# credit_risk_engine_v1.0.54 [ALL][ALL]Holmes 2.0-工单可配置化



## 1、背景及PRD

背景: Holme后台中支持在反欺诈环节输出工单加验，让local 在后台进行工单人审操作；目前新增工单的需求需要单独提需求定制开发；但在实际业务中risk team 可能经常需要快速新增工单应用于线上，或者对个别工单的展示内容进行灵活的修改（比如：工单展示新增图片对比、新增/修改拒绝工单时的拒绝原因和拒绝码等；）现有的工单定制化需求无法快速响应业务的此类需求；需要有更灵活快捷的方式承接工单灵活配置需求。holmes 后台提供工单可配置化功能，支持risk team 自行操作配置工单。

PRD: [ALL][ALL]Holmes 2.0-工单可配置化

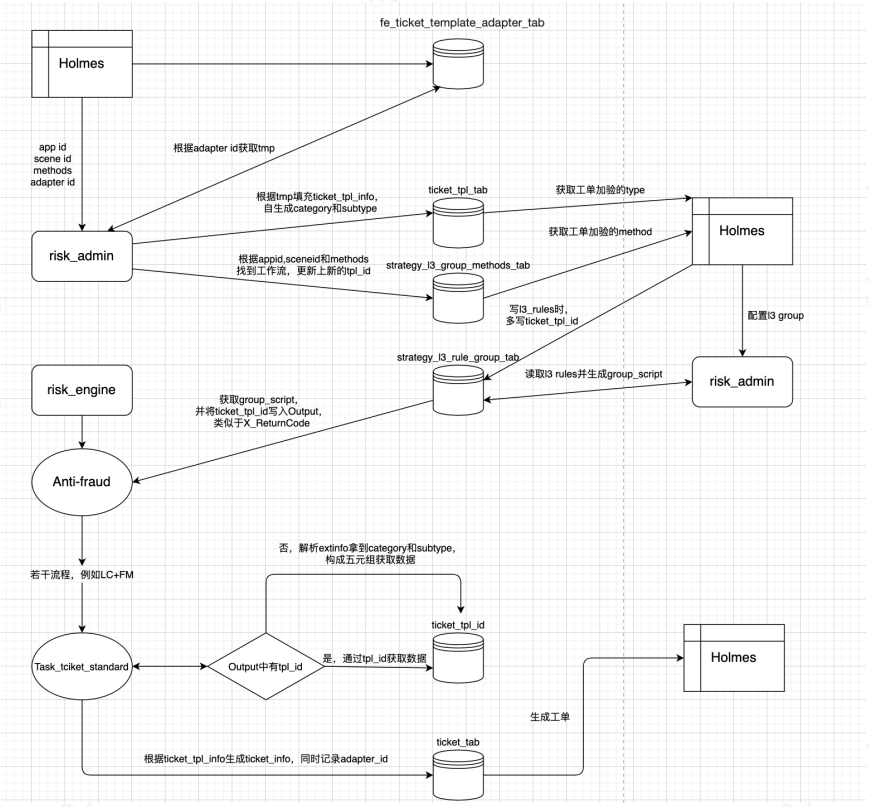
涉及、关联到的功能点

- risk_admin新增工单模板新增、修改接口，新增工单模板时，后端在 ticket_tpl_tab 和 strategy_l3_group_methods_tab 插入新数据，且将两张表通过 ticket_tpl_id 关联起来；修改时，直接修改 ticket_tpl_tab 这个表中的记录。
- risk_admin修改，在配置group时，前端将type（即ticket_tpl_tab表）对应的ticket_tpl_id传给后端，写入 strategy_l3_rule_group_tab 表中的 l3_rules 字段，并最终解析到group_script中（类似于X_ReturnCode）。在反欺诈流程结束后，将 ticket_tpl_id 写入Output。
- risk_engine修改，task_ticket_standard.go 中修改为 先从 task.Var的反欺诈output中获取 ticket_tpl_id，获取不到，则从 ext_info 里获取category和subtype，再构成五元组去ticket_tpl_tab中查询数据。
- db表结构变更，ticket_tpl_tab、ticket_tab 两个表新增 adapter_id 字段，ticket_tpl_tab 始终存储最新的adapter_id，ticket_tab 记录工作流运行时（ticket_tpl_tab）的adapter_id。

