课程目标

1、高仿真手写 Spring MVC 部分，充分实践设计模式。

2、用 30 个类搭建基本框架，满足核心功能。

内容定位

在完全掌握 Spring 系统结构、实现原理，在理解设计模式的基础上，自己动手写一个高仿真版本的 Spring 框架，以达到透彻理解 Spring 的目的，感受作者创作意图。

MVC 顶层设计

GPDispatcherServlet

```
package com.gupaoedu.vip.spring.formework.webmvc.servlet;

import com.gupaoedu.vip.spring.formework.annotation.GPController;
import com.gupaoedu.vip.spring.formework.annotation.GPRequestMapping;
import com.gupaoedu.vip.spring.formework.context.GPApplicationContext;
import com.gupaoedu.vip.spring.formework.webmvc.*;
import lombok.extern.slf4j.Slf4j;

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.lang.reflect.Method;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
```

```
 * Created by Tom.
 */
//Servlet 只是作为一个 MVC 的启动入口
@Slf4j
public class GPDispatcherServlet extends HttpServlet {

    private  final String LOCATION = "contextConfigLocation";

    //课后再去思考一下这样设计的经典之处
    //GPHandlerMapping 最核心的设计，也是最经典的
    //它牛 B 到直接干掉了 Struts、Webwork 等 MVC 框架
    private List<GPHandlerMapping> handlerMappings = new ArrayList<GPHandlerMapping>();

    private Map<GPHandlerMapping,GPHandlerAdapter> handlerAdapters = new HashMap<GPHandlerMapping,
GPHandlerAdapter>();

    private List<GPViewResolver> viewResolvers = new ArrayList<GPViewResolver>();

    private GPApplicationContext context;

    @Override
    public void init(ServletConfig config) throws ServletException {
        //相当于把 IOC 容器初始化了
        context = new GPApplicationContext(config.getInitParameter(LOCATION));
        initStrategies(context);
    }

    protected void initStrategies(GPApplicationContext context) {

        //有九种策略
        // 针对于每个用户请求，都会经过一些处理的策略之后，最终才能有结果输出
        // 每种策略可以自定义干预，但是最终的结果都是一致

        // =============  这里说的就是传说中的九大组件 ================
        initMultipartResolver(context);//文件上传解析，如果请求类型是 multipart 将通过
MultipartResolver 进行文件上传解析
        initLocaleResolver(context);//本地化解析
        initThemeResolver(context);//主题解析

        /** 我们自己会实现 */
        //GPHandlerMapping 用来保存 Controller 中配置的 RequestMapping 和 Method 的一个对应关系
        initHandlerMappings(context);//通过 HandlerMapping，将请求映射到处理器
        /** 我们自己会实现 */
        //HandlerAdapters 用来动态匹配 Method 参数，包括类转换，动态赋值
```

```
        initHandlerAdapters(context);//通过 HandlerAdapter 进行多类型的参数动态匹配

        initHandlerExceptionResolvers(context);//如果执行过程中遇到异常，将交给
HandlerExceptionResolver 来解析
        initRequestToViewNameTranslator(context);//直接解析请求到视图名

        /** 我们自己会实现 */
        //通过 ViewResolvers 实现动态模板的解析
        //自己解析一套模板语言
        initViewResolvers(context);//通过 viewResolver 解析逻辑视图到具体视图实现

        initFlashMapManager(context);//flash 映射管理器
    }

    private void initFlashMapManager(GPApplicationContext context) {}
    private void initRequestToViewNameTranslator(GPApplicationContext context) {}
    private void initHandlerExceptionResolvers(GPApplicationContext context) {}
    private void initThemeResolver(GPApplicationContext context) {}
    private void initLocaleResolver(GPApplicationContext context) {}
    private void initMultipartResolver(GPApplicationContext context) {}




    //将 Controller 中配置的 RequestMapping 和 Method 进行一一对应
    private void initHandlerMappings(GPApplicationContext context) {
        //按照我们通常的理解应该是一个 Map
        //Map<String,Method> map;
        //map.put(url,Method)

        //首先从容器中取到所有的实例
        String [] beanNames = context.getBeanDefinitionNames();
        try {
            for (String beanName : beanNames) {
                //到了 MVC 层，对外提供的方法只有一个 getBean 方法
                //返回的对象不是 BeanWrapper，怎么办？
                Object controller = context.getBean(beanName);
                //Object controller = GPAopUtils.getTargetObject(proxy);
                Class<?> clazz = controller.getClass();

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

