Spring和AOP编程

前言

实验内容

> 利用Spring技术实现一个校友信息的收集和展示页面。
>
> 要求采用MVC框架，要求加入依赖注入和面向切面的编程。
>
> 2.3 构建一个安全验证的切面。要求：
>
> a. 对于所有的Alumni表的查询操作，验证用户已经登录；如果用户没有登录，先导航到登录页面；
>
> b. 对于所有的Alumni表的更新（更新和删除）操作，在Read权限的基础上验证用户具有Update的权限。如果没有，该操作取消，并导航到错误页面。
>
> c. 对于所有的Alumni表的汇总和下载操作，验证用户具有Aggregate权限；如果没有，该操作取消，并导航到错误页面。

实验环境

> Manjaro 20.0 + jdk 1.8 + mariadb 10.4.12

AOP简介

1、什么是aop：

> - AOP（Aspect Oriented Programming）称为面向切面编程，在程序开发中主要用来解决一些系统层面上的问题，比如日志，事务，权限等待，Struts2的拦截器设计就是基于AOP的思想，是个比较经典的例子。
> - 在不改变原有的逻辑的基础上，增加一些额外的功能。代理也是这个功能，读写分离也能用aop来做。

> - AOP可以说是OOP（Object Oriented Programming，面向对象编程）的补充和完善。OOP引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过OOP允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（cross cutting），在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。

> - AOP技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。
> - 使用"横切"技术，AOP把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。

2、AOP的相关概念：

(1) 横切关注点：对哪些方法进行拦截，拦截后怎么处理，这些关注点称之为横切关注点;

(2) Aspect(切面):通常是一个类，里面可以定义切入点和通知;

(3) JointPoint(连接点):程序执行过程中明确的点，一般是方法的调用。被拦截到的点，因为Spring只支持方法类型的连接点，所以在Spring中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器;

(4) Advice(通知):AOP在特定的切入点上执行的增强处理，有before(前置),after(后置),afterReturning(最终),afterThrowing(异常),around(环绕);

(5) Pointcut(切入点):就是带有通知的连接点，在程序中主要体现为书写切入点表达式;

(6) weave(织入)：将切面应用到目标对象并导致代理对象创建的过程;

(7) introduction(引入)：在不修改代码的前提下，引入可以在**运行期**为类动态地添加一些方法或字段;

(8) AOP代理(AOP Proxy)：AOP框架创建的对象，代理就是目标对象的加强。Spring中的AOP代理可以使JDK动态代理，也可以是CGLIB代理，前者基于接口，后者基于子类;

(9) 目标对象（Target Object）: 包含连接点的对象。也被称作被通知或被代理对象。POJO.

3、Advice通知类型介绍：

(1)Before:在目标方法被调用之前做增强处理,@Before只需要指定切入点表达式即可

(2)AfterReturning:在目标方法正常完成后做增强,@AfterReturning除了指定切入点表达式后，还可以指定一个返回值形参名returning,代表目标方法的返回值

(3)AfterThrowing:主要用来处理程序中未处理的异常,@AfterThrowing除了指定切入点表达式后，还可以指定一个throwing的返回值形参名,可以通过该形参名

来访问目标方法中所抛出的异常对象

(4)After:在目标方法完成之后做增强，无论目标方法时候成功完成。@After可以指定一个切入点表达式

(5)Around:环绕通知,在目标方法完成前后做增强处理,环绕通知是最重要的通知类型,像事务,日志等都是环绕通知,注意编程中核心是一个ProceedingJoinPoint

4、AOP使用场景：

Authentication 权限

Caching 缓存

Context passing 内容传递

Error handling 错误处理

Lazy loading　懒加载

Debugging　　调试

logging, tracing, profiling and monitoring　记录跟踪　优化　校准

Performance optimization　性能优化

Persistence　　持久化

Resource pooling　资源池

Synchronization　同步

Transactions 事务

功能实现

DTO设计

- AlumniDTO

```java
public class AlumniDTO {

    private Integer id;
    private String testColumn;

    public AlumniDTO(Integer id, String testColumn) {
        this.id = id;
        this.testColumn = testColumn;
    }

    public Integer getId() { return id; }

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

    public String getTestColumn() { return testColumn; }

    public void setTestColumn(String testColumn) { this.testColumn = testColumn; }

    @Override
    public String toString() {
        return "AlumniDTO{" +
                "id=" + id +
                ", testColumn='" + testColumn + '\'' +
                '}';
    }
```

