轻量级权限管理系统

实体类定义

创建 `migo-shiro` 权限模块

菜单管理实体类定义

`com.migo.entity.SysMenuEntity`

角色定义

`com.migo.entity.SysRoleEntity`

角色与菜单对应关系

`com.migo.entity.SysRoleMenuEntity`

用户与角色对应关系

`com.migo.entity.SysUserRoleEntity`

系统用户

`com.migo.entity.SysUserEntity`

`DAO` 定义

创建 `migo-common` 公共模块

`baseDao`

`baseDao` 是通用的，所以设定在 `common` 模块内

回到 `migo-shiro` 权限模块

菜单管理

`com.migo.dao.SysMenuDao extends BaseDao<SysMenuEntity>` 包含功能:

- 根据父菜单，查询子菜单
- 获取不包含按钮的菜单列表
- 查询用户的权限列表

角色管理

`com.migo.dao.SysRoleDao extends BaseDao<SysRoleEntity>`

除基本方法外，需要独自实现的方法:**查询用户创建的角色ID列表**

角色与菜单对应关系

`com.migo.dao.SysRoleMenuDao extends BaseDao<SysRoleMenuEntity>`

除基本方法外，需要独自实现的方法:**根据角色ID，获取菜单ID列表**

系统用户

`com.migo.dao.SysUserDao extends BaseDao<SysUserEntity>`

除基本方法外，需要独自实现的方法:

- 查询用户的所有权限
- 查询用户的所有菜单ID
- 根据用户名，查询系统用户
- 修改密码

用户与角色对应关系

`com.migo.dao.SysUserRoleDao  extends BaseDao<SysUserRoleEntity>`

除基本方法外，需要独自实现的方法: **根据用户ID，获取角色ID列表**

Service定义

菜单管理

`com.migo.service.SysMenuService`

要实现的内容:

- 根据父菜单，查询子菜单
- 获取不包含按钮的菜单列表
- 获取用户菜单列表
- 查询菜单
- 查询菜单列表
- 查询总数
- 保存菜单
- 修改
- 删除
- 查询用户的权限列表

角色与菜单对应关系

`com.migo.service.SysRoleMenuService`

要实现的内容:

- 根据角色ID和菜单idlist进行保存或修改
- 根据角色ID，获取菜单ID列表

角色

`com.migo.service.SysRoleService`

要实现的内容:此处直接列代码了:

```
SysRoleEntity queryObject(Long roleId);

    List<SysRoleEntity> queryList(Map<String, Object> map);

    int queryTotal(Map<String, Object> map);

    void save(SysRoleEntity role);

    void update(SysRoleEntity role);

    void deleteBatch(Long[] roleIds);

    /**
     * 查询用户创建的角色ID列表
     */
    List<Long> queryRoleIdList(Long createUserId);
```

用户与角色对应关系

`com.migo.service.SysUserRoleService`

要实现的内容:

- 根据用户ID和角色idlist进行保存或修改
- 根据用户ID，获取角色ID列表
- 根据用户ID，做删除操作

系统用户

`com.migo.service.SysUserService`

要实现的内容:

- 查询用户的所有权限
- 查询用户的所有菜单ID
- 根据用户名，查询系统用户
- 根据用户ID，查询用户
- 查询用户列表
- 查询总数
- 保存用户
- 修改用户
- 删除用户
- 修改密码

Service实现

角色与菜单对应关系

`com.migo.service.impl.SysRoleMenuServiceImpl`

```
@Service("sysRoleMenuService")
public class SysRoleMenuServiceImpl implements SysRoleMenuService {
    @Autowired
    private SysRoleMenuDao sysRoleMenuDao;

    @Override
    @Transactional
    public void saveOrUpdate(Long roleId, List<Long> menuIdList) {
        if(menuIdList.size() == 0){
            return ;
        }
        //先删除角色与菜单关系
        sysRoleMenuDao.delete(roleId);

        //保存角色与菜单关系
        Map<String, Object> map = new HashMap<>();
        map.put("roleId", roleId);
        map.put("menuIdList", menuIdList);
        sysRoleMenuDao.save(map);
    }

    @Override
    public List<Long> queryMenuIdList(Long roleId) {
        return sysRoleMenuDao.queryMenuIdList(roleId);
    }

}
```

角色

```
@Service("sysRoleService")
public class SysRoleServiceImpl implements SysRoleService {
    @Autowired
    private SysRoleDao sysRoleDao;
    @Autowired
    private SysRoleMenuService sysRoleMenuService;
```

用户与角色对应关系

```java
@Service("sysUserRoleService")
public class SysUserRoleServiceImpl implements SysUserRoleService {
    @Autowired
    private SysUserRoleDao sysUserRoleDao;

    @Override
    public void saveOrUpdate(Long userId, List<Long> roleIdList) {
        if(roleIdList.size() == 0){
            return ;
        }

        //先删除用户与角色关系
        sysUserRoleDao.delete(userId);

        //保存用户与角色关系
        Map<String, Object> map = new HashMap<>();
        map.put("userId", userId);
        map.put("roleIdList", roleIdList);
        sysUserRoleDao.save(map);
    }

    @Override
    public List<Long> queryRoleIdList(Long userId) {
        return sysUserRoleDao.queryRoleIdList(userId);
    }

    @Override
    public void delete(Long userId) {
        sysUserRoleDao.delete(userId);
    }
```

