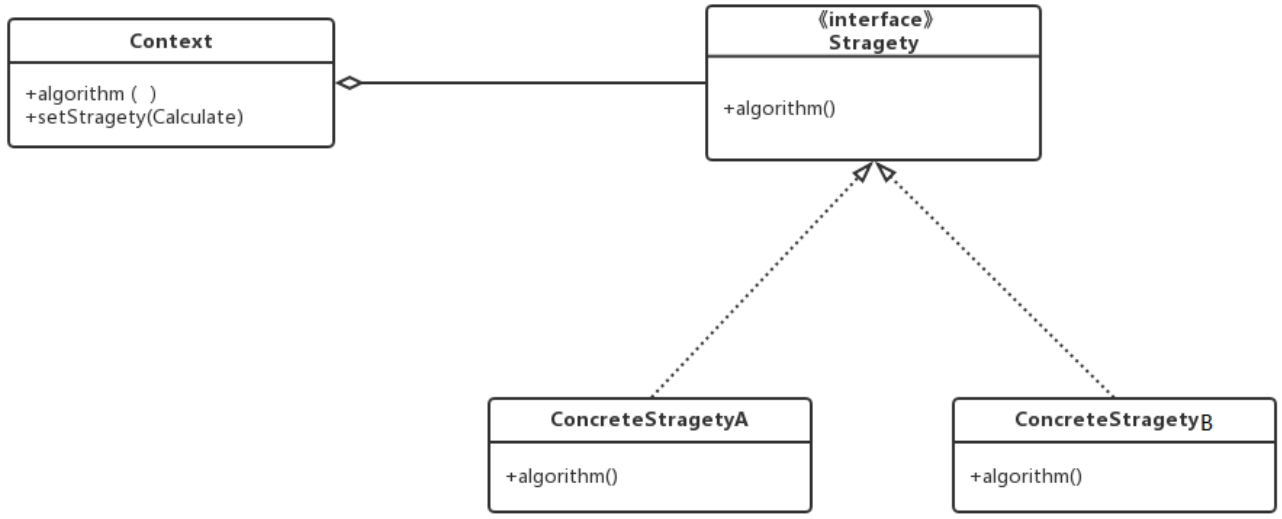
策略模式与状态模式

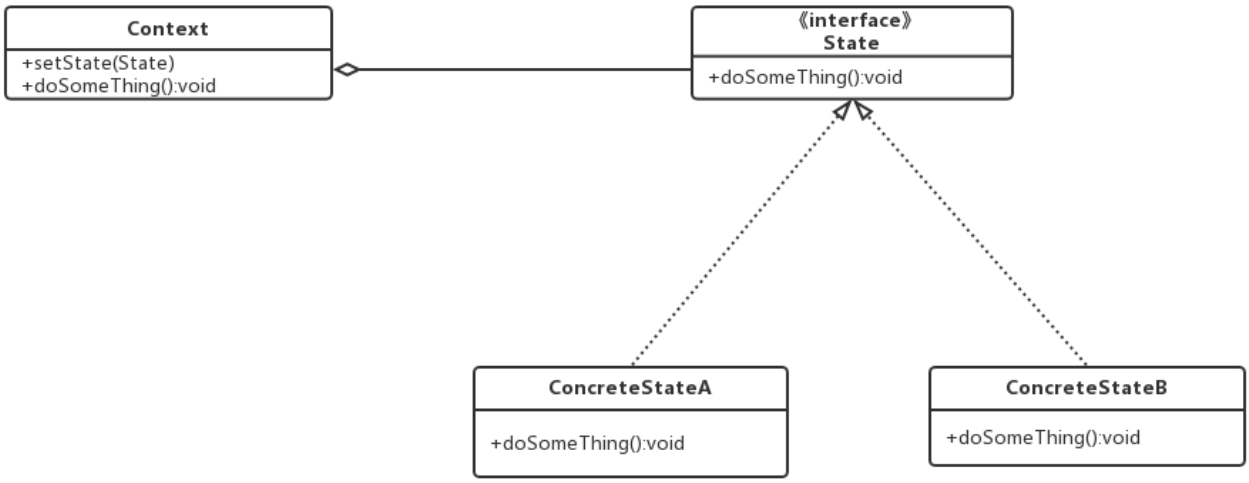
策略模式和状态模式

类图对比

策略模式：



状态模式：



两种模式的类图基本是一样的，即代码结构一样，即都 1个接口+N个实现类 +1个上下环境类。

把为什么还要划分为两种模式呢？

首先看看两者的定义：

	策略模式	状态模式
定义	策略模式定义了一系列算法，并将每一个算法封装起来，而且是它们还可以彼此独立、可相互替换	当一个对象的内在状态改变时允许改变其行为 这个对象看起（状态模式的行为时平行的，不可替换的）

