spring经典注解回顾

1.spring组件注入注解

2.spring中一些注入Bean的辅助注解

3.spring中Bean的声命周期相关注解

  3.1.spring的生命周期有哪写

  3.2.注解发现

  3.3.初始化和销毁

  3.4.初始化前和初始化后

4.Bean的属性赋值

5.Bean的自动装配

6.Autowired注解的使用

7.spring中的黑科技

8.profile环境部署注解

9.aop简单注解基础

10.aop原理

11.spring事务注解

12.spring的三级缓存

1.spring组件注入注解

- Bean注解

  多用于配置的方式注入bean，bean的名字就是方法名或者你指定好的名字。
- Import注解

  一次导入一个类也叫Bean;

  或者实现一个ImportSelector接口导入一组类;

```java
public class BeanSelectImport implements ImportSelector {
    @Override
    public String[] selectImports(AnnotationMetadata annotationMetadata)
{
        //组件全限定名
         String[] arrStr={"com.edu.bean.A","com.edu.bean.B"};
         return arrStr;
    }
}
```

或者实现一个叫ImportBeanDefinitionRegister接口；

```java

public class BeanSelectImportRegister implements
ImportBeanDefinitionRegistrar {
    @Override
    public void registerBeanDefinitions(AnnotationMetadata
importingClassMetadata, BeanDefinitionRegistry registry) {
        RootBeanDefinition rootBeanDefinition = new
RootBeanDefinition(Register.class);
        registry.registerBeanDefinition("register2",rootBeanDefinition);
    }
}
```

- CompentScan注解

  配置类的扫包注解：结合service restroity compent controller

```java
@ComponentScan(
basePackages = {"com.edu"}
,excludeFilters ={@ComponentScan.Filter(classes =
{UserMapperImpl.class},type = FilterType.ASSIGNABLE_TYPE)}
)
```

- FactoryBean接口（注意不是BeanFactory）

用于将对象放入IOC中，也就是自定义一个FactoryBean实现FactoryBean接口，返回想要注入的Bean,在配置 类使用Bean配置，当我们get的方法名的时候就是这个返回的Bean,如果想要自定义的FactoryBean。我们需要通过getObject("&color");

- Conditional注解

用于按照某种条件进行Bean的注入，如项目中有这个类我们在装配，或者没有那个类我们在生效。springboot用的比较多;

Java 复制代码

```java
//配置类  组件注入
@Configuration
//1.ComponetScan注解注入
@ComponentScan({"com.edu.service","com.edu.mapper"})
//4.Import
@Import({Cart.class,BeanSelectImport.class,
BeanSelectImportRegister.class})
//5.ImportSelector
//6.ImportBeanDefinitionRegistrar
public class ApplicationConfig {
    //2.factoryBean组件注入
    @Bean
    public StudentFactory studentFactory(){
        return new StudentFactory();
    }
    //3. bean组件注入注意用法
    @Bean
    public User myUser(){
        return new User("001","hehe","6666");
    }
    //7.condational
}
```

2.spring中一些注入Bean的辅助注解

1. filter 用于过滤那个包不需要扫描
2. scope 用于定义一个Bean的作用域
3. CompentScans 用于存放多个CompentScan注解
4. Configrution 配置类注解标志

3.spring中Bean的声命周期相关注解

3.1.spring的生命周期有哪写

注解发现--->构造推断--->属性注入--->初始化前---->初始化----->初始化后---->销毁

3.2.注解发现

就是spring传入配置类，到定义注解到发现注解的一个过程

3.3.初始化和销毁

- Bean(initMethod="",destoryMethod="")这两个方法就定义在Bean的类中;
- 实现两个接口InitializingBean和DisposableBean两个接口让bean去实现，即可
- JSR250风格：PostConstruct(构造之后） PreDestroy（销毁之前）

3.4.初始化前和初始化后

1. BeanPostProcessor（bean的后置处理器）

当我们要在一些Bean的初始化前和初始化后做一些事情我们可以实现这个接口.描述：处理器两个方法before after的重要参数bean(初始化bean的对象)，beanName(bean的名字)---->当前bean.