系统用户

```java
@Service("sysUserService")
public class SysUserServiceImpl implements SysUserService {
    @Autowired
    private SysUserDao sysUserDao;
    @Autowired
    private SysUserRoleService sysUserRoleService;
    @Autowired
    private SysRoleService sysRoleService;

    @Override
    public List<String> queryAllPerms(Long userId) {
        return sysUserDao.queryAllPerms(userId);
    }

    @Override
    public List<Long> queryAllMenuId(Long userId) {
        return sysUserDao.queryAllMenuId(userId);
    }

    @Override
    public SysUserEntity queryByUserName(String username) {
        return sysUserDao.queryByUserName(username);
    }

    @Override
    public SysUserEntity queryObject(Long userId) {
        return sysUserDao.queryObject(userId);
    }

    @Override
    public List<SysUserEntity> queryList(Map<String, Object> map){
        return sysUserDao.queryList(map);
    }

    @Override
    public int queryTotal(Map<String, Object> map) {
        return sysUserDao.queryTotal(map);
    }

    @Override
    @Transactional
    public void save(SysUserEntity user) {
        user.setCreateTime(new Date());
        //sha256加密
        user.setPassword(new Sha256Hash(user.getPassword()).toHex());
        sysUserDao.save(user);

        //检查角色是否越权
        checkRole(user);

        //保存用户与角色关系
        sysUserRoleService.saveOrUpdate(user.getUserId(), user.getRoleIdList());
    }
```

```
    @Override
    @Transactional
    public void update(SysUserEntity user) {
        if(StringUtils.isBlank(user.getPassword())){
            user.setPassword(null);
        }else{
            user.setPassword(new Sha256Hash(user.getPassword()).toHex());
        }
        sysUserDao.update(user);

        //检查角色是否越权
        checkRole(user);

        //保存用户与角色关系
        sysUserRoleService.saveOrUpdate(user.getUserId(), user.getRoleIdList());
    }

    @Override
    @Transactional
    public void deleteBatch(Long[] userId) {
        sysUserDao.deleteBatch(userId);
    }

    @Override
    public int updatePassword(Long userId, String password, String newPassword) {
        Map<String, Object> map = new HashMap<>();
        map.put("userId", userId);
        map.put("password", password);
        map.put("newPassword", newPassword);
        return sysUserDao.updatePassword(map);
    }

    /**
     * 检查角色是否越权
     */
    private void checkRole(SysUserEntity user){
        //如果不是超级管理员，则需要判断用户的角色是否自己创建
        if(user.getCreateUserId() == Constant.SUPER_ADMIN){
            return ;
        }

        //查询用户创建的角色列表
        List<Long> roleIdList = sysRoleService.queryRoleIdList(user.getCreateUserId());

        //判断是否越权
        if(!roleIdList.containsAll(user.getRoleIdList())){
            throw new RRException("新增用户所选角色，不是本人创建");
        }
    }
}
```

工具类之校验

hibernate-validator校验工具类

```
public class ValidatorUtils {
    private static Validator validator;

    static {
        validator = Validation.buildDefaultValidatorFactory().getValidator();
    }

    /**
     * 校验对象
     * @param object        待校验对象
     * @param groups        待校验的组
     * @throws RRException  校验不通过，则报RRException异常
     */
    public static void validateEntity(Object object, Class<?>... groups)
            throws RRException {
        Set<ConstraintViolation<Object>> constraintViolations = validator.validate(object,
groups);
        if (!constraintViolations.isEmpty()) {
            ConstraintViolation<Object> constraint =
(ConstraintViolation<Object>)constraintViolations.iterator().next();
            throw new RRException(constraint.getMessage());
        }
    }
}
```

更新数据 Group

```
public interface UpdateGroup {
}
```

新增数据 Group

```
public interface AddGroup {
}
```

定义校验顺序

如果**AddGroup**组失败，则**UpdateGroup**组不会再校验

```
@GroupSequence({AddGroup.class, UpdateGroup.class})
public interface Group {
}
```

Controller定义

Controller公共组件

```
public abstract class AbstractController {
    protected Logger logger = LoggerFactory.getLogger(getClass());

    protected SysUserEntity getUser() {
        return ShiroUtils.getUserEntity();
    }

    protected Long getUserId() {
        return getUser().getUserId();
    }
}
```

权限认证

`com.migo.shiro.UserRealm extends AuthorizingRealm`

重写:

`protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)` 授权(验证权限时调用)

`protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)` 认证(登录时调用)

返回数据包装

```
public class R extends HashMap<String, Object> {
    private static final long serialVersionUID = 1L;

    public R() {
        put("code", 0);
    }

    public static R error() {
        return error(500, "未知异常，请联系管理员");
    }

    public static R error(String msg) {
        return error(500, msg);
    }

    public static R error(int code, String msg) {
        R r = new R();
        r.put("code", code);
        r.put("msg", msg);
        return r;
    }

    public static R ok(String msg) {
        R r = new R();
        r.put("msg", msg);
        return r;
    }

    public static R ok(Map<String, Object> map) {
        R r = new R();
        r.putAll(map);
        return r;
    }

    public static R ok() {
        return new R();
    }

    public R put(String key, Object value) {
        super.put(key, value);
        return this;
    }
}
```

自定义异常

```
public class RRException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    private String msg;
    private int code = 500;
```

异常处理器

```
@Component
public class RRExceptionHandler implements HandlerExceptionResolver {
    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    public ModelAndView resolveException(HttpServletRequest request,
            HttpServletResponse response, Object handler, Exception ex) {
        R r = new R();
        try {
            response.setContentType("application/json;charset=utf-8");
            response.setCharacterEncoding("utf-8");

            if (ex instanceof RRException) {
                r.put("code", ((RRException) ex).getCode());
                r.put("msg", ((RRException) ex).getMessage());
            }else if(ex instanceof DuplicateKeyException){
                r = R.error("数据库中已存在该记录");
            }else if(ex instanceof AuthorizationException){
                r = R.error("没有权限，请联系管理员授权");
            }else{
                r = R.error();
            }

            //记录异常日志
            logger.error(ex.getMessage(), ex);

            String json = JSON.toJSONString(r);
            response.getWriter().print(json);
        } catch (Exception e) {
            logger.error("RRExceptionHandler 异常处理失败", e);
        }
        return new ModelAndView();
    }
}
```

查询参数

```java
public class Query extends LinkedHashMap<String, Object> {
    private static final long serialVersionUID = 1L;
    //当前页码
    private int page;
    //每页条数
    private int limit;

    public Query(Map<String, Object> params){
        this.putAll(params);

        //分页参数
        this.page = Integer.parseInt(params.get("page").toString());
        this.limit = Integer.parseInt(params.get("limit").toString());
        this.put("offset", (page - 1) * limit);
        this.put("page", page);
        this.put("limit", limit);

        //防止SQL注入（因为sidx、order是通过拼接SQL实现排序的，会有SQL注入风险）
        String sidx = params.get("sidx").toString();
        String order = params.get("order").toString();
        this.put("sidx", SQLFilter.sqlInject(sidx));
        this.put("order", SQLFilter.sqlInject(order));
    }
```

系统菜单

`com.migo.controller.SysMenuController`

`@RequestMapping("/sys/menu")`旗下的子访问路径有:

- 所有菜单列表:"/list"　　权限设定:`@RequiresPermissions("sys:menu:list")`
  - 选择菜单(添加、修改菜单):"/select"权限设定:`@RequiresPermissions("sys:menu:select")`
- 角色授权菜单:"/perms"权限设定:`@RequiresPermissions("sys:menu:perms")`
- 菜单信息:"/info/{menuId}"权限设定:`@RequiresPermissions("sys:menu:info")`
- 保存菜单:"/save"权限设定:`@RequiresPermissions("sys:menu:save")`
- 修改菜单:"/update"权限设定:`@RequiresPermissions("sys:menu:update")`
- 删除菜单:"/delete"权限设定:`@RequiresPermissions("sys:menu:delete")`
- 用户菜单列表:"/user"

其内涉及的校验:验证参数是否正确:

1. 菜单类型
2. 上级菜单类型:

