# Drools文档

JBoss Drools团队<https://www.drools.org/community/team.html>版本7.7.0.Final

目录

* [1.介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_welcome)
  + [1.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction)
  + [1.2。卷入](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_gettingstarted)
    - [1.2.1。注册jboss.org](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sign_up_to_jboss_org)
    - [1.2.2。签署贡献者协议](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sign_the_contributor_agreement)
    - [1.2.3。通过JIRA提交问题](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_submitting_issues_via_jira)
    - [1.2.4。叉GitHub](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fork_github)
    - [1.2.5。编写测试](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_writing_tests)
    - [1.2.6。坚持正确的惯例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commit_with_correct_conventions)
    - [1.2.7。提交合并请求](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_submit_pull_requests)
  + [1.3。安装和设置（核心和IDE）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installationandsetup)
    - [1.3.1。安装和使用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_and_using)
    - [1.3.2。从源头上构建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_building_from_source)
    - [1.3.3。蚀](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_eclipse)
* [2.发行说明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_droolsreleasenoteschapter)
  + [2.1。流口水中新增和值得注意的东西7.7](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.7.7.0)
    - [2.1.1。可执行模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_executable_model)
  + [2.2。KIE Workbench 7.7.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.7.0.final)
    - [2.2.1。面向项目的工作台](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_oriented_workbench)
    - [2.2.2。连接到远程Kie服务器控制器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_connecting_to_a_remote_kie_server_controller)
    - [2.2.3。内容管理增强](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_content_management_enhancements)
    - [2.2.4。新的迁移工具](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_migration_tool)
  + [2.3。KIE Workbench 7.6.0中的新增值得关注](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.6.0.final)
    - [2.3.1。内容管理增强](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_content_management_enhancements_2)
  + [2.4。新的Kie服务器控制器客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieserver.releasenoteskie.7.5.1)
  + [2.5。从7.0开始改变Kie Server 7.5.1的变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_breaking_changes_in_kie_server_7_5_1_from_7_0)
    - [2.5.1。Kie服务器控制器API更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_controller_api_changes)
    - [2.5.2。Kie服务器API更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_api_changes)
  + [2.6。KIE Workbench 7.5.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.5.0.final)
    - [2.6.1。内容管理增强](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_content_management_enhancements_3)
    - [2.6.2。额外的导航组件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_extra_navigation_components)
  + [2.7。KIE Workbench 7.4.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.4.0.final)
    - [2.7.1。指导决策表的改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_table_improvements)
    - [2.7.2。禁用实验编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_disable_experimental_editors)
  + [2.8。KIE Workbench 7.3.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.3.0.final)
    - [2.8.1。新家和菜单栏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_home_and_menu_bar)
    - [2.8.2。管理页面更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_admin_page_changes)
  + [2.9。什么是Drools中的新特性和值得注意的7.1](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.7.1.0)
  + [2.10。KIE Workbench 7.1.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.1.0.final)
    - [2.10.1。项目指标仪表板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_metrics_dashboard)
    - [2.10.2。团队指标仪表板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_teams_metrics_dashboard)
  + [2.11。Drools 7.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.7.0.0)
    - [2.11.1。核心引擎](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_core_engine)
    - [2.11.2。工作台](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_workbench)
  + [2.12。在6.x中打破Drools 7.0的变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_breaking_changes_in_drools_7_0_from_6_x)
    - [2.12.1。属性反应性默认启用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_property_reactivity_enabled_by_default)
    - [2.12.2。保存类型保存功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_type_preserving_accumulate_functions)
    - [2.12.3。重命名TimedRuleExecutionOption](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_renaming_timedruleexecutionoption)
    - [2.12.4。重命名和统一配置文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_renaming_and_unification_of_configuration_files)
  + [2.13。KIE Workbench 7.0.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.0.0.final)
    - [2.13.1。新创作（图书馆）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_authoring_library)
    - [2.13.2。创作 - 进口的例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_imports_of_examples)
    - [2.13.3。创作 - 弹出窗口的改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_pop_ups_improvements)
    - [2.13.4。创作 - 项目编辑器 - 重新导入按钮](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_project_editor_reimport_button)
    - [2.13.5。安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_management)
    - [2.13.6。kie-config-cli已被删除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_config_cli_has_been_removed)
    - [2.13.7。用户和项目管理页面和首选项](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_and_project_admin_pages_and_preferences)
    - [2.13.8。GAV冲突检查和儿童GAV版本](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_gav_conflict_check_and_child_gav_edition)
    - [2.13.9。数据源管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_source_management)
  + [2.14。在6.x中打破Kie Server 7.0的变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieserver.releasenoteskie.7.0.0)
    - [2.14.1。ServiceResponse XStream编组更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_serviceresponse_xstream_marshalling_changes)
    - [2.14.2。简化的Planner REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_simplified_planner_rest_api)
  + [2.15。什么是Drools 6.5.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.5.0)
    - [2.15.1。可配置的ThreadFactory](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configurable_threadfactory)
    - [2.15.2。使用任何表达式作为查询的输入](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_use_of_any_expressions_as_input_for_a_query)
    - [2.15.3。用修改后的属性更新](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_update_with_modified_properties)
    - [2.15.4。监测框架的改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_monitoring_framework_improvements)
  + [2.16。什么是Drools 6.4.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.4.0)
    - [2.16.1。更好的Java 8兼容性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_better_java_8_compatibility)
    - [2.16.2。更强大的增量编译](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_more_robust_incremental_compilation)
    - [2.16.3。改进了多线程行为](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_multi_threading_behaviour)
    - [2.16.4。OOPath改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_oopath_improvements_2)
  + [2.17。KIE Workbench 6.4.0中的新增值得关注](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.4.0.final)
    - [2.17.1。新的外观和感觉](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_look_and_feel)
    - [2.17.2。各种UI改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_various_ui_improvements)
    - [2.17.3。新的区域设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_locales)
    - [2.17.4。创作 - 导入 - 一致的术语](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_imports_consistent_terminology)
    - [2.17.5。禁用自动构建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_disable_automatic_build)
    - [2.17.6。支持SCP样式git存储库URL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_support_for_code_scp_code_style_code_git_code_repository_urls)
    - [2.17.7。创作 - 重复的GAV检测](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_duplicate_gav_detection)
    - [2.17.8。新的执行服务器管理用户界面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_execution_server_management_user_interface)
    - [2.17.9。用户和组管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_and_group_management)
  + [2.18。什么是Drools 6.3.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.3.0)
    - [2.18.1。使用OOPath浏览对象的图形](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_browsing_graphs_of_objects_with_oopath)
    - [2.18.2。Eclipse的Kie Navigator视图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_shared.releasenotes.kienavigator)
  + [2.19。KIE Workbench 6.3.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.3.0.final)
    - [2.19.1。实时验证和确认决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_real_time_validation_and_verification_for_the_decision_tables)
    - [2.19.2。改进的DRL编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_drl_editor)
    - [2.19.3。资产锁定](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_asset_locking)
    - [2.19.4。数据建模器工具窗口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_modeller_tool_windows)
    - [2.19.5。生成启用JPA的数据模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_generation_of_jpa_enabled_data_models)
    - [2.19.6。数据集创作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_set_authoring)
  + [2.20。什么是Drools 6.2.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.2.0)
    - [2.20.1。传播模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_propagation_modes)
  + [2.21。KIE Workbench 6.2.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.2.0.final)
    - [2.21.1。将版本库或版本库的一部分下载为ZIP](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_download_repository_or_part_of_the_repository_as_a_zip)
    - [2.21.2。项目编辑器权限](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_editor_permissions)
    - [2.21.3。在指导决策表向导中统一验证样式。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_unify_validation_style_in_guided_decision_table_wizard)
    - [2.21.4。改进的奇才](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_wizards)
    - [2.21.5。XLS，导向决策表和指导模板的一致行为](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_consistent_behaviour_of_xls_guided_decision_tables_and_guided_templates)
    - [2.21.6。改进的元数据选项卡](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_metadata_tab)
    - [2.21.7。改进的数据对象编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_data_objects_editor)
    - [2.21.8。执行服务器管理UI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_execution_server_management_ui)
    - [2.21.9。社交活动](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_social_activities)
    - [10年2月21日。贡献者仪表板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_contributors_dashboard)
    - [11年2月21日。包选择器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_package_selector)
    - [12年2月21日。改善视觉一致性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_visual_consistency)
    - [13年2月21日。指导决策树编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_tree_editor)
    - [14年2月21日。创建存储库向导](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_create_repository_wizard)
    - [15年2月21日。存储库结构屏幕](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_repository_structure_screen)
