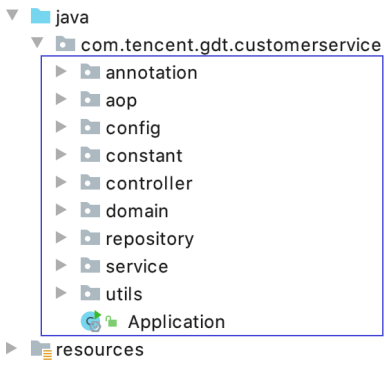
CS客户中心接口

整体认识：



1. annotation。存储自定义注解，包括`@UpdateESAnnotation`和`@NeedSyncAddAnnotation`注解，用于标注在执行过后，需要对ES进行更新的方法。比如Advertiser的增删改。
2. aop。存储定义的切面。利用spring的aop，在数据更改之后，将accountId更新到Redis队列中，供ES服务消费。
3. config。包含一些配置类，没有完全看完。`api.generate`配置Swagger，`cache`中包含配置操作redis的template，`db`配置数据库操作（JOOQ相关配置）,以及一些filter和interceptor。
4. constant。主要包含一些enum类，定义一些枚举类。
5. controller。各种controller接口。
6. domain。主要包含一些实体类。`api.generate`是Swagger生成的实体类，另外还有一些Model。`db`下是跟数据库表相关的实体类....
7. repository。dao层，利用JOOQ与数据库交互。
8. service。service层。
9. utils。一些工具类。

一个接口为例，查看代码如何运行，使用的示例接口：`/advertiser/add` （添加广告主账号）

（1）Controller层

Swgger生成实体类和controller接口。controller相关的类在`controller`包下，生成的相关实体类在`domain/api`下。一些enum类放在`constant`下，定义了一些常量，包括一些需要用到的账号ID等。

1. 接口定义（删去了跟文档相关以及一些日志等不影响阅读的代码）：

`com.tencent.gdt.customerservice.controller.api.generate.AdvertiserApi`

```
@RequestMapping(value = "/advertiser/add",
    produces = { "application/json" },
    consumes = { "application/json" },
    method = RequestMethod.POST)
ResponseEntity<AddAccountRet> addAdvertiser(@Valid @RequestBody
AddAdvertiserRequest body);
```

2. 接口实现：

`@UpdateESAnnotation`和`@NeedSyncAddAnnotation`注解用于在数据更改之后，将accountId更新到Redis队列中，供ES服务消费，后面会看具体如何实现。

添加的核心逻辑在add方法中。

`com.tencent.gdt.customerservice.controller.api.generate.AdvertiserApiController`

```
@Override
@UpdateESAnnotation
@NeedSyncAddAnnotation
public ResponseEntity<AddAccountRet> addAdvertiser(@Valid @RequestBody
AddAdvertiserRequest body) {
    String accept = request.getHeader("Accept");
    if (accept != null && accept.contains("application/json")) {
      // 核心逻辑在add()中
        AddAccountRet response = add(body);
        return new ResponseEntity<AddAccountRet>(response, HttpStatus.OK);
    } else {
        throw new CustomerApiException(ApiReturnCode.HEADER_ACCEPT_MISSING,
"Accept 'application/json' was expected");
    }
}
```

3. add()主要进行一系列的参数处理、一些没有的参数设置默认值、参数校验等。

```
private AddAccountRet add(AddAdvertiserRequest body) {
    // 检查是否有添加“财务标签”的权限
    checkFinancialTagThrown(body.getBornClientAppId(),
body.getFinancialTagId(), true);
    // 检查“销售划分”是否正常
    initializeSaleSegmentIfNeed(body);
    // 检查“外部行业”，并将其映射为“内部行业”
    initializeIndustryIdIfNeed(body);
    // 微信子客一些字段继承直客
    setMpSubAdvertiser(body);
    // ......
```

```
    // 核心代码
    // 新建一个账号
    ret = addAdvertiser(body, user);
```

4. 调用Service层方法

`com.tencent.gdt.customerservice.controller.api.generate.AdvertiserApiController`

```
private AddAccountRet addAdvertiser(AddAdvertiserRequest body, User user) {
    AddAccountRet ret;
    // 调用service的地方，传入的user是rootUserId指定的user，没有则为null
    ret = advertiserService.addAdvertiser(body, user);
    if (RetCodeUtils.isNotOk(ret)) {
        //新建账户失败
        // ......
    }
    return ret;
}
```