   目录或菜单的上级类型

   按钮上级类型

系统用户

`com.migo.controller.SysUserController`

`@RequestMapping("/sys/user")`旗下的子访问路径有:

所有用户列表:"/list" 权限设定:`@RequiresPermissions("sys:user:list")`

获取登录的用户信息:"/info" 权限设定:``

修改登录用户密码:"/password" 权限设定:``

用户信息:"/info/{userId}"权限设定:`@RequiresPermissions("sys:user:info")`

保存用户:"/save" 权限设定:`@RequiresPermissions("sys:user:save")`

修改用户:"/update"权限设定:`@RequiresPermissions("sys:user:update")`

删除用户:"/delete"权限设定:`@RequiresPermissions("sys:user:delete")`

分页工具类

```
public class PageUtils implements Serializable {
    private static final long serialVersionUID = 1L;
    //总记录数
    private int totalCount;
    //每页记录数
    private int pageSize;
    //总页数
    private int totalPage;
    //当前页数
    private int currPage;
    //列表数据
    private List<?> list;

    /**
     * 分页
     * @param list        列表数据
     * @param totalCount  总记录数
     * @param pageSize    每页记录数
     * @param currPage    当前页数
     */
    public PageUtils(List<?> list, int totalCount, int pageSize, int currPage) {
        this.list = list;
        this.totalCount = totalCount;
        this.pageSize = pageSize;
        this.currPage = currPage;
        this.totalPage = (int)Math.ceil((double)totalCount/pageSize);
    }
```

角色管理

`com.migo.controller.SysRoleController`

`@RequestMapping("/sys/role")`旗下的子访问路径有:

`角色列表(Map<String, Object> arg)`:"/list" 权限设定:`@RequiresPermissions("sys:role:list")`

角色列表(非参数查询):"/select" 权限设定:`@RequiresPermissions("sys:role:select")`

角色信息:"/info/{roleId}" 权限设定:`@RequiresPermissions("sys:role:info")`

保存角色:"/update" 权限设定:`@RequiresPermissions("sys:role:update")`

删除角色:"/delete" 权限设定:`@RequiresPermissions("sys:role:delete")`

Shiro工具类

```
public class ShiroUtils {

    public static Session getSession() {
        return SecurityUtils.getSubject().getSession();
    }

    public static Subject getSubject() {
        return SecurityUtils.getSubject();
    }

    public static SysUserEntity getUserEntity() {
        return (SysUserEntity)SecurityUtils.getSubject().getPrincipal();
    }

    public static Long getUserId() {
        return getUserEntity().getUserId();
    }

    public static void setSessionAttribute(Object key, Object value) {
        getSession().setAttribute(key, value);
    }

    public static Object getSessionAttribute(Object key) {
        return getSession().getAttribute(key);
    }

    public static boolean isLogin() {
        return SecurityUtils.getSubject().getPrincipal() != null;
    }

    public static void logout() {
        SecurityUtils.getSubject().logout();
    }

    public static String getKaptcha(String key) {
        String kaptcha = getSessionAttribute(key).toString();
        getSession().removeAttribute(key);
        return kaptcha;
    }

}
```

登录相关

`com.migo.controller.SysLoginController`

验证码逻辑:`@RequestMapping("captcha.jpg")`

登录访问:`@RequestMapping(value = "/sys/login", method = RequestMethod.POST)`

常量类工具类定义

`com.migo.utils.Constant`

```
/** 超级管理员ID */
public static final int SUPER_ADMIN = 1;
```

菜单类型

```
public enum MenuType {
        /**
         * 目录
         */
        CATALOG(0),
        /**
         * 菜单
         */
        MENU(1),
        /**
         * 按钮
         */
        BUTTON(2);

        private int value;

        private MenuType(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }
    }
```

系统页面视图

`com.migo.controller.SysPageController`

```
@Controller
public class SysPageController {

    @RequestMapping("sys/{url}.html")
    public String page(@PathVariable("url") String url){
        return "sys/" + url + ".html";
    }

    @RequestMapping("generator/{url}.html")
    public String generator(@PathVariable("url") String url){
        return "generator/" + url + ".html";
    }
}
```

定时任务

定时器实体类设定

`com.migo.entity.ScheduleJobEntity`

```
public class ScheduleJobEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    /**
     * 任务调度参数key
     */
    public static final String JOB_PARAM_KEY = "JOB_PARAM_KEY";

    /**
     * 任务id
     */
    private Long jobId;

    /**
     * spring bean名称
     */
    @NotBlank(message="bean名称不能为空")
    private String beanName;

    /**
     * 方法名
     */
    @NotBlank(message="方法名称不能为空")
    private String methodName;

    /**
     * 参数
     */
    private String params;

    /**
     * cron表达式
     */
    @NotBlank(message="cron表达式不能为空")
    private String cronExpression;

    /**
     * 任务状态
     */
    private Integer status;

    /**
     * 备注
     */
    private String remark;

    /**
     * 创建时间
     */
    private Date createTime;
```

常量类中设置定时任务状态

`com.migo.utils.Constant`

```
public enum ScheduleStatus {
        /**
         * 正常
         */
        NORMAL(0),
        /**
         * 暂停
         */
        PAUSE(1);

        private int value;

        private ScheduleStatus(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }
    }
```

定时任务Service

`com.migo.service.ScheduleJobService` 包含功能:

`根据ID，查询定时任务`

`查询定时任务列表`

`查询总数`

`保存定时任务`

`更新定时任务`

`批量删除定时任务`

`批量更新定时任务状态`

`立即执行`

`暂停运行`

`恢复运行`

定时任务Controller

`com.migo.controller.ScheduleJobController`

`@RequestMapping("/sys/schedule")` 旗下的子访问路径有:

`定时任务列表` :"/list" `权限设定` : `@RequiresPermissions("sys:schedule:list")`

`定时任务信息` :"/info/{jobId}" `权限设定` : `@RequiresPermissions("sys:schedule:info")`

保存定时任务:"/save"　　权限设定:`@RequiresPermissions("sys:schedule:save")`

修改定时任务:"/update"　　权限设定:`@RequiresPermissions("sys:schedule:update")`

删除定时任务:"/delete"　　权限设定:`@RequiresPermissions("sys:schedule:delete")`

立即执行任务:"/run"　　权限设定:`@RequiresPermissions("sys:schedule:run")`

暂停定时任务:"/pause"　　权限设定:`@RequiresPermissions("sys:schedule:pause")`

恢复定时任务:"/resume"　　权限设定:`@RequiresPermissions("sys:schedule:resume")`

Spring Context 工具类

```
@Component
public class SpringContextUtils implements ApplicationContextAware {
    public static ApplicationContext applicationContext;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext)
            throws BeansException {
        SpringContextUtils.applicationContext = applicationContext;
    }

    public static Object getBean(String name) {
        return applicationContext.getBean(name);
    }

    public static <T> T getBean(String name, Class<T> requiredType) {
        return applicationContext.getBean(name, requiredType);
    }

    public static boolean containsBean(String name) {
        return applicationContext.containsBean(name);
    }

    public static boolean isSingleton(String name) {
        return applicationContext.isSingleton(name);
    }

    public static Class<? extends Object> getType(String name) {
        return applicationContext.getType(name);
    }

}
```

