

## 课程目标

- 1、了解看源码最有效的方式，先猜测后验证，不要一开始就去调试代码。
- 2、浓缩就是精华，用 300 行最简洁的代码提炼 Spring 的基本设计思想。
- 3、结合设计模式，掌握 Spring 框架的基本脉络。

## 内容定位

- 1、具有 1 年以上的 SpringMVC 使用经验。
- 2、希望深入了解 Spring 源码的人群，对 Spring 有一个整体的宏观感受。
- 3、全程手写实现 SpringMVC 的核心功能，帮助大家更深刻地理解设计模式。从最简单的 v1 版本一步一步优化为 v2 版本，最后到 v3 版本。

## 实现思路

先来介绍一下 Mini 版本的 Spring 基本实现思路，如下图所示：



## 自定义配置

### 配置 application.properties 文件

为了解析方便，我们用 application.properties 来代替 application.xml 文件，具体配置内容如下：

```
scanPackage=com.gupaoedu.demo
```

### 配置 web.xml 文件

大家都知道，所有依赖于 web 容器的项目，都是从读取 web.xml 文件开始的。我们先配置好 web.xml 中的内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:javaee="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
```

```

<display-name>Gupao Web Application</display-name>
<servlet>
  <servlet-name>gpmvc</servlet-name>
  <servlet-class>com.gupaoedu.mvcframework.v1.servlet.GPDispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>application.properties</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>gpmvc</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
</web-app>

```

其中 GPDispatcherServlet 是有自己模拟 Spring 实现的核心功能类。

## 自定义 Annotation

@GPService 注解：

```

package com.gupaoedu.mvcframework.annotation;
import java.lang.annotation.*;
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface GPService {
  String value() default "";
}

```

@GPAutowired 注解：

```

package com.gupaoedu.mvcframework.annotation;
import java.lang.annotation.*;
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface GPAutowired {
  String value() default "";
}

```

@GPController 注解：

```

package com.gupaoedu.mvcframework.annotation;

```

```
import java.lang.annotation.*;
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface GPController {
    String value() default "";
}
```

@GPRequestMapping 注解：

```
package com.gupaoedu.mvcframework.annotation;
import java.lang.annotation.*;
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface GPRequestMapping {
    String value() default "";
}
```

@GPRequestParam 注解：

```
package com.gupaoedu.mvcframework.annotation;
import java.lang.annotation.*;
@Target({ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface GPRequestParam {
    String value() default "";
}
```

## 配置 Annotation

配置业务实现类 DemoService：

```
package com.gupaoedu.demo.service.impl;
import com.gupaoedu.demo.service.IDemoService;
import com.gupaoedu.mvcframework.annotation.GPService;
/**
 * 核心业务逻辑
 */
@GPService
public class DemoService implements IDemoService{
    public String get(String name) {
        return "My name is " + name;
    }
}
```

}

配置请求入口类 DemoAction :

```
package com.gupaoedu.demo.mvc.action;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.gupaoedu.demo.service.IDemoService;
import com.gupaoedu.mvcframework.annotation.GPAutowired;
import com.gupaoedu.mvcframework.annotation.GPController;
import com.gupaoedu.mvcframework.annotation.GPRequestMapping;
import com.gupaoedu.mvcframework.annotation.GPRequestParam;
@GPController
@GPRequestMapping("/demo")
public class DemoAction {
    @GPAutowired private IDemoService demoService;
    @GPRequestMapping("/query")
    public void query(HttpServletRequest req, HttpServletResponse resp,
        @GPRequestParam("name") String name){
        String result = demoService.get(name);
        try {
            resp.getWriter().write(result);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @GPRequestMapping("/add")
    public void add(HttpServletRequest req, HttpServletResponse resp,
        @GPRequestParam("a") Integer a, @GPRequestParam("b") Integer b){
        try {
            resp.getWriter().write(a + "+" + b + "=" + (a + b));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @GPRequestMapping("/remove")
    public void remove(HttpServletRequest req, HttpServletResponse resp,
        @GPRequestParam("id") Integer id){
    }
}
```

至此，配置阶段就已经完成。

# 容器初始化

## 实现 V1 版本

所有的核心逻辑全部写在一个 init()方法中。

