Design report

1. 项目描述

使用 restful 风格，较直观的体现如下：

1.1 视图都由 ModelAndView 进行解析

请求处理方法完成后，最终总会返回一个 ModelAndView 对象，而后借助视图解析器 ViewResolver 得到最终的视图

```
    }
    ModelAndView mav = new ModelAndView( viewName: "courses");
    mav.addObject( attributeName: "courselist",cache_course(loged));
    return mav;
}
```

返回 ModelAndView 对象

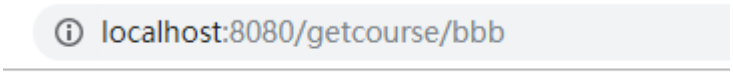
```
<mvc:annotation-driven/>

<!--ViewResolver 视图解析器-->
<!--用于支持Servlet、JSP视图解析-->
<bean id="jspViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/pages/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

视图解析器

1.2 URL 风格

localhost:8080/springmvc?name=小白&password=123;

未使用 restful 风格



使用 restful 风格

在项目中的体现（Controller 中）

```
@RequestMapping(value="getcourse/{name}")
public ModelAndView login(@PathVariable("name") String loged){
```

其中用{ }来匹配动态参数，方法的形参上用@PathVariable(“name”)来匹配上面的参数

2. 速率限制

使用 guava 的 RateLimiter

2.1 服务端（打包在 tomcat 上）

把限流服务封装到一个类中，提供 tryacquire（）方法用于获取令牌，根据返回值表示获取结果（每秒放出 5 个令牌）

```
@Service
public class AccessLimitService {
    RateLimiter rateLimiter = RateLimiter.create(5.0);

    public boolean tryAcquire(){
        return rateLimiter.tryAcquire();
    }
}
```

尝试获取令牌的调用类中方法

```java
@Autowired
private AccessLimitService accessLimitService;

@RequestMapping("/access")
@ResponseBody
public String access(){
    //尝试获取令牌
    if(accessLimitService.tryAcquire()){
        //模拟业务执行500毫秒
        try {
            Thread.sleep( millis: 500);
        }catch (InterruptedException e){
            e.printStackTrace();
        }
        return "aceess success";
    }else{
        return "aceess limit";
    }
}
}
```

RatelimitController

2.2 客户端

具体方法：建立连接获取令牌并把获取到的令牌数量储存到一个 token 实体类中用于接下来的判断

```java
public int access() throws Exception{
    final URL url = new URL( spec: "http://localhost:8080/access");

    for(int i=0;i<10;i++) {
        fixedThreadPool.submit(new Runnable() {
            public void run() {
                if(sendGet(url).equals("aceess success")) {
                    get_token.setToken_num(get_token.getToken_num()+1);
                    System.out.println("hhhhhh"+get_token.getToken_num());
                }
                System.out.println(sendGet(url));
            }
        });
    }
```

```
public static String sendGet(URL realUrl) {
    String result = "";
    BufferedReader in = null;
    try {
        // 打开和URL之间的连接
        URLConnection connection = realUrl.openConnection();
        // 设置通用的请求属性
        connection.setRequestProperty("accept", "*/*");
        connection.setRequestProperty("connection", "Keep-Alive");
        connection.setRequestProperty("user-agent",
                "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");
        // 建立实际的连接
        connection.connect();

        // 定义 BufferedReader输入流来读取URL的响应
        in = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
        String line;
```

调用：在用户登录的时候进行验证，根据获取到的令牌数判断用户的登录请求是否被通过

```
AccessClient accessClient = new AccessClient(token);
try {
    number=accessClient.access();
} catch (Exception e) {
    e.printStackTrace();
}
```

3. 分页

由于数据量较小，处于交互次数尽可能少的考虑，采用客户端分页，客户端通过展现组件进行分页的控制

3.1 获取数据（默认第一页开始，每页展示 3 条数据）