定义一个定时job

`com.migo.utils.ScheduleJob extends QuartzJobBean`

```
public class ScheduleJob extends QuartzJobBean {
    private Logger logger = LoggerFactory.getLogger(getClass());
    private ExecutorService service = Executors.newSingleThreadExecutor();

    @Override
    protected void executeInternal(JobExecutionContext context) throws JobExecutionException {
        ScheduleJobEntity scheduleJob = (ScheduleJobEntity) context.getMergedJobDataMap()
                .get(ScheduleJobEntity.JOB_PARAM_KEY);

        //获取spring bean
        ScheduleJobLogService scheduleJobLogService = (ScheduleJobLogService)
SpringContextUtils.getBean("scheduleJobLogService");

        //数据库保存执行记录
        ScheduleJobLogEntity log = new ScheduleJobLogEntity();
        log.setJobId(scheduleJob.getJobId());
        log.setBeanName(scheduleJob.getBeanName());
        log.setMethodName(scheduleJob.getMethodName());
        log.setParams(scheduleJob.getParams());
        log.setCreateTime(new Date());

        //任务开始时间
        long startTime = System.currentTimeMillis();

        try {
            //执行任务
            logger.info("任务准备执行，任务ID：" + scheduleJob.getJobId());
            ScheduleRunnable task = new ScheduleRunnable(scheduleJob.getBeanName(),
                    scheduleJob.getMethodName(), scheduleJob.getParams());
            Future<?> future = service.submit(task);

             future.get();

             //任务执行总时长
             long times = System.currentTimeMillis() - startTime;
             log.setTimes((int)times);
             //任务状态    0：成功    1：失败
             log.setStatus(0);

             logger.info("任务执行完毕，任务ID：" + scheduleJob.getJobId() + "  总共耗时：" + times
+ "毫秒");
         } catch (Exception e) {
             logger.error("任务执行失败，任务ID：" + scheduleJob.getJobId(), e);

             //任务执行总时长
             long times = System.currentTimeMillis() - startTime;
             log.setTimes((int)times);

             //任务状态    0：成功    1：失败
             log.setStatus(1);
             log.setError(StringUtils.substring(e.toString(), 0, 2000));
         }finally {

             scheduleJobLogService.save(log);
```

```
        }
    }
}
```

定时任务工具类-方便使用

`com.migo.utils.ScheduleUtils`

包含以下功能:

`获取触发器key`

`获取jobKey`

`获取表达式触发器`

`创建定时任务`

`更新定时任务`

`立即执行任务`

`暂停任务`

`恢复任务`

`删除定时任务`

执行定时任务

```
public class ScheduleRunnable implements Runnable {
    private Object target;
    private Method method;
    private String params;

    public ScheduleRunnable(String beanName, String methodName, String params) throws
NoSuchMethodException, SecurityException {
        this.target = SpringContextUtils.getBean(beanName);
        this.params = params;

        if(StringUtils.isNotBlank(params)){
            this.method = target.getClass().getDeclaredMethod(methodName, String.class);
        }else{
            this.method = target.getClass().getDeclaredMethod(methodName);
        }
    }

    @Override
    public void run() {
        try {
            ReflectionUtils.makeAccessible(method);
            if(StringUtils.isNotBlank(params)){
                method.invoke(target, params);
            }else{
                method.invoke(target);
            }
        }catch (Exception e) {
            throw new RRException("执行定时任务失败", e);
        }
    }

}
```

定时任务执行日志实体类

`com.migo.entity.ScheduleJobLogEntity`

```java
public class ScheduleJobLogEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    /**
     * 日志id
     */
    private Long logId;

    /**
     * 任务id
     */
    private Long jobId;

    /**
     * spring bean名称
     */
    private String beanName;

    /**
     * 方法名
     */
    private String methodName;

    /**
     * 参数
     */
    private String params;

    /**
     * 任务状态    0：成功    1：失败
     */
    private Integer status;

    /**
     * 失败信息
     */
    private String error;

    /**
     * 耗时(单位：毫秒)
     */
    private Integer times;

    /**
     * 创建时间
     */
    private Date createTime;
```

定时任务DAO

```
public interface ScheduleJobDao extends BaseDao<ScheduleJobEntity> {

    /**
     * 批量更新状态
     */
    int updateBatch(Map<String, Object> map);
}
```

定时任务日志DAO

```
public interface ScheduleJobLogDao extends BaseDao<ScheduleJobLogEntity> {

}
```

定时任务日志Service

```
public interface ScheduleJobLogService {

    /**
     * 根据ID，查询定时任务日志
     */
    ScheduleJobLogEntity queryObject(Long jobId);

    /**
     * 查询定时任务日志列表
     */
    List<ScheduleJobLogEntity> queryList(Map<String, Object> map);

    /**
     * 查询总数
     */
    int queryTotal(Map<String, Object> map);

    /**
     * 保存定时任务日志
     */
    void save(ScheduleJobLogEntity log);

}
```

定时任务日志Controller

`com.migo.controller.ScheduleJobLogController`

`@RequestMapping("/sys/scheduleLog")` 旗下的子访问路径有:

定时任务日志列表:"/list"　　权限设定:`@RequiresPermissions("sys:schedule:log")`

定时任务日志信息:"/info/{logId}"　　权限设定:``

Shiro权限标签

```
public class VelocityShiro {

    /**
     * 是否拥有该权限
     * @param permission  权限标识
     * @return   true：是     false：否
     */
    public boolean hasPermission(String permission) {
        Subject subject = SecurityUtils.getSubject();
        return subject != null && subject.isPermitted(permission);
    }

}
```

系统配置信息

系统配置信息实体类

```
public class SysConfigEntity {
    private Long id;
    @NotBlank(message="参数名不能为空")
    private String key;
    @NotBlank(message="参数值不能为空")
    private String value;
    private String remark;
```

系统配置信息DAO

```
public interface SysConfigDao extends BaseDao<SysConfigEntity> {

    /**
     * 根据key，查询value
     */
    String queryByKey(String paramKey);

    /**
     * 根据key，更新value
     */
    int updateValueByKey(@Param("key") String key, @Param("value") String value);

}
```

