# Spring的核心

1. 依赖注入（DI，dependency injection）
2. 面向切面编程（AOP,aspect-oriented programming）

# Spring之旅

## 1、创建Spring的主要目的

替代更加重量级的企业级Java技术，尤其是EJB，简化Java开发。

## 2、Spring采用4种关键策略来简化Java开发：

（1）基于POJO的轻量级和最小侵入性编程。

（2）通过依赖注入和面向接口实现松耦合。

（3）基于切面和惯例进行声明式编程。

（4）通过切面和模板减少样板式代码。

## 3、DI松耦合的实现

首先看看下面的例子（RescueDamselQuest.java和DamselRescuingKnight.java）

package com.springonaction.Knights;  
  
public class DamselRescuingKnight {  
 private RescueDamselQuest quest;  
  
 public DamselRescuingKnight(){  
 **this.quest = new RescueDamselQuest();**//与RescueDamselQuest紧耦合  
 }  
  
 public void embarkOnQuest(){  
 quest.embark();  
 }  
}

在这个类中，DamselRescuingKnight在构造是创建了RescueDamselQuest的对象，使得两者紧密地耦合在一起。

而构造器注入则将RescueDamselQuest对象作为**构造器参数**传入，实现了松耦合。（BraveKnight.java和SlayDragonQuest.java）

package com.springonaction.Knights;  
  
public class BraveKnight implements Knight {  
 private Quest quest;  
  
 **public BraveKnight(Quest quest){  
 this.quest = quest;  
 }**  
  
 @Override  
 public void embarkOnQuest() {  
 quest.embark();  
 }  
}

这里传入的类型是Quest,它是一个接口，因此可以传入任何实现了该接口的类的对象（比如原来的RescueDamselQuest）.

## 4、Spring装配的实现

*SlayDragonQuest.java和knights.xml*

将下面的Quest注入到BraveKnight中去。

package com.springonaction.Knights;  
  
import java.io.PrintStream;  
  
public class SlayDragonQuest implements Quest {  
 private PrintStream stream;  
  
 public SlayDragonQuest(PrintStream stream){  
 this.stream = stream;  
 }  
  
 @Override  
 public void embark() {  
 stream.println("Embark on quest to slay the dragon!");  
 }  
}

1. ***通过XML装配将PrintStream注入到SlayDragonQuest中，将SlayDragonQuest注入到BraveKnight中去。***

<bean id="BraveKnight" class="com.springonaction.Knights.BraveKnight">  
 <constructor-arg **ref**="quest"/>  
</bean>  
  
<bean id="quest" class="com.springonaction.Knights.SlayDragonQuest">  
 <constructor-arg **value**="#{T(System).out}"/>  
</bean>

***（2）通过基于Java的配置进行依赖注入***（KnightConfig.java）

@Configuration  
public class KnightConfig {  
 @Bean  
 public Knight knight(){  
 return new BraveKnight(quest());  
 }  
  
 @Bean  
 public Quest quest(){  
 return new SlayDragonQuest(System.*out*);  
 }  
}

基于Java配置进行依赖注入需要编写一个配置类，类中通过声明获取对象的方法并将其声明为bean.

## 5、工作过程

（1）***基于XML装配的运行函数***（KnightMain.java）

package com.springonaction.Knights;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class KnightMain {  
 public static void main(String[] args) {  
 ClassPathXmlApplicationContext context = new **ClassPathXmlApplicationContext**("knights.xml");//加载Spring上下文  
 // BraveKnight knight = **context.getBean(BraveKnight.class)**;//获取knight bean  
 */\*\*  
 \* 或者通过以下方式获取bean  
 \*/* BraveKnight knight = **(BraveKnight) context.getBean("BraveKnight")**;  
 knight.embarkOnQuest();//使用knight  
 context.close();  
 }  
}

1. ***基于Java配置的运行函数***（AnnotationConfig\_Main.java）

package com.springonaction.Knights;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class AnnotationConfig\_Main {

public static void main(String[] args) {

ApplicationContext context = new **AnnotationConfigApplicationContext**(**KnightConfig.class**);

BraveKnight knight = **context.getBean(BraveKnight.class);**

knight.embarkOnQuest();

}

}