```
                String baseUrl = "";

                if (clazz.isAnnotationPresent(GPRequestMapping.class)) {
                    GPRequestMapping requestMapping = clazz.getAnnotation(GPRequestMapping.class);
                    baseUrl = requestMapping.value();
                }

                //扫描所有的 public 方法
                Method[] methods = clazz.getMethods();
                for (Method method : methods) {
                    if (!method.isAnnotationPresent(GPRequestMapping.class)) {
                        continue;
                    }

                    GPRequestMapping requestMapping = method.getAnnotation(GPRequestMapping.class);
                    String regex = ("/" + baseUrl + requestMapping.value().replaceAll("\\*",
".*")).replaceAll("/+", "/");
                    Pattern pattern = Pattern.compile(regex);
                    this.handlerMappings.add(new GPHandlerMapping(pattern, controller, method));
                    log.info("Mapping: " + regex + " , " + method);

                }

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


    }

    private void initHandlerAdapters(GPApplicationContext context) {
        //在初始化阶段，我们能做的就是，将这些参数的名字或者类型按一定的顺序保存下来
        //因为后面用反射调用的时候，传的形参是一个数组
        //可以通过记录这些参数的位置 index,挨个从数组中填值，这样的话，就和参数的顺序无关了
        for (GPHandlerMapping handlerMapping : this.handlerMappings){
//            //每一个方法有一个参数列表，那么这里保存的是形参列表
            this.handlerAdapters.put(handlerMapping,new GPHandlerAdapter());
        }

    }

    private void initViewResolvers(GPApplicationContext context) {
        //在页面敲一个 http://localhost/first.html
```

```
        //解决页面名字和模板文件关联的问题
        String templateRoot = context.getConfig().getProperty("templateRoot");
        String templateRootPath =
this.getClass().getClassLoader().getResource(templateRoot).getFile();

        File templateRootDir = new File(templateRootPath);

        for (File template : templateRootDir.listFiles()) {
            this.viewResolvers.add(new GPViewResolver(templateRoot));
        }

    }



    @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("<font size='25' color='blue'>500
Exception</font><br/>Details:<br/>" + Arrays.toString(e.getStackTrace()).replaceAll("\\[|\\]","")
                    .replaceAll("\\s","\r\n") +  "<font
color='green'><i>Copyright@GupaoEDU</i></font>");
            e.printStackTrace();
        }
    }

    private void doDispatch(HttpServletRequest req, HttpServletResponse resp) throws  Exception{

        //根据用户请求的 URL 来获得一个 Handler
        GPHandlerMapping handler = getHandler(req);
        if(handler == null){
            processDispatchResult(req,resp,new GPModelAndView("404"));
            return;
        }
```

```
        GPHandlerAdapter ha = getHandlerAdapter(handler);

        //这一步只是调用方法，得到返回值
        GPModelAndView mv = ha.handle(req, resp, handler);

        //这一步才是真的输出
        processDispatchResult(req,resp, mv);

    }

    private void processDispatchResult(HttpServletRequest request,HttpServletResponse response,
GPModelAndView mv) throws Exception {
        //调用 viewResolver 的 resolveView 方法
        if(null == mv){ return;}

        if(this.viewResolvers.isEmpty()){ return;}

        if (this.viewResolvers != null) {
            for (GPViewResolver viewResolver : this.viewResolvers) {
                GPView view = viewResolver.resolveViewName(mv.getViewName(), null);
                if (view != null) {
                    view.render(mv.getModel(),request,response);
                    return;
                }
            }
        }

    }

    private GPHandlerAdapter getHandlerAdapter(GPHandlerMapping handler) {
        if(this.handlerAdapters.isEmpty()){return  null;}
        GPHandlerAdapter ha = this.handlerAdapters.get(handler);
        if (ha.supports(handler)) {
            return ha;
        }
        return null;
    }

    private GPHandlerMapping getHandler(HttpServletRequest req) {

        if(this.handlerMappings.isEmpty()){ return  null;}

        String url = req.getRequestURI();
        String contextPath = req.getContextPath();
```

