11 模板方法模式

在JS开发中用到继承场景其实不多，很多时候都喜欢用mix-in的方式来扩展。此外虽然也没有类和继承机制（ES6有了），但是可以通过原型变相继承

本章并非讨论继承，而是基于继承的设计模式--模板方法模式

定义和组成

两部分构成，抽象父类与实现字类。

- 在抽象父类封装子类的算法框架，包括实现公共方法以及封装子类方法的执行顺序
- 子类继承抽象父类，获取算法框架并可以选择性重写

其实模板方法模式是为了解决平行字类存在相同行为与不同行为时，相同部分复用上移的问题，体现了泛化的思想

第一个例子-Coffee or Tea

这个例子的原型来自《Head First 设计模式》。这一节我们使用JS来实现。当然过程稍微简化了点

泡一杯咖啡

```
class Coffee {
    constructor(){};

    boilWater = () => {console.log('煮沸水')};

    brewCoffee = () => {console.log('冲泡咖啡')}

    addMilk = () => {console.log('加牛奶')}

    init = () => {
        this.boilWater();
        this.brewCoffee();
        this.addMilk();
    }
}

const coffee = new Coffee();
```

```
coffee.init();
```

泡一壶茶

```
class Tea {
    constructor(){};

    boilWater = () => {console.log('煮沸水')};

    steepTea = () => {console.log('冲泡茶')}

    addLemon = () => {console.log('加柠檬')}

    init = () => {
        this.boilWater();
        this.steepTea();
        this.addLemon();
    }
}

const tea = new Tea();
tea.init();
```

抽象并分离共同点

我们发现泡茶和泡咖啡的不同点在于：原料，冲泡方式，调料不同，但是整个冲泡过程可以整理为三步走：

- 煮沸水
- 冲泡饮料
- 加调料

这里我们通过新的抽象方法名称与抽象父类来概述整个过程：

```
class Beverage {
    constructor(){};

    boilWater = () => {console.log('煮沸水')};

    brew = () => {}

    addCondiments = () => {}
```

```
9
10      init = () => {
11          this.boilWater();
12          this.brew();
13          this.addCondiments();
14      }
15  }
16
17  class newCoffee extends Beverage {
18      constructor(){};
19
20      brew = () => {console.log('冲泡咖啡')}
21
22      addCondiments = () => {console.log('加牛奶')}
23  }
24
25  const coffee2 = new newCoffee;
26  coffee2.init();
27
28  // newTea同理
```

上述ES6的继承背后通过原型链的方式来实现继承，最终newCoffee通过父类Beverage复用的共用行为，并通过重写实现了差异化。

上面射才是模板方法呢？答案是Beverage.prototype.init方法，因为该方法封装了整个子类的算法框架，作为一个算法的模板指导子类方法执行顺序