举个通俗的小例子：

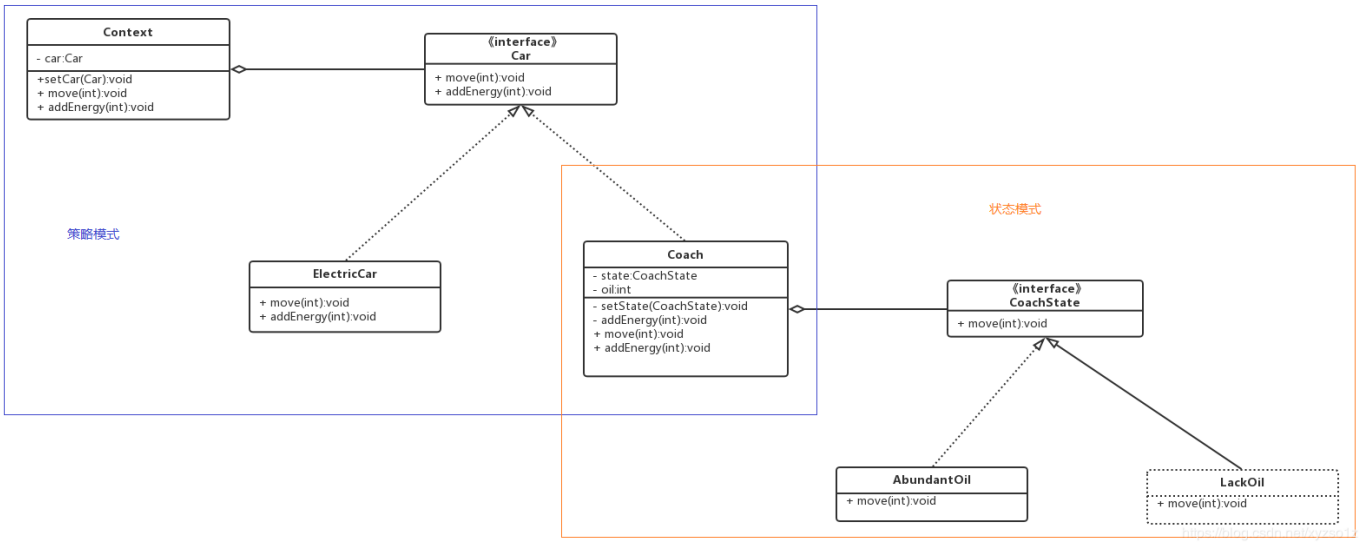
小明从A地到B地，有两种交通工具（高铁、长途客车）可以选择 ；长途客车有 有油\没油 两种状态，有油可以跑，没油就不能跑。

看看代码：

```
//车接口，
public interface Car {
     public void move(int distance);
     public void addEnergy(int energy);
}


//电动汽车
public class ElectricCar implements Car {
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        @Override
        public void move(int distance) {

                new Thread(new Runnable() {

                        @Override
                        public void run() {
                                System.out.print("开始时间: " + df.format(new Date()) + "    ");
                                try {
                                        Thread.sleep(distance * 1000 / 4);
                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }
                                System.out.print("电动汽车跑了: " + distance);
                                System.out.println("   结束时间: " + df.format(new Date()));
                        }
                }).start();

        }

        @Override
        public void addEnergy(int energy) {
                // 充电
        }
}


//长途客车状态接口，定义了客车的操作函数
public interface CoachState {
        public void move(int distance);
}
```

```
//客车缺油状态，此时不能移动
public class LackOil implements CoachState {

        @Override
        public void move(int distance) {
             System.out.println("没油了，跑不动了!");
        }

}

//客车有油的状态，此时可以移动
public class AbundantOil implements CoachState {
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        @Override
        public void move(int distance) {
                new Thread(new Runnable() {
                        @Override
                        public void run() {
                                System.out.print("开始时间: " + df.format(new Date()) + "    ");
                                try {
                                        Thread.sleep(distance * 1000 / 2);
                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }
                                System.out.print("汽车跑了: " + distance);
                                System.out.println("   结束时间: " + df.format(new Date()));
                        }
                }).start();

        }

}


public class Coach implements Car {
        private int oil = 5;
        private CoachState state;

        public Coach() {
                if (oil > 0) {
                         setState(new AbundantOil());
                } else {
                        setState(new LackOil());
                }
        }

        private void setState(CoachState state) {
                this.state = state;
        }

        // 消耗油
        public void useOil(int useOil) {
                oil =  oil - useOil>=0?oil - useOil:0 ;
                if (oil <= 0) {
```

```
                        setState(new LackOil());
                        System.out.println("没油了，请加油!");
                }
        }

        @Override
        public void move(int distance) {
                state.move(oil > distance ? distance : oil);
                useOil(oil > distance ? distance : oil);
        }

        @Override
        public void addEnergy(int energy) {
                oil = oil + energy;
                setState(new AbundantOil());
        }

}

//用来操作上下文的环境
public class Context {
        Car car;
        public void setCar(Car car) {
                this.car = car;
        }

        public void move(int distance) {
                car.move(distance);
        }
        public void addEnergy(int energy){
                car.addEnergy(energy);
        }
}

public class Client {
        public static void main(String[] args) {
                Context context=new Context();
                context.setCar(new ElectricCar());
                context.move(10);

        }

}
```


Context
- car:Car
+setCar(Car):void
+ move(int):void
+ addEnergy(int):void
《interface》
Car
+ move(int):void
+ addEnergy(int):void
策略模式
ElectricCar
+ move(int):void
+ addEnergy(int):void
状态模式
Coach
- state:CoachState
- oil:int
- setState(CoachState):void
- addEnergy(int):void
+ move(int):void
+ addEnergy(int):void
《interface》
CoachState
+ move(int):void
AbundantOil
+ move(int):void
LackOil
+ move(int):void