策略模式

概述

- 策略，具体可指算法，例如选择排序、插入排序，可指业务中具体的方法，如使用链表实现的定时器、使用最小堆实现的定时器等
- 在CS领域，没有通用最优解，只有在特定场景下的最优解，Redis中数据结构的实现诠释了该原则
  - 列表
    - 当元素数量少于512，且元素长度小于64字节时，采用ziplist实现
    - 当元素数量大于512，或者元素长度大于64字节时，采用双向列表实现
  - 哈希对象
    - 当元素数量少于512，且元素长度小于64字节时，采用ziplist实现
    - 当元素数量大于512，或者元素长度大于64字节时，采用字典实现
- 策略制定的规则
  - 在Redis数据结构的实现中，有着明显的性能优劣之分: 当数据量少时，尽可能地压缩内存使用，线性复杂度还是非线性复杂度关系影响不大；数据量多时，实现复杂度渐渐影响性能，此时应优化算法复杂度
  - 但是在业务场景中，可能策略是平级的，没有孰优孰劣，完全由业务系统决定。例如电商平台的满减功能，满100减10，满300减50
- **不管策略制定的规则如何，它们都有一个相同的特点: 多样化。多样化的选择是策略模式诞生的根本原因，将此类变化与原业务代码解耦，是策略模式的根本目的**

实现

策略的定义

策略的定义包括一个具体策略的接口和该接口的具体实现，使用接口的目的同其它模式一样: 动态运行时确定具体实现

```
interface Strategy {
    void Do();
}

class ConcreteStrategyA implements Strategy {
    @Override
    public void Do() { /* 策略A的具体实现 */ }
}

class ConcreteStrategyB implements Strategy {
    @Override
    public void Do() { /* 策略B的具体实现 */ }
}
```

策略的创建

通常来说，一个策略下会有多个不同的具体策略实现，为了更方便地使用策略，通常会使用简单工厂或者是工厂模式对创建对象进行封装

- 对象创建较为简单时可使用简单工厂模式
- 对象创建较为复杂时可使用工厂模式，将对象的创建下沉到具体工厂实现中

```
class StrategyFactory {
    private static final Map<String, Strategy> strategies = new HashMap<>();

    static {
        strategies.put("A", new ConcreteStrategyA());
        strategies.put("B", new ConcreteStrategyB());
    }

    public Strategy getStrategy(String value) {
        return strategies.get(value);  // 未做value的校验
    }
}
```

在使用简单工厂或者是工厂模式时，可根据实际策略的特性来决定如何创建实例

- 策略无状态
  - 可简单的使用一个map来保存对象，此时相当于一种单例模式
- 策略有状态
  - 使用if-else来决定创建哪个实例
  - 利用反射机制(Java)，或者是利用函数为一等公民的特性(Go、Python)来动态地创建

策略的使用

策略的使用则要相对简单一些，所有的具体策略均实现同一个接口，根据业务要求选择合适的策略注入至依赖类中即可

函数式编程语言中的"策略模式"

在函数式编程语言中，函数为一等公民，可作为右值对变量进行赋值，同时也可以作为函数参数传入至另外一个函数中。在策略模式中，策略其实就是一个函数，只不过因为具体编程语言的约束，无法直接传递，所以才会选择曲线救国的方式

以Python中的内置类filter为例

filter(function, iterable)，返回一个可迭代对象

```
data = [1, 2, 3, 4, 5, 6, 7]

result0 = list(filter(lambda x: x % 2 == 0, data))
result1 = list(filter(lambda x: x % 3 == 0, data))
result2 = list(filter(lambda x: x % 4 == 0, data))
```

使用lambda匿名函数表达式，我们可以对列表内的元素进行任意的筛选，该匿名函数就是筛选列表元素的具体策略

包括Python中的reduce函数，json模块中，均存在大量的以函数作为策略使用的场景。得益于语言的设计，策略模式在函数式编程语言中的使用相当简单