

Snaker-V1.0 参考手册

目录

Snaker V1.0 参考手册	1
1、入门	3
2、测试用例	8
3、流程设计器	9
4、项目整合	12
4.1、与 Spring 集成	14
4.2、使用帮助类整合	17
5、如何扩展	19

1、入门

snaker-core 流程引擎源码托管地址

<http://git.oschina.net/yuqs/snaker-core>

snaker-demo 流程示例源码托管地址

<http://git.oschina.net/yuqs/snaker-demo>

注：snaker-demo 基于 springmvc、shiro、spring、hibernate、snaker 整合的一套 demo，其中包含基础的安全管理、流程示例

下载流程示例的相关文件


http://git.oschina.net/yuqs/snaker-core/attach_files

➤ 下载流程示例的可运行 war 包 (**snaker-demo.war**)，直接复制到 tomcat 的 webapps 中。

➤ 流程引擎表及 snaker-demo 的数据初始化 (**snaker-database.rar**)。

首先选择数据库软件（以 mysql 为例），创建 snaker 数据库，执行 **snaker-mysql.sql** 初始化流程引擎表。**init-data.sql** 当启动应用后（即使用 hibernate 的自动建表配置 hbm2ddl.auto 设置为 update）再执行。

➤ 启动示例应用 (**admin/123456;test/123456**)



登录

账号：
admin

密码：
.....

☐ 记住我

登录

[忘记密码?](#) [找回密码](#)

➤ 发布测试流程



➤ 启动执行流程



SnakerEngine 常用 API 参考:

部署流程:

```
snakerEngine.process().deploy(InputStream input);
```

卸载流程:

```
snakerEngine.process().undeploy(String id);
```

更新流程:

```
snakerEngine.process().update(Process process, InputStream input);
```

获取流程:

```
snakerEngine.process().getProcess(String id);
```

```
snakerEngine.process().getProcesss(Page<Process> page, String name, Integer state);
```

启动流程:

```
snakerEngine.startInstanceById(String processId);
```

```
snakerEngine.startInstanceById(String processId, Long operator);
```

```
snakerEngine.startInstanceById(String processId, Long operator, Map<String, Object> args);
```

执行任务:

```
snakerEngine.executeTask(String taskId);
snakerEngine.executeTask(String taskId, Long operator);
snakerEngine.executeTask(String taskId, Long operator, Map<String, Object> args);
```

提取任务:

```
snakerEngine.takeTask(String taskId, Long operator)
```

终止任务:

```
snakerEngine.terminateById(String orderId)
snakerEngine.terminateById(String orderId, Long operator)
```

查询方法 (snakerEngine.query().查询方法):

根据流程实例ID获取流程实例对象

```
Order getOrder(String orderId);
```

根据任务ID获取任务对象

```
Task getTask(String taskId);
```

根据参与者获取任务集合

```
List<Task> getActiveTasks(Long... actorIds);
```

根据流程实例ID、任务名称查询所有活动任务列表

```
List<Task> getActiveTasks(String orderId, String... taskNames);
```

根据流程实例ID、排除的任务ID、任务名称查询所有活动任务列表

```
List<Task> getActiveTasks(String orderId, String excludedTaskId, String...  
taskNames);
```

根据参与者分页查询活动任务

```
List<Task> getActiveTasks(Page<Task> page, Long... actorIds);
```

根据流程定义ID查询流程实例列表

```
List<Order> getActiveOrders(String... processIds);
```

根据父流程实例ID，查询所有活动的子流程

```
List<Order> getActiveOrdersByParentId(String parentId, String...  
excludedId);
```

根据流程定义ID分页查询order

```
List<Order> getActiveOrders(Page<Order> page, String... processIds);
```

根据给定的分页对象page、流程定义processId、参与者ID列表，查询工作项（包含
process、order、task三个实体的字段集合）

```
List<WorkItem> getWorkItems(Page<WorkItem> page, String processId, Long...  
actorIds);
```

根据流程定义ID分页查询历史流程实例

```
List<HistoryOrder> getHistoryOrders(Page<HistoryOrder> page, String...  
processIds);
```

根据父流程ID查询子流程实例集合

```
List<HistoryOrder> getHistoryOrdersByParentId(String parentId);
```