```
package com.gupaoedu.mvcframework.v1.servlet;
import com.gupaoedu.mvcframework.annotation.GPAutowired;
import com.gupaoedu.mvcframework.annotation.GPController;
import com.gupaoedu.mvcframework.annotation.GPRequestMapping;
import com.gupaoedu.mvcframework.annotation.GPService;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.*;

public class GPDispatcherServlet extends HttpServlet {
    private Map<String, Object> mapping = new HashMap<String, Object>();
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
    IOException {this.doPost(req, resp);}
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
    IOException {
        try {
            doDispatch(req, resp);
        } catch (Exception e) {
            resp.getWriter().write("500 Exception " + Arrays.toString(e.getStackTrace()));
        }
    }
    private void doDispatch(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        String url = req.getRequestURI();
        String contextPath = req.getContextPath();
```

```

url = url.replace(contextPath, "").replaceAll("/+", "/");
if(!this.mapping.containsKey(url)){resp.getWriter().write("404 Not Found!!");return;}
Method method = (Method) this.mapping.get(url);
Map<String,String[]> params = req.getParameterMap();
method.invoke(this.mapping.get(method.getDeclaringClass().getName()),new
Object[] {req,resp,params.get("name")[0]});
}
@Override
public void init(ServletConfig config) throws ServletException {
    InputStream is = null;
    try{
        Properties configContext = new Properties();
        is =
this.getClass().getClassLoader().getResourceAsStream(config.getInitParameter("contextConfigLocat
ion"));
        configContext.load(is);
        String scanPackage = configContext.getProperty("scanPackage");
        doScanner(scanPackage);
        for (String className : mapping.keySet()) {
            if(!className.contains(".")){continue;}
            Class<?> clazz = Class.forName(className);
            if(clazz.isAnnotationPresent(GPController.class)){
                mapping.put(className,clazz.newInstance());
                String baseUrl = "";
                if (clazz.isAnnotationPresent(GPRequestMapping.class)) {
                    GPRequestMapping requestMapping =
clazz.getAnnotation(GPRequestMapping.class);
                    baseUrl = requestMapping.value();
                }
                Method[] methods = clazz.getMethods();
                for (Method method : methods) {
                    if (!method.isAnnotationPresent(GPRequestMapping.class)) { continue; }
                    GPRequestMapping requestMapping =
method.getAnnotation(GPRequestMapping.class);
                    String url = (baseUrl + "/" + requestMapping.value()).replaceAll("/+", "/");
                    mapping.put(url, method);
                    System.out.println("Mapped " + url + "," + method);
                }
            }else if(clazz.isAnnotationPresent(GPService.class)){
                GPService service = clazz.getAnnotation(GPService.class);
                String beanName = service.value();
                if("").equals(beanName)){beanName = clazz.getName();}
                Object instance = clazz.newInstance();
                mapping.put(beanName,instance);
            }
        }
    }
}

```

```

        for (Class<?> i : clazz.getInterfaces()) {
            mapping.put(i.getName(), instance);
        }
    }else {continue;}
}
for (Object object : mapping.values()) {
    if(object == null){continue;}
    Class clazz = object.getClass();
    if(clazz.isAnnotationPresent(GPController.class)){
        Field [] fields = clazz.getDeclaredFields();
        for (Field field : fields) {
            if(!field.isAnnotationPresent(GPAutowired.class)){continue; }
            GPAutowired autowired = field.getAnnotation(GPAutowired.class);
            String beanName = autowired.value();
            if("").equals(beanName)){beanName = field.getType().getName();}
            field.setAccessible(true);
            try {
                field.set(mapping.get(clazz.getName()), mapping.get(beanName));
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
    }
}
} catch (Exception e) {
}finally {
    if(is != null){
        try {is.close();} catch (IOException e) {
            e.printStackTrace();
        }
    }
}
System.out.print("GP MVC Framework is init");
}