国家业务举例

| | ID | TH | PH | MY | VN | TW | SG |
|---|---|---|---|---|---|---|---|
| consumer-loan | 本方案相关 | 本方案相关 | 本方案相关 | 本方案相关 | | 本方案相关 | |
| buyer cashloan | 本方案相关 | 本方案相关 | 本方案相关 | 本方案相关 | | 本方案相关 | |
| seller cashloan | 本方案相关 | 本方案相关 | 本方案相关 | 本方案相关 | | 本方案相关 | |
| fast-escrow | 本方案相关 | 本方案相关 | 本方案相关 | 本方案相关 | | 本方案相关 | |

## 2、架构设计




## 3、核心流程交互时序

参考第2点。

## 4、核心数据计算逻辑和字段设计

1. 用户配置生成新的工单模板

- holmes上用户配置新的工单模板，前端生成adapter，存入 fs_ticket_template_adapter_tab 中（参考前端方案：前端技术方案），并调用后端接口，表示需要新增tmp（接口定义在第5点 API设计）；
- 后端接收到前端请求后，在ticket_tpl_tab中生成一条新的记录。前端需要传值app_id，scene_id，region和adapter_id，这些数据会落库；同时后端需要自主生成category和subtype（对于ticket_tpl_tab来说，app_id，scene_id，region，category和subtype构成一个主键，同时这也唯一标识一个模板）。根据prd，如果是激活 category为2（表示KYC）；如果是支付或者提现category为6（表示Trabsaction）。但另外应该还会有 Upgrade 和同人工单，以及Common工单是否可以动态配置增加，这个需要PM澄清；
- ticket_tpl_tab最主要是增加两列数据，一列是 ticket_info_tpl，表示 ticket_info 的取数逻辑，目前 risk-engine 代码的函数库中已实现了许多方法，这一步需要做的就是拼接函数和入参（数据结构会在第5点 API设计中说明）。另一列是 ticket_other_info_tpl，因为 ticket_tab 表中还需要类似于 user_name 或者 id_card_no 这样的数据，而不同场景PB定义的字段取数逻辑各不相同，所以也提供了可配置的方案来取值，因为每个场景的这部分数据是固定的，risk-admin需要提供一个配置池，根据使用场景选择不同的配置；
- ticket_tpl_tab 增加数据后，拿到一个自增id，跟据前端传值 app_id，scene_id和methods，需要去 strategy_l3_group_methods_tab 中找到methods对应的记录。strategy_l3_group_methods_tab 表新增一列 ticket_tpl_ids，存放id数组，表示这个 method 所支持的所有工单模板。因为在holmes上新增模板时，考虑的是method维度，而在工作流执行过程中，考虑的应该是工作流维度，一个method可能会对应多条工作流，因此需要在多条记录中都加上新增的tpl的tpl_id，最终结果如图所示（之所以会出现图中的情况是因为，之前老流程里，工单和工作流是一对一的，新增工单即是新增工作流。这个地方是否需要再进一步抽象可以再讨论）；

| | id | app_id | scene_id | name | check_result | ticket_tpl_id | methods_descripti... | creator | version | create_time | update_time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 24 | kyc | 10012 | Screening | 30101 | 14,19 | for test | | 0 | 0 | 0 |
| | 18 | kyc | 10012 | Screening | 30102 | 11,19 | for test | | 0 | 0 | 0 |
| | 19 | kyc | 10012 | Screening | 30108 | 12,19 | for test | | 0 | 0 | 0 |
| | 232 | kyc | 10012 | Screening | 30125 | 18,19 | for test | | 0 | 0 | 0 |
| | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

- 以上 ticket_tpl_tab 和 strategy_l3_group_methods_tab 表的变更应该是原子操作，要考虑事务的情况，如果有其中一个表变更失败（例如 DB Connection Valid），则返回前端 System Error。