根据流程实例ID查询所有已完成的任务

```
List<HistoryTask> getHistoryTasks(String orderId);
```

根据参与者分页查询已完成的历史任务

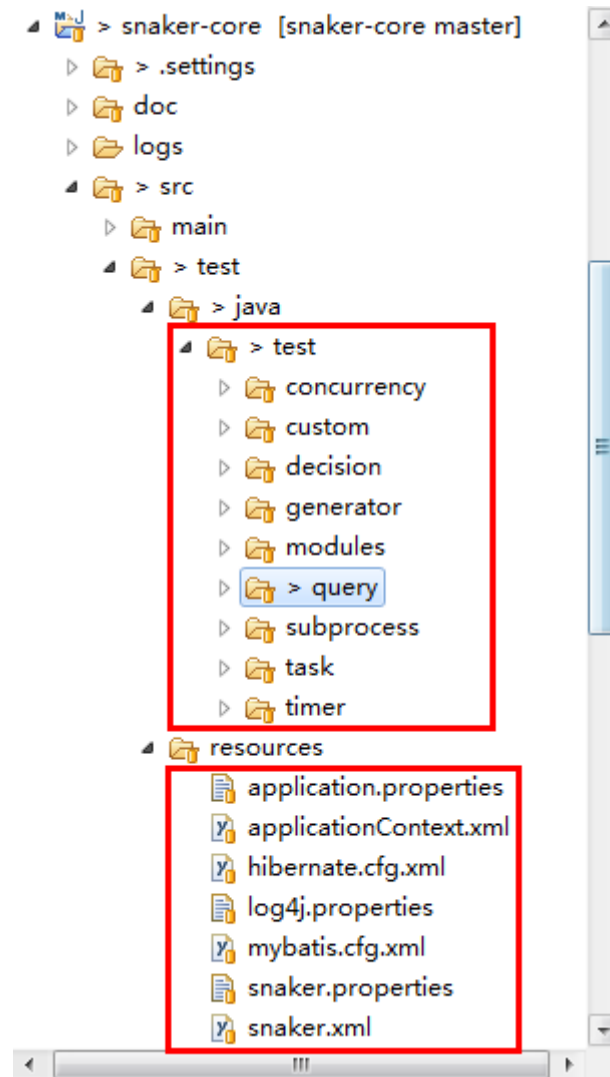
```
List<HistoryTask> getHistoryTasks(Page<HistoryTask> page, Long...  
actorIds);
```

根据流程定义ID、参与者分页查询已完成的历史任务项

```
List<WorkItem> getHistoryWorkItems(Page<WorkItem> page, String processId,  
Long... actorIds);
```

