前面所学习的HttpServlet类所使用的设计方式是23种设计模式中的模板方法设计模式，这种设计模式有什么优点和特点下面来探讨。

1、首先我们以普通的方式展示

创建一个学生类和一个工人类，两个类中都有一个day方法表示一天的行为

Student.java

```
package com.servlet.http.notemplate;

//不使用模板方法设计模式
public class Student {

    public void day(){
        System.out.println("学生起床");
        System.out.println("学生上课");
        System.out.println("学生放学");
        System.out.println("学生吃饭");
        System.out.println("学生睡觉");
    }

}
```

Worker.java

```
package com.servlet.http.notemplate;

public class Worker {
    public void day(){
        System.out.println("工人起床");
        System.out.println("工人上班");
        System.out.println("工人下班");
        System.out.println("工人吃饭");
        System.out.println("工人睡觉");
    }
}
```

一个测试类Test.java

```
package com.servlet.http.notemplate;

public class Test {

    public static void main(String[] args) {

        Student student = new Student();
        Worker worker = new Worker();
        student.day();
        worker.day();
    }
```

```
}
```

现实开发中完成业务需求需要多个平级的Servlet实现类，这些类中都有一套算法完成需求，有时候这些类中的算法骨架是一样的。

我们将day方法看作是一个算法，可以看到上面两个类中的核心算法骨架都是一样的，如果多出几个类的话，我们就需要在每个类中都编写同一种算法，消耗时间。

这样的后果就会使代码冗余，并且可能某个类中大意出错，不能很好的保护算法，算法会被修改

3、下面使用模板方法设计模式

编写一个模板类，这个类是一个抽象类，如下：

```
package com.servlet.http.usetemplate;


/**
 * PersonTemplate符合模板方法设计
 * PersonTemplate是一个模板类，抽象类
 */
public abstract class PersonTemplate {

    /**
     * templateMethod是一个模板方法，它定义算法核心骨架，具体实现步骤延迟到模板类的子类中实现
     * 为了使核心算法骨架受到保护，模板方法一般被final修饰
     * 作用：算法骨架不需要在每个业务类中编写了，只需要在模板方法中编写一次
     */
    public void templateMethod(){

        //核心算法骨架
        do1();
        do2();
        do3();

    }

    /**
     * 下面方法就是具体的实现步骤之一，可以延迟到子类中完成
     * 通常是抽象方法，所以模板类也是抽象类
     */
    public abstract void do1();
    public abstract void do2();
    public abstract void do3();

}
```

Student类和Worker类继承模板类，重写其中的抽象方法，如下：

Student

```
package com.servlet.http.usetemplate;

public class Student extends PersonTemplate{
    @Override
    public void do1() {
        System.out.println("学生起床");
    }

    @Override
    public void do2() {
        System.out.println("学生上学");
    }

    @Override
    public void do3() {
        System.out.println("学生放学");
    }
}
```

Worker

```
package com.servlet.http.usetemplate;

public class Worker extends PersonTemplate{
    @Override
    public void do1() {
        System.out.println("工人起床");
    }

    @Override
    public void do2() {
        System.out.println("工人上班");
    }

    @Override
    public void do3() {
        System.out.println("工人下班");
    }
}
```

测试类Test

```
package com.servlet.http.usetemplate;

public class Test {

    public static void main(String[] args) {

        //用模板类定义一个Student对象，明显是多态的运用
```

```
        PersonTemplate student = new Student();
        PersonTemplate worker = new Worker();

        //子类都可以调用模板方法templateMethod
        //模板发设计模式就是基于多态来实现的
        student.templateMethod();
        worker.templateMethod();

        //其特点就是，这个模板类的子类的核心算法骨架相同，保存在一个模板方法中，这个模板方法
        //在子类中无需重写，模板方法中的内容是各个Servlet实现类中都固定死，模板嘛，顾名思义
        //需要程序员编写的就是模板方法中调用的其他方法，这些方法就是需要程序员写的业务代码
        //且这些方法在子类中各自重写实现，子类含义不同，实现步骤就有所不同
    }

}
```

向上面这样的使用模板方法设计模式，如果我们想改算法，这里我们就修改do1与do2方法的执行顺序，只要在模板方法中修改就可以将其所有子类中的都修改到了，一改一大批。这体现着一种面向抽象，面向父类编程的思想，不能面向具体编程。

```
--模板方法设计模式
    Servlet规范中的：HttpServlet
    这个类是一个模板类
    其中有两个service方法，构成方法的重载，其中有protected权限关键字修饰的
    service(HttpServletRequest request,HttpServletResponse response)方法是典型的模板方法，其定义了核心算
    法骨架，如doGet(),doPost(),...具体的实现步骤延迟到子类中完成
--模板方法设计特点
    doXXX
    doYYY
    doNNN

--模板方法设计模式属于：行为型设计模式

--作用
    1、核心算法受到保护
    2、核心算法得到复用
    3、在不改变算法结构的前提下，可以重新定义算法具体步骤实现
```