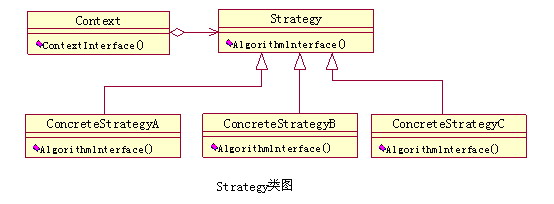
策略模式简介与使用

策略模式定义了一系列的算法，将每一个算法封装起来，使它们还可以相互替换。策略模式让算法和使用该算法的对象互相独立，彼此的变化互不影响。这里说的算法是一个抽象的概念，可以是一个类或一个方法。



上图是策略模式的类图，其中包含三种对象：

Context：使用算法的对象

Stategy：算法的基类，定义算法的公共接口

ConcreteStategy：具体的算法（具体的类或方法）

策略模式遵循的以下设计原则：

隔离系统中变化的部分

面向接口而非面向实现编程

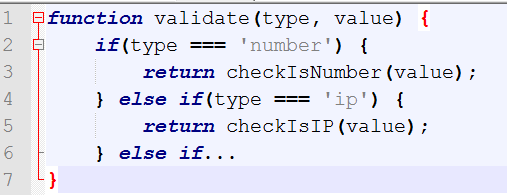
优先使用对象组合而非类继承

下面展示两个使用策略模式的实例。

表单验证器

需求：设计一个验证器，用于验证各种输入项的值是否合法。

最简单的实现，就是设计一个方法validate(type, value)，使用type标识验证的类型，方法内部使用条件语句if判断验证的类型type，进行针对性的验证。



这种做法会导致以下几个问题：

函数行数太多

内部充满判断语句

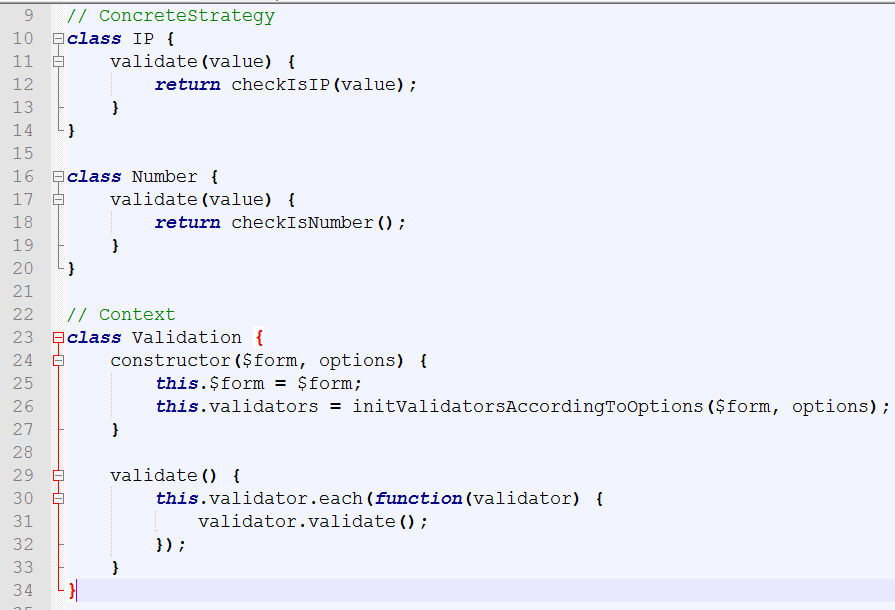
每添加一种验证方法都需要更改函数原码

组合判断不易实现

按照策略模式的进行设计，可以解决上面提到的问题。

隔离变化的部分，将每种验证方法封装成单独的验证器类

定义相同的接口

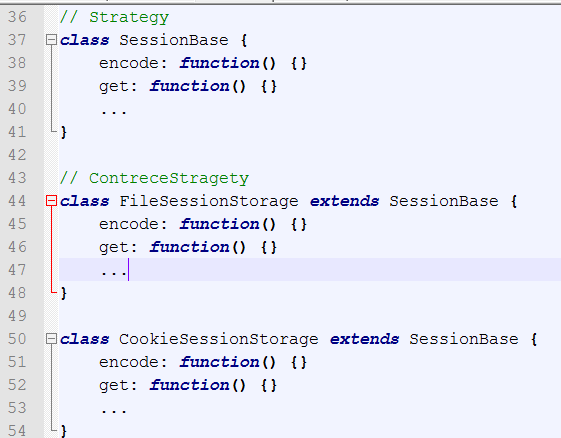


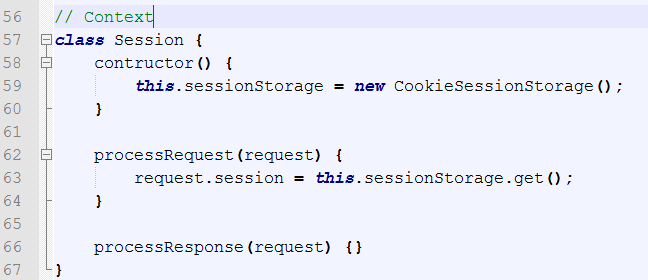
在验证方法validate内部，不需要进行验证类型的判断。只需在创建验证器时，指定验证类型即可。后续添加验证器对已有的验证器没有影响，也不需要更改已有的验证方法。

Session数据存储实现

需求：实现session数据的存储，支持多种方式和互相切换。

服务端的session数据有策略进行存储，可以存储在cookie、文件或数据库。





如上图所示，为每种存储方式定义一个类。每个类定义相同的接口，用于保存、获取session数据。这样，每种存储方式可以进行互相替换。至于使用何种存储方式，只需要在配置文件只指明使用的存储类即可。

使用场景

完成一个功能存在多个算法，且后续可能需要使用多种不同方法

优点

代码重用

灵活、容易扩展，易维护

减少代码中的条件判断语句

缺点

问题解决方式多时，策略类/方法数量可能较多