2、测试用例

snaker-core 项目同时提供了大量的 JUnit 测试用例，位于 test 包下。其中 resources 提供了各种场景的配置文件，如下图所示：

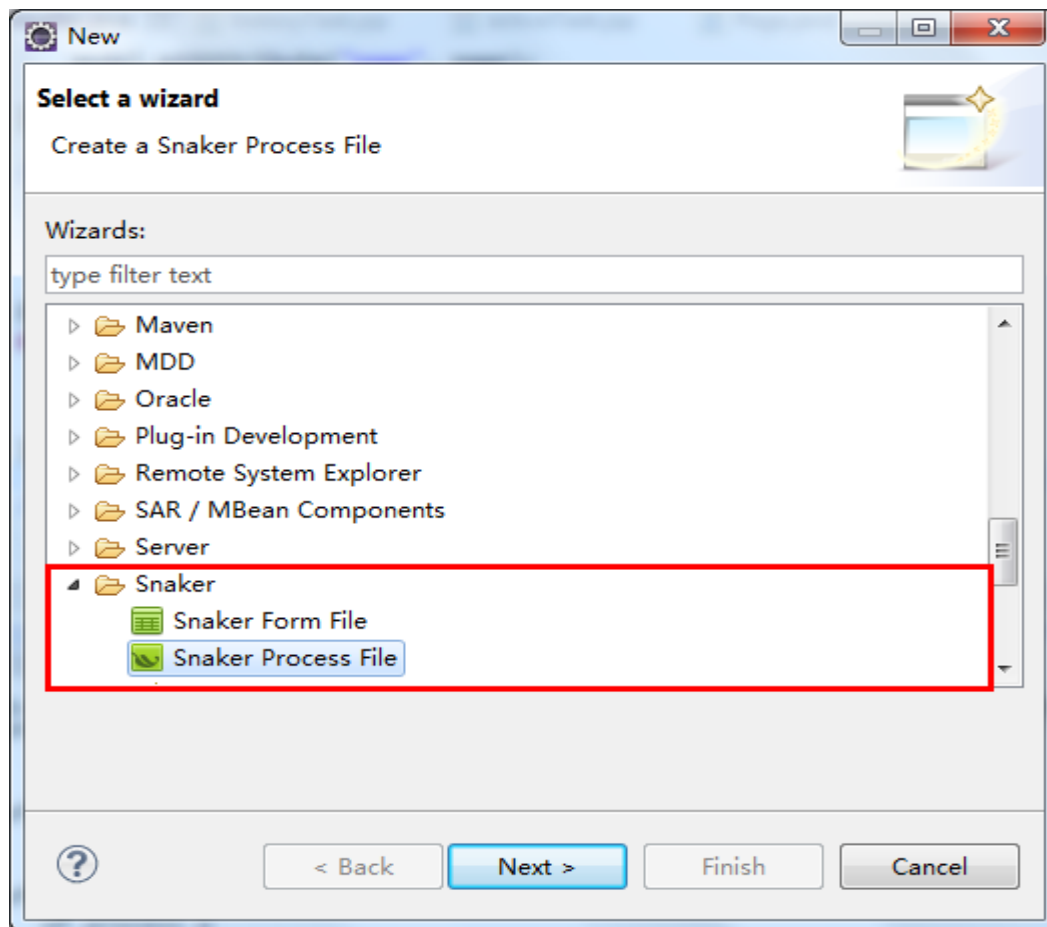


3、流程设计器

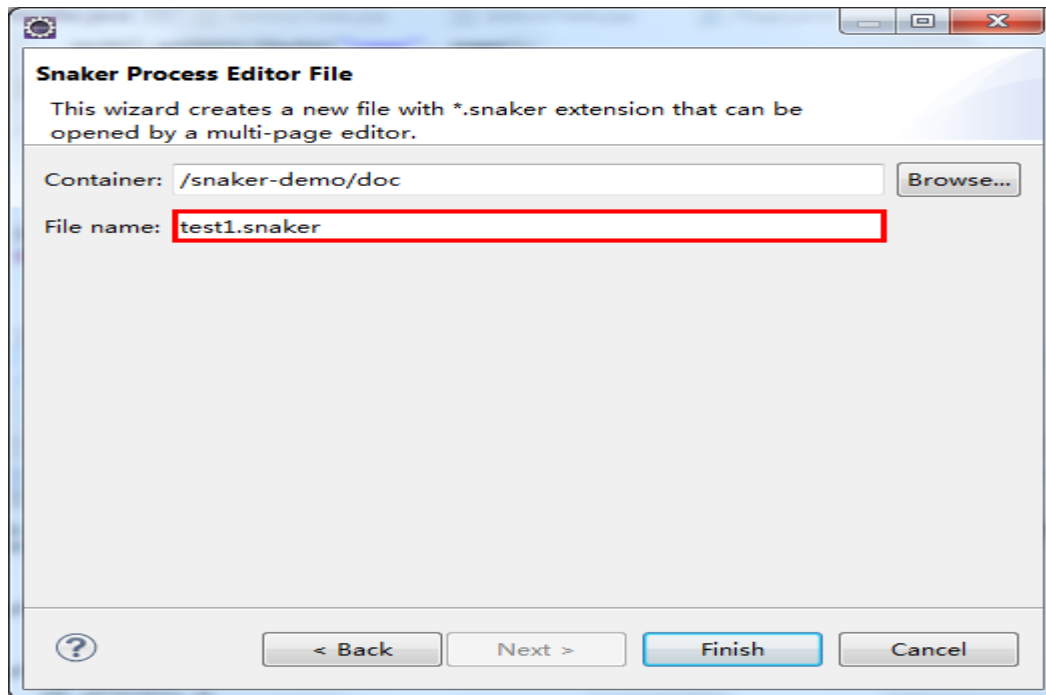
snaker-designer 流程设计器插件包(**snaker-designer_1.1.0.jar**)

http://git.oschina.net/yuqs/snaker-core/attach_files

复制 **snaker-designer_1.1.0.jar** 至 eclipse 的 plugins 文件夹下, 重新启动 eclipse 即可(经过测试的版本有 eclipse4.2、eclipse4.3), 依次选择: **File->New->Other->Snaker**, 如图所示:



选择 Snaker Process File 并输入文件名称, 如下图所示:



