**介绍**

模板方法（TemplateMethod）定义了一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

模板方法是一种代码复用的基本技术，在类库中尤为重要，因为他们提取了类库中的公共行为。模板方法导致一种反向的控制结构，这种结构就是传说中的“好莱坞法则”，即“别找找我们，我们找你”，这指的是父类调用一个类的操作，而不是相反。具体体现是面向对象编程编程语言里的抽象类（以及其中的抽象方法），以及继承该抽象类（和抽象方法）的子类。

**正文**

举个例子，泡茶和泡咖啡有同样的步骤，比如烧开水（boilWater）、冲泡（brew）、倒在杯子里（pourOnCup），加小料（addCondiments）等等。但每种饮料冲泡的方法以及所加的小料不一样，所以我们可以利用模板方法实现这个主要步骤。

首先先来定义抽象步骤：

var CaffeineBeverage = function () {  
  
};  
CaffeineBeverage.prototype.prepareRecipe = function () {  
 this.boilWater();  
 this.brew();  
 this.pourOnCup();  
 if (this.customerWantsCondiments()) {  
 // 如果可以想加小料，就加上  
 this.addCondiments();  
 }  
};  
CaffeineBeverage.prototype.boilWater = function () {  
 console.log("将水烧开!");  
};  
CaffeineBeverage.prototype.pourOnCup = function () {  
 console.log("将饮料到再杯子里!");  
};  
CaffeineBeverage.prototype.brew = function () {  
 throw new Error("该方法必须重写!");  
};  
CaffeineBeverage.prototype.addCondiments = function () {  
 throw new Error("该方法必须重写!");  
};  
// 默认加上小料  
CaffeineBeverage.prototype.customerWantsCondiments = function () {  
 return true;  
};

该函数在原型上扩展了所有的基础步骤，以及主要步骤，冲泡和加小料步骤没有实现，供具体饮料所对应的函数来实现，另外是否加小料（customerWantsCondiments ）默认返回true，子函数重写的时候可以重写该值。

下面两个函数分别是冲咖啡和冲茶所对应的函数：

// 冲咖啡  
var Coffee = function () {  
 CaffeineBeverage.apply(this);  
};  
Coffee.prototype = new CaffeineBeverage();  
Coffee.prototype.brew = function () {  
 console.log("从咖啡机想咖啡倒进去!");  
};  
Coffee.prototype.addCondiments = function () {  
 console.log("添加糖和牛奶");  
};  
Coffee.prototype.customerWantsCondiments = function () {  
 return confirm("你想添加糖和牛奶吗？");  
};  
  
//冲茶叶  
var Tea = function () {  
 CaffeineBeverage.apply(this);  
};  
Tea.prototype = new CaffeineBeverage();  
Tea.prototype.brew = function () {  
 console.log("泡茶叶!");  
};  
Tea.prototype.addCondiments = function () {  
 console.log("添加柠檬!");  
};  
Tea.prototype.customerWantsCondiments = function () {  
 return confirm("你想添加柠檬嘛？");  
};

另外使用confirm，可以让用户自己选择加不加小料，很不错，不是嘛？

**总结**

模板方法应用于下列情况：

1. 一次性实现一个算法的不变的部分，并将可变的行为留给子类来实现
2. 各子类中公共的行为应被提取出来并集中到一个公共父类中的避免代码重复，不同之处分离为新的操作，最后，用一个钓鱼这些新操作的模板方法来替换这些不同的代码
3. 控制子类扩展，模板方法只在特定点调用“hook”操作，这样就允许在这些点进行扩展

和策略模式不同，模板方法使用继承来改变算法的一部分，而策略模式使用委托来改变整个算法。

参考：https://github.com/tcorral/Design-Patterns-in-Javascript/blob/master/Template/withHook/index.html