private void doScanner(String scanPackage) {
    URL url = this.getClass().getClassLoader().getResource("/") +
scanPackage.replaceAll("\\.", "/");
    File classDir = new File(url.getFile());
    for (File file : classDir.listFiles()) {
        if(file.isDirectory()){ doScanner(scanPackage + "." + file.getName());}else {
            if(!file.getName().endsWith(".class")){continue;}
            String clazzName = (scanPackage + "." + file.getName().replace(".class", ""));
            mapping.put(clazzName, null);
        }
    }
}

```



```

    }
}
}

```

## 实现 V2 版本

在 V1 版本上进行了优化，采用了常用的设计模式（工厂模式、单例模式、委派模式、策略模式），将 init() 方法中的代码进行封装。按照之前的实现思路，先搭基础框架，再填肉注血，具体代码如下：

```

//初始化阶段
@Override
public void init(ServletConfig config) throws ServletException {

    //1、加载配置文件
    doLoadConfig(config.getInitParameter("contextConfigLocation"));

    //2、扫描相关的类
    doScanner(contextConfig.getProperty("scanPackage"));

    //3、初始化扫描到的类，并且将它们放入到 IOC 容器之中
    doInstance();

    //4、完成依赖注入
    doAutowired();

    //5、初始化 HandlerMapping
    initHandlerMapping();

    System.out.println("GP Spring framework is init.");

}

```

声明全局的成员变量，其中 IOC 容器就是注册时单例的具体案例：

```

//保存 application.properties 配置文件中的内容
private Properties contextConfig = new Properties();

//保存扫描的所有类名
private List<String> classNames = new ArrayList<String>();

//传说中的 IOC 容器，我们来揭开它的神秘面纱
//为了简化程序，暂时不考虑 ConcurrentHashMap
// 主要还是关注设计思想和原理

```

```
private Map<String,Object> ioc = new HashMap<String,Object>();

//保存 url 和 Method 的对应关系
private Map<String,Method> handlerMapping = new HashMap<String,Method>();
```

实现 doLoadConfig()方法：

```
//加载配置文件
private void doLoadConfig(String contextConfigLocation) {
    //直接从类路径下找到 Spring 主配置文件所在的路径
    //并且将其读取出来放到 Properties 对象中
    //相对于 scanPackage=com.gupaoedu.demo 从文件中保存到了内存中
    InputStream fis = this.getClass().getClassLoader().getResourceAsStream(contextConfigLocation);
    try {
        contextConfig.load(fis);
    } catch (IOException e) {
        e.printStackTrace();
    }finally {
        if(null != fis){
            try {
                fis.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

实现 doScanner()方法：

```
//扫描出相关的类
private void doScanner(String scanPackage) {
    //scanPackage = com.gupaoedu.demo ， 存储的是包路径
    //转换为文件路径，实际上就是把.替换为/就 OK 了
    //classpath
    URL url = this.getClass().getClassLoader().getResource("/") +
scanPackage.replaceAll("\\.", "/");
    File classPath = new File(url.getFile());
    for (File file : classPath.listFiles()) {
        if(file.isDirectory()){
            doScanner(scanPackage + "." + file.getName());
        }else{
            if(!file.getName().endsWith(".class")){ continue;}
            String className = (scanPackage + "." + file.getName().replace(".class",""));
            classNames.add(className);
        }
    }
}
```

```

    }
}