点击 Finish，则打开流程设计器，其中包括两大部分：流程组件、属性 Properties 视图

流程组件：

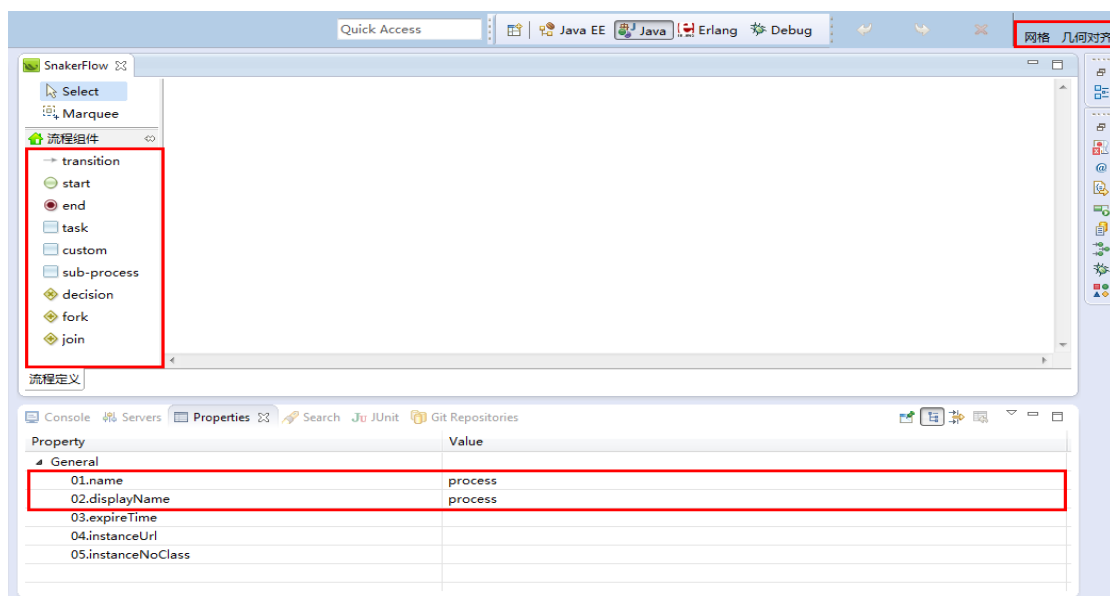
目前包括节点组件为：start、end、task、custom、sub-process、decision、fork、join，分别对应开始、结束、任务、自定义、子流程、决策、分支、合并组件模型。

属性视图：

对组件模型设置属性，包括常用的 name、displayName 等。

布局工具：

右上角的网格、几何对齐用于图形布局。

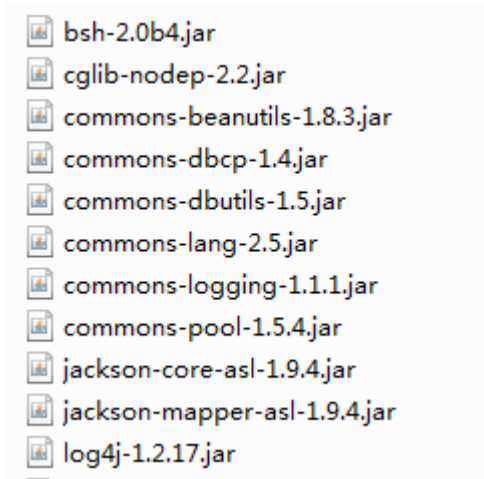


流程设计器组件属性说明：

组件模型	属性	描述
通用属性	name	组件名称，模型内名称唯一
	displayName	组件中文显示名称，方便阅读
Process	expireTime	期望完成时间，设置表达式变量由参数传递
	instanceUrl	流程定义列表页面直接启动流程实例的 URL
	instanceNoClass	流程实例编号生成类
Transition	expr	决策选择 Decision 节点的输出变迁表达式
Task	form	用户参与的表单任务对应的 URL
	assignee	任务参与者变量
	expireTime	期望完成时间，设置表达式变量由参数传递
	performType	任务参与类型（针对多个参与者），ANY 为其中一个参与者完成即往下流转；ALL 为所有参与者完成才往下流转
Custom	clazz	自定义节点的 Java 类路径，两种方式： 1、实现 IHandler 接口，实现接口时不需要设置下面三个属性。 2、无接口实现的普通 java 类，需要设置下面方法名称、参数属性
	methodName	定义需要执行的 java 类的方法名称
	args	定义传递的参数表达式
	var	定义返回值变量名称
SubProcess	processName	子流程名称（对应 process 的 name 属性）
Decision	expr	决策选择表达式
	handleClass	决策选择的处理类，实现 DecisionHandler 接口

4、项目整合

snaker 依赖的第三方开源库集合为：



整合 snaker 必须增加 snaker.properties，选择性的配置 snaker.xml。

snaker.properties:

