前端技术方案

PRD（可配置化）：https://confluence.shopee.io/pages/viewpage.action?pageId=589173243

PRD（历史迁移）：https://confluence.shopee.io/pages/viewpage.action?pageId=631447187

后端方案：credit_risk_engine_v1.0.54 [ALL][ALL]Holmes 2.0-工单可配置化

背景&概况

工单配置化：

1. 界面开放给 risk team 用户配置
2. 配置过程分为两步，第一步配模版（包含展示字段的 Label 是什么，字段类型是什么），第二步配适配器（选择模版，选择适用的 app_id + scene_id + type_id，Label 对应的数据来源是什么，对应展示的字段是什么）
3. ticket 表需新增 adapter_id，代表使用新的工单配置进行渲染

历史工单兼容：

1. 需要梳理目前所有工单所有特殊情况（定制情况）
    a. 若审核区里面包含审核类，往往需要定制组件（表单内操作的值，通过 eventBus 传递值至顶层工单组件，触发顶层工单组件审核操作）
    b. 若包含表格的，也需要特殊定制
    c. 带有 Tooltip 展示的
    d. 树形 Collapse 结构展示的
    e. 复杂照片 img 布局展示

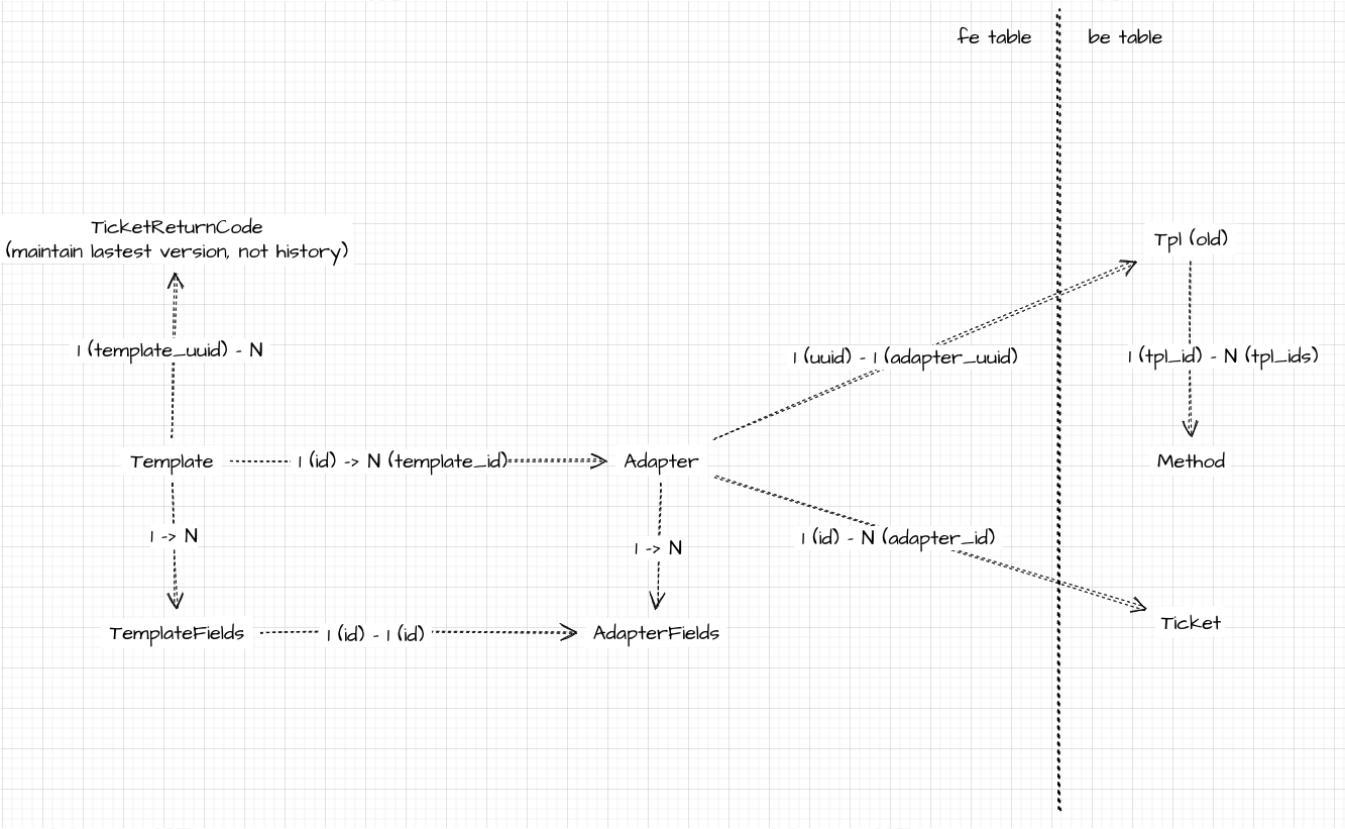
数据存储

template 下可使用多个 adapter：template ---(1-N)---→ adapter

template 下有多个 fields: template ---(1-N)---→ fields

adapter 下有多个 fields: adapter ---(1-N)---→ fields

1 个 adapter 下的 fields 应该与 关联的 template 下的 fields 一一对应



fe_ticket_template_tab 模版基础信息表

字段英文名	字段中文名	字段类型	字段描述	技术层关注细节点	
id	数据库自增 ID	number	id 唯一，主键	id 数据库自增	**模版展示（Web层）：** 1. category = hardcode 的，不允许配置字段配置层，只允许配置审核区域 2. 模版 List 展示，只展示所有模版ID的最新 Verison 记录 3. 模版配置内的 History List，展示的为该模版ID下的所有 Version 记录 4. 