  + [2.22。集成中的新增值得关注6.2.0](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.releasenotesintegration.6.2.0)
    - [2.22.1。KIE执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_execution_server)
  + [2.23。什么是Drools 6.1.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.1.0)
    - [2.23.1。JMX支持KieScanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jmx_support_for_kiescanner)
  + [2.24。KIE Workbench 6.1.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.1.0)
    - [2.24.1。Data Modeler - 往返和源代码保存](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_modeler_round_trip_and_source_code_preservation)
    - [2.24.2。Data Modeler - 改进了注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_modeler_improved_annotations)
    - [2.24.3。标准化表格数据的显示](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_standardization_of_the_display_of_tabular_data)
    - [2.24.4。生成modify(x) {…​}块](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_generation_of_code_modify_x_code_blocks)
  + [2.25。KIE A​​PI 6.0.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.releasenoteskie.6.0.0)
    - [2.25.1。新的KIE名字](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_kie_name)
    - [2.25.2。Maven对齐项目和模块以及Maven部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_aligned_projects_and_modules_and_maven_deployment)
    - [2.25.3。基于配置和约定的项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configuration_and_convention_based_projects)
    - [2.25.4。KieBase包容](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_inclusion)
    - [2.25.5。KieModules，KieContainer和KIE-CI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodules_kiecontainer_and_kie_ci)
    - [2.25.6。KieScanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiescanner)
    - [2.25.7。分层类加载器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hierarchical_classloader)
    - [2.25.8。旧版API适配器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_legacy_api_adapter)
    - [2.25.9。KIE文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_documentation)
  + [2.26。什么是Drools 6.0.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.0.0)
    - [2.26.1。PHREAK - 惰性规则匹配算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_phreak_lazy_rule_matching_algorithm)
    - [2.26.2。自动开启被动模式下的定时规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_automatically_firing_timed_rule_in_passive_mode)
    - [2.26.3。表达式定时器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_expression_timers)
    - [2.26.4。RuleFlowGroups和AgendaGroups被合并](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleflowgroups_and_agendagroups_are_merged)
  + [2.27。KIE Workbench 6.0.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.0.0)
  + [2.28。集成6.0.0中新增值得注意的一点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.releasenotesintegration.6.0.0)
    - [2.28.1。CDI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cdi)
    - [2.28.2。弹簧](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spring)
    - [2.28.3。白羊座蓝图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_aries_blueprints)
    - [2.28.4。OSGi就绪](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_osgi_ready)
* [3.兼容性矩阵](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_compatibilitymatrix)
* [4. KIE](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiechapter)
  + [4.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieoverviewsection)
    - [4.1.1。项目剖析](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieanatomysection)
    - [4.1.2。生命周期](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kielifecyclessection)
  + [4.2。构建，部署，利用和运行](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_builddeployutilizeandrunsection)
    - [4.2.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemoduleintroductionbuildingintroductionsection)
    - [4.2.2。建造](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebuildingsection)
    - [4.2.3。部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiedeployingsection)
    - [4.2.4。运行](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kierunningsection)
    - [4.2.5。安装和部署备忘单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiedeploycheatsheets)
    - [4.2.6。构建，部署和利用示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieexamplessection)
  + [4.3。安全](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesecuritysection)
    - [4.3.1。安全经理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securitymanager)
* [5.混合推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hybridreasoningchapter)
  + [5.1。人工智能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_artificial_intelligence)
    - [5.1.1。一点历史](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_little_history)
    - [5.1.2。知识表示与推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_representation_and_reasoning)
    - [5.1.3。规则引擎和生产规则系统（PRS）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_engines_and_production_rule_systems_prs)
    - [5.1.4。混合推理系统（HRS）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hybrid_reasoning_systems_hrs)
    - [5.1.5。专家系统](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_expert_systems)
    - [5.1.6。推荐阅读](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_recommended_reading)
  + [5.2。Rete算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reteoo)
  + [5.3。ReteOO算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reteoosection)
  + [5.4。PHREAK算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_phreak)
* [6.用户指南](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_guide)
  + [6.1。基础](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_basics)
    - [6.1.1。无状态知识会话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stateless_knowledge_session)
    - [6.1.2。有状态的知识会议](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stateful_knowledge_session)
    - [6.1.3。方法与规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_methods_versus_rules)
    - [6.1.4。交叉产品](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cross_products)
  + [6.2。执行控制](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_execution_control)
    - [6.2.1。议程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agenda)
    - [6.2.2。规则匹配和冲突集。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_matches_and_conflict_sets)
  + [6.3。推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_inference)
    - [6.3.1。巴士通行证示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_bus_pass_example)
  + [6.4。用逻辑对象维护真相](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_truth_maintenance_with_logical_objects)
    - [6.4.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview)
  + [6.5。电子表格中的决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_decision_tables_in_spreadsheets)
    - [6.5.1。何时使用决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_when_to_use_decision_tables)
    - [6.5.2。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview_2)
    - [6.5.3。决策表如何工作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_how_decision_tables_work)
    - [6.5.4。电子表格语法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spreadsheet_syntax)
    - [6.5.5。创建和集成基于电子表格的决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_and_integrating_spreadsheet_based_decision_tables)
    - [6.5.6。管理决策表中的业务规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_business_rules_in_decision_tables)
    - [6.5.7。规则模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_templates)
  + [6.6。记录](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_logging)
* [7.跑步](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_running)
  + [7.1。KieRuntime](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieruntime_3)
    - [7.1.1。入口点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_entrypoint)
    - [7.1.2。RuleRuntime](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleruntime)
    - [7.1.3。StatefulRuleSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_statefulrulesession)
  + [7.2。议程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agenda_2)
    - [7.2.1。解决冲突](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conflict_resolution_2)
    - [7.2.2。AgendaGroup](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agendagroup)
    - [7.2.3。的ActivationGroup](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_activationgroup)
    - [7.2.4。RuleFlowGroup](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleflowgroup)
  + [7.3。事件模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_model_2)
  + [7.4。StatelessKieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_statelesskiesession_2)
    - [7.4.1。顺序模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sequential_mode)
  + [7.5。规则执行模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_execution_modes)
    - [7.5.1。被动模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_passive_mode)
    - [7.5.2。活动模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_active_mode)
  + [7.6。传播模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_propagation_modes_2)
  + [7.7。命令和CommandExecutor](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commands_and_the_commandexecutor_2)
* [8.规则语言参考](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_droolslanguagereferencechapter)
  + [8.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview_3)
    - [8.1.1。规则文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_rule_file)
    - [8.1.2。什么制定规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_what_makes_a_rule)
  + [8.2。关键词](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_keywords)
  + [8.3。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_comments)
    - [8.3.1。单行评论](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_single_line_comment)
    - [8.3.2。多行评论](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_multi_line_comment)
  + [8.4。错误消息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_error_messages)
    - [8.4.1。消息格式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_message_format)
    - [8.4.2。错误消息说明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_error_messages_description)
    - [8.4.3。其他消息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_other_messages)
  + [8.5。包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_package)
    - [8.5.1。进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_import)
    - [8.5.2。全球](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_global)
  + [8.6。功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_function)
  + [8.7。类型声明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_type_declaration)
    - [8.7.1。声明新类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_new_types)
    - [8.7.2。声明元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_metadata)
    - [8.7.3。为现有类型声明元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_metadata_for_existing_types)
    - [8.7.4。声明类型的参数化构造函数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_parametrized_constructors_for_declared_types)
    - [8.7.5。非类型安全类](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_non_typesafe_classes)
    - [8.7.6。从应用程序代码访问已声明的类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_accessing_declared_types_from_the_application_code)
    - [8.7.7。类型声明'扩展'](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_type_declaration_extends)
  + [8.8。规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule)
    - [8.8.1。规则属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_attributes)
    - [8.8.2。定时器和日历](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_timers_and_calendars)
