流程插件开发说明

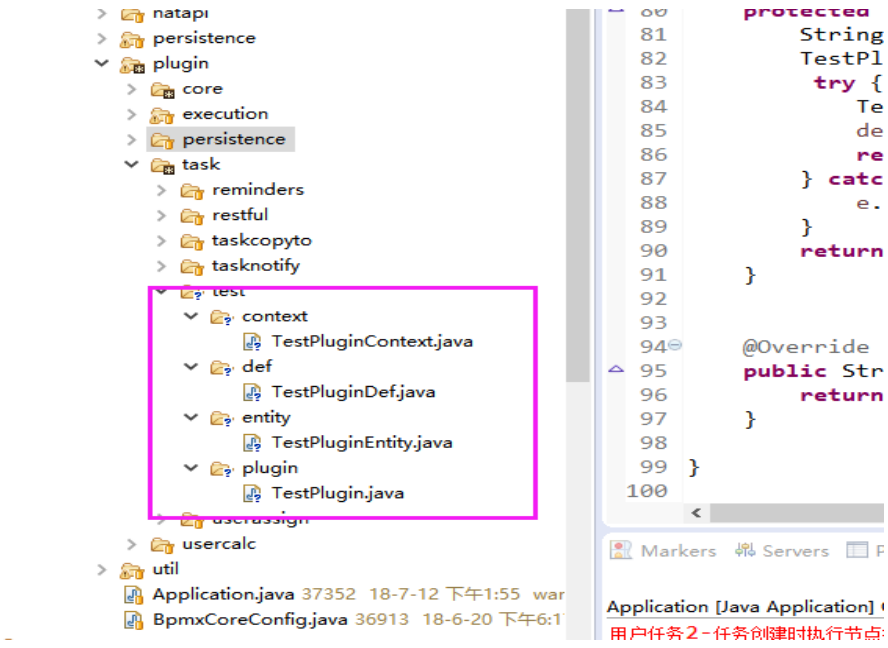
1 概要

流程插件用于节点任务创建时或任务完成时触发，根据用户在流程配置界面的配置，来实现一些业务需求。

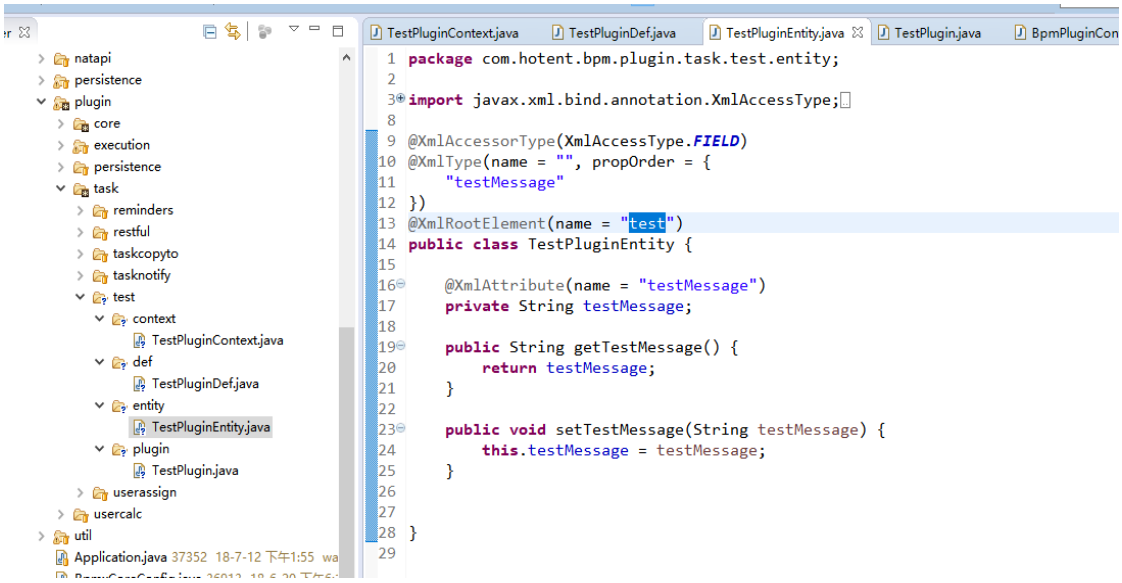
2 具体实现

2.1 插件文件目录

目前系统中包含三种类型的插件：execution（放置所有执行类插件）、task（放置所有任务类插件）、usercalc（放置所有用户计算策略插件），如果是其他类型的插件，请建一个新的包来放置。当然，在 bpmx-plugin-api 和 bpmx-plugin-core 也需要对该类的插件增加相应的接口和抽象类支持。当前插件（任务节点调用 restful 接口）定义为 task 任务类插件，放置位置如下：

