流畅接口(Fluent Interface)

波波老师~研发总监/资深架构师

波波微课

spring2go.com

问题域

- 传统Builder模式在实际场景中应用不多
- 流畅接口Fluent interface实际场景中应用更多
  - 方法级联(Method Cascading/Chaining)
  - 事实上的Builder模式
- 不可变(immutability)对象需求
  - 不容易出Bug
  - 多线程安全

构造对象实例的传统方式

- 构造函数(Telesoping Constructor Pattern)
- Setter方法(JavaBeans Pattern)
- 工厂

显微镜构造函数模式

```java
// 显微镜构造函数模式
public class User {

    // region 私有成员

    public User(String firstName, String lastName, int age, String phone) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.phone = phone;
    }

    public User(String firstName, String lastName, String phone, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.phone = phone;
        this.address = address;
    }

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // region public getter

}
```

问题

- 罗列各种参数组合繁琐复杂
- 扩展字段，增加构造函数麻烦
- 参数顺序易搞错
  - 程序员思考负担
  - 增加bug概率

JavaBean模式

```java
// JavaBeans模式
public class User {

    //region 私有成员

    public User() {}

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

JavaBean模式使用

```java
package io.spring2go.corespring.javabean;

public class JavaBeanMain {

    // JavaBean构造对象
    public static void main(String[] args) {
        User user = new User();
        user.setFirstName("william");
        user.setLastName("yang");
        user.setAge(35);
        user.setPhone("18001756666");
        user.setAddress("shanghai pudong");
    }

}
```

问题

- 状态可变，一致性保证
- 多线程安全

流畅接口~User

```
// 流畅接口(Fluent Interface)
public class User {
    // 所有字段final
    private final String firstName; // 必须
    private final String lastName; // 必须
    private final int age; // 可选
    private final String phone; // 可选
    private final String address; // 可选

    private User(UserBuilder builder) {
        this.firstName = builder.firstName;
        this.lastName = builder.lastName;
        this.age = builder.age;
        this.phone = builder.phone;
        this.address = builder.address;
    }

    // 全部是getter，没有setter，保证不可变性immutability
    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public int getAge() {
        return age;
    }

    public String getPhone() {
        return phone;
    }

    public String getAddress() {
        return address;
    }

    @Override
    public String toString() {
        return "User: " + this.firstName + ", "
                + this.lastName + ", " + this.age + ", "
                + this.phone + ", " + this.address;
    }
```

流畅接口~UserBuilder

```
public static class UserBuilder {
    private final String firstName;
    private final String lastName;
    private int age;
    private String phone;
    private String address;

    public UserBuilder(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public UserBuilder age(int age) {
        this.age = age;
        return this;
    }

    public UserBuilder phone(String phone) {
        this.phone = phone;
        return this;
    }

    public UserBuilder address(String address) {
        this.address = address;
        return this;
    }

    // 返回最终构造的用户对象
    public User build() {
        User user = new User(this);
        validateUserObject(user);
        return user;
    }

    private void validateUserObject(User user) {
        // 基本的验证检查
        // 确保用户对象不违反系统假设
    }
}
}
```

流畅模式使用

```
public static void main(String[] args) {
    User user1 = new User.UserBuilder("william", "Yang")
            .age(35)
            .phone("123456")
            .address("Fake address 1234")
            .build();

    System.out.println(user1);

    User user2 = new User.UserBuilder("Jack", "Liu")
            .age(40)
            .phone("5655")
            //no address
            .build();

    System.out.println(user2);

    User user3 = new User.UserBuilder("Frank", "Han")
            //no age
            //no phone
            //no address
            .build();

    System.out.println(user3);
}
```

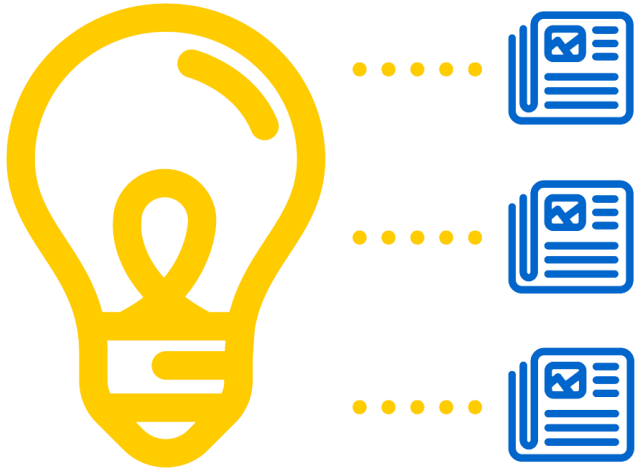
```
<terminated> ModernBuilderMain [Java Application] C:\Program
User: william, Yang, 35, 123456, Fake address 1234
User: Jack, Liu, 40, 5655, null
User: Frank, Han, 0, null, null
```

好处

- 参数组合更灵活
- 参数设置直观表意
- 一个构造器实例可以构造多个对象表示
- 构造出不可变对象(immutable objects)
- 使用代码简洁流畅

应用

- Core Java
  - java.lang.StringBuilder#append()
  - java.lang.StringBuffer#append()
  - java.nio.ByteBuffer#put()
- Spring Framework
  - EmbeddedDataBuilder
  - AuthenticationManagerBuilder
  - UriComponentsBuilder
  - BeanDefinitionBuilder
  - MockMvcWebClientBuilder



问题

- Fluent接口模式有啥不足？适用于什么样的场景？

参考

- Builder Design Pattern in Java
  - https://howtodoinjava.com/design-patterns/creational/builder-pattern-in-java/
- Builder Design Pattern
  - http://codepumpkin.com/builder-design-pattern/

代码

- https://github.com/spring2go/core-spring-patterns
- Builder模式目录下

波波微课

spring2go.com