（2）Service层

（跟新旧表相关的有account、advertiser、certification、operator、user）同时操作新旧表。最初有一套旧表，很多系统都在用。 后面才有了客户中心， 客户中心设计了一套新的模型来支撑业务，大部分业务方都接入了cs，但是还有一些业务方依旧是在读老库。 所以cs现在是双读双写的。保证一致性。

`com.tencent.gdt.customerservice.service.advertiser.AdvertiserServiceCombo`

```
// 这里的参数advertiser是request中的请求参数
// user是rootUserId指定用户
@Override
public AddAccountRet addAdvertiser(AddAdvertiserRequest advertiser, User user)
{
    long accountId;
    AddAccountRet retCode = null;

    // 获取账号ID
 if(UserUtils.isWechatAppId(user)||UserUtils.isWechatSpid(user)||UserUtils.isW
echatAppidSpid(user)){
        accountId = user.getUserId();
    }else {
        accountId = idService.newAccountId();
    }

    /**
     *  先写新库再写旧库是因为流水服务会根据cs的插库的流水更新financial_tag，如果先写旧库
的的话会出现问题
```

```
     */
    ConsistencyGuard cg = new ConsistencyGuard("addAdvertiser",
            ConsistencyGuard.Parameters.of(advertiser).add("newAccountId",
accountId)
                    .add("operatedTime",
TimeUtils.convertDateToString(TimeUtils.now(), TimeFmt.getTimefmt())));
    // 写新库
    retCode = cg.keepConsistency(() -> {
        return advertiserServiceNew.addAdvertiser(advertiser, accountId);
    });

    // 写旧库
    if (RetCodeUtils.isOk(retCode) && advertiserDoubleWrite) {
        retCode = cg.keepConsistencyOld(() -> {
            return advertiserServiceOld.addAdvertiser(advertiser, accountId,
user);
        });
    }

    return retCode;
}
```

（3）持久层

jooq持久层。dao相关操作在都在`repository`下，比如利用`DSLContext`编写sql与数据库交互。

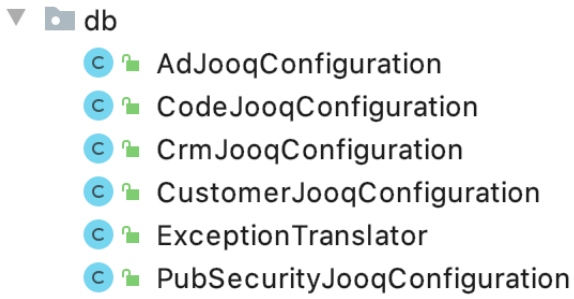
Advertiser与新表的持久层：

`com.tencent.gdt.customerservice.repository.AdvertiserRepository`

```
@DIAccessMo(table = "Advertiser_", db = "Customer_")
public int addAdvertiser(Long accountId, Long agencyAccountId,......)  {
        Timestamp now = TimeUtils.now();
        return customerDslContext.insertInto(
                ADVERTISER_, ADVERTISER_.ACCOUNTID,
ADVERTISER_.AGENCYACCOUNTID,.....)
          .values(accountId,agencyAccountId == null ? 0 : agencyAccountId,
                ......)
          .execute();
```