对于字段 type 设置（与前端渲染组件依赖数据源有关）一旦创建后，编辑状态不可再修改 5. **除了新增字段操作，在编辑时候保存，需要提示给到用户，会自动更新最新的 Adapter 版本** 6. 更多展示逻辑继续参考 PRD 描述 **模版存储（Node层）：** 1. 模版的任何修改，都只是新增记录，对应模版 ID 的 version + 1 进行存储，不删除记录 2. fields_config 存储时，Field Id 唯一生成（Id 需要与Adapter 的 FieldId 进行唯一关联，可使用 uuid，可考虑别的，此字段往往为前端定位所需，可考虑是否可以简短） 3. **模版一旦修改（无论修改 Label，字段顺序重排，删除字段，新增字段等）都需要拉取所有关联的 Adapter，新增一条对应的 Adapter verison 记录（并且 Operator 增加（auto）后缀）**
template_uuid	工单模版 ID，非自增	string	工单模版 ID，uuid	template_id + version 联合唯一	
version	工单模版 Version，非自增	number	工单模版 版本号，从 1 开始		
version_status	版本最新标志	tinyint	工单模版是否最新，1 - history，2 - current，方便搜索使用	筛选展示列表时，可以 where version_status = 1 进行搜索	
name	工单模版名称	string	工单模版名称	业务属性字段，产品所需	
category	工单模版分类	tinyint	工单模版分类，1 - default（默认），2 - hardcode（历史迁移的）		
description	工单模版描述	text	工单模版描述		
fields_config	工单模版字段配置	json	工单模版字段配置，包含字段展示 Label 和 字段类型	前端关注，渲染所需 **TplFieldItem[]**，内结构参考如下，默认结构	

layout	工单布局	json	工单布局参数配置	前端关注，保留字段，渲染所需，{ component, componentProps } 方便扩展使用，本期可先不考虑
rejection	拒绝信息	json	工单拒绝信息	**Rejection**，内结构参考如下
update_time	更新时间	number	时间戳，更新时间	默认字段
create_time	创建时间	number	时间戳，创建时间	
operator	操作人	varchar	操作人	