```
        url = url.replace(contextPath,"").replaceAll("/+","/");

        for (GPHandlerMapping handler : this.handlerMappings) {
            Matcher matcher = handler.getPattern().matcher(url);
            if(!matcher.matches()){ continue;}
            return handler;
        }

        return null;
    }
}
```

GPHandlerMapping

```
package com.gupaoedu.vip.spring.formework.webmvc;

import java.lang.reflect.Method;
import java.util.regex.Pattern;

/**
 * Created by Tom.
 */
public class GPHandlerMapping {
    private Object controller;
    private Method method;
    private Pattern pattern;  //url 的封装

    public GPHandlerMapping(Pattern pattern,Object controller, Method method) {
        this.controller = controller;
        this.method = method;
        this.pattern = pattern;
    }

    public Object getController() {
        return controller;
    }

    public void setController(Object controller) {
        this.controller = controller;
    }

    public Method getMethod() {
        return method;
    }
```

```
    public void setMethod(Method method) {
        this.method = method;
    }

    public Pattern getPattern() {
        return pattern;
    }

    public void setPattern(Pattern pattern) {
        this.pattern = pattern;
    }
}
```

GPHandlerAdapter

```
package com.gupaoedu.vip.spring.formework.webmvc;

import com.gupaoedu.vip.spring.formework.annotation.GPRequestParam;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.lang.annotation.Annotation;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

/**
 * Created by Tom.
 */
//专人干专事
public class GPHandlerAdapter {

    public boolean supports(Object handler){
        return (handler instanceof GPHandlerMapping);
    }

    public GPModelAndView handle(HttpServletRequest req, HttpServletResponse resp, Object handler)
throws Exception{
        GPHandlerMapping handlerMapping = (GPHandlerMapping)handler;

        //每一个方法有一个参数列表，那么这里保存的是形参列表
        Map<String,Integer> paramMapping = new HashMap<String, Integer>();

        //这里只是出来了命名参数
        Annotation[][] pa = handlerMapping.getMethod().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())){
                        paramMapping.put(paramName,i);
                    }
                }
            }
        }

        //根据用户请求的参数信息，跟 method 中的参数信息进行动态匹配
        //resp 传进来的目的只有一个：只是为了将其赋值给方法参数，仅此而已

        //只有当用户传过来的 ModelAndView 为空的时候，才会 new 一个默认的

        //1、要准备好这个方法的形参列表
        //方法重载：形参的决定因素：参数的个数、参数的类型、参数顺序、方法的名字
        //只处理 Request 和 Response
        Class<?>[] paramTypes = handlerMapping.getMethod().getParameterTypes();
        for (int i = 0;i < paramTypes.length; i ++) {
            Class<?> type = paramTypes[i];
            if(type == HttpServletRequest.class ||
                    type == HttpServletResponse.class){
                paramMapping.put(type.getName(),i);
            }
        }



        //2、拿到自定义命名参数所在的位置
        //用户通过 URL 传过来的参数列表
        Map<String,String[]> reqParameterMap = req.getParameterMap();

        //3、构造实参列表
        Object [] paramValues = new Object[paramTypes.length];

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

            if(!paramMapping.containsKey(param.getKey())){continue;}

            int index = paramMapping.get(param.getKey());
```

```
            //因为页面上传过来的值都是 String 类型的，而在方法中定义的类型是千变万化的
            //要针对我们传过来的参数进行类型转换
            paramValues[index] = caseStringValue(value,paramTypes[index]);
        }

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

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

        //4、从 handler 中取出 controller、method，然后利用反射机制进行调用

        Object result =
handlerMapping.getMethod().invoke(handlerMapping.getController(),paramValues);

        if(result == null){ return  null; }

        boolean isModelAndView = handlerMapping.getMethod().getReturnType() ==
GPModelAndView.class;
        if(isModelAndView){
            return (GPModelAndView)result;
        }else{
            return null;
        }
    }

    private Object caseStringValue(String value,Class<?> clazz){
        if(clazz == String.class){
            return value;
        }else if(clazz == Integer.class){
            return  Integer.valueOf(value);
        }else if(clazz == int.class){
            return Integer.valueOf(value).intValue();
        }else {
            return null;
        }
    }
```