    - [8.8.3。左手边（当）语法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_left_hand_side_when_syntax)
    - [8.8.4。右手边（当时）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_right_hand_side_then)
    - [8.8.5。有条件的命名后果](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_named_consequences)
    - [8.8.6。关于自动装箱和原始类型的注意事项](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_note_on_auto_boxing_and_primitive_types)
  + [8.9。询问](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_querysection)
  + [8.10。域特定语言](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_domain_specific_languages)
    - [8.10.1。何时使用DSL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_when_to_use_a_dsl)
    - [8.10.2。DSL基础知识](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dsl_basics)
    - [8.10.3。为事实添加约束](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adding_constraints_to_facts)
    - [8.10.4。开发DSL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_developing_a_dsl)
    - [8.10.5。DSL和DSLR参考](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dsl_and_dslr_reference)
* [9.复杂的事件处理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#DroolsComplexEventProcessingChapter)
  + [9.1。复杂的事件处理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_complex_event_processing)
  + [9.2。Drools融合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools_fusion)
  + [9.3。事件语义](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_semantics)
  + [9.4。事件处理模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_processing_modes)
    - [9.4.1。云端模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cloud_mode)
    - [9.4.2。流模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stream_mode)
  + [9.5。会话时钟](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_session_clock)
    - [9.5.1。可用的时钟实现](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_available_clock_implementations)
  + [9.6。滑动窗户](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sliding_windows)
    - [9.6.1。滑动时间窗口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sliding_time_windows)
    - [9.6.2。滑动长度窗口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sliding_length_windows)
    - [9.6.3。窗口声明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_window_declaration)
  + [9.7。流支持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_streams_support)
    - [9.7.1。声明和使用入口点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_and_using_entry_points)
  + [9.8。内存管理事件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_memory_management_for_events)
    - [9.8.1。显式到期抵消](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_explicit_expiration_offset)
    - [9.8.2。推断的过期抵消](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_inferred_expiration_offset)
  + [9.9。时间推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_temporal_reasoning)
    - [9.9.1。时间运算符](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_temporal_operators)
* [10.决策模型和表示法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#drools.DMN)
  + [10.1。决策模型和表示法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#decision_model_and_notation_overview)
    - [10.1.1。一致性级别](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#dmn_conformance_levels)
  + [10.2。执行嵌入在应用程序中的DMN服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#executing_decision_services_from_kie_container)
  + [10.3。在Kie服务器上远程执行DMN服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#executing_decision_services_on_the_kie_execution_server)
  + [10.4。FEEL语言参考](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_language_reference)
    - [10.4.1。FEEL语言语义](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_language_reference_semantics)
  + [10.5。DMN支持Drools](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#dmn_support_in_drools)
    - [10.5.1。DMN实施细节](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#dmn_support_in_drools_details)
* [11.实验特征](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_experimentalfeatureschapter)
  + [11.1。声明性议程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declarative_agenda)
  + [11.2。使用OOPath浏览对象的图形](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_browsing_graphs_of_objects_with_oopath_2)
    - [11.2.1。反应性和非反应性OOPath](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reactive_and_non_reactive_oopath)
  + [11.3。性状](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_traits)
    - [11.3.1。级联特征](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cascading_traits)
  + [11.4。规则单元](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_units_2)
    - [11.4.1。数据源](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_sources)
    - [11.4.2。必须运行并声明守卫RuleUnit](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_imperatively_running_and_declaratively_guarding_a_ruleunit)
    - [11.4.3。RuleUnit身份](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleunit_identity)
* [12. Drools命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.commands)
  + [12.1。API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commandsapisection)
    - [12.1.1。XStream的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_xstream)
    - [12.1.2。JSON](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_json)
    - [12.1.3。JAXB](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jaxb)
  + [12.2。支持的命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commands_supported)
    - [12.2.1。BatchExecutionCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_batchexecutioncommand)
    - [12.2.2。InsertObjectCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_insertobjectcommand)
    - [12.2.3。RetractCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_retractcommand)
    - [12.2.4。ModifyCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_modifycommand)
    - [12.2.5。GetObjectCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getobjectcommand)
    - [12.2.6。InsertElementsCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_insertelementscommand)
    - [12.2.7。FireAllRulesCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fireallrulescommand)
    - [12.2.8。StartProcessCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_startprocesscommand)
    - [12.2.9。SignalEventCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_signaleventcommand)
    - [12.2.10。CompleteWorkItemCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_completeworkitemcommand)
    - [12.2.11。AbortWorkItemCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_abortworkitemcommand)
    - [12.2.12。QueryCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_querycommand)
    - [12.2.13。SetGlobalCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_setglobalcommand)
    - [12.2.14。GetGlobalCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getglobalcommand)
    - [12.2.15。GetObjectsCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getobjectscommand)
* [13. CDI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cdichapter)
  + [13.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_2)
  + [13.2。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_annotations)
    - [13.2.1。@KReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kreleaseid)
    - [13.2.2。@KContainer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kcontainer)
    - [13.2.3。@KBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kbase)
    - [13.2.4。KieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_kiesession)
    - [13.2.5。用于StatelessKieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_statelesskiesession)
  + [13.3。API示例比较](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_api_example_comparison)
* [14.与Spring集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.spring)
  + [14.1。Drools 6.0的重要变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_important_changes_for_drools_6_0)
  + [14.2。与Drools Expert整合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_drools_expert)
    - [14.2.1。KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodule)
    - [14.2.2。KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_3)
    - [14.2.3。重要的提示](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_important_note)
    - [14.2.4。KieSessions](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesessions)
    - [14.2.5。纪伊：ReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_releaseid)
    - [14.2.6。纪伊：进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_import)
    - [14.2.7。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_annotations_2)
    - [14.2.8。事件监听器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_listeners)
    - [14.2.9。记录仪](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_loggers)
    - [14.2.10。定义批处理命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_batch_commands)
    - [14.2.11。坚持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_persistence)
    - [14.2.12。利用其他弹簧功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_leveraging_other_spring_features)
  + [14.3。与jBPM人工任务集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_jbpm_human_task)
    - [14.3.1。如何使用jBPM人工任务配置Spring](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_how_to_configure_spring_with_jbpm_human_task)
* [15.与Aries Blueprint集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.blueprint)
  + [15.1。与Drools Expert整合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_drools_expert_2)
    - [15.1.1。KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodule_2)
    - [15.1.2。KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_4)
    - [15.1.3。KieSessions](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesessions_2)
    - [15.1.4。纪伊：ReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_releaseid_2)
    - [15.1.5。纪伊：进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_import_2)
    - [15.1.6。事件监听器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_listeners_2)
    - [15.1.7。记录仪](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_loggers_2)
    - [15.1.8。定义批处理命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_batch_commands_2)
* [16. Android集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.android)
  + [16.1。与Drools Expert整合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_drools_expert_3)
    - [16.1.1。预序列化规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pre_serialized_rules)
    - [16.1.2。KieContainer API与drools编译器依赖关系](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiecontainer_api_with_drools_compiler_dependency)
  + [16.2。与Roboguice集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_roboguice)
    - [16.2.1。预先序列化规则与Roboguice](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pre_serialized_rules_with_roboguice)
    - [16.2.2。KieContainer与drools编译器依赖和Roboguice](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiecontainer_with_drools_compiler_dependency_and_roboguice)
* [17. Apache Camel集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.camel)
  + [17.1。骆驼](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_camel)
* [18. Drools骆驼服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.server)
  + [18.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_3)
  + [18.2。部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_deployment)
  + [18.3。组态](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configuration)
    - [18.3.1。REST /骆驼服务配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rest_camel_services_configuration)
