简述

模板方法模式需要开发抽象类和具体子类的设计师之间的协作。一个设计师负责给出一个算法的轮廓和骨架，另一些设计师则负责给出这个
辑步骤的方法称做基本方法(primitive method)；而将这些基本方法汇总起来的方法叫做模板方法(template method)，这个设计模式的名字

图中， 兔子就是一个模板，可以涂上 白色，黑色，灰色等。不管是什么颜色，但是兔子的形状都被模板规定了。

模式中的角色

- 抽象类（父类）：负责算法中每个计算步骤的抽象定义，并且定义具体算法来将抽象的计算步骤串联
- 具体类（子类）：负责实现父类中算法的每个抽象计算步骤的实现

代码实现

```
1 /**
```

```java
 * 玩游戏的抽象类
 */
public abstract class GameAbs {

    abstract void init();

    abstract void start();

     abstract void fun();

     abstract void end();

     final void play(){
         init();
         start();
         fun();
         end();
     }
}

/**
 * 足球游戏
 */
public class FootGame extends GameAbs {
    @Override
    void init() {
        System.out.println("找足球场地，召集小伙伴");
    }

    @Override
    void start() {
        System.out.println("比赛开始");
    }

    @Override
    void fun() {
        System.out.println("大家玩得很开心");
    }

    @Override
    void end() {
        System.out.println("比赛结束，结果 3: 3");
    }
}

/**
 * 篮球游戏
 */
public class Basketball extends GameAbs {
    @Override
    void init() {
        System.out.println("到篮球场结合");
    }

```

```
    @Override
    void start() {
        System.out.println("人员分配完成，开始");
    }

    @Override
    void fun() {
        System.out.println("你来我往，激烈比赛");
    }

    @Override
    void end() {
        System.out.println("时间到，比赛结果 100 : 100");
    }
}


public class Main {
    public static void main(String[] args) {
        GameAbs football = new FootGame();
        football.play();
        System.out.println();
        GameAbs basketball = new Basketball();
        basketball.play();
    }
}
```

总结

优点

- 封装不变部分，扩展可变部分。把认为不变部分的算法封装到父类中实现，而可变部分的则可以通过继承来继续扩展。
- 提取公共部分代码，便于维护。
- 行为由父类控制，子类实现

缺点

- 算法骨架需要改变时需要修改抽象类，还有实现类
- 每个不同的实现都需要定义一个子类，会导致类的个数增加，系统更加庞大