属性名称	描述	默认值
config	定义 snaker.xml 文件的位置	snaker.xml
jdbc.pageSize	查询结果的每页显示数量	15
subprocessurl	子流程运行状态显示页面	snaker/subprocess/view
jdbc.*	如果指定了 JDBC+dbutils 方式但未提供 DataSource 时使用	无
hibernate.*	如果指定了 Hibernate 方式但未提供 SessionFactory 时使用	无

snaker.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <!-- jdbc的数据库访问与事务拦截器
    <bean class="org.snaker.engine.access.jdbc.JdbcAccess"/>
    <bean
class="org.snaker.engine.access.transaction.DataSourceTransactionIntercepto
r"/>
```

```

-->

<!--hibernate3的数据库访问与事务拦截器-->

<bean class="org.snaker.engine.access.hibernate3.HibernateAccess"/>

<bean

class="org.snaker.engine.access.transaction.Hibernate3TransactionIntercepto
r"/>

<!--mybatis的数据库访问与事务拦截器

<bean class="org.snaker.engine.access.mybatis.MybatisAccess"/>

<bean

class="org.snaker.engine.access.transaction.MybatisTransactionInterceptor"/
>

-->

<bean class="org.snaker.engine.access.dialect.MySqlDialect"/>

<!-- 数据库方言配置。如果使用orm框架已自带了方言的支持，这里不需要配置

<bean class="org.snaker.engine.access.dialect.OracleDialect"/>

<bean class="org.snaker.engine.access.dialect.SQLServerDialect"/>

-->

<!-- 任务拦截器，这里可配置多个拦截器，在任务执行后进行拦截处理(如：记录日志、
短信提醒等) -->

<bean class="org.snaker.engine.impl.LogInterceptor"/>

</config>

```

如果与 spring 整合时，则不需要配置 DBAccess、TransactionInterceptor 实现类。使用 SpringJdbc、Mybatis 时，则需要配置 dialect 方言支持，使用其它 Hibernate 框架不需要配置 dialect。

此处配置的拦截器属于全局拦截器，只要产生 Task，就会执行拦截处理。

4.1、与 Spring 集成

Spring 集成 Snaker 时，需要配置事务管理、数据访问方式。

配置 Snaker:

```
<!-- SnakerEngine配置 -->  
<bean id="snakerEngine"  
class="org.snaker.engine.spring.SnakerEngineFactoryBean"/>
```

配置事务:

```
<tx:method name="start*" propagation="REQUIRED"/>  
<tx:method name="execute*" propagation="REQUIRED"/>  
<tx:method name="save*" propagation="REQUIRED"/>  
<tx:method name="delete*" propagation="REQUIRED" />  
<tx:method name="update*" propagation="REQUIRED" />  
<tx:method name="remove*" propagation="REQUIRED" />  
<tx:method name="assign*" propagation="REQUIRED" />  
<tx:method name="create*" propagation="REQUIRED" />  
<tx:method name="complete*" propagation="REQUIRED" />  
<tx:method name="finish*" propagation="REQUIRED" />  
<tx:method name="terminate*" propagation="REQUIRED" />  
<tx:method name="take*" propagation="REQUIRED" />  
<tx:method name="deploy*" propagation="REQUIRED" />  
<tx:method name="undeploy*" propagation="REQUIRED" />  
<tx:method name="get*" propagation="REQUIRED" read-only="true" />
```

增加拦截点:

```
execution(* org.snaker.engine..*.(..))
```

Spring+hibernate:

配置 hbm 映射文件:

需要在 sessionFactory 的 bean 配置中增加 mappingResources 属性

```
<property name="mappingResources">
```

```

        <list>
            <value>hbm/snaker.task.hbm.xml</value>
            <value>hbm/snaker.order.hbm.xml</value>
            <value>hbm/snaker.process.hbm.xml</value>
            <value>hbm/snaker.taskactor.hbm.xml</value>
            <value>hbm/snaker.workitem.hbm.xml</value>
        </list>
    </property>

```

配置 HibernateAccess:

```

<!-- hibernate access -->
<bean id="dbAccess"
class="org.snaker.engine.access.hibernate3.HibernateAccess">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

```

Spring+Mybatis:

配置 mybatis 的 sqlSessionFactory:

```

<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="configLocation"
value="classpath:mybatis.cfg.xml"></property>
    <property name="dataSource" ref="dataSource" />
    <property name="typeAliasesPackage" value="org.snaker.engine.entity"
/>
</bean>

```

mybatis.cfg.xml:

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