```
public ModelAndView listUser(@RequestParam(value = "page",defaultValue = "1") Integer page,
                             @RequestParam(value = "size",defaultValue = "3") Integer size) {
    ModelAndView modelAndView = new ModelAndView( viewName: "list");
    PageRequest request = new PageRequest( page: page - 1, size);
    Page<CoachEntity> coachPage = coachRepository.findAll(request);
    List<CoachEntity> coachEntityList = coachPage.getContent();
    if(coachEntityList.isEmpty()) {
        System.out.println("coach list is null");
    } else {
        System.out.println(coachEntityList.get(0).getCoachName());
    }
    Page<CoachEntity> coachEntityPage = new PageImpl<>(coachEntityList, request, coachPage.getTotalElements());
    modelAndView.addObject( attributeName: "coachPage", coachEntityPage);
    return modelAndView;
}
```

3.2 展示数据

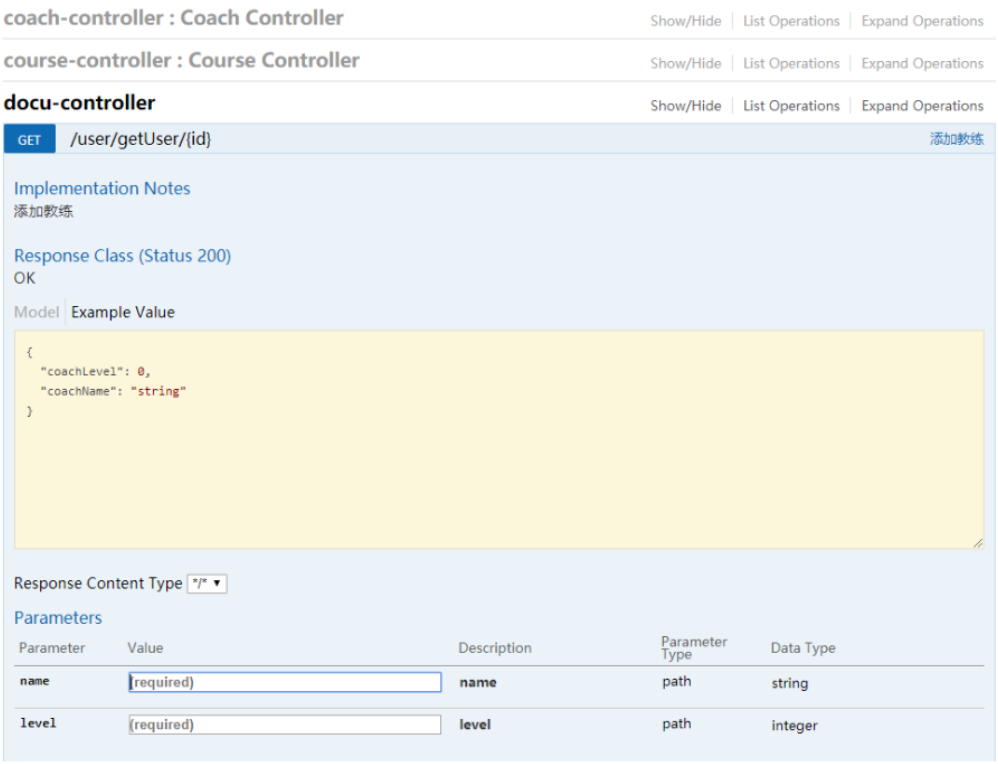
```
<table id="tableBody" border="0" cellspacing="0px" cellpadding="0">
    <tbody id="courseTbs">
    <c:forEach var="coach" items="${coached.content}" varStatus="status">
        <tr>
            <td>${coach.coachName}</td>
            <td>${coach.coachLevel}</td>
            <td style="">
```

3.3 跳转页面

```
<li>
        <a href="?page=${coached.number+2}" aria-label="Next">
            <span aria-hidden="true">»</span>
        </a>
</li>
```

4. API 文档

使用 swagger 框架实现



5. 缓存

5.1 cache

```
@Cacheable(value="cachedlist")
public List<CourseEntity> cache_course(String loged) {
    List<CourseEntity> list1=new ArrayList<>();
    list1=(List<CourseEntity>) courseRepository.findAll();
    for(CourseEntity test:list1) {
        if(!test.getUserName().equals(loged)) {
            list1.remove(test.getCourseId());
        }
    }
    return list1;
}
```

此注释的调用：

红框中即为从项目缓存中获取以该学员的 username 为 key 的缓存的值，结果即为其全部（高级）课程，用于在页面中直接显示而不需要再次访问数据库，

则加快了响应速度

```
    System.out.println("多表："+courseEntity.getCourseId()+" "+courseEntity.get(
}
ModelAndView mav = new ModelAndView( viewName: "courses");
mav.addObject( attributeName: "courselist",cache_course(loged));
return mav;
```

5.2 Http 缓存（LastModified）

```
@RestController
public class LinkController implements LastModified {
```

```
@Override
public long getLastModified(HttpServletRequest httpServletRequest) { return lastModified; }
```

实现 LastModified 接口，并重写 getLastModified()方法

```
@RequestMapping(value="getlinks")
public ModelAndView getlinks(WebRequest webRequest,HttpServletRequest reque
    HateoController hateoController=new HateoController();
    HttpEntity<Hateo> hateoHttpEntity= hateoController.greeting();
      System.out.println("LINKS: "+hateoHttpEntity.getBody().getLinks().get
    ArrayList<String> links=new ArrayList<~>();
    links.add(hateoHttpEntity.getBody().getLinks().get(0).getHref());
    links.add(hateoHttpEntity.getBody().getLinks().get(1).getHref());
    links.add(hateoHttpEntity.getBody().getLinks().get(2).getHref());

    System.out.println("start");
    if(webRequest.checkNotModified(lastModified)){
        System.out.println("check : "+lastModified);
        return null;
    }
    System.out.println("no check : "+lastModified);
    ModelAndView mav = new ModelAndView( viewName: "links");
    mav.addObject( attributeName: "linklist",links);
    return mav;
}
```

对一般的方法进行了适当修改，主动调用 checkNotModified()方法。可以看出第一次不会进入此方法，传回 ModelAndView 对象，输出“no check”+时间戳；若非第一次，则进入此方法，输出“check”+时间戳，同时返回 null,若没有缓存则无法正常跳转界面，实际正常实现了跳转，说明缓存实现了，输出截图如下：

```
course courseentity_
start
no check : 1557583302617
start
check : 1557583302617
```

6. Hateos

增加 link

```
ArrayList<String> links=new ArrayList<~>();
links.add(hateoHttpEntity.getBody().getLinks().get(0).getHref());
links.add(hateoHttpEntity.getBody().getLinks().get(1).getHref());
links.add(hateoHttpEntity.getBody().getLinks().get(2).getHref());
```

处理 link 将其置于页面用于跳转

跳转成功

Json 格式

links: [<http://localhost:8080>;rel="self", <http://localhost:8080/main>;rel="items", <www.baid