2. bean的postProcesser的原理

遍历容器中所有的BeanPostProcessor，依次执行所有的processor,一旦前置返回null,那么后面的后置也就不需要执行了，初始化都在属性注入之后。

注意：一定要注入到容器中

4.Bean的属性赋值

- PropertySource(value="classpath:db.propties") + Value("")

  value中字符串一共可以分为三种存放：${key} ， #{key}EL表达式 ,key的形式

  而我么的springboot配置呢与之有点不太一样：使用ConfigurtionProperty+compent+value和ConfigurtionProperty+EnableConfigtionProperty(car.class)

```java
@PropertySource(value = {"classpath:cart.properties"},encoding = "utf-8")
public class SpringBlackCreateConfig {
    @Bean
    public Cart cart(){
        return new Cart();
    }
}
public class Cart {
    @Value("guihaolecid")
    String cid;
    @Value("#{20-2}")
    String cart_sum;
    @Value("${cart_state}")
    String cart_state;
    @Value("${cart_price}")
    String cart_price;
}
```

5.Bean的自动装配

- Autowired先类型后值
- Qualifier指定形式装配
- Resource不可以于其他注解配合使用，作用于Autowired用法一致
- inject与Autowired一样，但需要导包

```java
    @Autowired
    //Qualifer----假如容器中一个类的类型对应两个bean对象,可以指定装配
    //@Resource----和autowired的使用放式相同，唯一不同的是它不可以配合其他注解
    //@Inject----要导包
    private UserMapper userMapper;
    @Override
    public void systemPrint() {
      userMapper.selectById();
    }
```

6.Autowired注解的使用

1. 存放的位置：构造器，参数，方法，属性
2. 参数和方法上： 就是让传的参数去spring容器找对应的bean对象

3. 构造器上：推断构造器，如果只有一个构造器，我们不需要加这个注解，spring会自动根据这个构造器构造bean对象，但有多个构造器并且没有无参，我们必须指定一个构造器，就用这个注解，如果多个构造器有无参我们默认使用这个无参。
4. 属性：自动注入么。

```java
    //测试一下AutoWried
    //1.bean  ----可以省略
    //参数上-----从spring容器中找
    @Bean
    public Cart cart1(@Autowired Student stu){
        Cart cart = new Cart();
        cart.setStudent(stu);
        return cart;
    }
    //2.属性上-----bytype---byname

    //3.构造器上----推断构造方法  一个可以省略

    //4.方法上-----从spring创建bean的时候在属性注入的阶段进行调用
    @Bean
    public Student stu(){
        return new Student();
    }

    public class Student {
        private String username;
        private String password;
        private String uid;
        @Autowired   //相当于属性注入阶段
        public void setInit(){
            this.username="guihaole";
            this.password="123456";
            System.out.println("我在使用autowired");
        }
        @PostConstruct  //初始化阶段
        public void init() {
            System.out.println("初始化initMethod");
        }
        @PreDestroy
        public void destory() {
            System.out.println("销毁initDestory");
        }

    }
```

7.spring中的黑科技

自定义组件使用spring容器底层的一些组件application beanfactory 自定义实现了aware回调

- applicationcontext实现他我们可以获取这个容器
- beanName　当前bean的名字
- stringvalueRestoveraware解析字符串　实现它我们可以用一个方法resolveStringValue("你好：${key}")；自动解析

8.profile环境部署注解

- 作用：在不同开发环境中使用不同的配置，可以利用这个注解定义多种配置，在不同环境下使用。
- bean注入的时候：profile("test")　profile("dev")　profile("prod")
- 切换环境：1.使用命令行参数-Dspring.profiles.active=test
  2.application.getEnviroment().setActProfies("test","dev")执行有参逻辑
- 位置：可以写在类上，也可以下载方法上

9.aop简单注解基础

1. 核心jar包：spring-aspects
2. 实现流程：

- 定义业务类和切面类，并告诉spring那个是切面那个是业务
- 定义好切点
- 加上EnableAspectJIntoProxy注解

3. 切点注解Before ,after,afterreturning,afterThrowing,around五种类型的通知也叫增强器
4. Pointcut(execution("修饰符　返回类型　全限定名（ 类型：参数")));Before("pointcut()");
5. JoinPoint可以获取参数，方法名等

