

在 Java 中有多种方式可以创建对象，总结起来主要有下面的 4 种方式：

正常创建。通过 new 操作符

反射创建。调用 Class 或 java.lang.reflect.Constructor 的 newInstance()方法

克隆创建。调用现有对象的 clone()方法

反序列化。调用 java.io.ObjectInputStream 的 getObject()方法反序列化

Java 对象的创建方式是其语法明确规定，用户不可能从外部改变的。本文仍然要使用上面的方式来创建对象，所以本文只能说是构建对象，而非创建对象也。

假设有这样一个场景，现在要构建一个大型的对象，这个对象包含许多个参数的对象，有些参数有些是必填的，有些则是选填的。那么如何构建优雅、安全地构建这个对象呢？

单一构造函数

通常，我们第一反应能想到的就是单一构造函数方式。直接 new 的方式构建，通过构造函数来传递参数，见下面的代码：

```
/**
 * 单一构造函数
 */
public class Person {

    // 姓名（必填）
    private String name;

    // 年龄（必填）
    private int age;

    // 身高（选填）
    private int height;

    // 毕业学校（选填）
    private String school;

    // 爱好（选填）
    private String hobby;

    public Person(String name, int age, int height, String school, String hobby) {
        this.name = name;
        this.age = age;
        this.height = height;
        this.school = school;
        this.hobby = hobby;
    }
}
```

上面的构建方式有下面的缺点：

有些参数是可以选填的（如 `height`，`school`），在构建 `Person` 的时候必须要传入可能并不需要的参数。

现在上面才 5 个参数，构造函数就已经非常长了。如果是 20 个参数，构造函数都可以直接上天了！

构建的这样的对象非常容易出错。客户端必须要对照 `Javadoc` 或者参数名来讲实参传入对应的位置。如果参数都是 `String` 类型的，一旦传错参数，编译是不会报错的，但是运行结果却是错误的。

多构造函数

对于第 1 个问题，我们可以通过构造函数重载来解决。见下面的代码：

```
/**
 * 多构造函数
 */
public class Person {

    // 姓名（必填）
    private String name;

    // 年龄（必填）
    private int age;

    // 身高（选填）
    private int height;

    // 毕业学校（选填）
    private String school;

    // 爱好（选填）
    private String hobby;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public Person(String name, int age, int height) {
        this.name = name;
        this.age = age;
        this.height = height;
    }

    public Person(String name, int age, int height, String school) {
```

```

        this.name = name;
        this.age = age;
        this.height = height;
        this.school = school;
    }

    public Person(String name, int age, String hobby, String school) {
        this.name = name;
        this.age = age;
        this.hobby = hobby;
        this.school = school;
    }
}

```

上面的方式确实能在一定程度上降低构造函数的长度，但是却有下面的缺陷：

导致类过长。这种方式会使得 **Person** 类的构造函数成阶乘级增长。按理来说，应该要写的构造函数数是可选成员变量的组合数（实际并没有这么多，原因见第 2 点）。如果让我调用这样的类，绝对会在心里默念 **xx!!**

有些参数组合无法重构。因为 **Java** 中重载是有限制的，相同方法签名的方法不能构成重载，编译时无法通过。譬如包含（**name, age, school**）和（**name, age, hobby**）的构造函数是不能重载的，因为 **school** 和 **hobby** 同为 **String** 类型。**Java** 只认变量的类型，管你变量是什么含义呢。（看脸的社会唉）

#### JavaBean 方式

上面的方法不行，莫急！还有法宝——**JavaBean**。一个对象的构建通过多个方法来完成。直接见下面的代码：

```

public class Person {

    // 姓名（必填）
    private String name;

    // 年龄（必填）
    private int age;

    // 身高（选填）
    private int height;

    // 毕业学校（选填）
    private String school;

    // 爱好（选填）
    private String hobby;

    public Person(String name, int age) {

```

```

        this.name = name;
        this.age = age;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public void setSchool(String school) {
        this.school = school;
    }

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

```

客户端使用这个对象的代码如下：

```

public class Client {

    public static void main(String[] args) {
        Person person = new Person("james", 12);
        person.setHeight(170);
        person.setHobby("reading");
        person.setSchool("xxx university");
    }
}

```

这样看起来完美的解决了 **Person** 对象构建的问题，使用起来非常优雅便捷。确实，在单一线程的环境中这确实是一个非常好的构建对象的方法，但是如果是在多线程环境中仍有其致命缺陷。在多线程环境中，这个对象不能安全地被构建，因为它不是不可变对象。一旦 **Person** 对象被构建，我们随时可通过 **setXXX()** 方法改变对象的内部状态。假设有一个线程正在执行与 **Person** 对象相关的业务方法，另外一个线程改变了其内部状态，这样得到莫名其妙的结果。由于线程运行的无规律性，使得这问题有可能不能重现，这个时候真的就只能哭了。（程序员真苦逼。。。）