* [19.使用RHQ / JON进行JMX监控](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rhqchapter)
  + [19.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_4)
    - [19.1.1。在Drools应用程序中启用JMX监视](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_enabling_jmx_monitoring_in_a_drools_application)
    - [19.1.2。安装并运行RHQ / JON插件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_and_running_the_rhq_jon_plugin)
* [20.工作台（一般）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbench)
  + [20.1。安装](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.installation)
    - [20.1.1。战争安装](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.warinstallation)
    - [20.1.2。工作台数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchdata)
    - [20.1.3。系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)
    - [20.1.4。故障排除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.troubleshooting)
  + [20.2。快速开始](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstart)
    - [20.2.1。导入示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartimportexamples)
    - [20.2.2。添加项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartaddproject)
    - [20.2.3。定义数据模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartdefinedatamodel)
    - [20.2.4。定义规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartdefinerule)
    - [20.2.5。构建和部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartbuildanddeloy)
  + [20.3。组态](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.configuration)
    - [20.3.1。基本的用户管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.usermanagement)
    - [20.3.2。角色](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.roles)
  + [20.4。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.introduction)
    - [20.4.1。登录并注销](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.loginandlogout)
    - [20.4.2。主屏幕](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.homescreen)
    - [20.4.3。工作台概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.administrationoverview)
    - [20.4.4。Workbench用户界面概念](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchconcepts)
  + [20.5。更改布局](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.changinglayout)
    - [20.5.1。调整](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.resizing)
  + [20.6。创作（一般）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.authoring)
    - [20.6.1。工件库](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.artifactrepository)
    - [20.6.2。资产编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.asseteditor)
    - [20.6.3。标签编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.tagseditor)
    - [20.6.4。项目浏览器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorer)
    - [20.6.5。项目编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projecteditor)
    - [20.6.6。验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.validation)
    - [20.6.7。数据建模器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datamodeller)
    - [20.6.8。数据集](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasets)
    - [20.6.9。数据源管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasources)
  + [20.7。安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.userandgroupmgmt)
    - [20.7.1。基本概念](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_basic_concepts)
    - [20.7.2。安装和设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installation_and_setup)
    - [20.7.3。用法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_usage_2)
    - [20.7.4。安全设置编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_settings_editor)
    - [20.7.5。安全策略存储](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_policy_storage)
  + [20.8。将嵌入式工作台嵌入您的应用程序中](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.embedding)
  + [20.9。执行服务器管理UI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserverui)
    - [20.9.1。服务器模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserveruitemplate)
    - [20.9.2。容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserveruicontainer)
    - [20.9.3。远程服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserveruiremoteserver)
* [21.编写规则资产](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#drools.AuthoringAssets)
  + [21.1。创建一个包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.newpackage)
    - [21.1.1。空包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_empty_package)
    - [21.1.2。复制，重命名和删除软件包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_copy_rename_and_delete_packages)
  + [21.2。业务规则与指导编辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditor)
    - [21.2.1。引导规则编辑器的各个部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorparts)
    - [21.2.2。规则的“WHEN”（左侧）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorlhs)
    - [21.2.3。规则的“THEN”（右侧）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorrhs)
    - [21.2.4。可选属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditoroptionalattributes)
    - [21.2.5。模式/操作工具栏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditortoolbar)
    - [21.2.6。用户驱动的下拉列表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorenumerations)
    - [21.2.7。增加DSL语句](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditordsl)
    - [21.2.8。一个更复杂的例子：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorcomplexexample)
  + [21.3。资产/规则模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditor)
    - [21.3.1。创建一个规则模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditorcreating)
    - [21.3.2。定义模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditordefining)
    - [21.3.3。定义模板数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditordefiningdata)
    - [21.3.4。生成的DRL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditorgenerateddrl)
  + [21.4。指导决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditor)
    - [21.4.1。决策表的类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditortypesofdecisiontable)
    - [21.4.2。定义一个基于Web的决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditordefining)
    - [21.4.3。命中政策](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorhitpolicy)
    - [21.4.4。审计日志](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorauditlog)
    - [21.4.5。实时验证和验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverification)
  + [21.5。引导决策表图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontablegrapheditor)
    - [21.5.1。编辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editor)
    - [21.5.2。创建一个新图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_a_new_graph)
    - [21.5.3。向图表中添加新表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adding_new_tables_to_the_graph)
    - [21.5.4。删除表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_removing_tables)
  + [21.6。指导决策树](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontreeeditor)
    - [21.6.1。最初的编辑布局](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_initial_editor_layout)
    - [21.6.2。第一步](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_first_steps)
    - [21.6.3。编辑数据对象节点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editing_data_object_nodes)
    - [21.6.4。编辑字段约束节点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editing_field_constraint_nodes)
    - [21.6.5。编辑动作节点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editing_action_nodes)
    - [21.6.6。管理树](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_the_tree)
  + [21.7。电子表格决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.xlsdecisiontable)
    - [21.7.1。转换为指导决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_convert_to_guided_decision_table)
  + [21.8。记分卡](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditor)
    - [21.8.1。（a）设置参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditorsetupparameters)
    - [21.8.2。（b）特点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditorcharacteristics)
  + [21.9。测试场景](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditor)
    - [21.9.1。知识会话选择器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.ksessionselector)
    - [21.9.2。给定部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorgivensection)
    - [21.9.3。期望部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorexpectedsection)
    - [21.9.4。全局部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorglobalsection)
    - [21.9.5。新的输入部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorinpuotsection)
  + [21.10。DSL编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.dsleditor)
  + [21.11。数据枚举（下拉列表配置）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.enumerationeditor)
    - [21.11.1。高级枚举概念](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.advancedenumconcepts)
  + [21.12。DRL文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.drleditor)
* [22.工作台集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchintegration)
  + [22.1。休息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.workbenchremoteapi)
    - [22.1.1。求职电话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_job_calls)
    - [22.1.2。项目调用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_calls)
    - [22.1.3。空间电话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_space_calls)
    - [22.1.4。Maven的电话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_calls)
    - [22.1.5。REST总结](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rest_summary)
    - [22.1.6。嵌入式Kie服务器控制器调用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_embedded_kie_server_controller_calls)
  + [22.2。Keycloak SSO集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.keycloakssointegration)
    - [22.2.1。脚本](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_scenario)
    - [22.2.2。安装并设置Keycloak服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_install_and_setup_a_keycloak_server)
    - [22.2.3。创建并设置演示领域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_create_and_setup_the_demo_realm)
    - [22.2.4。安装并设置jBPM Workbench](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_install_and_setup_jbpm_workbench)
    - [22.2.5。通过Keycloak确保工作台远程服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_workbench_remote_services_via_keycloak)
    - [22.2.6。通过Keycloak确保工作台的文件系统服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_workbench_s_file_system_services_via_keycloak)
    - [22.2.7。执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_execution_server)
    - [22.2.8。消费远程服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#consumingRemoteServices)
    - [22.2.9。Keycloak和工作台的安全管理区域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_keycloak_and_the_workbench_s_security_administration_area)
* [23.工作台高可用性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchhighavailability)
  + [23.1。VFS集群](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.vfsclustering)
  + [23.2。jBPM集群](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jbpm_clustering)
* [24. KIE执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.server)
  + [24.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview_4)
    - [24.1.1。词汇表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_glossary)
  + [24.2。安装KIE服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_the_kie_server)
    - [24.2.1。自举开关](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_bootstrap_switches)
    - [24.2.2。不同容器的安装细节](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installation_details_for_different_containers)
  + [24.3。Kie服务器设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_setup)
    - [24.3.1。托管Kie服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managed_kie_server)