系统配置信息Service

```
public interface SysConfigService {

    /**
     * 保存配置信息
     */
    public void save(SysConfigEntity config);

    /**
     * 更新配置信息
     */
    public void update(SysConfigEntity config);

    /**
     * 根据key，更新value
     */
    public void updateValueByKey(String key, String value);

    /**
     * 删除配置信息
     */
    public void deleteBatch(Long[] ids);

    /**
     * 获取List列表
     */
    public List<SysConfigEntity> queryList(Map<String, Object> map);
    /**
     * 获取总记录数
     */
    public int queryTotal(Map<String, Object> map);

    public SysConfigEntity queryObject(Long id);

    /**
     * 根据key，获取配置的value值
     *
     * @param key           key
     * @param defaultValue  缺省值
     */
    public String getValue(String key, String defaultValue);

    /**
     * 根据key，获取value的Object对象
     * @param key    key
     * @param clazz  Object对象
     */
    public <T> T getConfigObject(String key, Class<T> clazz);

}
```

系统配置信息Contorller

`com.migo.controller.SysConfigController extends AbstractController`

`@RequestMapping("/sys/config")`旗下的子访问路径有:

所有配置列表:"/list"　　权限设定:`@RequiresPermissions("sys:config:list")`

配置信息:"/info/{id}"　　权限设定:`@RequiresPermissions("sys:config:info")`

保存配置:"/save"　　权限设定:`@RequiresPermissions("sys:config:save")`

修改配置:"/update"　　权限设定:`@RequiresPermissions("sys:config:update")`

删除配置:"/delete"　　权限设定:`@RequiresPermissions("sys:config:delete")`

代码生成器

新建代码生成器模块项目

列的属性实体类

```
public class ColumnEntity {
    //列名
    private String columnName;
    //列名类型
    private String dataType;
    //列名备注
    private String comments;

    //属性名称(第一个字母大写)，如：user_name => UserName
    private String attrName;
    //属性名称(第一个字母小写)，如：user_name => userName
    private String attrname;
    //属性类型
    private String attrType;
    //auto_increment
    private String extra;
```

表数据对应实体类

```
public class TableEntity {
    //表的名称
    private String tableName;
    //表的备注
    private String comments;
    //表的主键
    private ColumnEntity pk;
    //表的列名(不包含主键)
    private List<ColumnEntity> columns;

    //类名(第一个字母大写)，如：sys_user => SysUser
    private String className;
    //类名(第一个字母小写)，如：sys_user => sysUser
    private String classname;
```

日期处理类

```
public class DateUtils {
    /** 时间格式(yyyy-MM-dd) */
    public final static String DATE_PATTERN = "yyyy-MM-dd";
    /** 时间格式(yyyy-MM-dd HH:mm:ss) */
    public final static String DATE_TIME_PATTERN = "yyyy-MM-dd HH:mm:ss";

    public static String format(Date date) {
        return format(date, DATE_PATTERN);
    }

    public static String format(Date date, String pattern) {
        if(date != null){
            SimpleDateFormat df = new SimpleDateFormat(pattern);
            return df.format(date);
        }
        return null;
    }
}
```

代码生成器DAO

```
public interface SysGeneratorDao {

    List<Map<String, Object>> queryList(Map<String, Object> map);

    int queryTotal(Map<String, Object> map);

    Map<String, String> queryTable(String tableName);

    List<Map<String, String>> queryColumns(String tableName);
}
```

代码生成器Service

```
public interface SysGeneratorService {

    List<Map<String, Object>> queryList(Map<String, Object> map);

    int queryTotal(Map<String, Object> map);

    Map<String, String> queryTable(String tableName);

    List<Map<String, String>> queryColumns(String tableName);

    /**
     * 生成代码
     */
    byte[] generatorCode(String[] tableNames);
}
```

代码生成器Controller

`com.migo.controller.SysGeneratorController`

`@RequestMapping("/sys/generator")`旗下的子访问路径有:

列表:"/list" 权限设定:`@RequiresPermissions("sys:generator:list")` 权限设定:``

生成代码:"/code" 权限设定:`@RequiresPermissions("sys:generator:code")`

封装代码生成器工具类

`com.migo.utils.GenUtils`

所要进行的逻辑如下:

获取静态模板列表

生成代码逻辑:

提前准备条件:

- 列名转换成Java属性名
- 表名转换成Java类名
- 获取配置信息
- 获取文件名

生成相应代码步骤: 传入的参数为`(Map<String, String> table,List<Map<String, String>> columns, ZipOutputStream zip)`

- 获取配置信息
- 对表进行表信息提取
- 将表名转换成Java类名
- 提取列信息:
    - 将传入的list类型的columns进行循环遍历

将每个列的列名转换成Java属性名

将每个列的数据类型，转换成Java类型

主键设置,判断有且对应实体类未设置，那么设置实体类的PK，没有则设置表第一个字段为主键到对应实体类中

设置velocity资源加载器

封装模板数据

获取模板列表,并对列表循环遍历

遍历的每一个模板进行渲染模板

添加到zip

系统日志

系统日志注解

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface SysLog {

    String value() default "";
}
```

通过日志注解来做系统日志的AOP切面处理

com.migo.aop.SysLogAspect

```java
@Aspect
@Component
public class SysLogAspect {
    @Autowired
    private SysLogService sysLogService;

    @Pointcut("@annotation(com.migo.annotation.SysLog)")
    public void logPointCut() {

    }

    @Before("logPointCut()")
    public void saveSysLog(JoinPoint joinPoint) {
        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        Method method = signature.getMethod();

        SysLogEntity sysLog = new SysLogEntity();
        SysLog syslog = method.getAnnotation(SysLog.class);
        if(syslog != null){
            //注解上的描述
            sysLog.setOperation(syslog.value());
        }

        //请求的方法名
        String className = joinPoint.getTarget().getClass().getName();
        String methodName = signature.getName();
        sysLog.setMethod(className + "." + methodName + "()");

        //请求的参数
        Object[] args = joinPoint.getArgs();
        String params = JSON.toJSONString(args[0]);
        sysLog.setParams(params);

        //获取request
        HttpServletRequest request = HttpContextUtils.getHttpServletRequest();
        //设置IP地址
        sysLog.setIp(IPUtils.getIpAddr(request));

        //用户名
        String username = ShiroUtils.getUserEntity().getUsername();
        sysLog.setUsername(username);

        sysLog.setCreateDate(new Date());
        //保存系统日志
        sysLogService.save(sysLog);
    }
}
```

系统日志实体类

```
public class SysLogEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    private Long id;
    //用户名
    private String username;
    //用户操作
    private String operation;
    //请求方法
    private String method;
    //请求参数
    private String params;
    //IP地址
    private String ip;
    //创建时间
    private Date createDate;
```

系统日志DAO

```
public interface SysLogDao extends BaseDao<SysLogEntity> {

}
```

系统日志Service

```
public interface SysLogService {

    SysLogEntity queryObject(Long id);