Java中写SQL的`JOOQ`

JOOQ相关配置在`com.tencent.gdt.customerservice.config.db`下，又相应的将DSLContext添加到容器中的配置。



（4）Redis队列添加Id的实现

Redis中只保存信息发生改变的账号的id，ES服务进行消费，会从DB中查，然后放到es中。

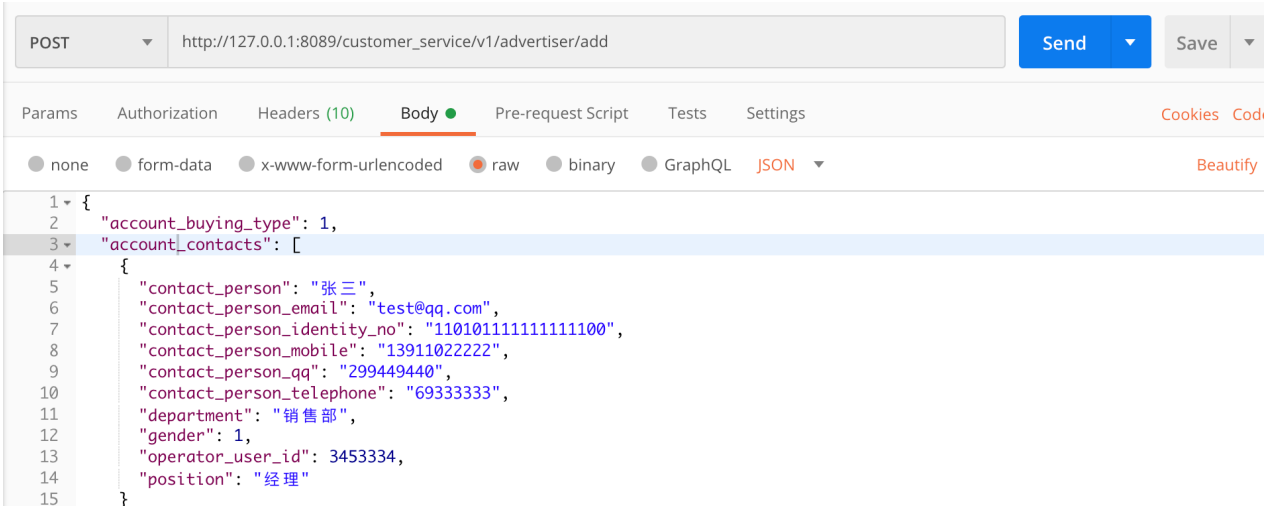
1. 任何对数据库的`增删改`操作都需要把最新的数据更新到elasticsearch中，用于查询，支持高并发。所以controller接口中，与数据库相关的增删改的接口上标注`@UpdateESAnnotation`和`@NeedSyncUpdateAnnotation`这几个自定义注解。
2. **定义切面**。`com.tencent.gdt.customerservice.aop.SyncRedisAspect`是一个切面，对所有`@NeedSyncAddAnnotation`的方法执行后，执行处理逻辑。用于在一个方法执行结束后（AfterReturning），同步accountId, accountType到Redis。

```
@AfterReturning(value =
"@annotation(com.tencent.gdt.customerservice.annotation.NeedSyncAddAnnotat
ion)",
            returning = "response")
    public void doAfterAddController(JoinPoint joinPoint, ResponseEntity
response) {
        ...
        SyncMsg msg = new SyncMsg();
        try {
            Method m = ret.getClass().getMethod("getData");
            Object data = m.invoke(ret);
            if (data != null) {
                m = data.getClass().getMethod("getAccountId");
                msg.setAccountId((Long) m.invoke(data));
            }
        } catch (Exception e) {
            LOG.error("doAfterAddController throws an exception! e = ",
e);
        }
        syncProducer.syncAccount(msg);
        // syncAccount():  RedisUtil.sadd(setKey,
JSONObject.toJSONString(msg));操作Redis
    }
```

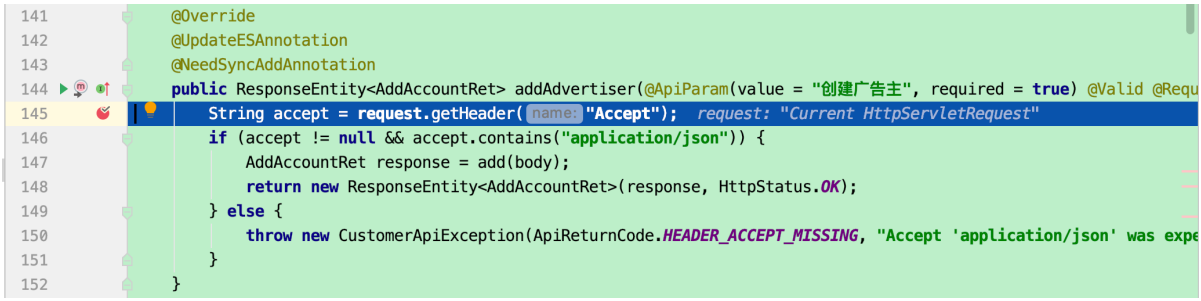
3. `com.tencent.gdt.customerservice.aop.UpdateESAspect`进行Monitor监控，这里还没看。