```

实现 doInstance()方法，doInstance()方法就是工厂模式的具体实现：

```

private void doInstance() {
    //初始化，为 DI 做准备
    if(classNames.isEmpty()){return;}

    try {
        for (String className : classNames) {
            Class<?> clazz = Class.forName(className);

            //什么样的类才需要初始化呢？
            //加了注解的类，才初始化，怎么判断？
            //为了简化代码逻辑，主要体会设计思想，只举例 @Controller 和@Service,
            // @Component...就一一举例了
            if(clazz.isAnnotationPresent(GPController.class)){
                Object instance = clazz.newInstance();
                //Spring 默认类名首字母小写
                String beanName = toLowerFirstCase(clazz.getSimpleName());
                ioc.put(beanName,instance);
            }else if(clazz.isAnnotationPresent(GPService.class)){
                //1、自定义的 beanName
                GPService service = clazz.getAnnotation(GPService.class);
                String beanName = service.value();
                //2、默认类名首字母小写
                if("").equals(beanName.trim())){
                    beanName = toLowerFirstCase(clazz.getSimpleName());
                }

                Object instance = clazz.newInstance();
                ioc.put(beanName,instance);
                //3、根据类型自动赋值,投机取巧的方式
                for (Class<?> i : clazz.getInterfaces()) {
                    if(ioc.containsKey(i.getName())){
                        throw new Exception("The "" + i.getName() + "" is exists!!");
                    }
                    //把接口的类型直接当成 key 了
                    ioc.put(i.getName(),instance);
                }
            }else {
                continue;
            }
        }
    }
}

```

```

    }
    }catch (Exception e){
        e.printStackTrace();
    }
}

```

为了处理方便，自己实现了 toLowerFirstCase 方法，来实现类名首字母小写，具体代码如下：

```

//如果类名本身是小写字母，确实会出问题
//但是我要说明的是：这个方法是我自己用，private 的
//传值也是自己传，类也都遵循了驼峰命名法
//默认传入的值，存在首字母小写的情况，也不可能出现非字母的情况

//为了简化程序逻辑，就不做其他判断了，大家了解就 OK
//其实写注释的时间都能够把逻辑写完了
private String toLowerFirstCase(String simpleName) {
    char [] chars = simpleName.toCharArray();
    //之所以加，是因为大小写字母的 ASCII 码相差 32，
    // 而且大写字母的 ASCII 码要小于小写字母的 ASCII 码
    //在 Java 中，对 char 做算学运算，实际上就是对 ASCII 码做算学运算
    chars[0] += 32;
    return String.valueOf(chars);
}

```

实现 doAutowired()方法：

```

//自动依赖注入
private void doAutowired() {
    if(ioc.isEmpty()){return;}

    for (Map.Entry<String, Object> entry : ioc.entrySet()) {
        //Declared 所有的，特定的 字段，包括 private/protected/default
        //正常来说，普通的 OOP 编程只能拿到 public 的属性
        Field[] fields = entry.getValue().getClass().getDeclaredFields();
        for (Field field : fields) {
            if(!field.isAnnotationPresent(GPAutowired.class)){continue;}
            GPAutowired autowired = field.getAnnotation(GPAutowired.class);

            //如果用户没有自定义 beanName，默认就根据类型注入
            //这个地方省去了对类名首字母小写的情况的判断，这个作为课后作业
            //小伙伴们自己去完善
            String beanName = autowired.value().trim();
            if("".equals(beanName)){
                //获得接口的类型，作为 key 待会拿这个 key 到 ioc 容器中去取值
                beanName = field.getType().getName();
            }
        }
    }
}

```

```

    }

    //如果是 public 以外的修饰符，只要加了@Autowired 注解，都要强制赋值
    //反射中叫做暴力访问， 强吻
    field.setAccessible(true);

    try {
        //用反射机制，动态给字段赋值
        field.set(entry.getValue(),ioc.get(beanName));
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    }

}

}

}

