Spring完整揭秘（一）：IoC

原创

智慧zhuhuix 于 2020-07-29 11:07:33 发布 159 收藏 1

分类专栏： java spring 文章标签： java spring

java同时被 2 个专栏收录

183 篇文章10 订阅

订阅专栏

spring

45 篇文章7 订阅

订阅专栏

文章目录

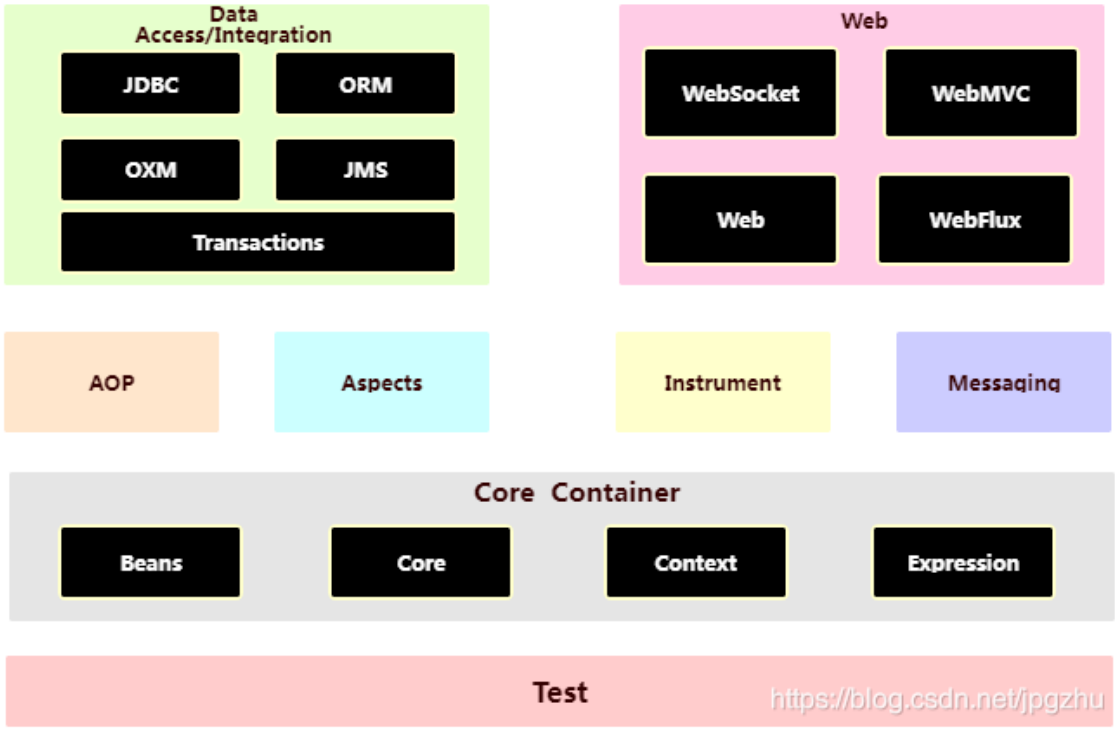
Spring的起源

在早期的J2EE平台开发实践过程中，过分强调分布式环境的使用，比如EJB，开发和部署复杂、测试困难，且代价昂贵，在这种历史背景情况下，倡导轻量级应用解决方案的Spring应运而生。

Spring框架

Spring倡导一切从实际出发，以敏捷、实用的态度来选择适合当前开发场景的解决方案。Spring框架的本质就是提供各种服务组件，以帮助我们简化基于POJO的Java企业级应用程序开发。

- Spring框架总体结构



组成整个Spring框架的各种服务实现被划分到了多个相互独立却又相互依赖的模块当中，它们组成了Spring框架的核心骨架。

Spring的IOC容器

- 整个Spring框架构建在Core核心模块之上，它是整个框架的基础。在该模块中，Spring为我们提供了一个IoC容器（IoC Container）实现，用于帮助我们以依赖注入的方式管理对象之间的依赖关系。

