

1. 唯一标示生成器 (Y9IdGenerator)

简介：生成器提供两生成方式雪花算法和 UUID，如果调用时不指定生成方式，则默认用雪花算法，雪花算法可以用于分布式系统中的数据分片和数据合并，避免了 ID 冲突的问题。

使用示例：

```
String id = Y9IdGenerator.genId();
```

2. 国密 SM 工具类 (SmUtil)

简介：该工具类是使用的 Hutool 的工具类，针对 Bouncy Castle 做了简化包装，用于实现国密算法中的 SM2、SM3、SM4

使用示例：

SM2:

```
public static void main(String[] args) {
    String text = "risesoft";
    // 使用随机生成的密钥对加密或解密
    System.out.println("使用随机生成的密钥对加密或解密====开始");
    SM2 sm2 = SmUtil.sm2();
    // 公钥加密
    String encryptStr = sm2.encryptBcd(text, KeyType.PublicKey);
    System.out.println("公钥加密: " + encryptStr);
    // 私钥解密
    String decryptStr = StrUtil.utf8Str(sm2.decryptFromBcd(encryptStr,
    KeyType.PrivateKey));
    System.out.println("私钥解密: " + decryptStr);
    System.out.println("使用随机生成的密钥对加密或解密====结束");

    // 使用自定义密钥对加密或解密
    System.out.println("使用自定义密钥对加密或解密====开始");

    KeyPair pair = SecureUtil.generateKeyPair("SM2");
    byte[] privateKey = pair.getPrivate().getEncoded();
    byte[] publicKey = pair.getPublic().getEncoded();

    SM2 sm22 = SmUtil.sm2(privateKey, publicKey);
    // 公钥加密
    String encryptStr2 = sm22.encryptBcd(text, KeyType.PublicKey);
    System.out.println("公钥加密: " + encryptStr2);
    // 私钥解密
    String decryptStr2 = StrUtil.utf8Str(sm22.decryptFromBcd(encryptStr2,
    KeyType.PrivateKey));
    System.out.println("私钥解密: " + decryptStr2);
    System.out.println("使用自定义密钥对加密或解密====结束");
}
```

SM3:

```
public static void main(String[] args) {
```

```
String text = "risesoft";
String digestHex = SmUtil.sm3(text);
System.out.println("加密后: " + digestHex);
}
```

3. 对称加密 AES 工具类 (AesUtil)

简介: 高级加密标准(AES,Advanced Encryption Standard)为最常见的对称加密算法。对称加密算法也就是加密和解密用相同的密钥。

使用示例:

```
try {
    String plaintext = "这是明文";
    String aesKey = AesUtil.getSecretKey();
    byte[] encryptByte = AesUtil.encryptByte(plaintext.getBytes(),
aesKey);

    System.out.println("加密后的 Base64 编码的字符串: " +
Base64.getEncoder().encodeToString(encryptByte));
    byte[] decryptByte = AesUtil.decryptByte(encryptByte, aesKey);
    System.out.println("解密后的字符串: " + new String(decryptByte));

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

4. 非对称加密 RSA 工具类 (RsaUtil)

简介: RSA 非对称加密算法中, 有两个密钥: 公钥和私钥。它们是一对, 如果用公钥进行加密, 只有用对应的私钥才能解密; 如果用私钥进行加密, 只有用对应的公钥才能解密。

使用示例:

```
try {
    Map<String, String> map = RsaUtil.initKey();
    String privateKey = map.get("privateKeyString");
    String publicKey = map.get("publicKeyString");
    String plaintext = "这是明文";
    String encString = RsaUtil.encryptByPriKey(plaintext, privateKey);
    System.out.println("私钥加密后字符串: " + encString);
    String decString = RsaUtil.decryptByPubKey(encString, publicKey);
    System.out.println("公钥解密后字符串: " + decString);
} catch (Exception e) {
    e.printStackTrace();
}
```

5. 消息摘要、加密工具类 (Y9MessageDigest)

简介: 主要用于数字底座密码等不可解密的字段的加密。

使用示例:

```
try {
    String password = "zheshimima";
    String sha1 = Y9MessageDigest.SHA1(password);
    System.out.println("sha1 加密后: "+sha1);
}
```

```

        String sha256 = Y9MessageDigest.SHA256(password);
        System.out.println("md5 加密后: "+sha256);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

6. 数据库元数据工具类 (DbMetaDataUtil)