TplFieldItem（单条模版字段配置）：

TplFieldItem

```
//
{
  id                          // uuid
  priority                  //
  type          //
  label         //  Labeljson  label  { name: 'UserName' }
  block         // picture / personal / other /
  dataKeys      // json arry ['country', 'province', 'city'] Adapter
  webConfig     // json object  { component(Name), formatter(), componentProps() }
}
```

TplFieldItem.webConfig（单条模版字段的 webConfig）：

值的渲染流程：原始值（value）-----(utils.formatter)-----→ 格式化后值（formatValue） ----------(component + props + value + formatValue)------------→ 组件渲染

TplFieldItem.webConfig

```
//
{
  //
  formatter: {
     //
         emptyValue,    //
     functionName,  //  formatter
     script,        //  script

        //  type
    ...
  },

   //
  component,
  //
  componentProps: {
     tooltip,             // tooltip
     className,           //
     style,               //
  }
}
```

优先实现以下字段类型：

字段类型	component	formatter / componentProps 特有参数	最终 JSON schema（只展示特殊部分）

text	TypographyText（默认）	**formatter**
		{
		  tpl // ${Address}.${Province}.${City}
		}
			{
			  type: 'text',
			  webConfig: {
			  formatter: {
			  tpl: '${Address}.${Province}.${City}'
			  }
			  }
			}
			（根据 dataKeys Array，处理成 dataValues Object，提供给到组件内渲染）

number	TypographyText（默认）	**formatter**
		{
		  decimals, //
		  isThousands, // 0-1-
		  divisor, //
		  appendUnit // +
		}
			{
			  type: 'number',
			  webConfig: {
			  formatter: {
			  decimals: 2,
			  isThousands: 0,
			  divisor: 1,
			  appendUnit: 'W'
			  }
			  }
			}

enum	TypographyText（默认）	**formatter**
		{
		  type, // staticdynamic
		  name, // namedynamic
		}
			{
			  type: 'enum',
			  webConfig: {
			  formatter: {
			  type: 'static',
			  name: 'AppId'
			  }
			  }
			}

link	TypographyLink（默认）	**formatter**
		{
		  tpl //
		}
			{
			  type: 'link',
			  webConfig: {
			  formatter: {
			  tpl: '/log/${RequestId}'
			  }
			  }
			}

img	Photo → ScalePhoto（默认）	**componentProps**
		{
		  width, //
		  height, //
		}
			{
			  type: 'img',
			  webConfig: {
			  componentProps: {
			  width: 100,
			  height: 100
			  }
			  }
			}

date	TypographyText（默认）	**formatter**
		{
		format // https://dayjs.gitee.io/docs/zh-CN/display/format
		}
			{
			type: 'date',
			webConfig: {
			formatter: {
			format: 'YYYY-MM-DD HH:mm:DD'
			}
			}
			}

datetime	TypographyText（默认）	**formatter**
		{
		format // https://dayjs.gitee.io/docs/zh-CN/display/format
		}
			{
			type: 'datetime',
			webConfig: {
			formatter: {
			format: 'YYYY-MM-DD HH:mm:DD'
			}
			}
			}

Rejection 结构：

TplFieldItem

```
//
{
        choiceType                  // single/multiple
        rejectCode                  // Y/N
        codeReturnType          // single/multiple
        codePriority        // Y/N
        rejectLabel                 // Y/N
}
```