Java | 复制代码

```java
//步骤aop
//1.向spring容器定义切面类和通知类 并告诉spring那个是切面类那个是通知类
public class springConfig(){
   @Bean
    public ActionClass actionClass(){
        return new ActionClass();
    }
    @Bean
    public AspectClass aspectClass(){
        return new AspectClass();
    }
}
----------------------------------------高级配置----------------------------
-------------------
@Aspect
public class ActionClassContext {

    @Pointcut("execution(public * 
com.edu.service.impl.UserServiceImpl.selectUser(..))")
    public void pointCut(){

    }
    @Before("pointCut()")
    public void BeforeProp(JoinPoint joinPoint){
            System.out.println("方法之前通知方法参数为：{"+
Arrays.toString(joinPoint.getArgs()) +"}");
    }
    @After("com.edu.Aop.ActionClassContext.pointCut()")
    public void AfterProp(){
        System.out.println("方法之后通知");
    }
    @AfterReturning(value = "pointCut()",returning = "result")
    public void AfterReturn(JoinPoint joinPoint,Object result){
        System.out.println("方法名
为"+joinPoint.getSignature().getName()+"正常返回通知-->返回的值："+result);
    }
    @AfterThrowing(value = "pointCut()",throwing = "exception")
    public void AfterExecption(Exception exception){
        System.out.println("方法异常通知，异常信息
为:"+exception.getMessage());
    }
//    @Around("pointCut()")
//    public void AroundProp(){
//        System.out.println("方法环绕通知");
//    }
```

```
41    }
42    //3.开启aop配置注解
43    @EnableAspectJAutoProxy
```

10.aop原理

1. @EnableAspectJAutoProxy注解实际上是使用引入一个AnnotationAwareAspectJAutoProxyCreator组件，这个组件其实是实现了BeanPostProcessor接口，实际他也是一个处理器
2. spring启动过程：spring容器创建，reflush刷新spring容器，然后会依次遍历所有beanpostprocessor,这个时候就会得到AnnotationAwareAspectJAutoProxyCreator这个处理器，在前置过程中，会寻找有没有切面类，也就是找@Aspect这个注解，如果找到了，就在后置处理器解析所有的增强器，并对其进行排序，然后在获取增强器不为null的条件下创建代理对象，保存到代理工厂中，proxyFactory,然后让spring自动决定JDK动态代理还是CGlib形式
3. cglib动态代理AOP：首先拦截目标方法，然后获取增强器链，然后挨个去执行增强方法，再去执行目标方法.

11.spring事务注解

1. 向spring容器中注入DataSourceFransactionManager类
2. 开启事务注解EnableTransactionManagement
3. 事务注解声明

Java | 复制代码

```
1       //事务
2       @Bean
3       public PlatformTransactionManager transactionManager(DataSource
    dataSource){
4           PlatformTransactionManager platformTransactionManager = new
    DataSourceTransactionManager(dataSource);
5           return platformTransactionManager;
6       }
```

12.spring的三级缓存

1. 场景问题：spring在生成bean代理对象的时候一般都会经理几个步骤，创建普通对象，属性注入，初始化，aop，生成bean对象。假如一个Aservice依赖于Bservice,也就是说：A创建普通对象，属性注入B，结果B还没有创建，然后去创建B，发现B需要A，属性注入A，A在创建导致循环依赖
2. 解决方案:

- 创建一个createSet集合用于存放正在创建的bean，创建完成后就删除了
- Aservice类--->实例化-------->Aservice对象原始对象（三级缓存）
- 填充B----->从单例池（一级缓存）找--->没有找到------>二级缓存------>set中找没有----->创建b对象
- Bservice类--->实例化-------->Bservice对象原始对象（三级缓存）
- 填充A----->单例池--->二级缓存----->set?有----->发现循环依赖------>去判断这个A需要AOP？需要先Aop，不需要不AOP--->放入二级缓存
- 填充B，放入单例池
- A填充属性B
- 放入单例池----->移除set中的bean

3. 总结：一级缓存是用来存放完整声明周期的bean的，二级缓存是存放未完整生命周期的bean的,三级缓存是用来打破循环依赖的；