    - [24.3.2。非托管KIE执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_unmanaged_kie_execution_server)
  + [24.4。创建Kie容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_a_kie_container)
  + [24.5。管理容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_containers)
    - [24.5.1。启动一个容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_starting_a_container)
    - [24.5.2。停止并删除容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stopping_and_deleting_a_container)
    - [24.5.3。更新容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_updating_a_container)
  + [24.6。Kie服务器REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.ksrestapi)
    - [24.6.1。[GET] /](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get)
    - [24.6.2。[POST] / config](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_config)
    - [24.6.3。[GET] /容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers)
    - [24.6.4。⁠[GET] / containers / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_id)
    - [24.6.5。[PUT] / containers / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_containers_id)
    - [24.6.6。[DELETE] / containers / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_containers_id)
    - [24.6.7。[POST] / containers / instances / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_instances_id)
    - [24.6.8。[GET] / containers / {id} / release-id](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_id_release_id)
    - [24.6.9。[POST] / containers / {id} / release-id](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_id_release_id)
    - [10年6月24日。[GET] / containers / {id} / scanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_id_scanner)
    - [11年6月24日。[POST] / containers / {id} / scanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_id_scanner)
  + [24.7。OptaPlanner REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_optaplanner_rest_api)
    - [24.7.1。[GET] /容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_2)
    - [24.7.2。[PUT] / containers / {containerId} / solvers / {solverId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_containers_containerid_solvers_solverid)
    - [24.7.3。[GET] / containers / {containerId} /求解器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers)
    - [24.7.4。[GET] / containers / {containerId} / solvers / {solverId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers_solverid)
    - [24.7.5。[POST] / containers / {containerId} / solvers / {solverId} / state / solve](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_containerid_solvers_solverid_state_solving)
    - [24.7.6。[POST] / containers / {containerId} / solvers / {solverId} / state / terminating-early](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_containerid_solvers_solverid_state_terminating_early)
    - [24.7.7。[GET] / containers / {containerId} / solvers / {solverId} / bestsolution](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers_solverid_bestsolution)
    - [24.7.8。[POST] / containers / {containerId} / solvers / {solverId} / problemfactchanges](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_containerid_solvers_solverid_problemfactchanges)
    - [24.7.9。[GET] / containers / {containerId} / solvers / {solverId} / problemfactchanges / processed](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers_solverid_problemfactchanges_processed)
    - [10年7月24日。[DELETE] / containers / {containerId} / solvers / {solverId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_containers_containerid_solvers_solverid)
  + [24.8。控制器REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_controller_rest_api)
    - [24.8.1。[GET] /管理/服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_servers)
    - [24.8.2。[GET] / management / server / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_server_id)
    - [24.8.3。[PUT] / management / server / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_management_server_id)
    - [24.8.4。[DELETE] / management / server / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_management_server_id)
    - [24.8.5。[GET] / management / server / {id} / containers](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_server_id_containers)
    - [24.8.6。[GET] / management / server / {id} / containers / {containerId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_server_id_containers_containerid)
    - [24.8.7。[PUT] / management / server / {id} / containers / {containerId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_management_server_id_containers_containerid)
    - [24.8.8。[DELETE] / management / server / {id} / containers / {containerId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_management_server_id_containers_containerid)
    - [24.8.9。[POST] / management / server / {id} / containers / {containerId} / status / started](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_management_server_id_containers_containerid_status_started)
    - [10年8月24日。[POST] / management / server / {id} / containers / {containerId} / status / stopped](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_management_server_id_containers_containerid_status_stopped)
  + [24.9。Kie服务器Java客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_java_client_api)
    - [24.9.1。Maven配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_configuration_2)
    - [24.9.2。客户端配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_client_configuration)
    - [24.9.3。服务器响应](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_server_response)
    - [24.9.4。服务器功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_server_capabilities)
    - [24.9.5。Kie容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_containers)
    - [24.9.6。管理容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_containers_2)
    - [24.9.7。Decision Server的可用客户端](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_available_clients_for_the_decision_server)
    - [24.9.8。发送命令到服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sending_commands_to_the_server)
    - [24.9.9。列出可用的业务流程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_listing_available_business_processes)
  + [24.10。使用密钥存储保护密码](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_password_using_key_store)
    - [24.10.1。简单的用例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_simple_usecase)
    - [24.10.2。实施和业务逻辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_implementation_and_business_logic)
    - [24.10.3。系统要求](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_system_requirements)
    - [24.10.4。初始化密钥存储](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_initialization_of_a_key_store)
    - [24.10.5。加载密钥库的系统参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_system_parameters_for_loading_key_store)
    - [24.10.6。例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_example_3)
  + [24.11。Kie服务器控制器客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_controller_client_api)
    - [24.11.1。Maven配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_configuration_3)
    - [24.11.2。客户端配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_client_configuration_2)
    - [24.11.3。设置Kie服务器控制器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_setting_up_a_kie_server_controller)
* [25.例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_examples_2)
  + [25.1。获取示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getting_the_examples)
  + [25.2。你好，世界](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hello_world)
  + [25.3。状态示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_state_example)
    - [25.3.1。了解状态示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_understanding_the_state_example)
  + [25.4。斐波那契示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fibonacci_example)
  + [25.5。银行教程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_banking_tutorial)
  + [25.6。定价规则决策表示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pricing_rule_decision_table_example)
    - [25.6.1。执行示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_executing_the_example)
    - [25.6.2。决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_decision_table)
  + [25.7。宠物商店示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pet_store_example)
  + [25.8。诚实的政治家例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_honest_politician_example)
  + [25.9。数独例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_example)
    - [25.9.1。数独概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_overview)
    - [25.9.2。运行示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_running_the_example)
    - [25.9.3。Java源代码和规则概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_java_source_and_rules_overview)
    - [25.9.4。数独验证器规则（validate.drl）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_validator_rules_validate_drl)
    - [25.9.5。数独解决规则（sudoku.drl）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_solving_rules_sudoku_drl)
  + [25.10。猜数字](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_number_guess)
  + [25.11。康威的人生游戏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conway_s_game_of_life)
  + [25.12。侵略者](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders)
    - [25.12.1。Invaders1Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders1main)
    - [25.12.2。Invaders2Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders2main)
    - [25.12.3。Invaders3Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders3main)
    - [25.12.4。Invaders4Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders4main)
    - [25.12.5。Invaders5Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders5main)
    - [25.12.6。Invaders6Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders6main)
    - [25.12.7。Invaders4Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders4main_2)
  + [25.13。冒险与流口水](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adventures_with_drools)
    - [25.13.1。使用游戏。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_the_game)
    - [25.13.2。代码](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_code)
  + [25.14。傍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pong)
  + [25.15。Wumpus世界](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wumpus_world)
  + [25.16。礼仪小姐和标杆管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_miss_manners_and_benchmarking)
    - [25.16.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_5)
    - [25.16.2。深入讨论](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_in_depth_discussion)
    - [25.16.3。输出摘要](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_output_summary)
  + [25.17。逆向链接](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_backward_chaining)
    - [25.17.1。后向链系统](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_backward_chaining_systems)
    - [25.17.2。克隆传递闭包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cloning_transitive_closures)
    - [25.17.3。定义查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_query)
    - [25.17.4。传递闭包示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_transitive_closure_example)
    - [25.17.5。反应性传递查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reactive_transitive_queries)
    - [25.17.6。带有未绑定参数的查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_queries_with_unbound_arguments)
    - [25.17.7。多个未绑定的参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_multiple_unbound_arguments)