2.用户修改已有的工单模板

- holmes上用户修改已有的工单模板，前端更新adapter后，调用后端接口，需要传值 app_id，scene_id，old_adapter_id，new_adapter_id，append_methods和remove_methods（接口定义在第5点 API设计）；

- 后端收到请求后，根据 app_id, scene_id和old_adapter_id在 ticket_tpl_tab 表中找到记录，更新adapter_id值为 new_adapter_id；根据 app_id，scene_id和append_methods在 strategy_l3_group_methods_tab 表中找到记录，在 ticket_tpl_ids 字段中增加刚刚在 ticket_tpl_tab 中操作记录的自增id；根据app_id，scene_id和remove_methods 在 strategy_l3_group_methods_tab 表中找到记录，在 ticket_tpl_ids 字段中移除刚刚在 ticket_tpl_tab 中操作记录的自增id；
- 以上 ticket_tpl_tab 和 strategy_l3_group_methods_tab 表的变更应该是原子操作，要考虑事务的情况，如果有其中一个表变更失败（例如 DB Connection Valid），则返回前端 System Error。

3.用户配置group加验方式

- 用户在 holmes 上操作配置L3 group的加验方式。根据 strategy_l3_group_methods_tab 表列出所有的method，再根据用户选择的 method，在 ticket_tpl_tab 中找到所支持的 type。因为 strategy_l3_group_methods_tab 表中是一个工作流一条记录，所以对于一个 method 来说，所拿到的所有工作流的 ticket_tpl_ids 会有数据重复的情况，需要做去重处理；
- 用户选择完 type 并点击确定后，前端在 strategy_l3_rule_group_tab 表中保存数据，除了之前会存的数据，还需要多存 ticket_tpl_id（及 type 对应记录的自增id）；
- risk_admin 解析 strategy_l3_rule_group_tab 表 l3_rule 列的数据，并将 tikcet_tpl_id 解析到 group_script 列中（类似于 X_Return_code）。

| id | ori_id | group_name | group_script | l3_rules | default_result_id | version | status | description | creator |
|---|---|---|---|---|---|---|---|---|---|
| 1397 | NULL | rejectall | V_DeviceFingerprint_1055 = "${device_fingerpri... | [{"L3RuleId":1144,"L3RuleName":"locauattest","... | 2 | 1 | 2 | locauattest | linfei.gui@shopee.c |
| 1398 | NULL | screen | V_DeviceFingerprint_1055 = "${device_fingerpri... | [{"L3RuleId":1144,"L3RuleName":"locauattest","... | 2 | 1 | 2 | 待审核 | linfei.gui@shopee.c |
| 1399 | NULL | checkeuserInfo | V_ClientIp_1066 = "${client_ip}" V_TimeLength... | [{"L3RuleId":1145,"L3RuleName":"test","L3Rule... | 1 | 1 | 2 | test | linfei.gui@shopee.c |

4.工作流执行过程中

- 反欺诈节点执行结束后，将 ticket_tpl_id 存入该反欺诈节点对应的 _Output 中，并在执行 tciket task 时告知前置反欺诈节点是哪个；
- 在 ticket task 中，先拿到前置反欺诈节点的namespace，找到 ticket_tpl_id；如果找不到 ticket_tpl_id，说明不是新流程动态增加的工单，则按老逻辑去 ext_info 中获取待执行工单的 category 和 subtype，和 app_id，scene_id，region组成五元组去 ticket_tpl_tab 中获取工单模板；
- 根据模板解析数据，得到 ticket_info，存入 ticket_tab 中。另外，ticket_tab新增一列 adapter_id，将 ticket_tpl_tab 表中拿到的 adapter_id 填充进去，这一步是为了说明该工单是由哪个模板（版本相关）生成得到的，方便以后定位问题。

## 5、API设计

1.配置化数据结构定义

```
{
    "type":"record",
    "fields":[
        {
            "name":"user_image_path",
            "value_type":{
                "fetch_type": "pb field/verify field/feature field",
                "args": "字段名/加验枚举/feature名"
            },
            "default": ""
        }
    ]
}
```