```

```

    <mappers>
      <mapper resource="mapper/process.xml"/>
      <mapper resource="mapper/order.xml"/>
      <mapper resource="mapper/task.xml"/>
      <mapper resource="mapper/task-actor.xml"/>
      <mapper resource="mapper/hist-order.xml"/>
      <mapper resource="mapper/hist-task.xml"/>
      <mapper resource="mapper/hist-task-actor.xml"/>
      <mapper resource="mapper/query.xml"/>
      <mapper resource="mapper/hist-query.xml"/>
    </mappers>
  </configuration>

```

配置MybatisAccess:

```

<bean id="mybatisAccess"
class="org.snaker.engine.access.mybatis.MybatisAccess">
  <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
</bean>

```

配置方言（在 snaker.xml 中增加方言）:

```

<bean class="org.snaker.engine.access.dialect.MySqlDialect"/>

```

Spring+Springjdbc:

配置SpringJdbcAccess:

```

<bean id="dbAccess"
class="org.snaker.engine.access.spring.SpringJdbcAccess">
  <property name="dataSource" ref="dataSource"/>
  <property name="lobHandler" ref="lobHandler"/>
</bean>

<bean id="lobHandler"
class="org.springframework.jdbc.support.lob.DefaultLobHandler" lazy-

```



```
init="true" />
```

配置方言（在 `snaker.xml` 中增加方言）：

```
<bean class="org.snaker.engine.access.dialect.MySqlDialect"/>
```

4.2、使用帮助类整合

使用帮助类整合时，示例代码如下：

```
import javax.sql.DataSource;
```

```
import org.snaker.engine.SnakerEngine;
```

```
import org.snaker.engine.access.jdbc.JdbcHelper;
```

```
import org.snaker.engine.cfg.Configuration;
```

```
public class SnakerHelper {
```

```
    private static final SnakerEngine engine;
```

```
    static {
```

```
        DataSource dataSource = JdbcHelper.getDataSource();
```

```
        engine = new Configuration()
```

```
            .initAccessDBObject(dataSource)
```

```
            .buildSnakerEngine();
```

```
    }
```

```
    public static SnakerEngine getEngine() {
```

```
        return engine;
```

```
    }
```

```
}
```

这里需要注意 `initAccessDBObject` 方法传递的参数值与 `snaker.xml` 配置的数据库访问对应关系：

参数类型	数据库访问对象	事务管理拦截器
DataSource	JdbcAccess	DataSourceTransactionInterceptor
SessionFactory	HibernateAccess	Hibernate3TransactionInterceptor
SqlSessionFactory	MybatisAccess	MybatisTransactionInterceptor

initAccessDBObject 方法也可以不调用,而使用 snaker.properties 中的 jdbc.*或 hibernate.* 来初始化 DataSource、SessionFactory、SqlSessionFactory。但在实际的项目,还是建议使用该项目已有的访问对象

5、如何扩展

DBAccess 扩展:

如果当前提供的数据库访问方式无法满足实际环境，可实现 **DBAccess** 接口或继承 **AbstractDBAccess** 抽象类，完成特定 **orm** 框架持久化操作。

Dialect 扩展:

目前流程引擎支持 **mysql**、**sqlserver**、**oracle** 三种方言，如果使用其它数据库，并且所使用的 **orm** 框架不支持方言时则可实现 **Dialect** 进行扩展。

INoGenerator 扩展:

流程实例编号生成器，作者从事很多的工作流项目都存在编号的概念，如请假流程的编号规则：**QJ_2013_12_0001**。

TaskInterceptor 扩展:

任务拦截器属于全局后置拦截器，对所产生的任务进行拦截处理。如常用的日志记录、邮件提醒、短信提醒等。目前只提供简单的 **LogInterceptor** 打印任务的日志信息。

流程组件模型扩展:

如果当前流程组件模型不满足实际的业务场景，可扩展新的组件模型。步骤如下：

1、增加组件模型：

继承 **NodeModel** 类，覆盖 **execute** 方法，完成组件的执行逻辑

2、增加组件模型解析：

继承 **AbstractNodeParser** 类，覆盖 **newModel**、**parseNode** 方法，**newModel** 方法返回模型对象，**parseNode** 方法用于解析 **xml** 数据设置模型属性

3、配置组件模型：

配置 **snaker.xml**，增加新增组件的解析器，如：

```
<bean name="custom" class="org.snaker.engine.parser.impl.CustomParser"/>
```