# 欢迎

欢迎和发行说明

## [1.介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_welcome)

### [1.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction)

KIE（Knowledge Is Everything）是一项综合性项目，旨在将我们的相关技术融合在一起。它也是我们项目之间共享的核心。

KIE包含以下不同但相关的项目，为业务自动化和管理提供完整的解决方案组合：

1. **Drools**是一个业务规则管理系统，具有前向链和后向链推理规则引擎，可以对业务规则和复杂事件处理进行快速可靠的评估。规则引擎也是创建专家系统的基本组成部分，在人工智能中，专家系统是模拟人类专家决策能力的计算机系统。
2. **jBPM**是一个灵活的业务流程管理套件，允许您通过描述为实现这些目标而需要执行的步骤来为业务目标建模。
3. **OptaPlanner**是一个约束求解器，可以优化员工队伍，车辆路线，任务分配和云优化等用例。
4. **Drools Workbench**是一个全功能的Web应用程序，用于定制业务规则和过程的视觉组合。
5. **UberFire**是一个基于Web的工作台框架，受Eclipse Rich Client Platform的启发。

7.x系列将采用更加灵活的方法，以更加规则和迭代的方式发布。我们计划对一系列次要版本做一些比正常更大的更改，并且用户需要意识到那些要在采用之前即将到来。

1. UI部分和链接将变为面向对象，而不是面向任务。<https://en.wikipedia.org/wiki/Object-oriented_user_interface>
2. 创作/图书馆将成为面向项目，而不是以存储库为导向。您将创建，浏览并打开项目而不是存储库。存储库的概念将被降低，例如当你创建projcet时它会被自动创建。
3. 旧的表单建模器将被删除，只有新的可用模型。尽管旧形式将继续呈现。
4. 新设计师将继续成熟，拥有更多节点和更好的用户体验。最终它将成为默认的编辑器，但我们不会删除旧的版本，除非在BPMN2支持中有功能奇偶校验。
5. 在许多地方继续提高UXD。
6. 我们将介绍AppFormer项目，这将是对现有项目的重组和整合，并导致一些神器重命名。UberFire将成为AppFormer-Core，表单，数据建模器和dashbuilder将归入AppFormer。Dashbuilder很可能会成为Appformer-Insight。

8.x系列将在今年年底推出。我们正在进行并行的工作，以引入改进的git支持的工作空间概念，该工作空间将具有用于分叉和拉取请求的内置工作流程。这将与水平缩放和改进的高可用性相结合。这些变化对可用性和云可扩展性而言非常重要，但是对于次要版本来说，变化太大了，因此碰到8.x

### [1.2。卷入](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_gettingstarted)

我们经常被问到“我如何参与”。幸运的是，答案很简单，只需编写一些代码并提交即可:)没有必要跳过或秘密握手。我们有一个非常小的“开销”，我们要求允许可扩展的项目开发。下面我们提供我们要求的工具和“工作流程”的一般概述，以及一些一般性建议。

如果你提供了一些好的工作，不要忘了写博客:)

#### [1.2.1。注册jboss.org](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sign_up_to_jboss_org)

登录到jboss.org将使您可以访问JBoss wiki，论坛和JIRA。转到<https://www.jboss.org/>并单击“注册”。

#### [1.2.2。签署贡献者协议](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sign_the_contributor_agreement)

您需要签署的唯一形式是贡献者协议，该协议通过网络完全自动化。如下图所示：“这为您的贡献建立了条款和条件，并确保源代码可以得到适当授权”

<https://cla.jboss.org/>

#### [1.2.3。通过JIRA提交问题](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_submitting_issues_via_jira)

为了能够与核心开发团队进行交互，您需要使用问题跟踪器JIRA。这可确保所有请求都被记录并分配到发布时间表以及在一个地方捕获所有讨论。错误报告，错误修复，功能请求和功能提交应该都在这里。一般问题应该在邮件列表中进行。

次要代码提交（如格式或文档修复）不需要创建关联的JIRA问题。

<https://issues.jboss.org/browse/DROOLS>

<https://issues.jboss.org/browse/JBPM>

<https://issues.jboss.org/browse/GUVNOR>

#### [1.2.4。叉GitHub](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fork_github)

通过签署贡献者协议并将您的请求提交给JIRA，您现在应该已准备好编写代码:)创建一个GitHub帐户，并分支Drools，jBPM或Guvnor存储库中的任何一个。分支将在您自己的GitHub空间中创建一个副本，您可以按自己的步调进行操作。如果你犯了一个错误，不要担心把它吹走并再次拨叉。请注意，每个GitHub存储库都为您提供了克隆（checkout）URL，GitHub将为您提供特定于您的分支的URL。

<https://github.com/kiegroup>

#### [1.2.5。编写测试](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_writing_tests)

在编写测试时，尽量让它们保持最小和自我包含。我们倾向于将DRL片段保存在测试中，因为它可以更快地进行检查。如果他们有大量的规则，那么使用一个字符串是不实际的，所以所有的方法都将它们放在单独的DRL文件中，而不是从类路径加载。如果您的测试需要使用模型，请尝试使用那些已经存在用于其他单元测试的测试; 如人员，奶酪或订单。如果不存在具有所需字段的类，请在添加新类之前尝试更新现有类的字段。

有大量的测试来看看想法，MiscTest是一个很好的开始。

[https://github.com/kiegroup/drools/blob/master/drools-compiler/src/test/java/org/drools/integrationtests/MiscTest.java](https://github.com/kiegroup)

#### [1.2.6。坚持正确的惯例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commit_with_correct_conventions)

当你提交时，确保你使用正确的约定。提交必须以JIRA问题标识开头，例如JBRULES-220。这确保提交通过JIRA进行交叉引用，因此我们可以在同一个地方看到针对特定问题的所有提交。在id之后，问题的标题应该接下来。然后使用一个带短划线的换行符来提供与此提交相关的其他信息。针对您希望制作的每个单独点使用额外的新行和短划线。如果合适，您可以将其他JIRA交叉引用添加到同一提交。通常尽量避免在同一提交中组合不相关的问题。

不要忘记重新分配原始大师的本地分支，然后将您的提交返回到您的分支。

#### [1.2.7。提交合并请求](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_submit_pull_requests)

随着您的代码从原始主人重新发布并推送到您的个人GitHub区域，您现在可以将您的工作作为拉取请求提交。如果您在GitHub的页面顶部查看您的工作区域，它们将成为“Pull Request”按钮。选择此项将提供一个gui来自动提交您的请求。

然后拉入请求进入队列，供所有人查看和评论。在下面你可以看到一个典型的拉动请求。拉取请求允许讨论，并显示所有相关的提交以及每次提交的差异。讨论通常涉及代码审查，这些代码审查提供了有助于改进的有用建议，并允许我们在代码的特定部分留下内联评论。如果我们不立即合并，请不要灰心，在我们接受拉取请求之前，通常需要进行多次修改。幸运的是，GitHub使得回到你的代码非常简单，做更多的提交，然后更新你的pull请求到你最新最好的。

