链滴

springboot 实战 - 一个简单的 springboot 项目

作者：someone9891

原文链接：https://ld246.com/article/1507908341981

来源网站：链滴

这两天学习 springboot，各种尝试，各种看文档，终于是跑起来几个 demo，在学习三天后，我在庆这一天完整了写了一个测试的 demo，虽然是一个测试项目，但是五脏六腑也差不了几样了。如果续有时间的话，还会再完善一下，把一般用到的组件都集成进去。

下面就对这个项目进行一下简单的介绍

说明

- 首先，这是一个 springboot 的入门学习 demo，不是什么完整的项目；
- 不过如果是拿来学习，还是有很多地方可以参考一下：
    - 日志的统一处理

代码中利用 spring **AOP** 的思想对日志进行了统一的处理，例子中只处理了 http 请求时的各种日志对于其他业务类的日志，参考依葫芦画瓢也不是什么难事

    - 异常统一处理

代码中用 **@ControllerAdvice** 进行了异常的统一处理，这个在实际项目中也是用途非常的广，一般目组中各组员能力参差不齐，所以对于异常、日志能够统一处理是非常好的，这里统一处理了异常，务代码就真的 “一切皆可抛” 了，无所顾忌的抛异常就可以了。

    - jpa 持久化

spring-data-jpa 是 spring 整合的 hibernate ，可以完全跟 hibernate 一样不用写 sql 语句，甚至连 xml 配置都不用写了

- 这个我可能会继续维护加东西进去，也可能不会……看时间吧，最近还是有很多东西要做，连国庆忙了一整天了。

构建

- 由于我之前已经写了一篇简单的构建 springboot 项目的帖子，今天就不重复写了，有需要了解的以移步这里：

springboot 入门搭建

- 而对于这个项目，我就不详细说明构建的经过了，相对来说还是比较痛苦的。
- 源码直接去　oschina获取就可以了

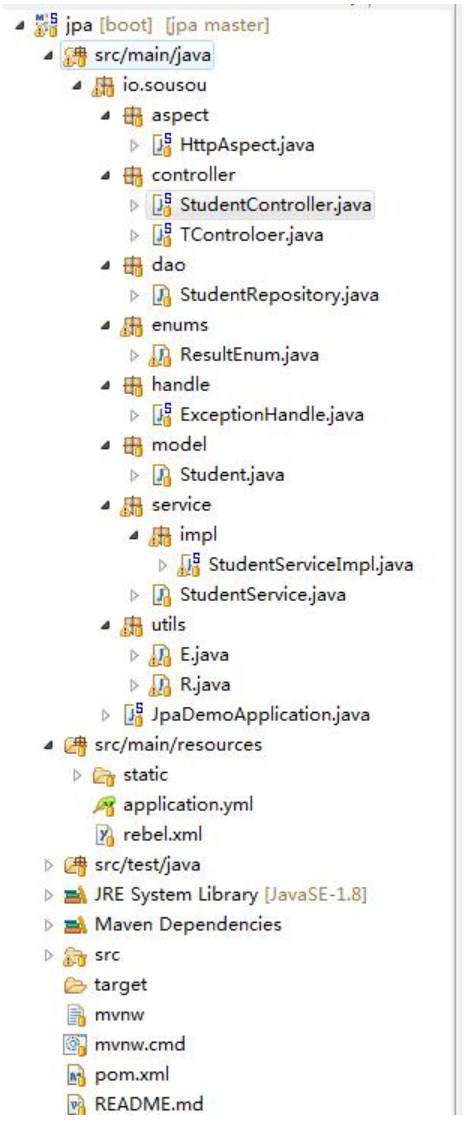
代码解析

项目结构

```
jap
|
├─src/main/java
|   ├─io.sousou.aspect 切面，统一日志处理
|   ├─io.sousou.controller 控制器
|   ├─io.sousou.dao 数据访问对象（持久化）
|   ├─io.sousou.enums  枚举
|   ├─io.sousou.handle  处理器
|   ├─io.sousou.model 实体对象
|   ├─io.sousou.service  service 接口
```

```
|      └─impl 接口实现类
|   └─io.sousou.enums   枚举
|
├─src/main/resources
|   ├─static  静态文件
|   ├─ * .yml 配置文件
|
├─src/main/test  单元测试
```