简介：主要用于获取数据源方言名称，数据库产品名称，批量执行 ddl 语句等
使用示例：

```

/**
 * 初始化数据库结构
 *
 * @param dsId 数据源 id
 * @param systemId 系统 id
 * @return
 */
@RiseLog(operationType = OperationTypeEnum.ADD, operationName = "初始化数据库结构")
@RequestMapping(value = "/dbSchemaSync")
public Y9Result<String> dbSchemaSync(@RequestParam String dsId,
    @RequestParam String systemId) throws Exception {
    SystemSqlFile systemSqlFile =
systemSqlFileService.findOne4MaxVersion(systemId, SqlFileTypeEnum.ALL);
    if (null == systemSqlFile) {
        return Y9Result.failure("同步失败：该系统不存在全量的 sql 文件");
    }
    String s =
y9FileStoreService.downloadFileToString(systemSqlFile.getFileStoreId());
    DataSource dataSource = y9DataSourceService.getDataSource(dsId);
    List<String> sqlList = Y9FileUtil.loadSql(s);
    DbMetaDataUtil.batchExecuteDdl(dataSource, sqlList);

    systemSqlFile.setSync(true);
    systemSqlFileService.save(systemSqlFile);
    return Y9Result.successMsg("初始化数据库结构成功！");
}

```

7. 数据库表操作工具类 (Ddl)

简介：主要用于数据库表重命名、表新增列、修改表的列、删除表的列、删除表等操作。
使用示例：

```

/**
 * 重命名表
 *
 * @param dsId 数据库 id
 * @param tableNameOld 旧表名称
 * @param tableNameNew 新表名称

```

```

    * @return
    * @throws Exception
    */
    @RequestMapping(value = "/table/rename")
    public Y9Result<String> tableRename(@NotBlank String dsId, @NotBlank String
tableNameOld,
        @NotBlank String tableNameNew) throws Exception {
        DataSource dataSource = y9DataSourceService.getDataSource(dsId);
        Ddl.renameTable(dataSource, tableNameOld, tableNameNew);
        return Y9Result.success("操作成功!");
    }

```

8. 数据库分页工具类（SqlPaginationUtil）

简介：针对不同的数据库，自己写 sql 查询时，便于分页处理
使用示例：

```

/**
 * 分页获取指定的表数据
 * @param page 页数
 * @param rows 条数
 * @param dsId 数据源 id
 * @param tableName 表名称
 * @return
 * @throws Exception
 */
@RequestMapping(value = "/getTableData")
public Y9Page<Map<String, Object>> getTableData(int page, int rows,
@NotBlank String dsId,
        @NotBlank String tableName) throws Exception {
    Long totalCount;
    DataSource dataSource = y9DataSourceService.getDataSource(dsId);
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
    totalCount = jdbcTemplate.queryForObject("select count(*) from " +
tableName, Long.class);
    List<Map<String, Object>> items = jdbcTemplate.queryForList(
        SqlPaginationUtil.generatePagedSql(dataSource, "select * from " +
tableName, (page - 1) * rows, rows));
    return Y9Page.success(page, totalCount == 0 ? 1 :
(int)Math.ceil((double)totalCount / (double)rows), totalCount,
        items);
}

```

9. JSON 工具类（Y9JsonUtil）

简介：基于 Jackson 实现的工具类，主要用于将对象转换为字符串，将字符串转为对象，将字符串转为数组，将字符串转为 Map 等功能。

使用示例：

```

public static void main(String[] args) {

```

```

String s = "[\"aaa\", \"bbb\"]";

String[] ss = Y9JsonUtil.readValue(s, String[].class);
for (String item : ss) {
    System.out.println("array item==" + item);
}

ss = Y9JsonUtil.readArray(s, String.class);
for (String item : ss) {
    System.out.println("array2 item==" + item);
}

List<String> list = Y9JsonUtil.readList(s, String.class);
for (String item : list) {
    System.out.println("list item==" + item);
}

String s1 = "{\"aaa\":\"111\", \"bbb\":\"222\"}";
HashMap<String, String> map1 = Y9JsonUtil.readHashMap(s1, String.class,
String.class);
System.out.println("aaa==" + map1.get("aaa"));

String s2 =
"[{\"aaa\":\"1a\", \"bbb\":\"1b\"}, {\"aaa\":\"2a\", \"bbb\":\"2b\"}]";
List<Map<String, Object>> list2 = Y9JsonUtil.readListOfMap(s2);
for (Map<String, Object> map : list2) {
    System.out.println("bbb==" + map.get("bbb"));
}

List<Map<String, Object>> list3 = Y9JsonUtil.readListOfMap(s2,
String.class, Object.class);
for (Map<String, Object> map : list3) {
    System.out.println("bbb==" + map.get("bbb"));
}

String s3 =
"{\"aaa\":\"111\", \"bbb\": [{\"q\":\"q1111\", \"t\":\"t1111\"}, {\"q\":\"q2222\", \"t\":\"t2222\"}]}";
HashMap<String, Object> map2 = Y9JsonUtil.readHashMap(s3, String.class,
Object.class);
System.out.println("aaa==" + map2.get("aaa"));
System.out.println("bbb==" + map2.get("bbb"));

List<Map<String, Object>> list4 =

