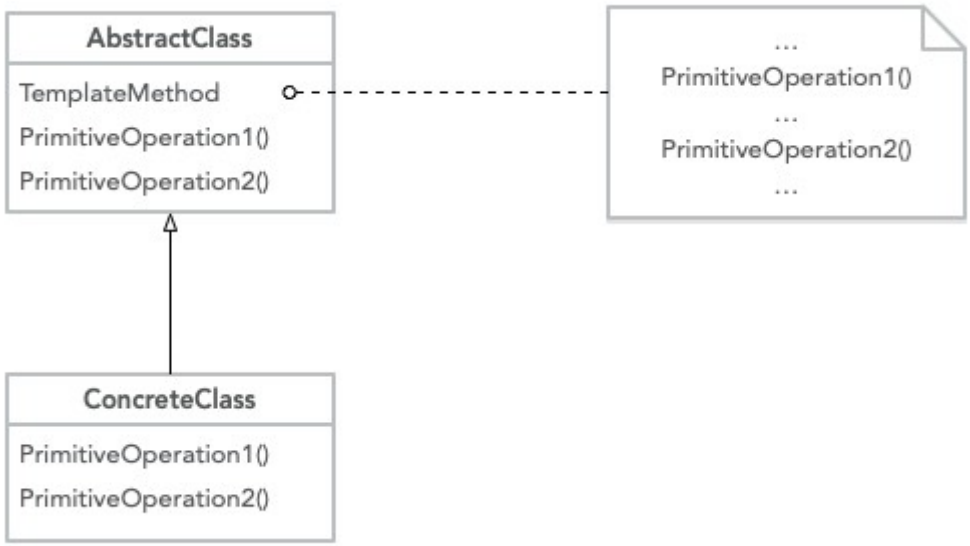
# PHP设计模式之模板方法模式

模板方法模式，也是我们经常会在不经意间有会用到的模式之一。这个模式是对继承的最好诠释。当子类中有重复的动作时，将他们提取出来，放在父类中进行统一的处理，这就是模板方法模式的最简单通俗的解释。就像我们平时做项目，每次的项目流程实都差不多，都有调研、开发、测试、部署上线等流程。而具体到每个项目中，这些流程的实现又不会完全相同。这个流程，就像是模板方法，让我们每次都按照这个流程进行开发。

## Gof类图及解释

**GoF定义：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。TemplateMethod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。**

> GoF类图




> 代码实现

```php
abstract class AbstractClass
{
    public function TemplateMethod()
    {
        $this->PrimitiveOperation1();
        $this->PrimitiveOperation2();
    }

    abstract public function PrimitiveOperation1();
    abstract public function PrimitiveOperation2();
}
```

定义一个抽象类，有一个模板方法TemplateMethod()，这个方法中我们对算法操作方法进行调用。而这些算法抽象方法是在子类中去实现的。

```php
class ConcreteClassA extends AbstractClass
{
    public function PrimitiveOperation1()
    {
        echo '具体类A实现方法1', PHP_EOL;
    }
    public function PrimitiveOperation2()
    {
        echo '具体类A实现方法2', PHP_EOL;
    }
}

class ConcreteClassB extends AbstractClass
{
    public function PrimitiveOperation1()
    {
        echo '具体类B实现方法1', PHP_EOL;
    }
    public function PrimitiveOperation2()
    {
        echo '具体类B实现方法2', PHP_EOL;
    }
}
```

具体的实现类，它们只需要去实现父类所定义的算法就可以了。

```php
$c = new ConcreteClassA();
$c->TemplateMethod();

$c = new ConcreteClassB();
$c->TemplateMethod();
```

