

利用多态重构为带参方法

张逸

www.agiledon.com

我在阅读遗留代码时，经常发现存在这样一种情形。在一个类中存在两个方法，它们做了相似的工作，区别仅在于方法内部某些对象的类型。例如：

```
public class WorkSheet{
    private void fillHeader() {
        Header header = createHeader();
        for (String title:header.getTitles()) {
            fillCell(title);
        }
    }
    Private void fillBody() {
        CellGroup cellGroup = createCellGroup();
        for (Cell cell:cellGroup.getCells()) {
            fillCell(cell.getText());
        }
    }
}
```

方法 `fillHeader()` 和 `fillBody()` 的目的都是从对象中获得字符串数组，然后将其填充到单元格中。区别在于，获得字符串数组的对象并不相同。前者为 `Header` 对象，后者为 `CellGroup` 对象。我们可以为其提供一个抽象的接口，以实现代码的有效重用：

```
public interface TextDataSource {
    public String[] getTextArea();
}
```

然后，让 `Header` 和 `CellGroup` 类均实现该接口。由于 `CellGroup` 并没有直接定义返回字符串数组的方法，而是通过返回的 `Cell` 对象获得 `text` 值，因此，需要将这部分实现封装到 `getTextArea()` 方法中：

```
public class Header implements TextDataSource {
    @Override
    public String[] getTextArea() {
```

```
//这里的实现即为原有的 getTitles()实现
//可以保留原有的方法，并在本方法中指向该方法
//也可以利用 Rename Method 手法，直接更名该方法
}
}
public class CellGroup implements TextDataSource {
    @Override
    public String[] getTextArea() {
        List<String> textArea = new ArrayList<String>();
        for (Cell cell:this.getCells()) {
            textArea.add(cell.getText());
        }
        return textArea.toArray();
    }
}
```

现在，就可以重构原有的 WorkSheet 类了。

```
public class WorkSheet{
    private void fillSheet(TextDataSource dataSource) {
        for (String text:dataSource.getTextArea()) {
            fillCell(text);
        }
    }
}
```

具体需要填充什么内容，可以在调用 fillSheet()方法时，根据传入的参数对象来决定。经过重构之后，WorkSheet 类中的重复代码得到了移除，且具有了更好的扩展性。

这一重构手法与 Parameterize Method 要解决的坏味道相似，同样对相似方法提取了共同的参数，但实现的本质完全不同。它利用了多态的原理，通过对抽象方法体中的相似对象，抹去了不同类型对象之间的差异性，使得方法体中的相似结构能够被抽取出来。

我将这一重构手法命名为 Parameterize Method by Polymorphism。让我仿照 Martin Fowler 的风格，给出这一重构方式的作法 (Mechanics):

- 1) 新建一个接口，并使原有方法中的差异对象实现该接口。

- 2) 如果原有对象的方法与该接口定义的方法签名不同，则运用 `Rename Method`。
- 3) 编译。
- 4) 新建一个参数为新接口类型的方法，使它可以替换先前所有的重复方法。
- 5) 编译。
- 6) 将对旧方法的调用替换为对新函数的调用。
- 7) 编译，测试。
- 8) 对所有旧方法重复上述步骤，每次替换后，修改并测试

如果在调用新方法时，发现创建参数实参对象是一件麻烦事，可以考虑在原有类中引入一个创建新接口对象的工厂方法，从而对复杂的创建逻辑进行封装。