```

```

Y9JsonUtil.readListOfMap(Y9JsonUtil.writeValueAsString(map2.get("bbb")),
String.class, Object.class);
    for (Map<String, Object> map : list4) {
        System.out.println("q==" + map.get("q"));
    }
}

```

10. 实体新建事件工具类（Y9EntityCreatedEvent）

简介：该工具类主要用于某个实体进行新增操作时，可以发布一个携带该实体对象的事件，事件的监听者监听到这个事件后，触发自定义逻辑。

使用示例：

```

public Y9PersonsToGroups addY9PersonsToGroups(String personId, String groupId,
Integer maxGroupsOrder,
    Integer maxPersonsOrder) {
    Y9PersonsToGroups y9PersonsToGroups = new Y9PersonsToGroups();
    y9PersonsToGroups.setId(Y9IdGenerator.genId());
    y9PersonsToGroups.setGroupId(groupId);
    y9PersonsToGroups.setPersonId(personId);
    y9PersonsToGroups.setGroupOrder(maxGroupsOrder);
    y9PersonsToGroups.setPersonOrder(maxPersonsOrder);
    y9PersonsToGroups =
y9PersonsToGroupsRepository.save(y9PersonsToGroups);

    Y9Person person = y9PersonManager.getById(personId);
    Y9Group group = y9GroupManager.getById(groupId);
    Y9MessageOrg<PersonsGroups> msg =
        new Y9MessageOrg<>(ModelConvertUtil.convert(y9PersonsToGroups,
PersonsGroups.class),
            Y9OrgEventConst.RISEORGEVENT_TYPE_GROUP_ADDPERSON,
Y9LoginUserHolder.getTenantId());
    Y9PublishServiceUtil.persistAndPublishMessageOrg(msg, "添加用户组人员
",
        group.getName() + "添加用户组成员" + person.getName());

    Y9Context.publishEvent(new
Y9EntityCreatedEvent<>(y9PersonsToGroups));

    return y9PersonsToGroups;
}

```

11. 实体删除事件工具类（Y9EntityDeletedEvent）

简介：该工具类主要用于某个实体进行新增操作时，可以发布一个携带该实体对象的事件，事件的监听者监听到这个事件后，触发自定义逻辑。

使用示例：

```

public void delete(Y9PersonsToGroups y9PersonsToGroups) {

```

```

        y9PersonsToGroupsRepository.delete(y9PersonsToGroups);

        Y9Person person =
y9PersonManager.getById(y9PersonsToGroups.getPersonId());
        Y9Group group =
y9GroupManager.getById(y9PersonsToGroups.getGroupId());
        Y9MessageOrg<PersonsGroups> msg =
            new Y9MessageOrg<>(ModelConvertUtil.convert(y9PersonsToGroups,
PersonsGroups.class),
                Y9OrgEventConst.RISEORGEVENT_TYPE_GROUP_REMOVEPERSON,
Y9LoginUserHolder.getTenantId());
        Y9PublishServiceUtil.persistAndPublishMessageOrg(msg, "移除用户组人员",
            group.getName() + "移除用户组成员" + person.getName());

        Y9Context.publishEvent(new
Y9EntityDeletedEvent<>(y9PersonsToGroups));
    }

```

12. 实体更新事件工具类（Y9EntityUpdatedEvent）

简介：该工具类主要用于某个实体进行新增操作时，可以发布一个携带该实体对象的事件，事件的监听者监听到这个事件后，触发自定义逻辑。

使用示例：

```

public Y9Person changeDisabled(String id) {
    Y9Person y9Person = this.getById(id);
    boolean disabled = !y9Person.getDisabled();
    y9Person.setDisabled(disabled);
    final Y9Person savedPerson = y9PersonManager.save(y9Person);
    if (disabled) {
        // 禁用人员的时候，将人员岗位移除
        y9PersonsToPositionsManager.deleteByPersonId(id);
    }
    Y9Context.publishEvent(new Y9EntityUpdatedEvent<>(y9Person,
savedPerson));
    return savedPerson;
}