我们可能需要一段时间才能回复请求，因此请耐心等待。提交修补程序时提交的测试通常会很快应用，因为只有在我们有时间提交修补程序时才进行测试。不要忘记重新提交并不时重新提交请求，否则随着时间的推移它会合并冲突，核心开发人员将会忽略这些冲突。

### [1.3。安装和设置（核心和IDE）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installationandsetup)

#### [1.3.1。安装和使用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_and_using)

Drools提供了一个基于Eclipse的IDE（这是可选的），但其核心只需要Java 1.5（Java SE）。

一个简单的入门方法是下载并安装Eclipse插件 - 这还需要安装Eclipse GEF框架（如果您尚未安装Eclipse GEF框架，请参阅下文）。这将为您提供所需的所有依赖关系：您可以简单地创建一个新的规则项目，一切都将为您完成。请参阅Rule Workbench和IDE中的章节以获取有关此方面的详细说明。安装Eclipse插件通常就像将文件解压缩到Eclipse插件目录一样简单。

不需要使用Eclipse插件。规则文件只是文本输入（或电子表格，视情况而定），IDE（也称为规则工作台）只是一种方便。人们已经在许多方面整合了规则引擎，没有“一刀切”。

或者，您可以下载二进制发行版，并将相关的JAR包含在您的项目类路径中。

##### [1.3.1.1。依赖和JAR](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dependencies_and_jars)

Drools分解成几个模块，在规则开发/编译过程中需要一些模块，在运行时需要一些模块。在很多情况下，人们只是想在运行时包含所有的依赖关系，这很好。它可以让你拥有最大的灵活性。然而，有些人可能更喜欢将他们的“运行时间”精简到最低限度，因为他们将以二进制形式部署规则 - 这也是可能的。核心运行时引擎可以非常紧凑，在3个JAR文件中只需要几千个字节。

以下是组成JBoss Drools的重要库的描述

* knowledge-api.jar - 这提供了接口和工厂。它还有助于清楚地显示用户API的用途以及引擎API。
* knowledge-internal-api.jar - 这提供了内部接口和工厂。
* drools-core.jar - 这是核心引擎，运行时组件。包含RETE引擎和LEAPS引擎。如果您是预编译规则（并通过Package或RuleBase对象进行部署），则这是唯一的运行时依赖项。
* drools-compiler.jar - 这包含编译器/构建器组件以获取规则源，并构建可执行规则库。这通常是应用程序的运行时依赖关系，但如果您正在预编译规则，则不必这样做。这取决于drools-core。
* drools-jsr94.jar - 这是符合JSR-94的实现，它本质上是drools-compiler组件上的一个层。请注意，由于JSR-94规范的性质，并非所有功能都可以通过此界面轻松公开。在某些情况下，直接访问Drools API会更容易，但在某些环境中，JSR-94是强制的。
* drools-decisiontables.jar - 这是决策表的“编译器”组件，它使用drools-compiler组件。这支持excel和CSV输入格式。

上述组件需要其他的依赖关系，其中大部分依赖于drools编译器，drools-jsr94或drools-decisiontables模块。一些关键要注意的是“POI”，它提供电子表格解析能力，“antlr”提供规则语言本身的解析。

|  |  |
| --- | --- |
|  | 如果您在J2EE或servlet容器中使用Drools，并且遇到了使用“JDT”的类路径问题，那么您可以切换到janino编译器。设置系统属性“drools.compiler”：例如：-Ddrools.compiler = JANINO。 |

有关发行版中依赖关系的最新信息，请参阅发布的POM，它可以在Maven存储库中找到。

##### [1.3.1.2。使用Maven，Gradle，Ivy，Buildr或Ant](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_use_with_maven_gradle_ivy_buildr_or_ant)

将广口瓶同时有[中央Maven仓库](http://search.maven.org/#search|ga|1|org.drools)（也在<https://repository.jboss.org/nexus/index.html#nexus-search;gav>org.drools 〜 [JBoss的Maven仓库]）。

如果您使用Maven，请将KIE和Drools依赖项添加到项目的pom.xml中， 如下所示：

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-bom</artifactId>

<type>pom</type>

<version>...</version>

<scope>import</scope>

</dependency>

...

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.kie</groupId>

<artifactId>kie-api</artifactId>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-compiler</artifactId>

<scope>runtime</scope>

</dependency>

...

<dependencies>

这与Gradle，Ivy和Buildr类似。要确定最新版本，请检查Maven存储库。

如果您仍在使用Ant（不含常青藤），请从下载的zip 二进制文件 目录复制所有JAR，并手动确认您的类路径不包含重复的JAR。

##### [1.3.1.3。运行](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_runtime)

这里提到的“运行时”要求是，如果您将规则部署为二进制形式（作为KnowledgePackage对象或KnowledgeBase对象等）。这是一个可选功能，可让您保持运行时非常轻。您可以使用drools编译器生成“过程外”的规则包，然后将它们部署到运行时系统。这个运行时系统只需要drools-core.jar和knowledge-api来执行。这是一种可选的部署模式，许多人不需要“修剪”他们的应用程序，但它是某些环境的理想选择。

##### [1.3.1.4。安装IDE（规则工作台）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_ide_rule_workbench)

规则工作台（对于Eclipse）要求您具有Eclipse 3.4或更高版本以及Eclipse GEF 3.4或更高版本。您可以通过下载插件或使用更新站点进行安装。

另一种选择是使用JBoss IDE，它提供了预先打包的所有插件需求，以及与规则分开的其他工具。您可以选择仅从JBoss IDE附带的“捆绑”中安装规则。

###### [安装GEF（必需的依赖项）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_gef_a_required_dependency)

GEF是Eclipse图形编辑框架，用于插件中的图形查看组件。

如果您没有安装GEF，可以使用内置更新机制进行安装（或者从Eclipse.org网站下载不推荐的GEF）。JBoss IDE已经有了GEF，就像Eclipse的许多其他“发行版”一样，所以对于某些人来说，这一步可能是多余的。

从帮助菜单中打开帮助→软件更新→可用软件→添加站点...。位置是：

http://download.eclipse.org/tools/gef/updates/releases/

接下来你选择GEF插件：

按next，并同意安装插件（可能需要重新启动Eclipse）。一旦完成，您可以继续安装规则插件。

###### [从zip文件安装GEF](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_gef_from_zip_file)

要从zip文件安装，请下载并解压缩文件。在zip里面你会看到一个插件目录，以及插件JAR本身。将插件JAR放入Eclipse应用程序插件目录中，然后重新启动Eclipse。

###### [从zip文件安装Drools插件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_drools_plug_in_from_zip_file)

从下面的链接下载Drools Eclipse IDE插件。将下载的文件解压缩到您的主eclipse文件夹中（不要只是将文件复制到那里，将其提取出来，以便功能和插件JAR最终位于eclipse的features和plugin目录中）并（重新）启动Eclipse。

<https://www.drools.org/download/download.html>

要检查安装是否成功，请尝试打开Drools透视图：单击Eclipse窗口右上角的“Open Perspective”按钮，选择“Other ...”并选择Drools透视图。如果您无法找到Drools视角作为可能的角度之一，则安装可能不成功。检查是否正确执行了每个必需的步骤：您是否拥有正确版本的Eclipse（3.4.x）？你是否安装了Eclipse GEF（检查org.eclipse.gef\_3.4 **.jar**是否**存在于你的eclipse根文件夹的plugins目录中）？您是否正确提取了Drools Eclipse插件（检查org.drools.eclipse\_**.jar存在于你的eclipse根文件夹的plugins目录中）？如果您找不到问题，请尝试与我们联系（例如在irc或用户邮件列表上），更多信息可以在我们的主页找到：