fe_ticket_template_adapter_tab 模版适配器表

字段英文名	字段中文名	字段类型	字段描述	技术层关注细节点	
id	数据库自增 ID	number	id 唯一，主键	id 数据库自增	**适配器展示（Web层）：** 1. 适配器 List 展示，只展示所有适配器 ID 的最新 Verison 记录 2. 适配器配置内的 History List，展示的为该适配器 ID下的所有 Version 记录 3. 更多展示逻辑继续参考 PRD 描述 **适配器存储（Node层）：** 1. 适配器的任何修改，都只是新增记录，对应适配器 ID 的 version + 1 进行存储，不删除记录，也不修改历史版本 2. fields_config 存储时，需要同时处理一份处理到 be_fields_config 字段，后端只关注此数据格式即可 3. category / scene / app, 一旦创建不可再编辑，method 可以再编辑 4. **创建 adapter 需要校验 application + scene + screening type 是否已经存在（判断也需要带上 pause 的），若存在，提示该适配已存在，无法新增成功**
adapter_uuid	工单适配器 ID，非自增	string	工单适配器 ID，uuid	adapter_id + version 联合唯一	
version	工单适配器 Version，非自增	number	工单适配器 版本号，从 1 开始		
version_status	版本最新标志	tinyint	工单适配器是否最新，1 - history，2 - current，方便搜索使用 -1 - disabled，为了跟后端保持事务性，用于标记错误的数据	筛选展示列表时，可以 where version_status = 1 进行搜索	
template_id	依赖工单模版 ID	number	依赖工单模版ID（关联 ticket_template_tab 的 id 自增字段）	需要关联到这个 Adapter 用的哪个模版	
app_id	应用ID	number	应用ID	后端会根据这个字段去捞当前生成工单用哪个	

scene_id	场景ID	number	场景ID	Adapter
method_ids	加验方法ID	json	多项，json array	
category	分类	number	默认 default	业务预留字段
status	状态	tinyint	1-Active、2-Paused	
fe_fields_config	前端字段列表，渲染所需	json	前端字段列表，渲染所需	前端关注，渲染所需 **AdapterFeFieldItem[]**，内结构参考如下，默认结构
be_fields_config	后端字段列表，后端生成数据所需	json	后端字段列表，后端生成数据所需	前端关注，渲染所需 **AdapterBeFieldItem[]**，内结构参考如下，默认结构
update_time	更新时间	number	时间戳，更新时间	默认字段
create_time	创建时间	number	时间戳，创建时间	
operator	操作人	varchar	操作人	

AdapterFeFieldItem（单条适配器 FE 字段配置）：

AdapterFeFieldItem

```
//
{
  templateFieldId             // ID
  key                                    //
  dataIndex                                 //  key lodash get
  dataSource                          // , 1-pb, 2-verify, 3-feature,
}
```

AdapterBeFieldItem（单条适配器 BE 字段配置，对比 FE 字段，只少了 template_field_id，后端不关注模版字段怎么关联）：

AdapterBeFieldItem

```
// be_fields_config
{
    "type":"record",
    "fields":[
        {
            "name":"user_image_path",
            "value_type":{
                "fetch_type": "pb field/verify field/feature field",
                "args": "//feature"
            },
            "default": ""
        }
    ]
}

// AdapterBeFieldItem
{
  default                          //
  name                                  //  ticket_info
  value_type: {
        fetch_type                 // pb field/verify field/feature field
        args                             //  name
  }
}
```

ticket_tab 工单表（已有）

ticket_tab

```
{
  //
  adapter_id                 // ada ID DB  ID

  //
  ticket_category
  ticket_ticket_subtype
}
```

ticket_return_code_tab 工单拒绝码列表（已有）

ticket_tab

```
{
  //
  template_uuid                 //  template  template_uuid
}
```

1. 对于新方案生成的工单，通过 获取动态配置的拒绝原因列表。
2. 针对历史工单，通过旧方式 category & subtype → ticket_return_code_tab 获取手动插 db 的拒绝原因列表。
3. 历史工单如果需要迁移，需要将过去手动插 db 的数据添加上 template_id。

后端协议对齐

前后端方案对齐，需要基于 fe_ticket_template_adapter_tab 表 (简称 adapter_tab)。adapter_tab 存放了前端渲染相关的 template_id，以及后端构造工单数据相关的 be_fields_config，因此一些关联的表需要新增字段 adapter_id (adapter 表自增 ID)。