- pb field。即是从请求中拿指定字段的值，这个已实现（可参考 task_ticket_func.go），拼接成 GetFromReq('args',RiskReqParamsMap)即可。至于 args，前端可以选择从 strategy_metric_tab 表中拿透传字段（格式为__XXX，是全路径的）展示给用户下拉选择，但是存DB时，需要将 ”__“ 处理为 ”.“；
- verify field。即加验数据，目前会有 LC图片1，LC图片2，LC结果和FM结果四类取值，定义为字符串枚举，分别是 VERIFY_LIVENESS_CHECK_URL1，VERIFY_LIVENESS_CHECK_URL2，VERIFY_LIVENESS_CHECK_RESULT 和 VERIFY_FACE_MACHING_RESULT。目前 task_ticket_func.go 中已实现获取LC加验图片的逻辑，及函数 func GetLCPicUrlFromVar(index int, ctx flow_entity.Variable) string，可以参照该函数进行改造，需要前端传值枚举，最终也是拼接字符串；
- feature field。需要从 RiskRsp 中拿指定feature的值，可以参考 task_ticket_func.go 中的 func GetFMScore(scoreType string, riskRsp *credit_risk_gateway.RiskGatewayRsp) interface{} 函数进行改造，需要前端传值 feature 名，最终也是拼接字符串。

2.工单模板新增接口定义

```
type RiskAdminReq struct {
    RiskAdminReqHeader *RiskAdminReqHeader `protobuf:"bytes,1,opt,name=RiskAdminReqHeader" json:"
RiskAdminReqHeader"`
    Params             []byte              `protobuf:"bytes,2,opt,name=Params,proto3" json:"Params"`
}

#Param
type TicketTplAdd struct {
        AppId       string
        SceneId     uint64
        AdapterUuid string       //
        MethodIds   []uint64
    Region      string
        Name        string
}

type RiskAdminRsp struct {
    RiskRspHeader *RiskAdminRspHeader `protobuf:"bytes,1,opt,name=RiskRspHeader" json:"RiskRspHeader"`
    Data          []byte              `protobuf:"bytes,2,opt,name=Data,proto3" json:"Data"`
}

#Data
```

3.工单模板修改接口定义

```
type RiskAdminReq struct {
    RiskAdminReqHeader *RiskAdminReqHeader `protobuf:"bytes,1,opt,name=RiskAdminReqHeader" json:"
RiskAdminReqHeader"`
    Params             []byte              `protobuf:"bytes,2,opt,name=Params,proto3" json:"Params"`
}

#Param
type TicketTplMod struct {
        AppId             string
        SceneId           uint64
        AdapterUuid       string     //
         AppendMethodIds   []uint64
        RemoveMethodIds   []uint64
}


type RiskAdminRsp struct {
    RiskRspHeader *RiskAdminRspHeader `protobuf:"bytes,1,opt,name=RiskRspHeader" json:"RiskRspHeader"`
    Data          []byte              `protobuf:"bytes,2,opt,name=Data,proto3" json:"Data"`
}

#Data
```

4.ticket_tpl_tab 表结构变更

```
alter table ticket_tpl_tab add column adapter_uuid vachar(255);
```

5.strategy_l3_group_methods_tab 表结构变更

```
alter table strategy_l3_group_methods_tab add column ticket_tpl_ids vachar(255);
```

6.ticket_tab 表结构变更

```
alter table ticket_tab add column adapter_id bigint(20);
alter rable ticket_tab add column code_return_nums bigint(20);    // 1999Int
```

## 6、兼容性及扩展性

(1) 兼容性: 兼容已有的工单

(2) 扩展性: 支持给已有的工作流配置新的工单模板

## 7、性能

不另外增加性能负担。

## 8、线上监控

(1) 接口监控: 工作流支持配置多类工单后，可以考虑监控细分，当前只是对工作流有监控（老流程一个工单对应一条工作流）。

## 9、其他

1.新流程开始后，前端采用 template_id 区分工单类型，return_code 也通过 template_id 关联。category 和 subtype 除了兼容老工单，只对工单类型起作用（体现在 holmes 菜单tab页）。