链滴

# 简单工厂模式

作者：kevin2020
原文链接：https://ld246.com/article/1513702003253
来源网站：链滴

# 为什么要引入工厂类

通过引入工厂类，客户类不涉及对象的创建，对象的创建者也不会涉及对象的使用。

在Java语言中，我们通常有以下几种创建对象的方式：

(1) 使用new关键字直接创建对象；

(2) 通过反射机制创建对象；

(3) 通过clone()方法创建对象；

(4) 通过工厂类创建对象。

在所有的工厂模式中，我们都强调一点：**两个类A和B之间的关系应该仅仅是A创建B或是A使用B，而能两种关系都有。**将对象的创建和使用分离，也使得系统更加符合“单一职责原则”，有利于对功能复用和系统的维护。

这在Joshua Kerievsky的《重构与模式》一书中有专门的一节来进行介绍。因为有时候我们创建一个象不只是简单调用其构造函数，还需要设置一些参数，可能还需要配置环境，如果将这些代码散落在一个创建对象的客户类中，势必会出现**代码重复、创建蔓延**的问题，而这些客户类其实无须承担对象创建工作，它们只需使用已创建好的对象就可以了。此时，可以引入工厂类来封装对象的创建逻辑和户代码的实例化/配置选项。

但如果将对象的创建过程封装在工厂类中，我们**可以提供一系列名字完全不同的工厂方法，每一个工方法对应一个构造函数**，客户端可以以一种更加可读、易懂的方式来创建对象。

# 定义

**简单工厂模式是属于创建型模式，又叫做静态工厂方法（Static Factory Method）模式，但不属于2种GOF设计模式之一。简单工厂模式是由一个工厂对象决定创建出哪一种产品类的实例。简单工厂模是工厂模式家族中最简单实用的模式，可以理解为是不同工厂模式的一个特殊实现。**

# 实例

我们写出如下filter。

```
package com.web.filter;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;

import com.web.factory.ServletFactory;
//用来分派请求的filter
```

```
public class DispatcherFilter implements Filter{

    private static final String URL_SEPARATOR = "/";

    private static final String SERVLET_PREFIX = "servlet/";

    private String servletName;

    public void init(FilterConfig filterConfig) throws ServletException {}

    public void destroy() {}

    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,Filter
hain filterChain) throws IOException, ServletException {
        parseRequestURI((HttpServletRequest) servletRequest);
        //这里为了体现我们本节的重点，我们采用一个工厂来帮我们制造Action
        if (servletName != null) {
            //这里使用的正是简单工厂模式，创造出一个servlet，然后我们将请求转交给servlet处理
            Servlet servlet = ServletFactory.createServlet(servletName);
            servlet.service(servletRequest, servletResponse);
        }else {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }

    //负责解析请求的URI，我们约定请求的格式必须是/contextPath/servlet/servletName
    //不要怀疑约定的好处，因为LZ一直坚信一句话，约定优于配置
    private void parseRequestURI(HttpServletRequest httpServletRequest){
        String validURI = httpServletRequest.getRequestURI().replaceFirst(httpServletRequest.ge
ContextPath() + URL_SEPARATOR, "");
        if (validURI.startsWith(SERVLET_PREFIX)) {
            servletName = validURI.split(URL_SEPARATOR)[1];
        }
    }

}
```

这个filter需要在web.xml中加入以下配置，这个不多做介绍，直接贴上来。

```
<filter>
      <filter-name>dispatcherFilter</filter-name>
      <filter-class>com.web.filter.DispatcherFilter</filter-class>
  </filter>
  <filter-mapping>
      <filter-name>dispatcherFilter</filter-name>
      <url-pattern>/servlet/*</url-pattern>
  </filter-mapping>
```

本章的重点是简单工厂模式，所以我们这里突出我们本章的主角，使用简单工厂来创造servlet去处理户的请求，当然如果你是一个JAVA造诣比较深的程序猿，出于好奇进来一观，或许会对这种简单工模式的处理方式不屑一顾，不过我们不能偏离主题，我们的目的不是模拟一个struts2，而是介绍简单厂。

下面给出我们的主角，我们的servlet工厂，它就相当于上面的Creator。

```
package com.web.factory;

import javax.servlet.Servlet;

import com.web.exception.ServletException;
import com.web.servlet.LoginServlet;
import com.web.servlet.LoginoutServlet;
import com.web.servlet.RegisterServlet;

public class ServletFactory {

    private ServletFactory(){}
    //一个servlet工厂，专门用来生产各个servlet，而我们生产的依据就是传入的servletName，
    //这个serlvetName由我们在filter截获，传给servlet工厂。
    public static Servlet createServlet(String servletName){
        if (servletName.equals("login")) {
            return new LoginServlet();
        }else if (servletName.equals("register")) {
            return new RegisterServlet();
        }else if (servletName.equals("loginout")) {
            return new LoginoutServlet();
        }else {
            throw new ServletException("unknown servlet");
        }
    }
}
```

看到这里，是不是有点感觉了呢，我们一步一步去消除servlet的XML配置的过程，其实就是在慢慢的出一个简单工厂模式.

了解简单工厂模式，为接下来的工厂方法模式与抽象方法模式做准备。

一般应用于管理的类比较少，场景不复杂时使用，正如其名 简单。