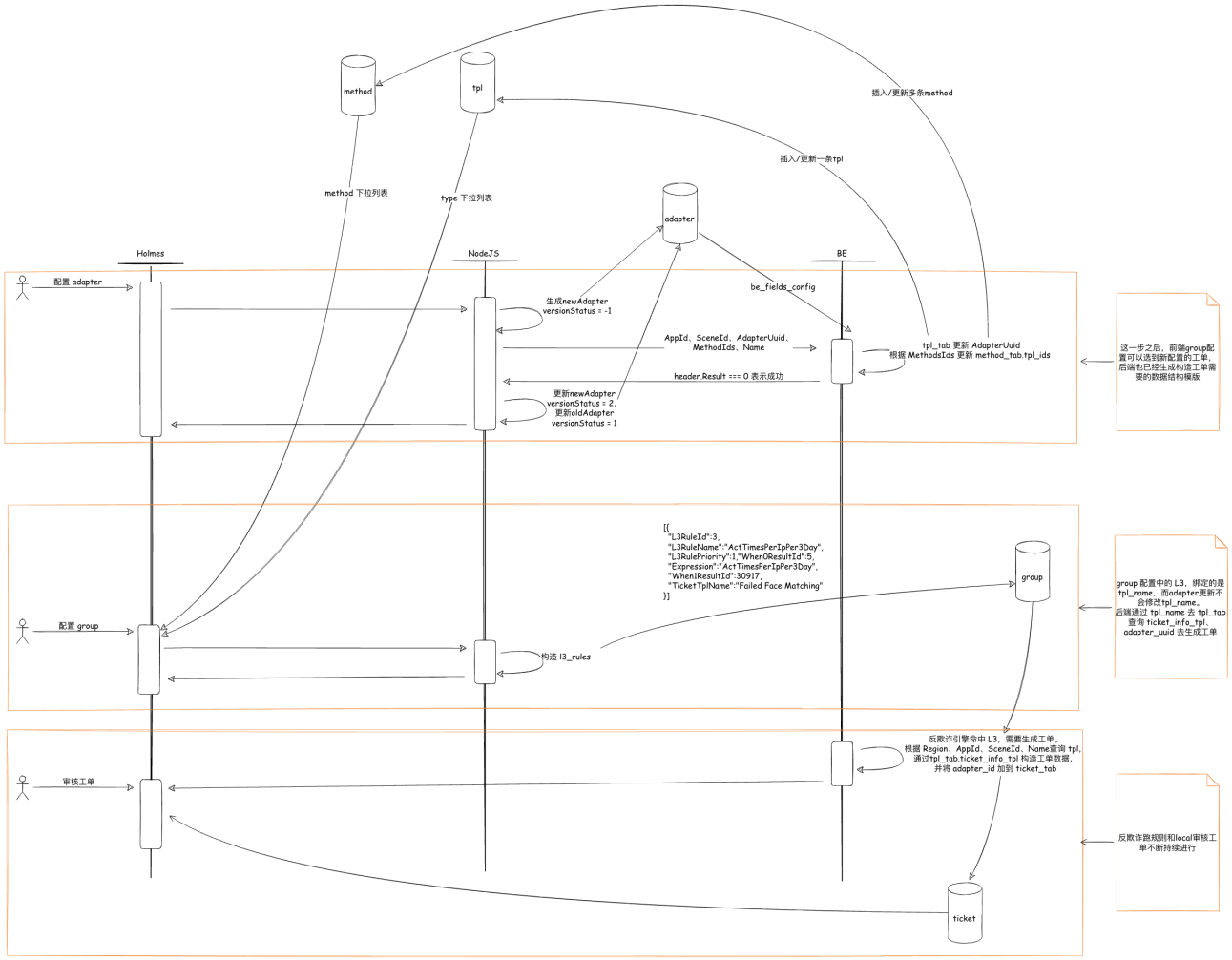
以下是需要添加字段的相关表

表名	添加字段
ticket_tpl_tab (简称 tpl_tab)	adapter_uuid
ticket_tab	adapter_id

主要流程：

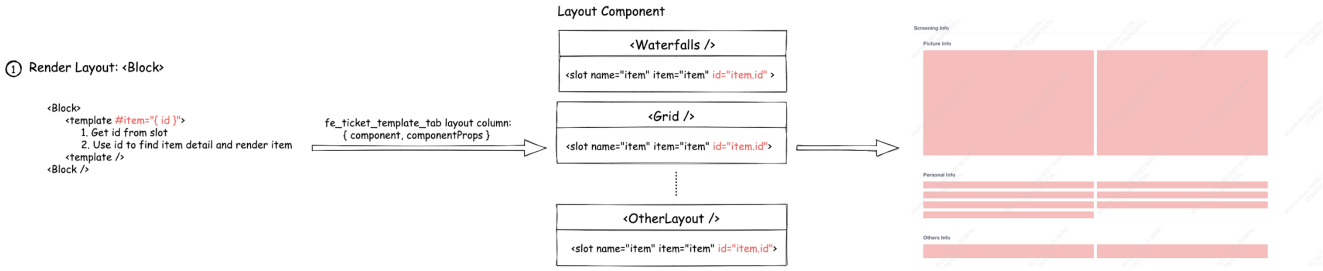
1. 配置(新增/编辑) adapter 时，前端更新 adapter_tab。然后调后端接口，通知后端将 adapter_uuid 更新到 tpl_tab：
   a. 后端根据 adapter_uuid 去 adapter_tab 查询获取(version_status = current) be_fields_config 处理后生成 数据模版 json 添加到 tpl_tab。
   b. 后端根据更新的 tpl ，联动修改 method_tab，保证 method_tab 和 tpl_tab 能够关联查询。
2. 配置 group 时，选择 method 和 subtype 后，前端需要将 type_name 也写入db，后端根据 type_name 去 tpl_tab 查询获取 数据模版 json，进而生成新工单。
3. 工单渲染时，前端根据 adapter_id 去 adapter_tab 查询获取 template_id，进行渲染。

以上表添加 adapter_id 只考虑最新版本，所以没有历史版本的概念。但 ticket_tab 旧工单记录存放的有可能是旧的 adapter_id ，因此工单渲染情况还是按当时关联的 adapter_id 为准。