- UserDTO

```java
public class UserDTO {
    private Integer id;
    private String username;
    private String password;

    /**
     * 用户权限等级，其中
     * 1为Read权限
     * 2为Update权限
     * 3为Aggregate权限
     * 比1大则代表拥有Read权限
     */
    private Integer privilege;

    public UserDTO(Integer id, String username, String password, Integer privilege) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.privilege = privilege;
    }

    public Integer getId() {
        return id;
    }
```

Mapper

通过`mybatis`实现数据库连接操作。

先是总体配置`application.yml`：

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/Middleware?useUnicode=true&characterEncoding=utf8
    username: joker
    password: joker
    type: com.alibaba.druid.pool.DruidDataSource
    druid:
      #初始化时建立物理连接的个数
      initial-size: 3
      #  最小连接池数量
      min-idle: 1
      #最大连接池数量
      max-active: 5
      #获取连接时最大等待时间
      max-wait: 60000
      validation-query: select 1

mybatis:
  type-aliases-package: model
  mapper-locations: classpath:mapper/*.xml
  configuration:
    map-underscore-to-camel-case: true
```

然后是用户登录获取信息的mapper:

```java
package mapper;

import model.UserDTO;
import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface UserMapper {

    /**
     * 根据username和password进行登录
     * @param username username
     * @param password password
     * @return userDTO
     */
    UserDTO getUser(String username, String password);

}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/myba
<mapper namespace="mapper.UserMapper">

    <resultMap id="userMap" type="UserDTO">
        <id property="id" column="id"/>
        <result property="username" column="username"/>
        <result property="password" column="password"/>
        <result property="privilege" column="privilege"/>
    </resultMap>

    <select id="getUser" resultType="userMap">
        select * from UserInfo
        where username = #{username}
        and password = #{password};
    </select>

</mapper>
```

接着是对`Alumni`进行三种操作的mapper：

```java
package mapper;

import model.AlumniDTO;
import org.apache.ibatis.annotations.Mapper;

import java.util.List;

@Mapper
public interface AlumniMapper {

    List<AlumniDTO> search();

    Integer update(Integer id,String info);

    Integer aggregate();
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/my
<mapper namespace="mapper.AlumniMapper">

    <select id="search" resultType="AlumniDTO">
        select * from alumni;
    </select>

    <update id="update">
        update alumni set testColumn = #{info} where id = #{id};
    </update>

    <select id="aggregate">
        select count(*) from alumni;
    </select>
</mapper>
```

Controller接口

先实现`/login`：

```java
@RequestMapping("/login")
public void login(HttpServletRequest httpServletRequest, HttpServletResponse httpServletRespons
    HttpSession session = httpServletRequest.getSession();
    //如果已登录则自动退出登录
    if (session.getAttribute( s: "userDetail") != null) {
        httpServletResponse.sendRedirect( s: "/logout");
    }
    //验证用户是否成功登录
    else {
        String username = httpServletRequest.getParameter( s: "username");
        String password = httpServletRequest.getParameter( s: "password");
        UserDTO user = userMapper.getUser(username, password);
        if (user != null) {
            System.out.println(user.getId() + ":" + "login");
            session = httpServletRequest.getSession();
            session.setAttribute( s: "userDetail", user);
            //登录成功，将response中的status设置为200，以便aop获得此信息
            httpServletResponse.setStatus(200);
        } else {
            //登录成功，将response中的status设置为403，以便aop获得此信息
            httpServletResponse.setStatus(403);
        }
    }
}
```

顺带实现一下`logout`：

```java
@GetMapping("/logout")
public void logout(HttpServletRequest httpServletRequest, HttpServletResponse httpServletRespon
    HttpSession session = httpServletRequest.getSession();
    UserDTO user = (UserDTO) session.getAttribute( s: "userDetail");
    //验证用户是否还未登录就登出
    if (user != null) {
        System.out.println(user.getId() + ":" + "logout");
        //登出成功，将response中的status设置为200，以便aop获得此信息
        httpServletResponse.setStatus(200);
    } else {
        //登录失败，将response中的status设置为403，以便aop获得此信息
        httpServletResponse.setStatus(403);
    }
}
```

然后是对`Alumni`的三个操作：

```
@GetMapping("/alumni")
public Object search() throws Exception {
    return alumniMapper.search();
}

@PutMapping("/alumni/{id}")
public Object update(@PathVariable("id") Integer id, @RequestParam String info) {
    return alumniMapper.update(id, info);
}

@GetMapping("/alumni/aggregate")
public Object aggregate() {
    return alumniMapper.aggregate();
}
```

AOP

```
package aspect;

import model.UserDTO;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

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

@Aspect
public class SafetyVerification {

    @Pointcut("execution(* controller.InfoController.search())")
    public void searchPoint() {