```
}
```

GPModelAndView

```
package com.gupaoedu.vip.spring.formework.webmvc;

import java.util.Map;

/**
 * Created by Tom.
 */
public class GPModelAndView {

    private String viewName;
    private Map<String,?> model;

    public GPModelAndView(String viewName) {
        this(viewName,null);
    }
    public GPModelAndView(String viewName, Map<String, ?> model) {
        this.viewName = viewName;
        this.model = model;
    }

    public String getViewName() {
        return viewName;
    }

    public void setViewName(String viewName) {
        this.viewName = viewName;
    }

    public Map<String, ?> getModel() {
        return model;
    }

    public void setModel(Map<String, ?> model) {
        this.model = model;
    }
}
```

GPViewResolver

```
package com.gupaoedu.vip.spring.formework.webmvc;
```

```
import java.io.File;

import java.util.Locale;

/**
 * Created by Tom.
 */

//设计这个类的主要目的是：
//1、讲一个静态文件变为一个动态文件
//2、根据用户传送参数不同，产生不同的结果
//最终输出字符串，交给 Response 输出
public class GPViewResolver {
    private final String DEFAULT_TEMPLATE_SUFFIX = ".html";

    private File templateRootDir;
    private String viewName;

    public GPViewResolver(String templateRoot){
        String templateRootPath =
this.getClass().getClassLoader().getResource(templateRoot).getFile();
        this.templateRootDir = new File(templateRootPath);
    }

    public GPView resolveViewName(String viewName, Locale locale) throws Exception {
        this.viewName = viewName;
        if(null == viewName || "".equals(viewName.trim())){ return null;}
        viewName = viewName.endsWith(DEFAULT_TEMPLATE_SUFFIX) ? viewName : (viewName +
DEFAULT_TEMPLATE_SUFFIX);
        File templateFile = new File((templateRootDir.getPath() + "/" + viewName).replaceAll("/+",
"/"));
        return new GPView(templateFile);
    }

    public String getViewName() {
        return viewName;
    }
}
```

GPView

```
package com.gupaoedu.vip.spring.formework.webmvc;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import java.io.RandomAccessFile;
import java.util.Map;
import java.io.File;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Created by Tom.
 */
public class GPView {

    public static final String DEFAULT_CONTENT_TYPE = "text/html;charset=utf-8";

    private File viewFile;

    public GPView(File viewFile){
        this.viewFile = viewFile;
    }

    public String getContentType(){
        return DEFAULT_CONTENT_TYPE;
    }

    public void render(Map<String, ?> model,HttpServletRequest request, HttpServletResponse
response) throws Exception{
        StringBuffer sb = new StringBuffer();
        RandomAccessFile ra = new RandomAccessFile(this.viewFile,"r");


        try {
            String line = null;
            while (null != (line = ra.readLine())) {
                line = new String(line.getBytes("ISO-8859-1"),"utf-8");
                Pattern pattern = Pattern.compile("￥\\{[^\\}]+\\}",Pattern.CASE_INSENSITIVE);
                Matcher matcher = pattern.matcher(line);

                while (matcher.find()) {

                    String paramName = matcher.group();
                    paramName = paramName.replaceAll("￥\\{|\\}","");
                    Object paramValue = model.get(paramName);
                    if (null == paramValue) { continue; }
                    //要把￥{}中间的这个字符串给取出来
                    line = matcher.replaceFirst(makeStringForRegExp(paramValue.toString()));
```

```
                matcher = pattern.matcher(line);

            }

            sb.append(line);
        }
    }finally {
        ra.close();
    }
    response.setCharacterEncoding("utf-8");
    //response.setContentType(DEFAULT_CONTENT_TYPE);
    response.getWriter().write(sb.toString());
}