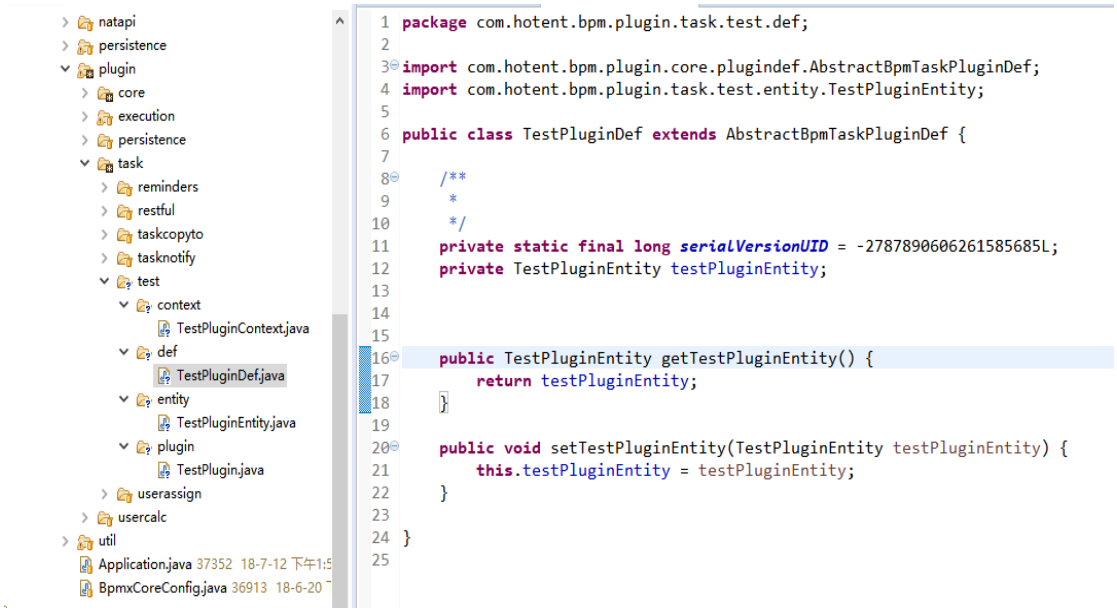


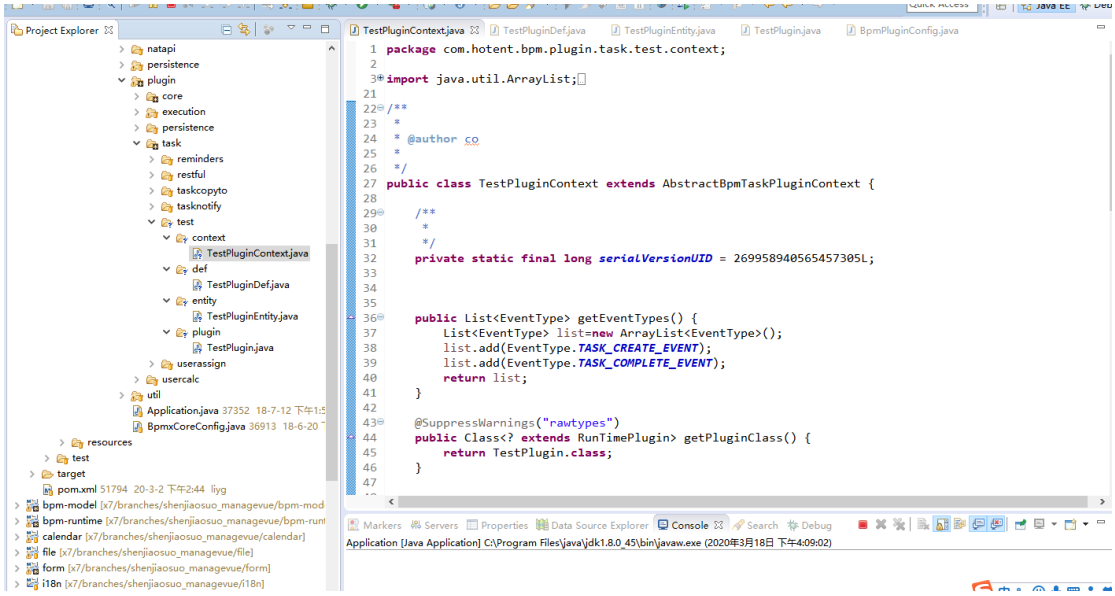
2.2 定义插件运行实体

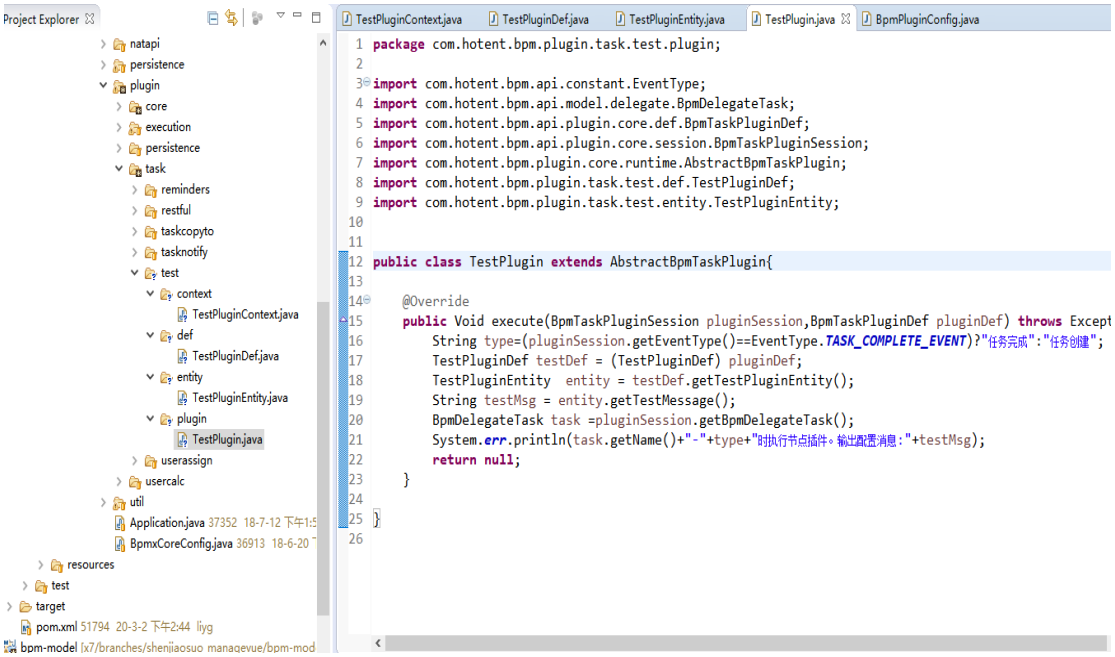


2.3 定义插件运行时、插件上下文、插件定义

在 restful 插件目录下添加插件定义（def）文件，插件上下文（context）、插件（plugin）目录如下：





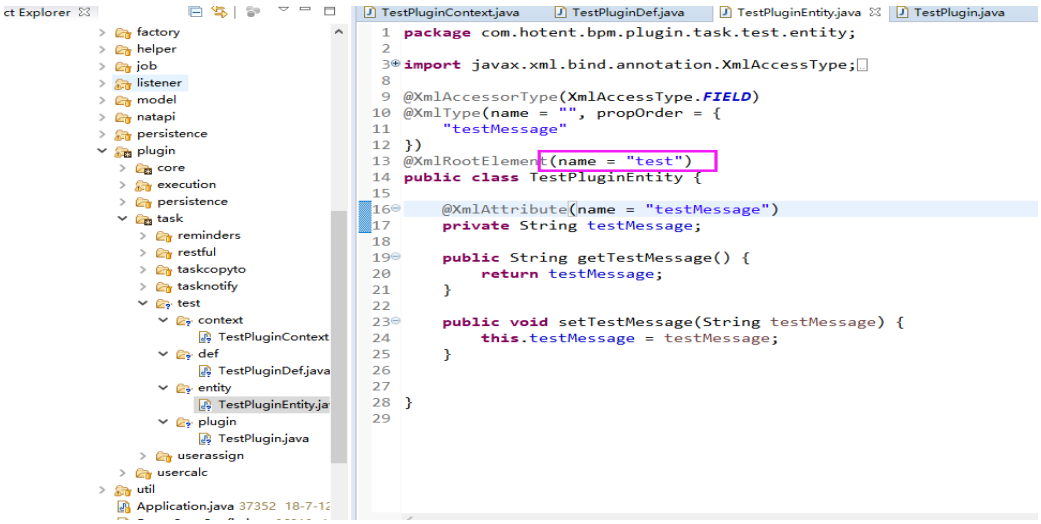


