

用大白话聊聊JavaSE -- 如何理解Java Bean（一）

首先，在开始本章之前，先说一个总的概念：所谓的Java Bean，就是一个java类，编译后成为了一个后缀名是 .class的文件。这就是Java Bean，很多初学者，包括当年的我自己，总是被这些专有名词搞的晕头转向。去公司面试，对方一口一个controller，一口一个service，dao，搞得我很紧张。其实都是很简单的东西，只是自己当时不知道罢了，接触之后才发现，不就是Java类吗？

1. 什么是 Java Bean？

很多培训机构在讲java基础的时候，基本都会写这样的代码：

```
package com.springmvc.bean;

public class Person {
    private String name; // 姓名
    private Integer age; // 年龄
    private String gender; // 性别
    private String hobby; // 爱好

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String getHobby() {
        return hobby;
    }

    public void setHobby(String hobby) {
        this.hobby = hobby;
    }
}
```

毋庸置疑，这就是一个java bean。

在很多教材上，我们都被告知，现实中有人，分为男人和女人，这是一个类。然后，我们用java的面向对象将人抽象成一个Java类——Person类。

这固然不错，然而，也就只是如此而已了。没有什么其他的東西，弄了半天，我们初学者学java，都在那写人类，车子类，房子类，等等。学继承的时候，总是写这样的代码，老师也举类似这样的例子。

```
package com.springmvc.bean;

/**
 * 水果类
 *
 */
public class Fruit {

    private String name; //水果的名称
    private Float price; //价格

    public Fruit(String name, Float price) {
        super();
        this.name = name;
        this.price = price;
    }
}
```

```

        // getters 和 setters 省略
    }

package com.springmvc.bean;
/**
 * 苹果类
 */
public class Apple extends Fruit {

    private String color;//颜色

    public Apple(String name, Float price,String color) {
        super(name, price);
        this.color = color;
    }

    // getters 和 setters 省略

}

```

没错，这些都是Java Bean，对于初学者，这些例子的确比较好理解。但是，这样就有个弊端，它很容易让人产生一种迷茫，就是说，这些例子我固然听得懂，但是，我不知道学这些有什么用？我看过很多培训机构的视频，里面多半都是这么讲的，这导致我整个JavaSE都学完了，还是不知道自己干嘛？

去企业面试，人家一口一个专业词汇，弄得我好不尴尬。

想想也知道，去企业里不可能让我写这种代码的。归根到底，还是这些例子不实用。再举个例子，学校学数据库的时候，肯定会举一个被举了N多次的例子，就是有一张学生表，学生有姓名，性别，课程，分数。然后来一个

```
select * from t_student;
```

恩，我的确听懂了，可是，然后呢。。。？

然后？我们只负责领你入门，你学费也交了，你赶紧去企业里面学吧！

好吧，说多了都是泪。我就见过好多简历被包装过的培训生，结果一进来，连JSON都不知道是啥，ajax也不清楚怎么用。让写个多表查询或者连表查询吧，琢磨个半天还是写不出来。

我个人主张实用主义，所以，本节就 JavaBean 的基本概念来稍微延伸一下，看看我学这些东西，什么String，Integer啊，数据库啊，到底可以用在哪里？

2. 需求分析

举个例子咯，就拿简书来说好了，比如一个文章列表：

每一条数据，说实话，就是一个对象，对象有各种属性，从图片中，我们起码就可以获得这些信息：

一篇文章具备的属性：

- 1.发布时间
- 2.文章标题
- 3.点击量
- 4.评论数
- 5.点赞数
- 6.显示图片

当然，还需要有文章内容（因为是举例子，我就先不写了），这些属性，不就是Java类里面修饰符为private的私有属性吗？对应数据库表里面，这些就是所谓的字段。

你可能会说，对啊，可是我为什么要专门去写一个JavaBean来承载这些数据呢？我用一个HashMap不是也一样吗？

是的，从某种程度上，的确可以，可是，用JavaBean的话更加清晰明了。

你可能写了很久的代码，都不清楚到底为什么要定义JavaBean，尤其是做JavaWeb开发的时候，Java的作用其实就是两句话：

1.1 控制数据的流向，将前台传过来的数据包起来，然后一个一个地插入数据库永久保存。

1.2 从数据库中用jdbc取出数据，然后包起来，最终传递到前台页面进行公开展览

JavaBean就是一个中转载体。

不就是这么回事吗？作为码农的我们，每天不就是在做这些事情吗？这就是所谓的增删改查。

最多就是，我们很根据业务需求，通过写Java代码，来进行一些逻辑的控制，说穿了就是：

数据不是你想增，想增就能增。

数据不是你想删，想删就能删。

数据不是你想改，想改就能改。

数据不是你想查，想查就能查。

查个权限，做个判断，放手你的爱。

不就是这么回事吗？

什么JavaBean，dao，事务管理，切面，这些我认为反而是次要的，都是一些专业术语的堆积，作为一名码农，我认为首先得知道，自己每天到底在干嘛，然后根据需要再去学对应的知识或者理论来给自己充电。

不要连本职工作都还没做好，就一会跟风去学大数据，一会又去学bootstrap，node.js，我不是说多学点技术不好。而是，我认为还是要分一个轻重缓急，比如你现在待的公司，根本用不到大数据，你花了那么多精力去学了又怎样呢？