在客户端的调用中，实例化子类，但调用的是子类所继承的父类的模板方法。就可以实现统一的算法调用了。

- 模板方法模式相信只要是做过一点面向对象开发的朋友都会多多少少使用过。因为真的非常常见
- 一些框架中经常会有某些功能类有初始化的功能，在初始化的函数中都会调用很多内部的其他函数，这其实也是一种模板方法模式的应用
- 模板方法模式可以很方便的实现钩子函数。就像很多模板或者开源系统中给你准备好的钩子函数。比如某些博客开源程序会预留一些广告位或者特殊位置的钩子函数让使用者自己按需实现
- 模板方法模式适用于：一次性实现一个算法中不变的部分，并将可变的部分留给子类来实现；将子类中公共的行为提取出来并集中到父类中；控制子类的扩展；
- 这个模式体现了一个叫“好莱坞法则”的原则，那就是“别找我们，我们来找你”

**在公司中，我非常的推崇敏捷式的项目管理，当然，这里也不是说传统的项目管理有多么不好，只是敏捷更适合我们这种短平快的公司。在敏捷中，我们采用的是Scurm框架，它其实就是一个模板。它定义了四种会议、三种人员、三个工具。在每个项目的具体实现中，我们都会遵守这些规则，但具体的实现又不会一样。比如有时我们是一周一个迭代，有时是一个月一个迭代。有时我们不需要回顾会议，而是将回顾和评审会议放在了一起进行。不管怎么样，我们会在Scurm的基础上进行灵活的项目开发。而做为领导的我，只需要在每个项目中调取Scurm的基本流程就可以了。所以说，公司的强大和大家的学习是分不开的，好用的东西当然要时刻学习分享并应用啦！！**

**完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/20.template-method/source/template-method.php**

## 实例

不发短信了，这次我们实现的是一个Cache类的初始化部分。就像上文说过的一些框架中的工具类。一般Cache我们会使用Memcached或者Redis来实现，所以我们抽取一个公共Cache类，然后让Memcached和Redis的Cache实现类都继承它。在公共类中，通过模板方法来进行实现类的一些初始化工作，这些工作由父类统一调用，实现类只需要实现每一个步骤的具体内容就可以了。

> 缓存类图

**完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/20.template-method/source/template-method-cache.php**

```php
<?php

abstract class Cache
{
    private $config;
    private $conn;

```

```php
    public function __construct()
    {
        $this->init();
    }
    public function init()
    {
        $this->GetConfig();
        $this->OpenConnection();
        $this->CheckConnection();
    }

    abstract public function GetConfig();
    abstract public function OpenConnection();
    abstract public function CheckConnection();
}

class MemcachedCache extends Cache
{
    public function GetConfig()
    {
        echo '获取Memcached配置文件！', PHP_EOL;
        $this->config = 'memcached';
    }
    public function OpenConnection()
    {
        echo '链接memcached!', PHP_EOL;
        $this->conn = 1;
    }
    public function CheckConnection()
    {
        if ($this->conn) {
            echo 'Memcached连接成功！', PHP_EOL;
        } else {
            echo 'Memcached连接失败，请检查配置项！', PHP_EOL;
        }
    }
}

class RedisCache extends Cache
{
```

```php
    public function GetConfig()
    {
        echo '获取Redis配置文件！', PHP_EOL;
        $this->config = 'redis';
    }
    public function OpenConnection()
    {
        echo '链接redis!', PHP_EOL;
        $this->conn = 0;
    }
    public function CheckConnection()
    {
        if ($this->conn) {
            echo 'Redis连接成功！', PHP_EOL;
        } else {
            echo 'Redis连接失败，请检查配置项！', PHP_EOL;
        }
    }
}

$m = new MemcachedCache();

$r = new RedisCache();
```

> 说明

- 这样一个简单的缓存类我们就实现了。是不是和很多框架中的代码非常类似。
- 子类只需要定义自己的实现就可以了，剩下的重复代码都让父类去完成，如果没有父类，它们都需要自己实现一个init()方法
- 当然，需要增加其它的实现类时，也只需要继承这个Cache父类后完成自己的实现就可以了，客户端面对这些实现类都能非常轻松，因为它们知道自己只需要先调用一下初始化方法可以使用这个类了，不管是哪一个实现类都是一样的