- 插件运行时（plugin）文件需要继承实现对应抽象类（目前系统中有：执行类（AbstractBpmExecutionPlugin）、任务类（AbstractBpmTaskPlugin）、用户计算类（AbstractUserCalcPlugin）），这里继承的是任务类（AbstractBpmTaskPlugin）；插件定义和上下文也同样需要继承实现对应抽象类（Task、Execution、UserCalc），这里是继承任务类。
- 插件中具体业务逻辑请根据实际业务需求来实现，task 对象可以获取当前流程的相关数据。此处为输出任务名称和配置的消息。

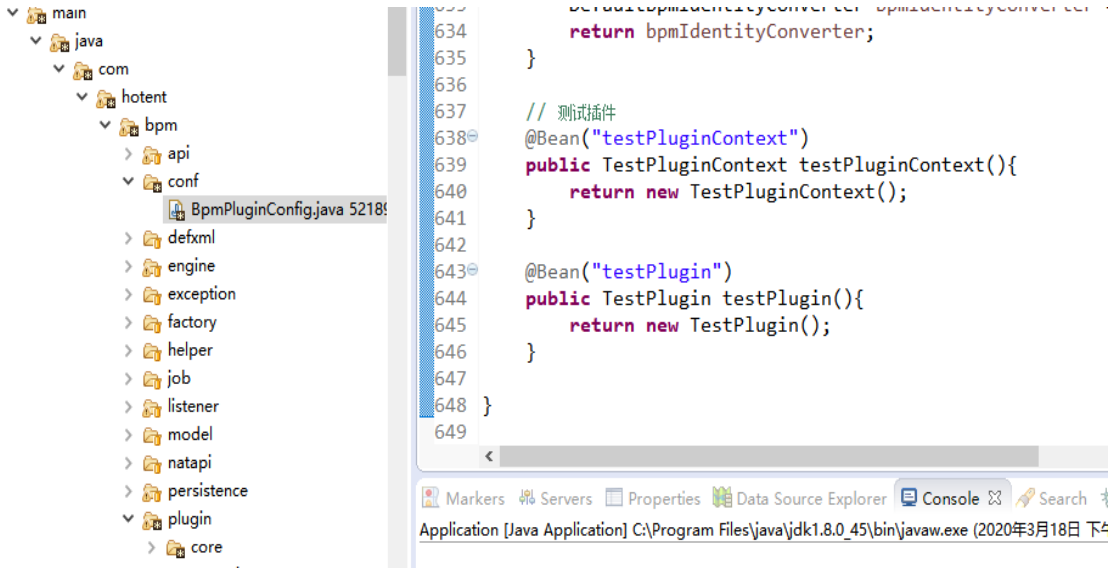
2.4 插件在配置文件中声明、配置

插件目录开发完成后需要在 BpmPluginConfig.java 文件中声明、配置：

注意：插件 bean 注册的名称一定要按照插件的根节点名称+“PluginContext”的规则命名，因为代码中是通过 String pluginContextBeanId = el.getLocalName() + PluginContext.PLUGINCONTEXT 来获取的解析类。这里的根节点名称为：