### Builder 方式

为了完美地解决这个问题，下面引出本文中的主角（等等等等！）。我们使用构建器（**Builder**）来优雅、安全地构建 **Person** 对象。废话不说，直接代码：

```

/**
 * 待构建的对象。该对象的特点：
 * <ol>
 * <li>需要用户手动的传入多个参数，并且有多个参数是可选的、顺序随意</li>
 * <li>该对象是不可变的（所谓不可变，就是指对象一旦创建完成，其内部状态不可变，

```

更通俗的说是其成员变量不可改变)。

- \* 不可变对象本质上是线程安全的。</li>

- \* <li>对象所属的类不是为了继承而设计的。</li>

- \* </ol>

\* 满足上面特点的对象构建可是使用下面的 **Build** 方式构建。这样构建对象有下面的好处:

- \* <ol>

- \* <li>不需要写多个构造函数，使得对象的创建更加便捷</li>

- \* <li>创建对象的过程是线程安全的</li>

- \* </ol>

- \* @author xialei

- \* @date 2015-5-2

- \*/

```
public class Person {
```

```
    // 姓名（必填），final 修饰 name 一旦被初始化就不能再改变，保证了对象的不可变性。
```

```
    private final String name;
```

```
    // 年龄（必填）
```

```
    private final int age;
```

```
    // 身高（选填）
```

```
    private final int height;
```

```
    // 毕业学校（选填）
```

```
    private final String school;
```

```
    // 爱好（选填）
```

```
    private final String hobby;
```

```
    /**
```

```
     * 这个私有构造函数的作用：
```

```
     * <ol>
```

```
     * <li>成员变量的初始化。final 类型的变量必须进行初始化，否则无法编译成功</li>
```

```
     * <li>私有构造函数能够保证该对象无法从外部创建，并且 Person 类无法被继承</li>
```

```
     * </ol>
```

```
     */
```

```
    private Person(String name, int age, int height, String school, String hobby) {
```

```
        this.name = name;
```

```
        this.age = age;
```

```
        this.height = height;
```

```
        this.school = school;
```

```
        this.hobby = hobby;
```

```

}

/**
 * 要执行的动作
 */
public void doSomething() {
    // TODO do what you want!!
}

/**
 * 构建器。为什么 Builder 是内部静态类?
 * <ol>
 * <li>必须是 Person 的内部类。否则，由于 Person 的构造函数私有，不能通过 new 的
方式创建 Person 对象</li>
 * <li>必须是静态类。由于 Person 对象无法从外部创建，如果不是静态类，则外部无
法引用 Builder 对象。</li>
 * </ol>
 * <b>注意</b>: Builder 内部成员变量要与 Person 的成员变量保持一致。
 * @author xialei
 *
 */
public static class Builder {
    // 姓名（必填）。注意：这里不能是 final 的
    private String name;

    // 年龄（必填）
    private int age;

    // 身高（选填）
    private int height;

    // 毕业学校（选填）
    private String school;

    // 爱好（选填）
    private String hobby;

    public Builder(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public Builder setHeight(int height) {
        this.height = height;
    }
}

```

```

        return this;
    }

    public Builder setSchool(String school) {
        this.school = school;
        return this;
    }

    public Builder setHobby(String hobby) {
        this.hobby = hobby;
        return this;
    }

    /**
     * 构建对象
     * @return 返回待构建的对象本身
     */
    public Person build() {
        return new Person(name, age, height, school, hobby);
    }
}

```

客户端构建对象的方式见下面的代码：

```

/**
 * 使用 Person 对象的客户端
 * @author xialei
 * @date 2015-5-2
 */
public class Client {

    public static void main(String[] args) {
        /*
         * 通过链式调用的方式创建 Person 对象，非常优雅！
         */
        Person person = new Person.Builder("james", 12)
            .setHeight(170)
            .setHobby("reading")
            .build();

        person.doSomething();
    }
}

```

如果不想看代码，可看下面对于上面代码的总结：

通过 `private Person(..)` 使得 `Person` 类不可被继承

通过将 `Person` 类的成员变量设置为 `final` 类型，使得其不可变

通过 `Person` 内部的 `static Builder` 类来构建 `Person` 对象

通过将 `Builder` 类内部的 `setXXX()` 方法返回 `Builder` 类型本身，实现链式调用构建 `Person` 对象

总结

至此，我们就相对完美地解决这一类型的对象创建问题！下面来总结一下本文的重点。待创建的对象特点：

需要用户手动地传入多个参数，并且有多个参数是可选的、顺序任意

对象不可变

对象所属的类不是为了继承而设计的。即类不能被继承

依次使用的对象构建方法：

单一构造函数

多构造函数

JavaBean 方式

Builder 方式

最终，通过比较得出 `Builder` 方法最为合适的解决。