IoC的基本概念

> IoC的全称为Inversion of Control，中文通常翻译为“控制反转”。用大白话来讲的意思就是“你不用找我们，我们会找你的”。

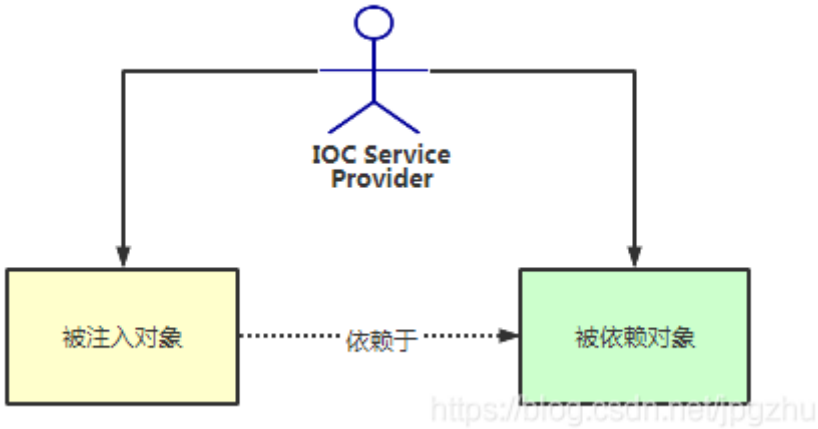
- 我们进食午餐时需要依赖于食物（Food），传统方式的做法就是主动new一个Food对象，然后完成进餐过程。

```
/**
 * IOC的理解--传统方式
 *
 * @author zhuhuix
 * @date 2020-07-29
 */
public class Person {
    // 食物
    private Food food;
    // 进食午餐
    public void haveLunch(){
        // 主动创建食物对象
        this.food= new Food("面食","馄饨面",1);
        // 进餐过程
        // ...
        System.out.println("进食午餐：“+this.food.toString());
    }
    // 食物类
    class Food {
```

```
        // 类型
        private String foodType;
        // 名称
        private String foodName;
        // 数量
        private int foodNum;

        public Food(){}

        public Food(String foodType, String foodName, int foodNum) {
            this.foodType = foodType;
            this.foodName = foodName;
            this.foodNum = foodNum;
        }

        public String getFoodType() {
            return foodType;
        }

        public void setFoodType(String foodType) {
            this.foodType = foodType;
        }

        public String getFoodName() {
            return foodName;
        }

        public void setFoodName(String foodName) {
            this.foodName = foodName;
        }

        public int getFoodNum() {
            return foodNum;
        }

        public void setFoodNum(int foodNum) {
            this.foodNum = foodNum;
        }

        @Override
        public String toString() {
            return "Food{" +
                    "foodType='" + foodType + '\'' +
                    ", foodName='" + foodName + '\'' +
                    ", foodNum=" + foodNum +
                    '}';
        }
    }
}

12345678910111213141516171819202122232425262728293031323334353637383940414243444
5464748495051525354555657585960616263646566676869
```

- IoC的场景
    - 二者之间通过IoC ServiceProvider来打交道，所有的被注入对象和依赖对象现在由IoC Service Provider统一管理。被注入对象需要什么，通行IoC Service Provider，就会把相应的

被依赖对象注入到被注入对象中，从而达到IoC Service Provider为被注入对象服务的目的。



IoC的注入方式

1. 构造方法注入

> 这种注入方式的优点就是，对象在构造完成之后，即已进入就绪状态，可以马上使用。缺点就是，当依赖对象比较多的时候，构造方法的参数列表会比较长。

```
/**
 * IOC的注入方式--构造方法注入
 *
 * @author zhuhuix
 * @date 2020-07-29
 */
public class PersonIoC1 {
    private Food food;

    // 构造方法注入
    public PersonIoC1(Food food) {
        this.food = food;
    }

    // 进食午餐
    public void haveLunch(){
        // 进餐过程
        // ...
        System.out.println("进食午餐："+this.food.toString());
    }
}

12345678910111213141516171819202122
```