```

实现 initHandlerMapping()方法，handlerMapping 就是策略模式的应用案例：

```

//初始化 url 和 Method 的一对一对应关系
private void initHandlerMapping() {
    if(ioc.isEmpty()){ return; }

    for (Map.Entry<String, Object> entry : ioc.entrySet()) {
        Class<?> clazz = entry.getValue().getClass();

        if(!clazz.isAnnotationPresent(GPController.class)){continue;}

        //保存写在类上面的@GPRequestMapping("/demo")
        String baseUrl = "";
        if(clazz.isAnnotationPresent(GPRequestMapping.class)){
            GPRequestMapping requestMapping = clazz.getAnnotation(GPRequestMapping.class);
            baseUrl = requestMapping.value();
        }

        //默认获取所有的 public 方法
        for (Method method : clazz.getMethods()) {
            if(!method.isAnnotationPresent(GPRequestMapping.class)){continue;}

            GPRequestMapping requestMapping = method.getAnnotation(GPRequestMapping.class);

```

```

        //优化
        // //demo///query
        String url = ("/" + baseUrl + "/" + requestMapping.value())
            .replaceAll("/+", "/");
        handlerMapping.put(url, method);
        System.out.println("Mapped :" + url + "," + method);
    }

}

}

}

```

到这里位置初始化阶段就已经完成，接下实现运行阶段的逻辑，来看 doPost/doGet 的代码：

```

@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    this.doPost(req, resp);
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {

    //6、调用，运行阶段
    try {
        doDispatch(req, resp);
    } catch (Exception e) {
        e.printStackTrace();
        resp.getWriter().write("500 Exection,Detail : " + Arrays.toString(e.getStackTrace()));
    }
}
}

```

doPost()方法中，用了委派模式，委派模式的具体逻辑在 doDispatch()方法中：

```

private void doDispatch(HttpServletRequest req, HttpServletResponse resp) throws Exception {
    String url = req.getRequestURI();
    String contextPath = req.getContextPath();
    url = url.replaceAll(contextPath, "").replaceAll("/+", "/");
    if(!this.handlerMapping.containsKey(url)){
        resp.getWriter().write("404 Not Found!!");
        return;
    }
}

```

```

Method method = this.handlerMapping.get(url);
//第一个参数：方法所在的实例
//第二个参数：调用时所需要的实参

Map<String,String[]> params = req.getParameterMap();
//投机取巧的方式
String beanName = toLowerFirstCase(method.getDeclaringClass().getSimpleName());
method.invoke(ioc.get(beanName),new Object[]{req,resp,params.get("name")[0]});
//System.out.println(method);
}

```

在以上代码中，doDispatch()虽然完成了动态委派并反射调用，但对 url 参数处理还是静态代码。要实现 url 参数的动态获取，其实还稍微有些复杂。我们可以优化 doDispatch()方法的实现逻辑，代码如下：

```

private void doDispatch(HttpServletRequest req, HttpServletResponse resp) throws Exception {
    //绝对路径
    String url = req.getRequestURI();
    //处理成相对路径
    String contextPath = req.getContextPath();
    url = url.replaceAll(contextPath,"").replaceAll("/+","/");

    if(!this.handlerMapping.containsKey(url)){
        resp.getWriter().write("404 Not Found!!!");
        return;
    }

    Method method = this.handlerMapping.get(url);

    //从 request 中拿到 url 传过来的参数
    Map<String,String[]> params = req.getParameterMap();

    //获取方法的形参列表
    Class<?> [] parameterTypes = method.getParameterTypes();

    Object [] paramValues = new Object[parameterTypes.length];

    for (int i = 0; i < parameterTypes.length; i++) {
        Class parameterType = parameterTypes[i];
        //不能用 instanceof, parameterType 它不是实参，而是形参
        if(parameterType == HttpServletRequest.class){
            paramValues[i] = req;
            continue;
        }
    }
}