其他的就不列出来了，大家应该都知道。可以参照下图：



统一日志处理

废话不多说了，直接上代码

```
package io.sousou.aspect;

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

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

/**
 * 切面
 * 处理请求访问日志
 * @author allen
 * @website sousou.io
 *
 */
@Aspect
@Component
public class HttpAspect {
    private final static Logger logger = LoggerFactory.getLogger(HttpAspect.class);//日志类
    @Pointcut("execution(public * io.sousou.controller.*.*(..))")
    public void log(){
    }

    @Before("log()")
    public void doBefore(JoinPoint joinPoint){
        ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.
etRequestAttributes();
        HttpServletRequest request = attributes.getRequest();
        logger.info("------------------------start request-------------------------");
        logger.info("--url={}",request.getRequestURL());
        logger.info("--method_type={}",request.getMethod());
        logger.info("--class_method={}", joinPoint.getSignature().getDeclaringTypeName() + "."
 joinPoint.getSignature().getName());
        logger.info("--args={}", joinPoint.getArgs());
    }
    @After("log()")
    public void doAfter(){
        logger.info("------------------------ent request-------------------------");
    }

    @AfterReturning(returning = "object", pointcut = "log()")
    public void doAfterReturning(Object object) {
        logger.info("--response={}", object.toString());
        logger.info("-------------------------end response-------------------------");
    }

}
```

统一异常处理

异常处理中，我自己封装了一个异常类 E，平时自己项目中的异常可以手动抛，
做出不同的处理，而对于意料之外的异常，则统一捕获处理，前台返回为未知的异常，具体异常信息录日志。

```
package io.sousou.handle;

import io.sousou.utils.E;
import io.sousou.utils.R;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * @author allen
 * 2017-10-01 13:59
 */
@ControllerAdvice
public class ExceptionHandle {

    private final static Logger logger = LoggerFactory.getLogger(ExceptionHandle.class);

    @ExceptionHandler(value = Exception.class)
    @ResponseBody
    public R handle(Exception e) {
        if (e instanceof E) {
            E ex = (E) e;
            logger.info(ex.getMessage());
            return R.error(ex.getCode(), ex.getMessage());
        }else {
            logger.error("【系统异常】{}", e);
            return R.unknow_error();
        }
    }
}
```

服务端表单验证

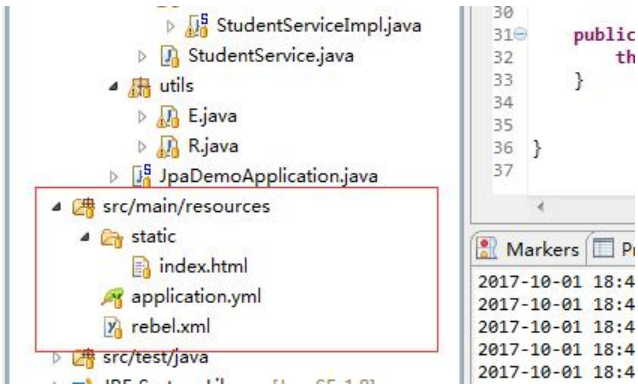
其实对于表单验证，很多时候都是交给前端去处理的，但是有时候根据业务，也是需要在服务端进行证的，
这面的类中只做了一个为空验证，就是在属性上加 @NotBlack 注解。然后再controller中进行判断可以了。
具体请看代码，稍后我会加多点注释进去

```java
package io.sousou.model;

import javax.persistence.Entity;

@Entity
public class Student {
    @Id
    @GeneratedValue
    private Integer id;
    @NotBlank(message = "学生姓名不能为空")
    private String name;
    public Student(){

    }
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


}
```

另外，在这里我有一个index.html ， 这个static目录下面的静态文件是直接可以访问的，
访问路径就是 http://localhost:8081/index.html

我这里只是用来测试的，本来想顺便作为api接口的测试，但是后来发现比较麻烦，
就用postman来测试api接口测试了

postman 谷歌插件安装



结束

原文在此!