    }

    @Pointcut("execution(* controller.InfoController.update())")
    public void updatePoint() {

    }

    @Pointcut("execution(* controller.InfoController.aggregate())")
    public void aggregate() {

    }

    @Before("searchPoint()")
    public void beforeSearch(ProceedingJoinPoint joinPoint) throws Throwable {
        // 拿到session并从session中拿到user信息
        HttpSession session = (HttpSession)
HttpServletRequest.class.getMethod("getSession").invoke(joinPoint.getArgs()[0]);
        UserDTO user = (UserDTO) session.getAttribute("userDetail");

        if (user == null) {
```

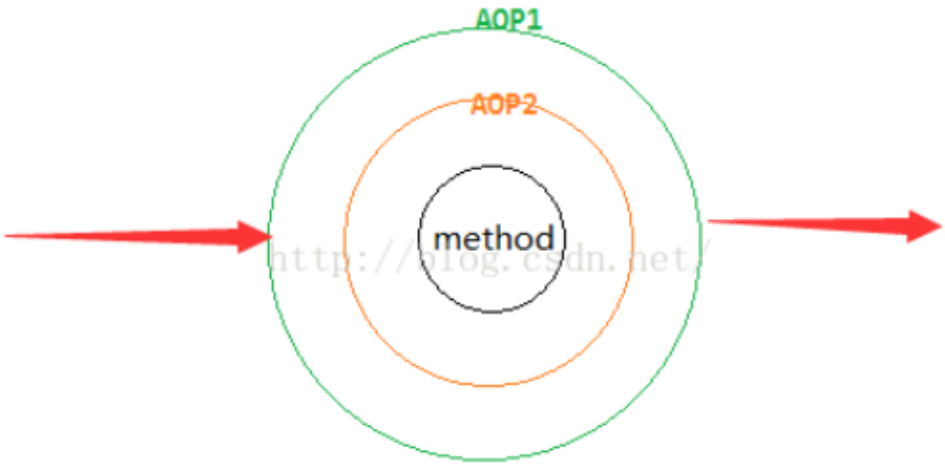
```
            // 未登录，重定向到登录界面
            System.out.println("Not Login!");
            HttpServletResponse.class.getMethod("sendRedirect",
String.class).invoke(joinPoint.getArgs()[1], "/login");
        } else if (user.getPrivilege() >= 1) {

            joinPoint.proceed();
        }
    }

    @Before("updatePoint()")
    public void beforeUpdate(ProceedingJoinPoint joinPoint) throws Throwable {
        // 拿到session并从session中拿到user信息
        HttpSession session = (HttpSession)
HttpServletRequest.class.getMethod("getSession").invoke(joinPoint.getArgs()[0]);
        UserDTO user = (UserDTO) session.getAttribute("userDetail");

        if (user == null) {
            // 未登录，重定向到登录界面
            System.out.println("Not Login!");
            HttpServletResponse.class.getMethod("sendRedirect",
String.class).invoke(joinPoint.getArgs()[1], "/login");
        } else if (user.getPrivilege() != 2) {
            // 权限不足，重定向到错误页面
            System.out.println("No Permission!");
            HttpServletResponse.class.getMethod("sendRedirect",
String.class).invoke(joinPoint.getArgs()[1], "/error.html");
        } else {
            joinPoint.proceed(joinPoint.getArgs());
        }
    }

    @Before("aggregate()")
    public void beforeAggregate(ProceedingJoinPoint joinPoint) throws Throwable
{
        // 拿到session并从session中拿到user信息
        HttpSession session = (HttpSession)
HttpServletRequest.class.getMethod("getSession").invoke(joinPoint.getArgs()[0]);
        UserDTO user = (UserDTO) session.getAttribute("userDetail");