    List<SysLogEntity> queryList(Map<String, Object> map);

    int queryTotal(Map<String, Object> map);

    void save(SysLogEntity sysLog);

    void update(SysLogEntity sysLog);

    void delete(Long id);

    void deleteBatch(Long[] ids);
}
```

系统日志Controller

`com.migo.controller.SysLogController`

`@RequestMapping("/sys/log")`旗下的子访问路径有:

`列表`:"/list"　　`权限设定`:`@RequiresPermissions("sys:log:list")`

API接口模块

用户实体类

```
public class UserEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    //用户ID
    private Long userId;
    //用户名
    private String username;
    //手机号
    private String mobile;
    //密码
    transient private String password;
    //创建时间
    private Date createTime;
```

用户DAO

```
public interface UserDao extends BaseDao<UserEntity> {

    UserEntity queryByMobile(String mobile);
}
```

用户Service

```
public interface UserService {

    UserEntity queryObject(Long userId);

    List<UserEntity> queryList(Map<String, Object> map);

    int queryTotal(Map<String, Object> map);

    void save(UserEntity user);

    void update(UserEntity user);

    void delete(Long userId);

    void deleteBatch(Long[] userIds);

    UserEntity queryByMobile(String mobile);

    /**
     * 用户登录
     * @param mobile    手机号
     * @param password  密码
     * @return          返回用户ID
     */
    long login(String mobile, String password);
}
```

用户Token实体类

```
public class TokenEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    //用户ID
    private Long userId;
    //token
    private String token;
    //过期时间
    private Date expireTime;
    //更新时间
    private Date updateTime;
```

用户TokenDAO

```
public interface TokenDao extends BaseDao<TokenEntity> {

    TokenEntity queryByUserId(Long userId);

    TokenEntity queryByToken(String token);

}
```

用户TokenService

```
public interface TokenService {

    TokenEntity queryByUserId(Long userId);

    TokenEntity queryByToken(String token);

    void save(TokenEntity token);

    void update(TokenEntity token);

    /**
     * 生成token
     * @param userId  用户ID
     * @return        返回token相关信息
     */
    Map<String, Object> createToken(long userId);

}
```

数据校验

```
public abstract class Assert {

    public static void isBlank(String str, String message) {
        if (StringUtils.isBlank(str)) {
            throw new RRException(message);
        }
    }

    public static void isNull(Object object, String message) {
        if (object == null) {
            throw new RRException(message);
        }
    }
}
```

API登录授权

`com.migo.api.ApiLoginController`

`@RequestMapping("/api")`旗下的子访问路径有:

`登录`:"login"

自定义忽略Token验证注解

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface IgnoreAuth {
}
```

权限(Token)验证

```
@Component
public class AuthorizationInterceptor extends HandlerInterceptorAdapter {
    @Autowired
    private TokenService tokenService;

    public static final String LOGIN_USER_KEY = "LOGIN_USER_KEY";

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {
        IgnoreAuth annotation;
        if(handler instanceof HandlerMethod) {
            annotation = ((HandlerMethod) handler).getMethodAnnotation(IgnoreAuth.class);
        }else{
            return true;
        }

        //如果有@IgnoreAuth注解，则不验证token
        if(annotation != null){
            return true;
        }

        //从header中获取token
        String token = request.getHeader("token");
        //如果header中不存在token，则从参数中获取token
        if(StringUtils.isBlank(token)){
            token = request.getParameter("token");
        }

        //token为空
        if(StringUtils.isBlank(token)){
            throw new RRException("token不能为空");
        }

        //查询token信息
        TokenEntity tokenEntity = tokenService.queryByToken(token);
        if(tokenEntity == null || tokenEntity.getExpireTime().getTime() <
System.currentTimeMillis()){
            throw new RRException("token失效，请重新登录");
        }

        //设置userId到request里，后续根据userId，获取用户信息
        request.setAttribute(LOGIN_USER_KEY, tokenEntity.getUserId());

        return true;
    }
}
```

API测试接口

```
@RestController
@RequestMapping("/api")
public class ApiTestController {

    /**
     * 获取用户信息
     */
    @GetMapping("userInfo")
    public R userInfo(@LoginUser UserEntity user){

        return R.ok().put("user", user);
    }

    /**
     * 忽略Token验证测试
     */
    @IgnoreAuth
    @GetMapping("notToken")
    public R notToken(){

        return R.ok().put("message", "无需token也能访问。。。");
    }
}
```

API接口模块提供注册接口

```
@RestController
@RequestMapping("/api")
public class ApiRegisterController {
    @Autowired
    private UserService userService;

    /**
     * 注册
     */
    @IgnoreAuth
    @PostMapping("register")
    public R register(@RequestBody UserEntity user){
        Assert.isBlank(user.getMobile(), "手机号不能为空");
        Assert.isBlank(user.getPassword(), "密码不能为空");

        userService.save(user);

        return R.ok();
    }
}
```

注:对外登录见注册接口同一包下代码

自定义登录用户信息注解

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface LoginUser {
}
```

有@LoginUser注解的方法参数，注入当前登录用户

```
public class LoginUserHandlerMethodArgumentResolver implements HandlerMethodArgumentResolver {
    private UserService userService;

    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        return parameter.getParameterType().isAssignableFrom(UserEntity.class) &&
parameter.hasParameterAnnotation(LoginUser.class);
    }

    @Override
    public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer container,
                                  NativeWebRequest request, WebDataBinderFactory factory) throws
Exception {
        //获取用户ID
        Object object = request.getAttribute(AuthorizationInterceptor.LOGIN_USER_KEY,
RequestAttributes.SCOPE_REQUEST);
        if(object == null){
            return null;
        }

        //获取用户信息
        UserEntity user = userService.queryObject((Long)object);

        return user;
    }
}
```

文件上传

文件上传实体类

```
public class SysOssEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    //
    private Long id;
    //URL地址
    private String url;
    //创建时间
    private Date createDate;
```

文件上传DAO

```
public interface SysOssDao extends BaseDao<SysOssEntity> {
}
```

文件上传Service

```
public interface SysOssService {

    SysOssEntity queryObject(Long id);

    List<SysOssEntity> queryList(Map<String, Object> map);

    int queryTotal(Map<String, Object> map);

    void save(SysOssEntity sysOss);

    void update(SysOssEntity sysOss);

    void delete(Long id);

    void deleteBatch(Long[] ids);
}
```

云存储配置信息

```
public class CloudStorageConfig implements Serializable {
    private static final long serialVersionUID = 1L;

    //类型 1：七牛  2：阿里云  3：腾讯云
    @Range(min=1, max=3, message = "类型错误")
    private Integer type;

