建造者模式

基本概念

在绝大部分场景中，我们所设计的类在创建实例时仅需少量的参数即可。但是，仍然存在需要较多参数、且构造函数内部校验较为复杂的对象

并且由于Java支持函数重载，所以丢弃了"默认参数"这一语言特性，如果该创建复杂的对象包含默认参数实现的话，情况会变得更加复杂

```java
public class ConnectionPool {

    // 默认值的实现，通常使用环境变量注入
    private int maxTotal = 150;                 // 连接池最大数量
    private int maxIdle = 50;                   // 连接池中最大闲置连接数量
    private int minIdle = 10;                   // 连接池中最小闲置连接数量

    public ConnectionPool(Integer maxTotal, Integer maxIdle, Integer minIdle) {

        /* 若maxTotal不为null，校验其是否大于0 */
        if (maxTotal != null) {
            if (maxTotal <= 0) {
                throw new IllegalArgumentException("");
            }
            this.maxTotal = maxTotal;
        }

        /* 若maxIdle不为null，校验其是否大于0，是否大于maxTotal */

        /* 若minIdle不为null，校验其是否大于0，是否大于maxIdle */
    }

    public static void main(String[] args) {
        ConnectionPool pool = new ConnectionPool(200, null, 25);
    }
}
```

以创建连接池为例，其中最大连接数量、最大闲置连接数以及最小闲置连接数均为可选选项，若在构造类对象时传入null，则使用默认值

对参数的判断与校验均在构造函数中进行，如果参数数量稍微多一些、校验规则稍微复杂一点，构造函数就会变得很长，且逻辑较为复杂

此外，由于Java本身并不支持具名参数的传递，所以当创建一个类时，需要非常小心的检查参数的顺序以及个数

解决构造函数过长的一个方法就是使用set()方法，构造函数不做任何处理，所有可选参数均使用set方法写入

```java
public class ConnectionPool {

    // 默认值的实现，通常使用环境变量注入
    private int maxTotal = 150;                 // 连接池最大数量
    private int maxIdle = 50;                   // 连接池中最大闲置连接数量
    private int minIdle = 10;                   // 连接池中最小闲置连接数量

    public ConnectionPool() {}

    public void setMaxTotal(int maxTotal) {
        if (maxTotal <= 0) {
            throw new IllegalArgumentException();
        }
        this.maxTotal = maxTotal;
    }

    public void setMaxIdle(int maxIdle) {/*...*/}

    public void setMinIdle(int minIdle) {/*...*/}
}
```

但是set()方法又引入了新的问题

连接池相关属性可在运行时被修改，通常来讲，连接池属性一旦确定，应无法修改

无法进行具有依赖关系的参数校验，例如minIdle应小于maxIdle，由于无法保证用户调用set()方法的顺序，也就无法进行校验

为了解决以上方案存在的种种问题，包括语言本身的局限性(函数不支持默认值，不支持具名参数传递等)，以及业务逻辑的复杂性以及依赖性，才会有建造者模式的诞生

使用建造者模式

与大多数设计模式类似，解决方法就是再加一层抽象: 由Builder类来处理这些复杂的参数设置，并在最终的build()方法中一次性校验所有需要校验的内容

```java
public class ConnectionPool {
    private int maxTotal;
    private int maxIdle;
    private int minIdle;

    private ConnectionPool(Builder builder) {
        this.maxTotal = builder.maxTotal;
        this.maxIdle = builder.maxIdle;
        this.minIdle = builder.minIdle;
    }

    public static class Builder {
        private int maxTotal = 150;
        private int maxIdle = 50;
        private int minIdle = 10;

        public ConnectionPool build() {
            /* 在该方法内，需要对所有的成员变量再次进行校验，且进行依赖性校验 */
            return new ConnectionPool(this);
        }

        public Builder setMaxTotal(int maxTotal) {
            /* 仅校验是否为正整数 */
            this.maxTotal = maxTotal;
            return this;
        }

        public Builder setMaxIdle(int maxIdle) {/*...*/}

        public Builder setMinIdle(int minIdle) {/*...*/}
    }

    public static void main(String[] args) {
        ConnectionPool connectionPool = new ConnectionPool.Builder()
                .setMaxIdle(500)
                .setMaxTotal(1500)
                .build();
    }
}
```

在ConnectionPool内部，使用了一个内部静态类Builder协助用户完成对象的构建，也可以使用外部类，但是内部静态类的使用要更为方便一些

建造者模式使得构建一个类对象分为了两步: 使用set()方法填充成员变量，以及最终对传入的参数进行汇总校验，并实际创建出所需要的对象

建造者模式解决了大量构造参数传递的问题，以及保证了内部成员变量符合设计者所定义的校验规则

但是使用建造者模式所设计的类避免不了代码冗长的问题，并且成员变量以及校验规则均会重复

Java特有的设计模式

产生建造者模式的原因可以归为以下几点原因

Java因为支持函数重载，所以移除了函数默认参数的语言特性

创建一个类本身较为复杂，其中存在默认参数值、用户传递参数值以及参数值依赖关系的校验

对于Python语言而言，由于存在默认参数以及*args和**kwargs等黑魔法，基本不需要建造者模式，同时实现建造者模式也比较困难

但是，在Python中，如Django REST framework 的 Serializers，Marshmallow中的Schema，多多少少存在着建造者模式的思想在其中

建造者模式和工厂模式的区别

工厂模式: 提供多个不同类对象的创建

建造者模式: 提供一个复杂类对象的创建

在工厂模式的对象创建中，完全可以使用建造者模式，两者并不冲突且互补