```

13. 远程调用工具类（RemoteCallUtil）

简介：该工具类主要用于对第三方接口进行调用，调用的接口可以是 GET、POST 带请求头或者不带请求头等多种方式。

使用示例：

```

@Override
    public Y9Result<Object> changeStatus(String tenantId, String userId,
Integer docId, Integer status) {
        try {
            String url = Y9CommonApiUtil.cmsBaseUrl + "/article/changeStatus";

```

```

        List<NameValuePair> params = new ArrayList<>();
        params.add(new NameValuePair(TENANT_ID, tenantId));
        params.add(new NameValuePair("userId", userId));
        params.add(new NameValuePair("docId", docId + ""));
        params.add(new NameValuePair("status", status + ""));
        return RemoteCallUtil.postCallRemoteService(url, params,
Y9Result.class);
    } catch (Exception e) {
        e.printStackTrace();
        return Y9Result.failure(e.getMessage());
    }

}

```

14. 实体属性值复制工具类（Y9BeanUtil）

简介：主要用于将更新过字段的实体类拷贝到老的数据库查出的实体类中。

使用示例：

```

public Y9Department move(String deptId, String parentId) {
    Y9Department originDepartment = getById(deptId);
    Y9Department updatedDepartment = new Y9Department();
    Y9BeanUtil.copyProperties(originDepartment, updatedDepartment);

    checkMoveTarget(originDepartment, parentId);

    Y9OrgBase parent =
compositeOrgBaseManager.getOrgUnitAsParent(parentId);
    updatedDepartment.setParentId(parent.getId());

    updatedDepartment.setDn(OrgLevelConsts.getOrgLevel(OrgTypeEnum.DEPARTMENT) +
updatedDepartment.getName()
        + OrgLevelConsts.SEPARATOR + parent.getDn());
    updatedDepartment.setGuidPath(parent.getGuidPath() +
OrgLevelConsts.SEPARATOR + updatedDepartment.getId());

    updatedDepartment.setTabIndex(compositeOrgBaseManager.getMaxSubTabIndex(par
entId));
    final Y9Department savedDepartment =
y9DepartmentManager.save(updatedDepartment);

    compositeOrgBaseManager.recursivelyUpdateProperties(savedDepartment);

    Y9Context.publishEvent(new Y9EntityUpdatedEvent<>(originDepartment,
savedDepartment));
}

```



```

        TransactionSynchronizationManager.registerSynchronization(new
TransactionSynchronization() {
            @Override
            public void afterCommit() {
                Y9MessageOrg<Department> msg =
                    new
Y9MessageOrg<>(Y9ModelConvertUtil.convert(savedDepartment,
Department.class),
                    Y9OrgEventConst.RISEORGEVENT_TYPE_UPDATE_DEPARTMENT,
Y9LoginUserHolder.getTenantId());
                Y9PublishServiceUtil.persistAndPublishMessageOrg(msg, "移动部
门", "移动" + savedDepartment.getName());
            }
        });

        return savedDepartment;
    }

```

15. 当前登录用户工具类（Y9LoginUserHolder）

简介：数字底座的每个系统都会有一个处理当前登录用户的过滤器，从登录的 token 中解析到当前登录的用户信息后，会把得到的信息设置到 Y9LoginUserHolder 中，后续的请求可以直接从该对象中获取当前登录用户的详细信息。

使用示例：

```

/**
 * 获取当前登陆用户有权限的菜单
 *
 * @return
 */
@GetMapping("/getMenus")
@RiseLog(operationName = "获取当前登陆用户有权限的菜单", moduleName = "日志
测试系统")
public Y9Result<List<Menu>> getMenus() {
    // 租户 id
    String tenantId = Y9LoginUserHolder.getTenantId();
    // 用户 id
    String personId = Y9LoginUserHolder.getPersonId();
    // 当前系统所有应用
    List<App> appList =
appApi.listBySystemName(Y9Context.getSystemName()).getData();
    // 获取当前系统第一个应用，当前用户有权限的菜单
    List<Menu> menuList = new ArrayList<>();
    if (!appList.isEmpty()) {
        menuList = personResourceApi.listSubMenus(tenantId, personId,
AuthorityEnum.BROWSE, appList.get(0).getId())
            .getData();
    }
}

```

```
}  
return Y9Result.success(menuList);  
}
```