```

```

    }else if(parameterType == HttpServletResponse.class){
        paramValues[i] = resp;
        continue;
    }else if(parameterType == String.class){
        GPRequestParam requestParam =
(GPRequestParam)parameterType.getAnnotation(GPRequestParam.class);
        if(params.containsKey(requestParam.value())) {
            for (Map.Entry<String,String[]> param : params.entrySet()){
                String value = Arrays.toString(param.getValue())
                    .replaceAll("\\[|\\]", "")
                    .replaceAll("\\s", ",");
                paramValues[i] = value;
            }
        }
    }
}

//投机取巧的方式
//通过反射拿到 method 所在 class，拿到 class 之后还是拿到 class 的名称
//再调用 toLowerFirstCase 获得 beanName
String beanName = toLowerFirstCase(method.getDeclaringClass().getSimpleName());
method.invoke(ioc.get(beanName),paramValues);
}

//url 传过来的参数都是 String 类型的，HTTP 是基于字符串协议
//只需要把 String 转换为任意类型就好
private Object convert(Class<?> type,String value){
    //如果是 int
    if(Integer.class == type){
        return Integer.valueOf(value);
    }
    //如果还有 double 或者其他类型，继续加 if
    //这时候，我们应该想到策略模式了
    //在这里暂时不实现，希望小伙伴自己来实现
    return value;
}

```

## 实现 V3 版本

在 V2 版本中，基本功能以及完全实现，但代码的优雅程度还不如人意。譬如 HandlerMapping 还不能像 SpringMVC 一样支持正则，url 参数还不支持强制类型转换，在反射调用前还需要重新获取 beanName，在 V3 版本中，下面我们



继续优化。

首先，改造 HandlerMapping，在真实的 Spring 源码中，HandlerMapping 其实是一个 List 而非 Map。List 中的元素是一个自定义的类型。现在我们来仿真写一段代码，先定义一个内部类 Handler 类：

```
/**
 * Handler 记录 Controller 中的 RequestMapping 和 Method 的对应关系
 * @author Tom
 * 内部类
 */
private class Handler{
    protected Object controller;    //保存方法对应的实例
    protected Method method;        //保存映射的方法
    protected Pattern pattern;
    protected Map<String,Integer> paramIndexMapping;    //参数顺序
    /**
     * 构造一个Handler 基本的参数
     * @param controller
     * @param method
     */
    protected Handler(Pattern pattern,Object controller,Method method){
        this.controller = controller;
        this.method = method;
        this.pattern = pattern;
        paramIndexMapping = new HashMap<String,Integer>();
        putParamIndexMapping(method);
    }
    private void putParamIndexMapping(Method method){
        //提取方法中加了注解的参数
        Annotation [] [] pa = method.getParameterAnnotations();
        for (int i = 0; i < pa.length ; i ++) {
            for(Annotation a : pa[i]){
                if(a instanceof GPRequestParam){
                    String paramName = ((GPRequestParam) a).value();
                    if(!"".equals(paramName.trim())){
                        paramIndexMapping.put(paramName, i);
                    }
                }
            }
        }
    }
    //提取方法中的 request 和 response 参数
    Class<?> [] paramsTypes = method.getParameterTypes();
    for (int i = 0; i < paramsTypes.length ; i ++) {
```

```

        Class<?> type = paramsTypes[i];
        if(type == HttpServletRequest.class ||
            type == HttpServletResponse.class){
            paramIndexMapping.put(type.getName(),i);
        }
    }
}
}
}
}

```

然后，优化 HandlerMapping 的结构，代码如下：

```

//保存所有的 Url 和方法的映射关系
private List<Handler> handlerMapping = new ArrayList<Handler>();