//处理特殊字符
public static String makeStringForRegExp(String str) {
     return str.replace("\\", "\\\\").replace("*", "\\*")
    .replace("+", "\\+").replace("|", "\\|")
    .replace("{", "\\{").replace("}", "\\}")
    .replace("(", "\\(").replace(")", "\\)")
    .replace("^", "\\^").replace("$", "\\$")
    .replace("[", "\\[").replace("]", "\\]")
    .replace("?", "\\?").replace(",", "\\,")
    .replace(".", "\\.").replace("&", "\\&");
}

}
```

业务代码实现

IQueryService

```
package com.gupaoedu.vip.spring.demo.service;

/**
 * 查询业务
 * @author Tom
 *
 */
public interface IQueryService {
   /**
    * 查询
    */
   public String query(String name);
```

```
}
```

QueryService

```
package com.gupaoedu.vip.spring.demo.service.impl;

import java.text.SimpleDateFormat;
import java.util.Date;

import com.gupaoedu.vip.spring.demo.service.IQueryService;
import com.gupaoedu.vip.spring.formework.annotation.GPService;
import lombok.extern.slf4j.Slf4j;

/**
 * 查询业务
 * @author Tom
 *
 */
@GPService
@Slf4j
public class QueryService implements IQueryService {

   /**
    * 查询
    */
   public String query(String name) {
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
      String time = sdf.format(new Date());
      String json = "{name:\"" + name + "\",time:\"" + time + "\"}";
      log.info("这是在业务方法中打印的：" + json);
      return json;
   }

}
```

IModifyService

```
package com.gupaoedu.vip.spring.demo.service;
/**
 * 增删改业务
 * @author Tom
 *
 */
public interface IModifyService {
   /**
    * 增加
```

```
    */
   public String add(String name, String addr) ;
   /**
    * 修改
    */
   public String edit(Integer id, String name);
   /**
    * 删除
    */
   public String remove(Integer id);
}
```

ModifyService

```
package com.gupaoedu.vip.spring.demo.service.impl;
import com.gupaoedu.vip.spring.demo.service.IModifyService;
import com.gupaoedu.vip.spring.formework.annotation.GPService;
/**
 * 增删改业务
 * @author Tom
 *
 */
@GPService
public class ModifyService implements IModifyService {
   /**
    * 增加
    */
   public String add(String name,String addr) {
      return "modifyService add,name=" + name + ",addr=" + addr;
   }
   /**
    * 修改
    */
   public String edit(Integer id,String name) {
      return "modifyService edit,id=" + id + ",name=" + name;
   }
   /**
    * 删除
    */
   public String remove(Integer id) {
      return "modifyService id=" + id;
   }
}
```

MyAction

```java
package com.gupaoedu.vip.spring.demo.action;

import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.gupaoedu.vip.spring.demo.service.IModifyService;
import com.gupaoedu.vip.spring.demo.service.IQueryService;
import com.gupaoedu.vip.spring.formework.annotation.GPAutowired;
import com.gupaoedu.vip.spring.formework.annotation.GPController;
import com.gupaoedu.vip.spring.formework.annotation.GPRequestMapping;
import com.gupaoedu.vip.spring.formework.annotation.GPRequestParam;
import com.gupaoedu.vip.spring.formework.webmvc.GPModelAndView;

/**
 * 公布接口url
 * @author Tom
 *
 */
@GPController
@GPRequestMapping("/web")
public class MyAction {

   @GPAutowired IQueryService queryService;
   @GPAutowired IModifyService modifyService;

   @GPRequestMapping("/query.json")
   public GPModelAndView query(HttpServletRequest request, HttpServletResponse response,
                     @GPRequestParam("name") String name){
      String result = queryService.query(name);
      return out(response,result);
   }
   @GPRequestMapping("/add*.json")
   public GPModelAndView add(HttpServletRequest request,HttpServletResponse response,
           @GPRequestParam("name") String name,@GPRequestParam("addr") String addr){
      String result = modifyService.add(name,addr);
      return out(response,result);
   }
   @GPRequestMapping("/remove.json")
   public GPModelAndView remove(HttpServletRequest request,HttpServletResponse response,
         @GPRequestParam("id") Integer id){
      String result = modifyService.remove(id);
      return out(response,result);
   }
   @GPRequestMapping("/edit.json")
```