（5）调试

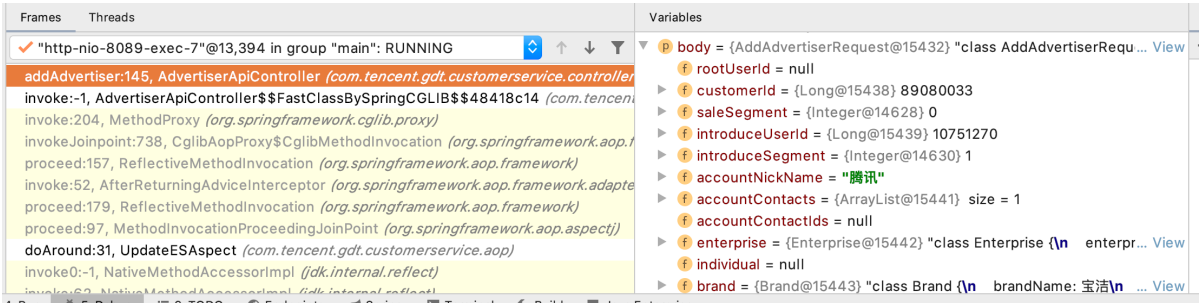
使用postman发请求，在代码中调试：`advertiser/add`接口的处理流程。（断点都是提前打好的）



1. 首先在controller方法上加断点，代码停住

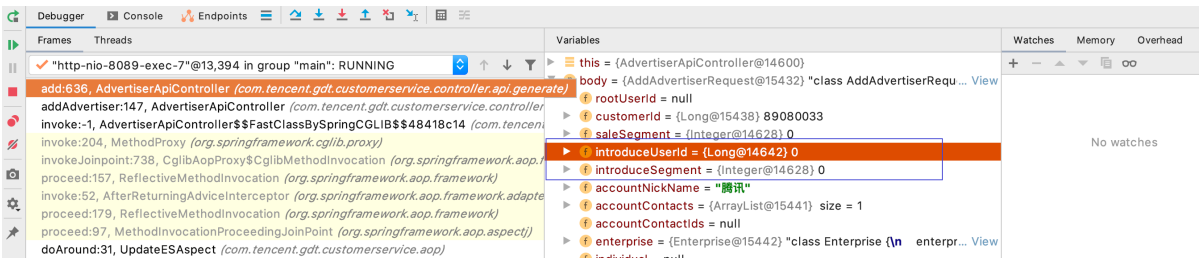


查看body中参数都没有问题，就是在postman中输入的参数



2. 接下来，继续走，代码停到了add()方法，进行一系列的`校验`，或者为一些空值赋予默认值。

例如，在上面的校验等过程走完后，body参数中，介绍人和介绍部门被初始化为0



3. 下一个断点：调用service的`addAdvertiser`，新旧表插入

   `com.tencent.gdt.customerservice.service.advertiser.AdvertiserServiceCombo`

```
298     @Override
299     public AddAccountRet addAdvertiser(AddAdvertiserRequest advertiser, User user) {  advertiser: "class AddAdvertise
300         long accountId;
301         AddAccountRet retCode = null;
302
303         // 获取账号ID
304         if(UserUtils.isWechatAppId(user)||UserUtils.isWechatSpid(user)||UserUtils.isWechatAppidSpid(user)){
305             accountId = user.getUserId();
306         }else {
```