test，所以 bean 的名称为：testPluginContext。



2.5 插件应用

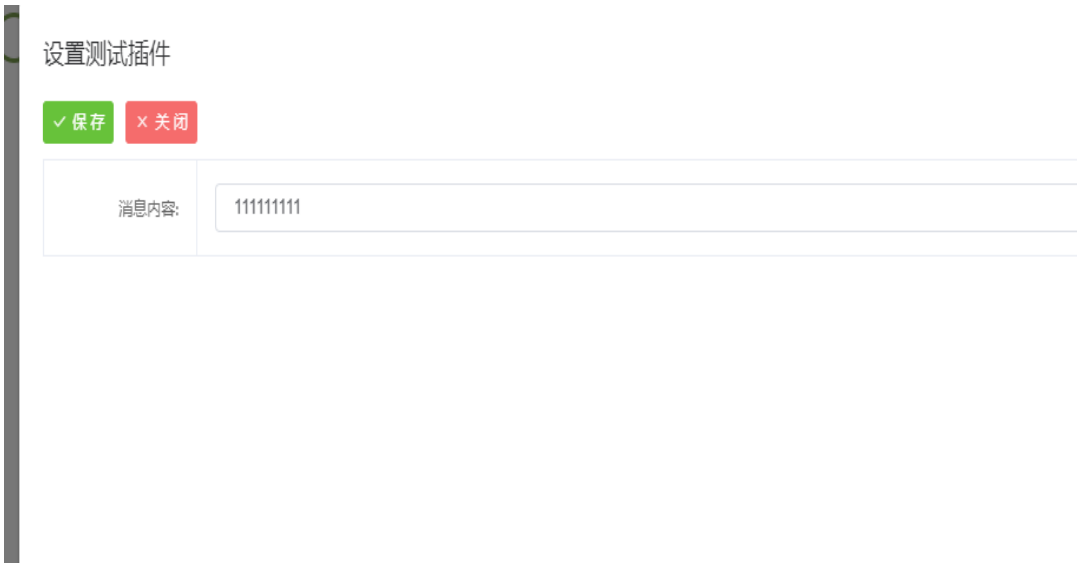
1、 保存插件数据（一般在 PluginsController 中新增方法保存数据）：

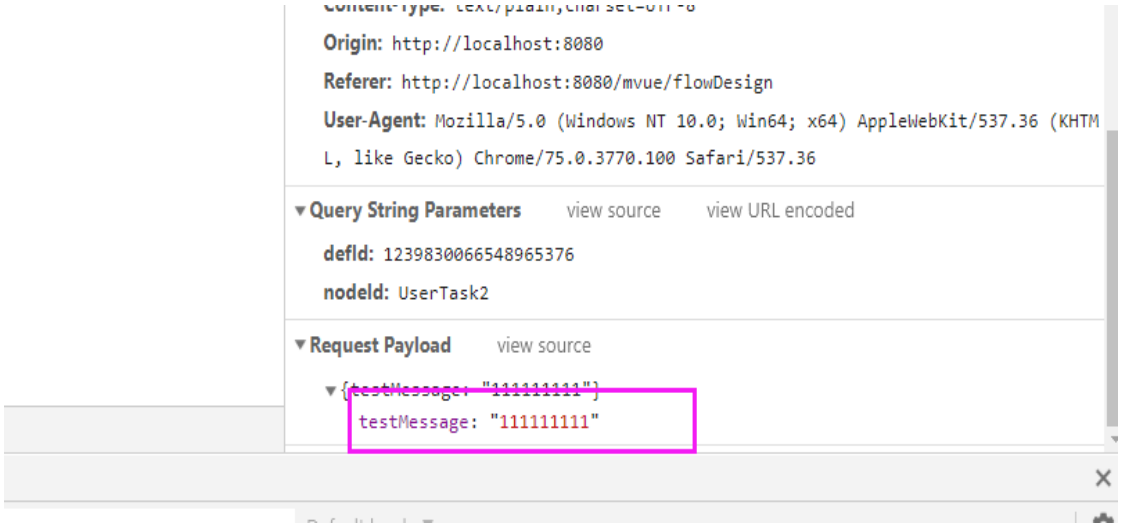
```
    /**
     * 保存测试插件
     * @exception
     * @since  1.0.0
     */
    @RequestMapping(value="saveTestPlugin", method=RequestMethod.POST, produces={"application/json; charset=utf-8" })
    @ApiOperation(value = "保存测试插件", httpMethod = "POST", notes = "保存测试插件")
    public CommonResult<String> saveTestPlugin(
            @ApiParam(name="defId",value="流程定义id", required = true) @RequestParam String defId,
            @ApiParam(name="nodeId",value="节点id", required = true) @RequestParam String nodeId,
            @ApiParam(name="json",value="json", required = true) @RequestBody String json) throws Exception{
        try {
            BpmNodeDef nodeDef = bpmDefinitionAccessor.getBpmNodeDef(defId, nodeId);

            TestPluginContext context = new TestPluginContext();
            context.parse(json);
            List<BpmPluginContext> plugins = changeOnePluginContextForSave(nodeDef.getBpmPluginContexts(),context);
            PluginsBpmDefXmlHandler bpmDefXmlHandler = (PluginsBpmDefXmlHandler) AppUtil.getBean(PluginsBpmDefXmlHan
            bpmDefXmlHandler.saveNodeXml(defId, nodeId, plugins);
            return new CommonResult<String>("保存成功");
        } catch (Exception e) {
            e.printStackTrace();
            return new CommonResult<String>(false,"保存失败: "+e.getMessage());
        }
    }
}
```

