### JAVA第三阶段—DAY04-JAVA案例

1. IOC入门案例，使用XML配置

带学员熟悉IOC的相关配置

/\*\*

\* 电脑

\*/

public class Computer {

private String brand;

private Cpu cpu;

private Memory memory;

public Cpu getCpu() {

return cpu;

}

public void setCpu(Cpu cpu) {

this.cpu = cpu;

}

public Memory getMemory() {

return memory;

}

public void setMemory(Memory memory) {

this.memory = memory;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public void start(){

System.out.println(brand + "电脑启动了！");

cpu.run();

memory.read();

memory.write();

}

}

配置

<?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="cpu" class="com.blb.ioc\_demo.AMDCpu"></bean>

<bean id="memory" class="com.blb.ioc\_demo.KingstonMemory"></bean>

<!--Java对象配置-->

<bean id="computer" class="com.blb.ioc\_demo.Computer">

<property name="brand" value="联想"></property>

<property name="cpu" ref="cpu"></property>

<property name="memory" ref="memory"></property>

</bean>

</beans>

测试

public class TestComputerSpring {

public static void main(String[] args) {

//创建应用程序上下文

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

//获得对象

Computer computer = (Computer) context.getBean("computer");

computer.start();

}

}

1. 注解配置案例

讲解注解的配置方法

@Component

public class AMDCpu implements Cpu {

public void run() {

System.out.println("AMD的CPU在运行。。。");

}

}

@Component

public class KingstonMemory implements Memory {

public void read() {

System.out.println("金士顿内存读取数据");

}

public void write() {

System.out.println("金士顿内存写入数据");

}

}

值类型属性用@Value注入，引用类型用@Autowired注入

@Component

public class Computer {

@Value("戴尔")

private String brand;

@Autowired

private Cpu cpu;

@Autowired

private Memory memory;

配置类

/\*\*

\*配置类

\*/

@ComponentScan(basePackages = "com.blb.ioc\_demo")

@Configuration

public class ComputerConfig {

public static void main(String[] args) {

//创建注解应用程序上下文

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(ComputerConfig.class);

//获得对象

Computer computer = context.getBean(Computer.class);

computer.start();

}

}

1. 反射+注解实现IOC案例

讲解IOC的原理

/\*\*  
\* 注入值的注解  
\*/  
@Target(ElementType.FIELD)  
@Retention(RetentionPolicy.RUNTIME)  
public @interface MyValue {  
  //注入的值  
  String value();  
}

/\*\*  
\* 注入对象的注解  
\*/  
@Target(ElementType.FIELD)  
@Retention(RetentionPolicy.RUNTIME)  
public @interface MyComponent {  
//注入的类型  
  Class value();  
}

/\*\*  
\* 电脑  
\*/  
public class Computer {  
​  
  @MyValue("戴尔")  
  private String brand;  
​  
  @MyComponent(IntelCpu.class)  
  private Cpu cpu;  
​  
  @MyComponent(SumsungMemory.class)  
  private Memory memory;  
  ...省略其余部分...

/\*\*  
\* 电脑工厂  
\*/  
public class ComputerFactory {  
​  
  /\*\*  
    \* 创建电脑对象  
    \* @param computerClass  
    \* @return  
    \*/  
  public Computer createComputer(Class computerClass) throws Exception {  
      //反射创建对象  
      Object computer = computerClass.newInstance();  
      //遍历所有的属性  
      Field[] fields = computerClass.getDeclaredFields();  
      for(Field field : fields){  
          String fName = field.getName();  
          //读取自定义注解  
          MyValue myValue = field.getDeclaredAnnotation(MyValue.class);  
          if(myValue != null){  
              //通过反射调用set方法注入值  
              String mName = "set" + fName.substring(0,1).toUpperCase() + fName.substring(1);  
              Method set = computerClass.getDeclaredMethod(mName, field.getType());  
              set.invoke(computer,myValue.value());  
          }  
          MyComponent myComponent = field.getDeclaredAnnotation(MyComponent.class);  
          if(myComponent != null){  
              //通过反射调用set方法注入对象  
              String mName = "set" + fName.substring(0,1).toUpperCase() + fName.substring(1);  
              Method set = computerClass.getDeclaredMethod(mName, field.getType());  
              //通过配置的类型创建对象  
              Object obj = myComponent.value().newInstance();  
              set.invoke(computer,obj);  
          }  
      }  
      return (Computer) computer;  
  }  
}

public class TestFactory {  
​  
  public static void main(String[] args) {  
      ComputerFactory factory = new ComputerFactory();  
      try {  
          Computer computer = factory.createComputer(Computer.class);  
          computer.start();  
      } catch (Exception e) {  
          e.printStackTrace();  
      }  
  }  
}