4. 插入完成，controller方法结束。配置的切面起作用（`@NeedSyncAddAnnotation`）：将信息更新的账号Id放到Redis中：

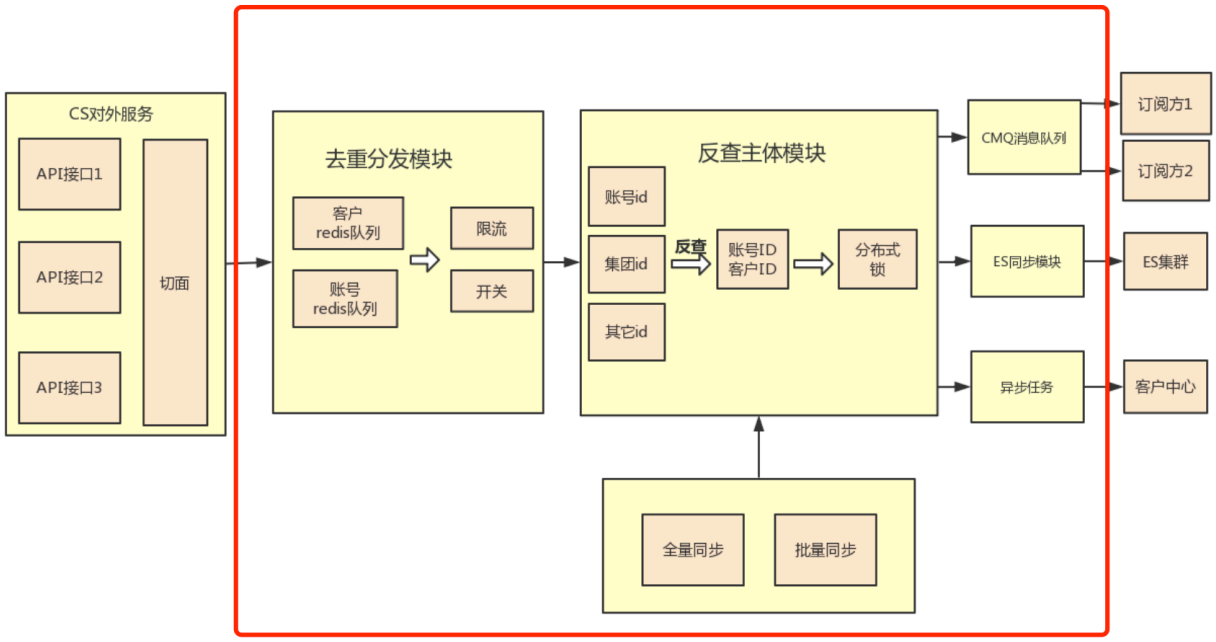
```
51      @AfterReturning(value = "@annotation(com.tencent.gdt.customerservice.annotation.NeedSyncAddAnnotation)",
52              returning = "response")  response: "<200 OK,class AddAccountRet {\n    class ReturnCode {\n        code
53      public void doAfterAddController(JoinPoint joinPoint, ResponseEntity response) {  joinPoint: "execution(Respons
54          if (joinPoint.getArgs().length < 1) {  joinPoint: "execution(ResponseEntity com.tencent.gdt.customerservic
55              return;
56          }
57
58          Object ret = response.getBody();
59          if (ret == null) {
```

   这里的66行就是Redis的添加操作，因为外网，连接不了。信息更改后，需要手动调用ES的服务接口更新es。

```
53      public void syncAccount(SyncMsg msg) {  msg: "SyncMsg{accountId=1585468729, accountType=null, createdTime=null}"
54          // 如果没有开启同步，直接返回
55          if (isSync == false) {  isSync: true
56              return;
57          }
58
59          try {
60              if (msg.getAccountId() == null) {
61                  LOGGER.error("syncAccount account change log error! illegal param, msg = {}", JSONObject.toJSONString(
62                  return;
63              }
64
65              msg.setCreatedTime(TimeUtils.now().getTime());
66  //              RedisUtil.sadd(setKey, JSONObject.toJSONString(msg));
67          } catch (Exception e) {
```

ES服务

CS异步处理服务-数据流程图



（1）手动更新es

可以通过redis类似于消息队列来获取需要更新的accountId等。也可以通过访问指定接口，带上id参数，可手动更新es信息：

1. contoller：`/sync/sub/account`，更新account

`com.tencent.gdt.customerservice.es.controller.SyncController`

```
// 参数格式：  123423;23432    即accountId1;accountId2
@ResponseBody
@RequestMapping(value = "/sync/sub/account", method = RequestMethod.POST,
produces = "application/json;charset=UTF-8")
public String syncSubAccount(@RequestParam(name = "accountIds") String
accountIds) {
  // ......
  new Thread(new Runnable() {
    @Override
    public void run() {
      // 新建线程，使用线程池对每一个ID进行同步操作
      sycnAccountToEsProcessor.syncSubAccountToEs(accountIdList);
    }
  }).start();
  // ......
}
```

2. 将每一个账号的同步操作当作一个任务，交给线程池处理

`com.tencent.gdt.customerservice.es.processor.BatchSyncAccountToEsProcessor`

```
public void syncSubAccountToEs(List<Long> accountIdList) {
  // ......
  for (Long accountId : accountIdList) {
    rateLimiter.acquire();
    // 线程池，executorService，提交一个任务
    executorService.submit(new Runnable() {
      @Override
      public void run() {
        // 将每一个账号的同步操作当作一个任务
        accountServiceCombo.syncAccount(accountId, null, null);
      }
    });
  }
  // ......
}
```

3. Service逻辑

`com.tencent.gdt.customerservice.es.service.AccountServiceComboImpl`

```
public Boolean syncAccount(Long accountId, Integer accountType, Long mdmId,
boolean needSyncMq) {
  long startTime = System.currentTimeMillis();
  // 查询是否有accoutType
  if (accountType == null) {
    accountType = accountServiceNew.getAccountType(accountId);
    if (accountType == null) {
      accountType = accountServiceOld.getServiceAccountType(accountId);
    }
  }
  // 同步到CMQ队列？？干啥的
  if(needSyncMq) {
    cmqService.pushToAccountQueue(accountId, accountType);
  }
  // 放到ES中
  boolean success = pushAccountToEsWithLock(accountId, mdmId);
  // 报告监控
  MoUtils.reportEsSyncData("syncAccount", System.currentTimeMillis() -
startTime, success);
  return success;
}
```