2、 获取插件数据（一般在 PluginsController 中新增方法获取数据）：

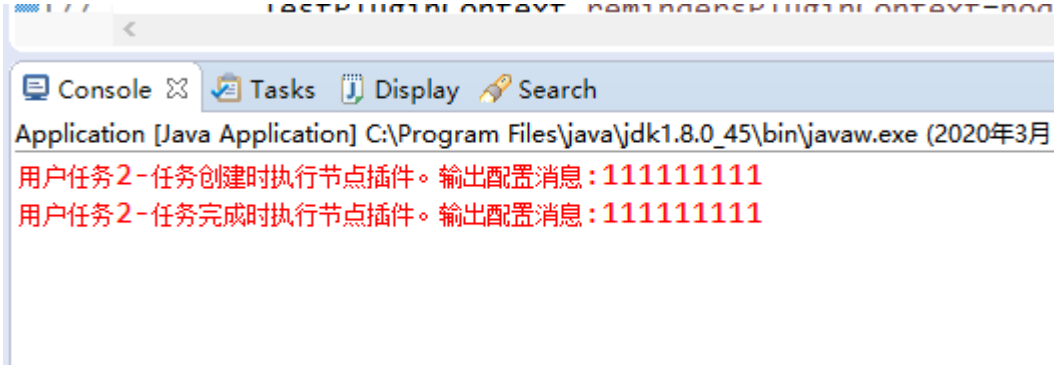
```
    /**
     * 获取测试插件json
     */
    @RequestMapping(value="getTestPlugin", method=RequestMethod.GET, produces={"application/json; charset=utf-8" })
    @ApiOperation(value = "获取测试插件json", httpMethod = "GET", notes = "获取测试插件json")
    public String getTestPlugin(@ApiParam(name="defId",value="流程定义id", required = true) @RequestParam String defId,
            @ApiParam(name="nodeId",value="节点id", required = true) @RequestParam String nodeId ) throws Exception{

        BpmNodeDef nodeDef = bpmDefinitionAccessor.getBpmNodeDef(defId, nodeId);
        TestPluginContext remindersPluginContext=nodeDef.getPluginContext(TestPluginContext.class);
        if (BeanUtils.isEmpty(remindersPluginContext)) {
            return "";
        }
        return remindersPluginContext.getJson();
    }

    /**
```

3，在前端按照插件运行实体(TestPluginEntity)调用后台接口传入 json 保存插件配置





4，配置插件数据后，任务创建、完成时（可在 TestPluginContext 的 getEventTypes 中自定义）就会检测到添加的插件，满足条件则触发运行。


Console
Tasks
Display
Search
Application [Java Application] C:\Program Files\java\jdk1.8.0_45\bin\javaw.exe (2020年3月
用户任务2-任务创建时执行节点插件。输出配置消息：111111111
用户任务2-任务完成时执行节点插件。输出配置消息：111111111