1. setter 方法注入

> setter方法注入在描述性上要比构造方法注入好一些。另外，setter方法可以被继承，允许设置默认值，而且有良好的IDE支持。缺点当然就是对象无法在构造完成后马上进入就绪状态。

```
/**
 * IOC的注入方式--setter方法注入
 *
 * @author zhuhuix
 * @date 2020-07-29
 */
public class PersonIoC2 {
```

```
    private Food food;

    public PersonIoC2() { }

    // setter方法注入
    public void setFood(Food food) {
        this.food = food;
    }

    // 进食午餐
    public void haveLunch(){
        // 进餐过程
        // ...
        System.out.println("进食午餐：" +this.food.toString());
    }
}
```

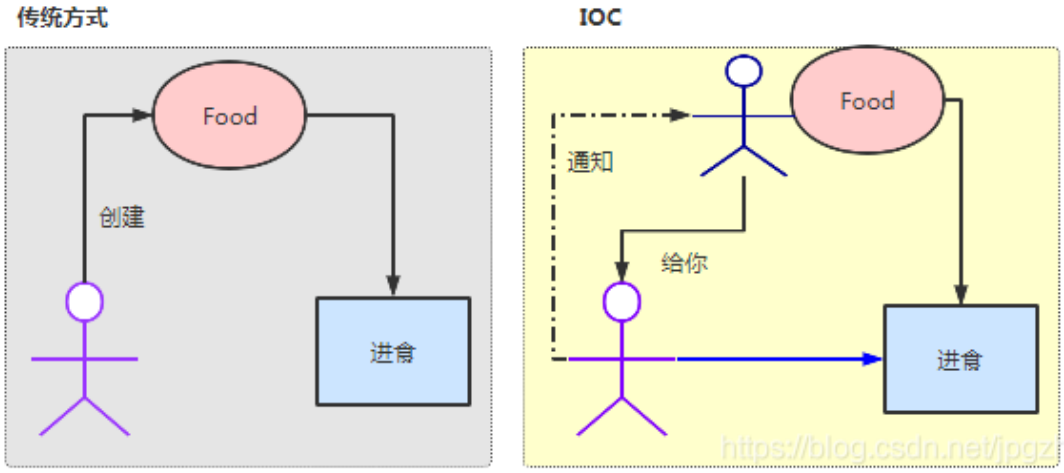
1. 接口注入

> 需要被依赖的对象实现不必要的接口，带有侵入性。一般都不推荐这种方式

```
/**
 * IOC的注入方式--接口注入
 *
 * @author zhuhuix
 * @date 2020-07-29
 */
public class PersonIoC3 implements injectFood{
    private Food food;

    public PersonIoC3() { }

    // 实现接口进行注入
    @Override
    public void injectFood(Food food) {
        this.food = food;
    }

    // 进食午餐
    public void haveLunch(){
        // 进餐过程
        // ...
        System.out.println("进食午餐：" +this.food.toString());
    }


}

//注入接口
interface injectFood{
     void injectFood(Food food);
}
```

我对IOC的理解

- 我们要完成某个业务逻辑时需要用到其他的对象来协作完成，在没有使用Spring的时候，均要使用像new object() 这样的语法来将合作对象创建出来，这个合作对象是由自己主动创建出来的，**在本例中，由Person对象主动创建了Food对象。** 这样两个对象就紧密地耦合在了一起。



- 而使用了Spring之后，创建协作对象的工作是由Spring来完成，Spring创建好对象，存储到一个容器里面，当需要使用协作对象时，可以从Spring存放对象的容器里面取出并直接使用，至于Spring是如何创建那个对象，以及什么时候创建好对象的，完全不需要关心这些细节问题。**在本例中，Person对象向Spring容器请求Food对象。** 这样两个对象通过Spirng Ioc容器完成了解耦。
- 从图上也可以看出来，IoC实际描述的是创建对象的控制权进行了转移，以前创建对象的主动权和创建时机是由自己把控的，而现在这种权力转移到第三方。
- 用一句话来总结：IoC是一种可以帮助我们解耦各业务对象间依赖关系的对象绑定方式