    //七牛绑定的域名
    @NotBlank(message="七牛绑定的域名不能为空", groups = QiniuGroup.class)
    @URL(message = "七牛绑定的域名格式不正确", groups = QiniuGroup.class)
    private String qiniuDomain;
    //七牛路径前缀
    private String qiniuPrefix;
    //七牛ACCESS_KEY
    @NotBlank(message="七牛AccessKey不能为空", groups = QiniuGroup.class)
    private String qiniuAccessKey;
    //七牛SECRET_KEY
    @NotBlank(message="七牛SecretKey不能为空", groups = QiniuGroup.class)
    private String qiniuSecretKey;
    //七牛存储空间名
    @NotBlank(message="七牛空间名不能为空", groups = QiniuGroup.class)
    private String qiniuBucketName;

    //阿里云绑定的域名
    @NotBlank(message="阿里云绑定的域名不能为空", groups = AliyunGroup.class)
    @URL(message = "阿里云绑定的域名格式不正确", groups = AliyunGroup.class)
    private String aliyunDomain;
    //阿里云路径前缀
    private String aliyunPrefix;
    //阿里云EndPoint
    @NotBlank(message="阿里云EndPoint不能为空", groups = AliyunGroup.class)
    private String aliyunEndPoint;
    //阿里云AccessKeyId
    @NotBlank(message="阿里云AccessKeyId不能为空", groups = AliyunGroup.class)
    private String aliyunAccessKeyId;
    //阿里云AccessKeySecret
    @NotBlank(message="阿里云AccessKeySecret不能为空", groups = AliyunGroup.class)
    private String aliyunAccessKeySecret;
    //阿里云BucketName
    @NotBlank(message="阿里云BucketName不能为空", groups = AliyunGroup.class)
    private String aliyunBucketName;

    //腾讯云绑定的域名
    @NotBlank(message="腾讯云绑定的域名不能为空", groups = QcloudGroup.class)
    @URL(message = "腾讯云绑定的域名格式不正确", groups = QcloudGroup.class)
    private String qcloudDomain;
    //腾讯云路径前缀
    private String qcloudPrefix;
    //腾讯云AppId
    @NotNull(message="腾讯云AppId不能为空", groups = QcloudGroup.class)
    private Integer qcloudAppId;
    //腾讯云SecretId
    @NotBlank(message="腾讯云SecretId不能为空", groups = QcloudGroup.class)
```

```
    private String qcloudSecretId;
    //腾讯云SecretKey
    @NotBlank(message="腾讯云SecretKey不能为空", groups = QcloudGroup.class)
    private String qcloudSecretKey;
    //腾讯云BucketName
    @NotBlank(message="腾讯云BucketName不能为空", groups = QcloudGroup.class)
    private String qcloudBucketName;
    //腾讯云COS所属地区
    @NotBlank(message="所属地区不能为空", groups = QcloudGroup.class)
    private String qcloudRegion;
```

云存储服务抽象类(用到谁家各自拓展实现就好)

```java
public abstract class CloudStorageService {
    /** 云存储配置信息 */
    CloudStorageConfig config;

    /**
     * 文件路径
     * @param prefix 前缀
     * @return 返回上传路径
     */
    public String getPath(String prefix) {
        //生成uuid
        String uuid = UUID.randomUUID().toString().replaceAll("-", "");
        //文件路径
        String path = DateUtils.format(new Date(), "yyyyMMdd") + "/" + uuid;

        if(StringUtils.isNotBlank(prefix)){
            path = prefix + "/" + path;
        }

        return path;
    }

    /**
     * 文件上传
     * @param data    文件字节数组
     * @param path    文件路径，包含文件名
     * @return        返回http地址
     */
    public abstract String upload(byte[] data, String path);

    /**
     * 文件上传
     * @param data    文件字节数组
     * @return        返回http地址
     */
    public abstract String upload(byte[] data);

    /**
     * 文件上传
     * @param inputStream   字节流
     * @param path          文件路径，包含文件名
     * @return              返回http地址
     */
    public abstract String upload(InputStream inputStream, String path);

    /**
     * 文件上传
     * @param inputStream  字节流
     * @return             返回http地址
     */
    public abstract String upload(InputStream inputStream);

}
```

七牛云存储实现

```
public class QiniuCloudStorageService extends CloudStorageService{
    private UploadManager uploadManager;
    private String token;

    public QiniuCloudStorageService(CloudStorageConfig config){
        this.config = config;

        //初始化
        init();
    }

    private void init(){
        uploadManager = new UploadManager(new Configuration(Zone.autoZone()));
        token = Auth.create(config.getQiniuAccessKey(), config.getQiniuSecretKey()).
                uploadToken(config.getQiniuBucketName());
    }

    @Override
    public String upload(byte[] data, String path) {
        try {
            Response res = uploadManager.put(data, path, token);
            if (!res.isOK()) {
                throw new RuntimeException("上传七牛出错：" + res.toString());
            }
        } catch (Exception e) {
            throw new RRException("上传文件失败，请核对七牛配置信息", e);
        }

        return config.getQiniuDomain() + "/" + path;
    }

    @Override
    public String upload(InputStream inputStream, String path) {
        try {
            byte[] data = IOUtils.toByteArray(inputStream);
            return this.upload(data, path);
        } catch (IOException e) {
            throw new RRException("上传文件失败", e);
        }
    }

    @Override
    public String upload(byte[] data) {
        return upload(data, getPath(config.getQiniuPrefix()));
    }

    @Override
    public String upload(InputStream inputStream) {
        return upload(inputStream, getPath(config.getQiniuPrefix()));
    }
}
```

阿里云存储实现

```
public class AliyunCloudStorageService extends CloudStorageService{
    private OSSClient client;

    public AliyunCloudStorageService(CloudStorageConfig config){
        this.config = config;

        //初始化
        init();
    }

    private void init(){
        client = new OSSClient(config.getAliyunEndPoint(), config.getAliyunAccessKeyId(),
                config.getAliyunAccessKeySecret());
    }

    @Override
    public String upload(byte[] data, String path) {
        return upload(new ByteArrayInputStream(data), path);
    }

    @Override
    public String upload(InputStream inputStream, String path) {
        try {
            client.putObject(config.getAliyunBucketName(), path, inputStream);
        } catch (Exception e){
            throw new RRException("上传文件失败，请检查配置信息", e);
        }

        return config.getAliyunDomain() + "/" + path;
    }

    @Override
    public String upload(byte[] data) {
        return upload(data, getPath(config.getAliyunPrefix()));
    }

    @Override
    public String upload(InputStream inputStream) {
        return upload(inputStream, getPath(config.getAliyunPrefix()));
    }
}
```

腾讯云存储实现

```
public class QcloudCloudStorageService extends CloudStorageService{
    private COSClient client;

    public QcloudCloudStorageService(CloudStorageConfig config){
        this.config = config;