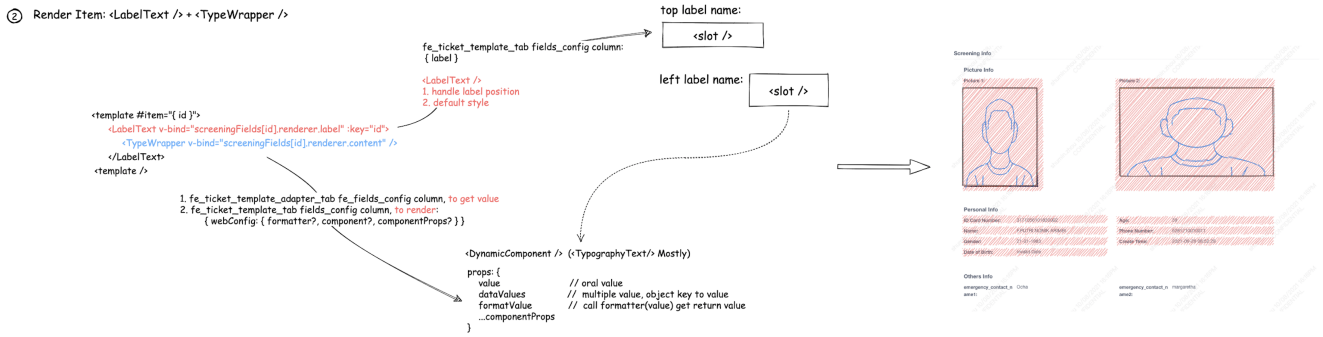


前端渲染层（ScreeningAuto / Renderer）

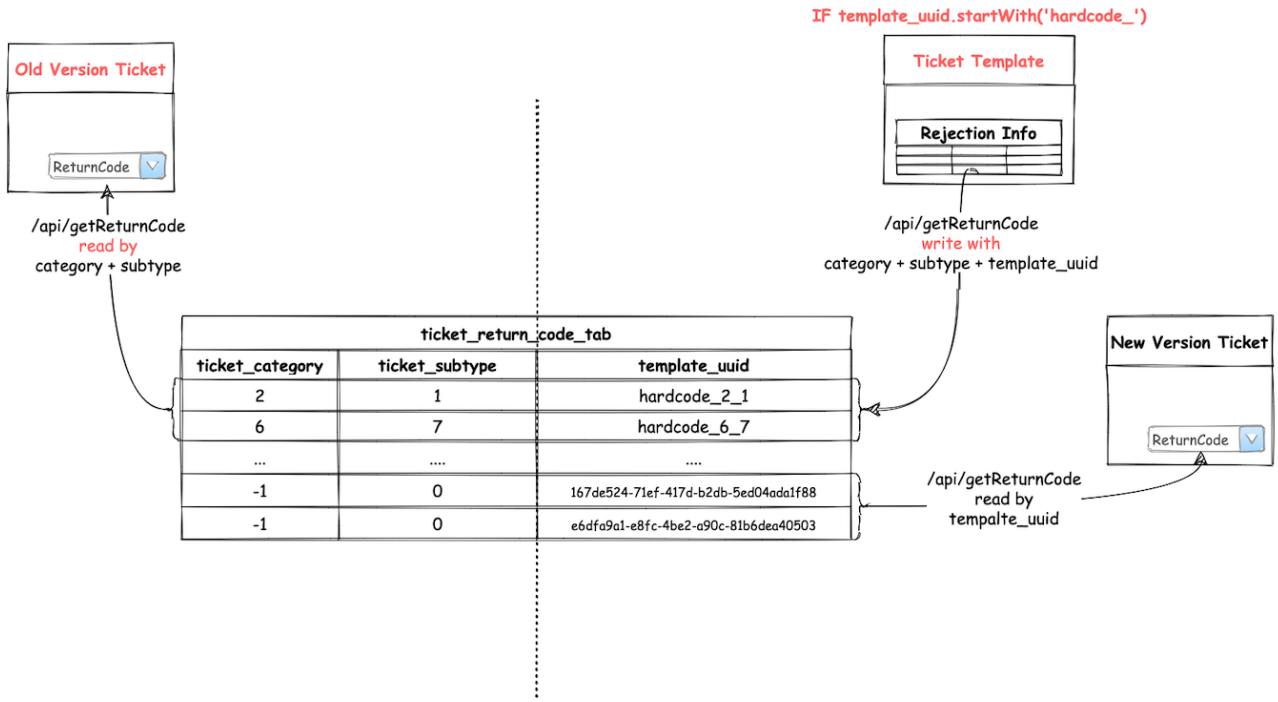
- <Block /> 布局适配器，根据用户设置的 layout.component 参数可指定具体渲染什么布局（暂未开放，此功能保留开发，便于后续扩展），默认为 Waterfalls，布局类组件多为可复用组件，需要暴露 item slot，并且使用 item.id 保证每项唯一



- 每一项为 <LabelText /> + <TypeWrapper />，LabelText 处理 name 与 content 之间的位置关系，并且包含默认展示样式。TypeWrapper 为动态渲染部分，依然保留定制开发部分（通过 field item 的 webConfig.component 指定渲染什么组件）



历史工单迁移



1. 旧有工单渲染逻辑**不变更，依然为 `category + subtype`**，template 存储时通过 template_uuid 区分哪些为旧有工单，根据规范解析出 category + subtype，并且存为对应字段中
2. 新工单渲染不变，依然根据 template_uuid 拉取对应 code 渲染即可
3. 迁移 hardcode 包含哪些 category + subtype，根据 ticket_tpl_tab 过滤出地域已支持的工单，结合前端 Operate 操作区域渲染控件整理 Operate 审核，对应只迁移以下 hardcode 工单：

Region	category-subtype
ID	2-1, 2-2, 2-4
~~MY~~	~~2-6~~
PH	2-1, 2-2, 2-6, 2-11, 2-12, 6-7, 6-9
TH	2-4, 2-8, 2-9, 2-10, 6-5
TW	2-7, 6-4