        if (user == null) {
            // 未登录，重定向到登录界面
            System.out.println("Not Login!");
            HttpServletResponse.class.getMethod("sendRedirect",
String.class).invoke(joinPoint.getArgs()[1], "/login");
        } else if (user.getPrivilege() != 3) {
            // 权限不足，重定向到错误页面
            System.out.println("No Permission!");
            HttpServletResponse.class.getMethod("sendRedirect",
String.class).invoke(joinPoint.getArgs()[1], "/error.html");
        } else {
            joinPoint.proceed();
        }
    }
}
```

权限验证可以使用拦截器进行操作而不需要使用AOP。拦截器和AOP一样采用了**动态代理**的方式，可以设置指定的url被拦截或不被拦截，配置方便，很好用，但这里不做深入介绍。

设计思路：

- 实现登陆拦截器，用于拦截所有除登录的请求，如果用户未登录，则跳转到登录页面，同时将登录前访问的url保存至session中，以便登录成功后重新跳转到之前的页面。
- 实现`PermissionMetadata`组件，用于加载所有需要验证的URL的信息。
- 实现权限验证拦截器，用于拦截除登录登出以及获取当前用户信息外的所有请求，先查看此url是否在`PermissionMetadata`中，如果不在，则放行；如果在，根据sesssion中保存的用户的信息查看用户是否具有此权限，如果有，则放行。

这个拦截器也可以通过Zuul网关实现。附上一篇Zuul网教程：https://www.cnblogs.com/jing99/p/11696192.html

> 动态代理：
>
> Q：如果一个method实现了两个AOP，那么在两个AOP中调用了两次JoinPoint.proceed，那么会不会原方法也调用了两次呢？两次调用都修改了原方法的返回值，最终取谁的返回值呢？
>
> A：假设原方法的类为class，切面1的代理类proxy1，切面2的代理类为proxy2。事实上，SpringAOP使proxy2代理了class，proxy1代理了proxy2。调用的入口在proxy2，返回值的决定权也在proxy2，即最外层的代理类。（代理类的代理顺序可以进行配置）
> 打个比方：
> spring aop就是一个同心圆，要执行的方法为圆心，proxy2相应的方法为圆AOP2，proxy1相应的方法为圆AOP1
>
> 
>
>
> 所以AOP可以嵌套代理，但是被代理类只被调用一次。

数据库设计

由于DBeaver无法转储SQL文件，这里只能截图展示。

UserInfo表：

字段名	#	数据类型	非空	自增	键	缺省	额外的	Expression	注释
id	1	int(11)	☑	☐					
username	2	varchar(100)	☑	☐					
password	3	varchar(100)	☑	☐					
privilege	4	tinyint(4)	☑	☐					

属性 数据 ER 图

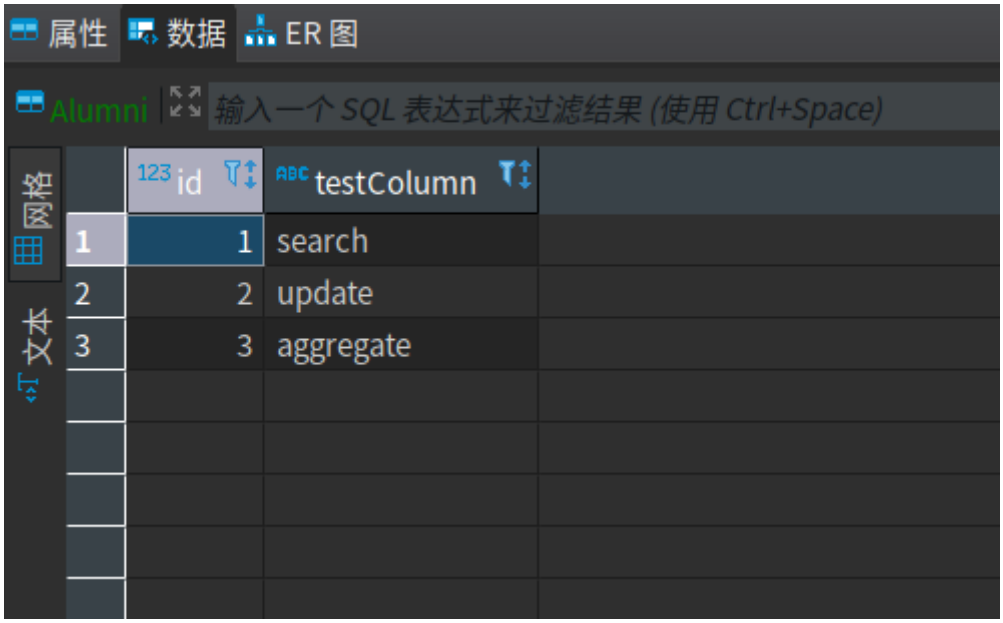
UserInfo 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

网格 文本

	id	username	password	privilege
1	1	searchAop	test1	1
2	1	searchAop	test2	2
3	1	searchAop	test3	3

Alumni表：

字段名	#	数据类型	非空	自增	键	缺省	额外的	Expression	注释
id	1	int(11)	☑	☐					
testColumn	2	varchar(100)	☑	☐					



参考资料

http://shouce.jb51.net/spring/aop.html

https://howtodoinjava.com/spring-aop/aspectj-around-annotation-example/

https://www.iteye.com/blog/bjtdeyx-656021

https://blog.csdn.net/weixin_30954607/article/details/96816549

https://howtodoinjava.com/spring-aop/aspectj-around-annotation-example/