```

修改 initHandlerMapping()方法：

```

private void initHandlerMapping(){
    if(ioc.isEmpty()){ return; }
    for (Entry<String, Object> entry : ioc.entrySet()) {
        Class<?> clazz = entry.getValue().getClass();
        if(!clazz.isAnnotationPresent(GPController.class)){ continue; }
        String url = "";
        //获取 Controller 的 url 配置
        if(clazz.isAnnotationPresent(GPRequestMapping.class)){
            GPRequestMapping requestMapping = clazz.getAnnotation(GPRequestMapping.class);
            url = requestMapping.value();
        }
        //获取 Method 的 url 配置
        Method [] methods = clazz.getMethods();
        for (Method method : methods) {
            //没有加 RequestMapping 注解的直接忽略
            if(!method.isAnnotationPresent(GPRequestMapping.class)){ continue; }
            //映射 URL
            GPRequestMapping requestMapping = method.getAnnotation(GPRequestMapping.class);
            String regex = ("/" + url + requestMapping.value()).replaceAll("/+", "/");
            Pattern pattern = Pattern.compile(regex);
            handlerMapping.add(new Handler(pattern,entry.getValue(),method));
            System.out.println("mapping " + regex + "," + method);
        }
    }
}

```

修改 doDispatch()方法：

```

/**
 * 匹配 URL

```

```

* @param req
* @param resp
* @return
* @throws Exception
*/
private void doDispatch(HttpServletRequest req, HttpServletResponse resp) throws Exception {

    Handler handler = getHandler(req);
    if(handler == null){
//        if(!this.handlerMapping.containsKey(url)){
//            resp.getWriter().write("404 Not Found!!!");
//            return;
//        }

//获得方法的形参列表
Class<?> [] paramTypes = handler.getParamTypes();

Object [] paramValues = new Object[paramTypes.length];

Map<String,String[]> params = req.getParameterMap();
for (Map.Entry<String, String[]> parm : params.entrySet()) {
    String value = Arrays.toString(parm.getValue()).replaceAll("\\[|\\]", "")
        .replaceAll("\\s", ",");

    if(!handler.paramIndexMapping.containsKey(parm.getKey())){continue;}

    int index = handler.paramIndexMapping.get(parm.getKey());
    paramValues[index] = convert(paramTypes[index],value);
}

if(handler.paramIndexMapping.containsKey(HttpServletRequest.class.getName())) {
    int reqIndex = handler.paramIndexMapping.get(HttpServletRequest.class.getName());
    paramValues[reqIndex] = req;
}

if(handler.paramIndexMapping.containsKey(HttpServletResponse.class.getName())) {
    int respIndex = handler.paramIndexMapping.get(HttpServletResponse.class.getName());
    paramValues[respIndex] = resp;
}

Object returnValue = handler.method.invoke(handler.controller,paramValues);
if(returnValue == null || returnValue instanceof Void){ return; }
resp.getWriter().write(returnValue.toString());
}

```

```
private Handler getHandler(HttpServletRequest req) throws Exception{
    if(handlerMapping.isEmpty()){ return null; }
    String url = req.getRequestURI();
    String contextPath = req.getContextPath();
    url = url.replace(contextPath, "").replaceAll("/+", "/");
    for (Handler handler : handlerMapping) {
        try{
            Matcher matcher = handler.pattern.matcher(url);
            //如果没有匹配上继续下一个匹配
            if(!matcher.matches()){ continue; }
            return handler;
        }catch(Exception e){
            throw e;
        }
    }
    return null;
}

private Object convert(Class<?> type,String value){
    if(Integer.class == type){
        return Integer.valueOf(value);
    }
    return value;
}
```

在以上代码中，增加了两个方法，一个是 getHandler()方法，主要负责处理 url 的正则匹配；一个是 convert()方法，主要负责 url 参数的强制类型转换。

至此，手写 Mini 版 SpringMVC 框架就已全部完成。

## 运行效果演示

在浏览器输入：<http://localhost:8080/demo/query.json?name=Tom>，就会得到下面的结果：



# My name is Tom

当然，真正的 Spring 要复杂很多，本课中主要通过手写的形式，了解 Spring 的基本设计思路以及设计模式如何应用，在以后的课程中，我们还会继续手写更加高仿真版本的 Spring2.0。