```
    public GPModelAndView edit(HttpServletRequest request,HttpServletResponse response,
          @GPRequestParam("id") Integer id,
          @GPRequestParam("name") String name){
       String result = modifyService.edit(id,name);
       return out(response,result);
    }

    private GPModelAndView out(HttpServletResponse resp,String str){
       try {
          resp.getWriter().write(str);
       } catch (IOException e) {
          e.printStackTrace();
       }
       return null;
    }
}
```

PageAction

```
package com.gupaoedu.vip.spring.demo.action;

import java.util.HashMap;
import java.util.Map;
import com.gupaoedu.vip.spring.demo.service.IQueryService;
import com.gupaoedu.vip.spring.formework.annotation.GPAutowired;
import com.gupaoedu.vip.spring.formework.annotation.GPController;
import com.gupaoedu.vip.spring.formework.annotation.GPRequestMapping;
import com.gupaoedu.vip.spring.formework.annotation.GPRequestParam;
import com.gupaoedu.vip.spring.formework.webmvc.GPModelAndView;

/**
 * 公布接口url
 * @author Tom
 */
@GPController
@GPRequestMapping("/")
public class PageAction {

   @GPAutowired IQueryService queryService;

   @GPRequestMapping("/first.html")
   public GPModelAndView query(@GPRequestParam("teacher") String teacher){
      String result = queryService.query(teacher);
      Map<String,Object> model = new HashMap<String,Object>();
      model.put("teacher", teacher);
```

```
        model.put("data", result);
        model.put("token", "123456");
        return new GPModelAndView("first.html",model);
    }

}
```

定制模板页面

first.html

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <title>咕泡学院 SpringMVC 模板引擎演示</title>
</head>
<center>
    <h1>大家好，我是￥{teacher}老师<br/>欢迎大家一起来探索 Spring 的世界</h1>
    <h3>Hello,My name is ￥{teacher}</h3>
    <div>￥{data}</div>
    Token 值：￥{token}
</center>
</html>
```

404.html

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <title>页面去火星了</title>
</head>
<body>
    <font size='25' color='red'>404 Not Found</font><br/><font
color='green'><i>Copyright@GupaoEDU</i></font>
</body>
</html>
```

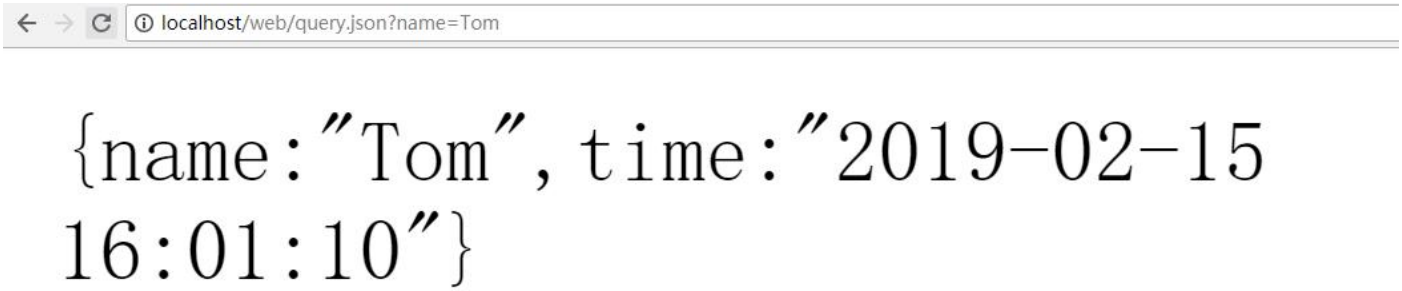
500.html

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <title>服务器好像累了</title>
```

```
</head>
<body>
    <font size='25' color='blue'>500 服务器好像有点累了，需要休息一下</font><br/>
    <b>Message:￥{detail}</b><br/>
    <b>StackTrace:￥{stackTrace}</b><br/>
    <font color='green'><i>Copyright@GupaoEDU</i></font>
</body>
</html>
```

运行效果演示

/query.json



/add*.json

/remove.json

/edit.json

modifyService edit,id=666,name=Tom

/first.html

大家好，我是Tom老师
欢迎大家一起来探索Spring的世界

Hello,My name is Tom

{name:"Tom",time:"2019-02-15 16:05:03"}
Token值：123456