再说个实际的，如果你的网站访问量每个月1千都达不到，你高并发，分布式学得再好，又能怎么样呢？

再说了，如果一个网站真的做大了，到最后其实都是拼的服务器，而不是技术。

当你确实需要用到大数据了，再去学，也行啊。到那个时候，公司肯定会想办法的，实在不行，向外招人也是非常必要的。

可是，还没有达到那个层次之前，说句大白话，你的工资只跟你对当前岗位的适应程度挂钩。你不可能说，我已经学了大数据了，就跑到老板的办公室要求涨工资吧。也不现实啊，你觉得呢？

好了，回到正题，刚才我们从图片中获取了这么多字段：

- 1.发布时间
- 2.文章标题
- 3.点击量
- 4.评论数
- 5.点赞数
- 6.显示图片

其实可以分为必要字段和辅助字段。所谓的必要字段，就是作为一篇文章，必须要有的属性。

像发布时间，文章标题，还有文章内容，这些都是必要字段。

辅助字段就是，为了方便对这篇文章进行控制，需要设置的字段。比如，

这篇文章是否能够被看到？

这篇文章是否已经删除？

这篇文章是否应该被排在前面？

对应的，我们可以设置这些字段，

1. 是否发布 : 0 未发布 1 发布
2. 是否删除 : 0 未删除 1 已删除
3. 点击量，评论数，点赞数 这些字段可用于生成一个热度，如果热度很高，那么就on应该被排在前面，被更多的人看到。（这些字段看做主要字段也可以）

顺便提一句，实际开发的时候，基本用不到delete语句的，删除都是用的逻辑删除，就是说，将某一个字段（is_delete）从0变为1，表示已删除。这样的好处就是，万一这条数据以后想恢复的话，直接改变那个字段的值就OK了。我查询数据的时候，在where条件里面加一个 is_delete = 1 不就好了，那么删除的数据就不会被查出来了。

为什么要这么做呢？

比如，游戏中物品丢失了，怎么找回呢？如果直接delete掉了，那么就真的GG了。只要数据还在，一切都好说，我大不了不让他显示就是了，万一要找回就能直接找回了。嗯，一般都是这么做的。

3. JavaBean设计

接下来，我们来设计这个JavaBean。打开Eclipse，我以之前那个springmvc的案例来举例，现在新建一个包，就叫做bean，里面是专门用来存放这些JavaBean的，然后新建一个类——Article（文章类）

New Java Class

Java Class

Create a new Java class.

Source folder: Browse...

Package: Browse...

☐ Enclosing type: Browse...

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Finish Cancel

```
public class Article {  
  
}
```

开始设计字段。

首先，我们知道，这些数据最终是要存储到数据库表的，那么就肯定需要有一个id，作为它的主键，我们就用String吧，然后用uuid主键生成策略。

```
private String id; //主键 UUID
```

主键就是这一条数据的身份证，是唯一的，不允许重复。

必要字段，或者叫主要字段

```
private Date publishTime; //发布时间  
private String title; //文章标题  
private String pictureLine; //图片链接地址
```

辅助字段：

```
private Integer hitNum; //点击量  
private Integer commentNum; //评论数  
private Integer loveNum; //点赞数  
private Integer isPublished; //是否发布 0 未发布 1 发布  
private Integer isDelete; //是否删除 0 未删除 1 已删除
```

最后，一般来说，我们还需要记录一些类似于日志的信息，比如这篇文章是什么时候创建的，这个创建的时间是不是可以记录下来呢？另外，是谁创建的，我们是不是可以记录下用户的id呢？还有，文章是可以被多次修改的，那么，最后一次修改的时间是不是需要被记录下来呢？

先就说这么多吧，JavaBean的设计需要根据具体的业务需求来定，我这里只是举一个例子而已。

```
private String userId; //用户ID  
private Date createTime; //创建时间  
private Date updateTime; //最后更新时间
```

这样一来就差不多了，最后，利用Eclipse的快捷键 alt + s，点击Generate Getters and Setters...，选择全部字段，再点击OK。

最后，鼠标右键，Source —— Format，格式化一下，一个JavaBean就做好了。

当你写完了JavaBean，差不多也就相当于设计好了数据库表。我们在写JavaBean的时候，属性一般都用驼峰法来命名，而数据库表有点不一样，就是在驼峰的地方，大写字母要改为小写字母，然后加上一个下划线。

比如

userId，对应的数据库表字段名称就是 user_id，一般都是这么命名的。

在实际开发过程中，差不多就是这样：

前台页面（可能是在一张表单Form里面填写数据） —— 保存 —— ajax传递到Controller层 —— 与JavaBean做映射，将这些数据保存到JavaBean中 —— 进入Service层，这里对数据进行一些逻辑操作和判断。有歌为证：

数据不是你想增，想增就能增。

数据不是你想删，想删就能删。

数据不是你想改，想改就能改。

数据不是你想查，想查就能查。

查个权限，做个判断，放手你的爱。

—— 最后，进入Dao层，直接访问数据库，进行各种操作。

好了，这一节对JavaBean做了一些简短的，额，简短的介绍。下一节我会稍微深入一些。总之，初学Java的时候，千万不要被这些专业术语给吓到，它真的没你想的那么难。

如果你总是想着复杂，那么你永远看不到简单。直到有一天你发现，所有的麻烦都只在你的心中。