualifier注解来指明到底转配哪一

个实例。

@Resource注解。该注解默认通过ByName进行装配，如果在容器中找

不到，则再通过ByType装配。@Resource注解中有一个name属性，如

果容器中出现多个name相同的实例时，可以通过name属性指明到底转

配哪一个实例。注意: @Resource注解是JDK中自带的，但从JDK11开始，

该注解从JDK中移除。

**spring中常用的注解**

@Autowired

@Qualifier

@Resource

@Component 该注解放在类上,说明这个类被spring管理了，spring容器会

将这个类部署成bean

@Service

@Repository

@Controller

注: @Service，@Repository和@Controller是@Component的衍生注解，它

们的功能一样，都是将某个类注入到spring容器，并装配bean。

@Scope 该注解用于指定bean的作用域

**代理模式**

代理模式的核心思想是借助代理类在不修改原有代码(核心代码)的基础上，

拓展新的功能。代理模式分为静态代理和动态代理。

**静态代理**

抽象角色（接口）

真实角色（接口的实现类）

代理角色（接口的调用者）

**动态代理**

动态生成的代理类是是一个虚拟的类，不是直接写好的。动态代理分两类:

基于接口的动态代理和基于类的动态代理。常见的实现动态代理的方法有:

JDK动态代理，即使用jdk反射包下的接口和类实现动态代理

javassist动态代理，即使用javassist工具实现动态代理

在学习JDK实现动态代理前，需要先了解以下3个重要的API:

- Method类，代表类中的方法，切确的说就是目标类中的方法。

作用: 通过Method可以执行某个目标类中的方法，也就是目标对象的方

法。

该类中有一个重要方法，如下所示:

public Object invoke(Object obj, Object... args); //Method对象使用该

方法，表示执行方法的调用。obi是目标对象，args是目标对象方法的参

数，即调用哪个目标对象的方法。invoke方法的返回值就是目标对象方法

的返回值。

- InvocationHandler接口，调用处理器

该接口中只有一个invoke方法，在该方法中编写代理对象要执行的功能代

码，简单讲代理类要完成的功能就写在这个invoke方法中。该方法如下:

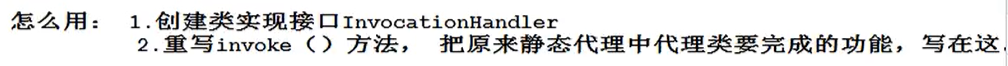
public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable;

该方法中的三个参数详解:

proxy 为jdk创建的代理对象，无需赋值

method 为目标类中的方法，jdk会提供这个method对象，无需赋值

args 为目标类中方法的参数，jdk会提供这个参数，无需赋值。 

注: InvocationHandler就代表动态生成的代理类要干什么。

- proxy类

该类中有一个静态的方法newProxyInstance，用于创建代理对象。该方法

如下:

public static Object newProxyInstance(ClassLoader loader,

Class<?>[] interfaces,

InvocationHandler h)

throws IllegalArgumentException;

该方法中的三个参数详解:

loader 是目标对象的类加载器。假如a是目标对象，通过a.getClass().classLoader(); 就可以获取对象的类加载器。

Interfaces 是目标对象实现的接口

h 是调用处理器对象，该对象会自动调用其invoke()方法。前面提到过，

invoke方法中就包含代理类要完成的功能。

注: Method类中的invoke方法和InvocationHandler接口中的invoke方

法，不是同一个方法，别弄混淆。

总结: 动态创建代理类的步骤