        //初始化
        init();
    }

    private void init(){
       Credentials credentials = new Credentials(config.getQcloudAppId(),
config.getQcloudSecretId(),
                config.getQcloudSecretKey());

       //初始化客户端配置
        ClientConfig clientConfig = new ClientConfig();
        //设置bucket所在的区域，华南：gz 华北：tj 华东：sh
        clientConfig.setRegion(config.getQcloudRegion());

       client = new COSClient(clientConfig, credentials);
    }

    @Override
    public String upload(byte[] data, String path) {
        //腾讯云必需要以"/"开头
        if(!path.startsWith("/")) {
            path = "/" + path;
        }

        //上传到腾讯云
        UploadFileRequest request = new UploadFileRequest(config.getQcloudBucketName(), path,
data);
        String response = client.uploadFile(request);

        JSONObject jsonObject = JSON.parseObject(response);
        if(jsonObject.getIntValue("code") != 0) {
            throw new RRException("文件上传失败，" + jsonObject.getString("message"));
        }

        return config.getQcloudDomain() + path;
    }

    @Override
    public String upload(InputStream inputStream, String path) {
       try {
            byte[] data = IOUtils.toByteArray(inputStream);
            return this.upload(data, path);
        } catch (IOException e) {
            throw new RRException("上传文件失败", e);
        }
    }
```

```
    @Override
    public String upload(byte[] data) {
        return upload(data, getPath(config.getQcloudPrefix()));
    }

    @Override
    public String upload(InputStream inputStream) {
        return upload(inputStream, getPath(config.getQcloudPrefix()));
    }
}
```

系统参数相关Key

```
public class ConfigConstant {
    /**
     * 云存储配置KEY
     */
    public final static String CLOUD_STORAGE_CONFIG_KEY = "CLOUD_STORAGE_CONFIG_KEY";
}
```

文件上传Factory

```
public final class OSSFactory {
    private static SysConfigService sysConfigService;

    static {
        OSSFactory.sysConfigService = (SysConfigService)
SpringContextUtils.getBean("sysConfigService");
    }

    public static CloudStorageService build(){
        //获取云存储配置信息
        CloudStorageConfig config =
sysConfigService.getConfigObject(ConfigConstant.CLOUD_STORAGE_CONFIG_KEY,
CloudStorageConfig.class);

        if(config.getType() == Constant.CloudService.QINIU.getValue()){
            return new QiniuCloudStorageService(config);
        }else if(config.getType() == Constant.CloudService.ALIYUN.getValue()){
            return new AliyunCloudStorageService(config);
        }else if(config.getType() == Constant.CloudService.QCLOUD.getValue()){
            return new QcloudCloudStorageService(config);
        }

        return null;
    }

}
```

文件上传Controller

`com.migo.controller.SysOssController`

`@RequestMapping("sys/oss")`旗下的子访问路径有:

`列表`:"/list" `权限设定`:`@RequiresPermissions("sys:oss:all")` `权限设定`:``

`获取云存储配置信息`:"/config" `权限设定`:`@RequiresPermissions("sys:oss:all")`

`保存云存储配置信息`:"/saveConfig" `权限设定`:`@RequiresPermissions("sys:oss:all")`

`上传文件`:"/upload" `权限设定`:`@RequiresPermissions("sys:oss:all")`

`删除`:"/delete" `权限设定`:`@RequiresPermissions("sys:oss:all")`

代码防御

XSS过滤处理

其实就是用`HTMLFilter`来做下处理

```
public class XssHttpServletRequestWrapper extends HttpServletRequestWrapper {
    //没被包装过的HttpServletRequest（特殊场景，需求自己过滤）
    HttpServletRequest orgRequest;
    //html过滤
    private final static HTMLFilter htmlFilter = new HTMLFilter();

    public XssHttpServletRequestWrapper(HttpServletRequest request) {
        super(request);
        orgRequest = request;
    }

    @Override
    public String getParameter(String name) {
        String value = super.getParameter(xssEncode(name));
        if (StringUtils.isNotBlank(value)) {
            value = xssEncode(value);
        }
        return value;
    }

    @Override
    public String[] getParameterValues(String name) {
        String[] parameters = super.getParameterValues(name);
        if (parameters == null || parameters.length == 0) {
            return null;
        }

        for (int i = 0; i < parameters.length; i++) {
            parameters[i] = xssEncode(parameters[i]);
        }
        return parameters;
    }

    @Override
    public Map<String,String[]> getParameterMap() {
        Map<String,String[]> map = new LinkedHashMap<>();
        Map<String,String[]> parameters = super.getParameterMap();
        for (String key : parameters.keySet()) {
            String[] values = parameters.get(key);
            for (int i = 0; i < values.length; i++) {
                values[i] = xssEncode(values[i]);
            }
            map.put(key, values);
        }
        return map;
    }

    @Override
    public String getHeader(String name) {
        String value = super.getHeader(xssEncode(name));
        if (StringUtils.isNotBlank(value)) {
            value = xssEncode(value);

        }
```

```
        return value;
    }

    private String xssEncode(String input) {
        return htmlFilter.filter(input);
    }

    /**
     * 获取最原始的request
     */
    public HttpServletRequest getOrgRequest() {
        return orgRequest;
    }

    /**
     * 获取最原始的request
     */
    public static HttpServletRequest getOrgRequest(HttpServletRequest request) {
        if (request instanceof XssHttpServletRequestWrapper) {
            return ((XssHttpServletRequestWrapper) request).getOrgRequest();
        }

        return request;
    }

}
```

XSS过滤

```
public class XssFilter implements Filter {

   @Override
   public void init(FilterConfig config) throws ServletException {
   }

   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
      XssHttpServletRequestWrapper xssRequest = new XssHttpServletRequestWrapper(
            (HttpServletRequest) request);
      chain.doFilter(xssRequest, response);
   }

   @Override
   public void destroy() {
   }

}
```

SQL注入过滤

注:写完后加入到Query工具类中，详情参加代码

```
public class SQLFilter {

    /**
     * SQL注入过滤
     * @param str  待验证的字符串
     */
    public static String sqlInject(String str){
        if(StringUtils.isBlank(str)){
            return null;
        }
        //去掉'|"|;|\字符
        str = StringUtils.replace(str, "'", "");
        str = StringUtils.replace(str, "\"", "");
        str = StringUtils.replace(str, ";", "");
        str = StringUtils.replace(str, "\\", "");

        //转换成小写
        str = str.toLowerCase();

        //非法字符
        String[] keywords = {"master", "truncate", "insert", "select", "delete", "update",
"declare", "alert", "create", "drop"};

        //判断是否包含非法字符
        for(String keyword : keywords){
            if(str.indexOf(keyword) != -1){
                throw new RRException("包含非法字符");
            }
        }

        return str;
    }
}
```