4. 获得分布式锁

`com.tencent.gdt.customerservice.es.service.AccountServiceComboImpl`

- redis分布式锁：（LOCK_ACCONT + accountId）"lockAccout139857184"
- **DEFAULT_EXPIRE_TIME**，锁过期时间，5s

```
private Boolean pushAccountToEsWithLock(Long accountId, Long mdmId) {
    try {
      return synchronizer.synchronize(LOCK_ACCONT + accountId,
DEFAULT_EXPIRE_TIME, () -> {
        return pushAccountToEsWithMmdId(accountId, mdmId);
      });
    } catch (Exception e) {
      // 日志等错误信息.....
    }
  }
```

分布式同步器：`com.tencent.gdt.customerservice.es.utils.DistributedSynchronizer`

```
public <V> V synchronize(String lockName,int expireAfterMilSec,Callable<V>
callable) {
  long threadId=Thread.currentThread().getId();
  long timestamp=System.currentTimeMillis();

  long expireTime=timestamp+expireAfterMilSec*1000L;
  String lockValue=new
StringBuilder().append(timestamp).append("_").append(threadId).toString();
  V result=null;
  try {
    while(true) {
      // 尝试加锁，获得锁之后，做更新操作
      if(RedisUtil.putIfNotExists(lockName,lockValue, expireAfterMilSec)) {
        result=callable.call();
        break;
      }
      // 超时结束
      if(System.currentTimeMillis()>expireTime) {
        break;
      }
    }
    // ......
  return result;
}
```

5. 更新es

```
/**
   * 如果mdmId 为null，则会去请求mdmId的值。
   */
private Boolean pushAccountToEsWithMmdId(long accountId, Long mdmId) {
  boolean success = false;

  Account account = null;
  // 这里尝试通过accountId获取account的值和mdmId
```

```
  try {
    account = accountServiceNew.getAccount(accountId);
    // ......
    //如果传入的mdmId为null，则去ckv redis中去取
    if (mdmId == null) {
      mdmId = getMdmId(accountId);
    }
    // ......
    account.setMdmId(mdmId);
    }
  }
  // 将数据转为map属性值对
  String json =
JSONObject.toJSONString(account,SerializerFeature.WriteMapNullValue);
  Map<String, Object> dataMap = JSONObject.parseObject(json);
  for (int index = 0; index < MAX_RETRY_TIME; index++) {
    // 核心语句。利用esService更新elasticsearch，供查询搜索
    UpdateResponse response = esService.upsertAccount(accountId, dataMap);
  }
  // ......
  return success;
}
```

@Scheduled定时任务

SpringBoot启动类加`@EnableScheduling`注解，开启定时任务功能。Spring定时器(定时执行一次或定时轮询执行一段代码)，注解在方法上，用于将方法设置为调度任务。

在ES服务中，定义了这样一个任务

`com.tencent.gdt.customerservice.es.task.RedisSizeCheckTask`

```
//1分钟执行一次
@Scheduled(cron = "0 */1 * * * ?")
public void scheduled() {
  checkSize(redisKeyConfig.getAccountListKey());
  checkSize(redisKeyConfig.getCustomerListKey());
  checkSize(redisKeyConfig.getAddtionalDataListKey());
  checkSize(redisKeyConfig.getMdmIdListKey());
}
// redis队列大小检测，监控上报
void checkSize(String redisKey) {
  long size = RedisUtil.getSetSize(redisKey);
  LOGGER.info("customer redis size = {}, redisKey = {}", size, redisKey);

  MoUtils.reportQueueSize(redisKey, size);

  if (size > MAX_SIZE) {
```

```
    String msg = String.format("customer redis size is too big! size = %d ,
redisKey = %s", size, redisKey);
    LOGGER.error(msg);

 Tnm2ServiceFactory.getServiceByEnv().agentRepStr(Tnm2Const.REDIS_SET_SIZE_TOO
_BIG_ID, msg);
  }
}
```

（2）Redis队列

----------end----------

勿令妄动，静重如山。不要轻率的行动，自古以来，为男人者都要沉着冷静，举止要稳如泰山。