<https://www.drools.org/>

###### [Drools运行时](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools_runtimes)

Drools运行时是您的文件系统上的JAR集合，代表Drools项目JAR的一个特定版本。要创建运行时，您必须将IDE指向您选择的版本。如果你想创建一个基于插件本身包含的最新Drools项目JAR的新运行时，你也可以轻松地做到这一点。您需要为Eclipse工作区指定默认的Drools运行时，但每个单独的项目都可以覆盖缺省值，并专门为该项目选择适当的运行时。

[定义一个Drools运行时](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_drools_runtime)

您需要使用Eclipse首选项视图来定义一个或多个Drools运行时。要打开您的偏好设置，请在菜单窗口中选择首选项菜单项。新的首选项对话框应显示您的所有首选项。在该对话框的左侧，在Drools类别下，选择“已安装的Drools运行时”。然后右侧的面板应显示当前定义的Drools运行时间。如果你还没有定义任何运行时间，它应该像下图那样。

要定义一个新的Drools运行时，请点击添加按钮。应弹出如下所示的对话框，要求您的运行时的名称以及您的文件系统上可找到它的位置。

一般来说，你有两个选择：

1. 如果您只想使用Drools Eclipse插件中包含的默认JAR，则可以通过单击“创建新的Drools 5运行时...”按钮来自动创建新的Drools运行时。文件浏览器将显示出来，要求您选择要在其中创建运行时的文件系统上的文件夹。该插件会自动将所有必需的依赖项复制到指定的文件夹。选择该文件夹后，对话框应如下图所示。
2. 如果您想使用Drools项目的特定版本，则应在文件系统上创建一个包含所有必需的Drools库和依赖关系的文件夹。如上所述，不要创建一个新的Drools运行时，请为运行时提供一个名称，并选择包含所有必需JAR的此文件夹的位置。

单击确定按钮后，运行时应显示在已安装的Drools运行时表中，如下所示。单击新创建的运行时前面的复选框，使其成为默认的Drools运行时。默认的Drools运行时将用作所有未选择项目特定运行时的Drools项目的运行时间。

您可以根据需要添加尽可能多的Drools运行时。例如，下面的屏幕截图显示了一个定义了三个运行时的配置：一个Drools 4.0.7运行时，一个Drools 5.0.0运行时和一个Drools 5.0.0.SNAPSHOT运行时。Drools 5.0.0运行时被选为默认运行时。

请注意，如果更改了默认运行时并且要确保所有使用默认运行时的项目都相应地更新其类路径，则需要重新启动Eclipse。

[为您的Drools项目选择运行时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_selecting_a_runtime_for_your_drools_project)

无论何时创建Drools项目（使用New Drools项目向导或将现有Java项目转换为Drools项目时，都使用Drools透视图中显示的“Convert to Drools Project”操作，并右键单击现有Java项目），插件会自动将所有必需的JAR添加到项目的类路径中。

当创建一个新的Drools项目时，插件会自动为该项目使用默认的Drools运行时，除非您指定了特定项目。您可以在New Drools Project向导的最后一步执行此操作，如下所示，取消选中“使用默认Drools运行时”复选框并在下拉框中选择适当的运行时。如果您单击“配置工作区设置...”链接，将会打开显示当前安装的Drools运行时的工作区首选项，以便您可以在其中添加新的运行时。

您可以随时通过打开项目属性（右键单击项目并选择“属性”）并选择Drools类别来更改Drools项目的运行时间，如下所示。选中“启用项目特定设置”复选框并从下拉框中选择适当的运行时间。如果您单击“配置工作区设置...”链接，将会打开显示当前安装的Drools运行时的工作区首选项，以便您可以在其中添加新的运行时。如果取消选中“启用项目特定设置”复选框，它将使用全局首选项中定义的默认运行时。

#### [1.3.2。从源头上构建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_building_from_source)

##### [1.3.2.1。获取来源](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getting_the_sources)

每个Maven工件的源代码在JBoss Maven存储库中作为源JAR提供。下拉链接中也包含相同的源JAR。但是，如果您想从源代码构建，强烈建议从源代码管理中获取我们的源代码。

Drools和jBPM使用[Git](http://git-scm.com/)来控制源代码。[GitHub上](https://github.com/)托管着有福的git仓库：

* <https://github.com/kiegroup>

Git允许您分发代码，独立地对其进行个人更改，但仍会定期合并我们的最新更改，并可选择与我们分享您的更改。要了解更多关于git的信息，请阅读免费的书[Git Pro](http://progit.org/book/)。

##### [1.3.2.2。建立资源](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_building_the_sources)

在本质上，从源代码构建非常容易，例如，如果您想构建guvnor 项目：

$ git clone git@github.com:kiegroup/guvnor.git

...

$ cd guvnor

$ mvn clean install -DskipTests -Dfull

...

但是，有很多潜在的缺陷，所以如果您认真构建源代码并可能对项目做出贡献，请**按照**[**droolsjbpm-build-bootstrap中的README文件中**](https://github.com/kiegroup/droolsjbpm-build-bootstrap/blob/master/README.md)**的说明进行操作 。**

#### [1.3.3。蚀](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_eclipse)

##### [1.3.3.1。导入Eclipse项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_importing_eclipse_projects)

通过生成Eclipse项目文件，他们现在可以导入到Eclipse中。当启动Eclipse时，打开Subversion签出的根目录中的工作空间。

当调用mvn install所有的项目依赖关系被下载并添加到本地Maven仓库时。除非您告诉它该存储库在哪里，否则Eclipse无法找到这些依赖关系。为此设置一个M2\_REPO类路径变量。

## [2.发行说明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_droolsreleasenoteschapter)

### [2.1。流口水中新增和值得注意的东西7.7](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.7.7.0)

#### [2.1.1。可执行模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_executable_model)

可执行模型的目的是提供规则集的基于Java的纯粹表示形式，以及一个方便的Java DSL以编程方式创建此模型。该模型属于低级别，专为用户提供所需的所有信息，例如用于索引评估的lambda。这使得它保持快速并避免在这个级别建立太多的假设。预计未来可能会有更高层次的表示，可能会更多地关注最终用户。这项工作也高度赞扬了单元工作，它提供了一种面向Java的方式来提供数据和控制编排。

##### [2.1.1.1。细节](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_details)

该模型通用性足以独立于Drools，但可以编译为普通的Drools知识库。出于这个原因，可执行模型的实现已被分成两个子项目：

1. **drools-canonical-model**是完全独立于Drools的规则集模型的标准表示
2. **drools-model-compiler**将规范模型编译为Drools内部数据结构，使其可以由引擎执行

可执行模型的引入为不同领域带来了一系列好处：

* **编译时间**：在Drools 6中，一个kjar包含drl文件和其他Drools工件的列表，其中定义了规则库以及一些实现约束和后果的预生成类。这些drl文件需要从头开始进行分析和编译，这时kjar从Maven存储库下载并安装在KieContainer中，使得此过程非常缓慢，特别是对于大型规则集。相反，现在可以在kjar内部封装实现项目规则库可执行模型的Java类，并以更快的方式重新创建KieContainer及其KieBase。在编译过程中，kie-maven-plugin会自动从drl文件生成可执行模型源文件。
* **运行时**：在可执行模型中，所有约束都被定义为Java lambda表达式。相同的lambda也用于约束条件评估，这允许为解释评估摆脱两个mvel，并且在字节码中转换基于mvel的约束，从而导致缓慢的预热过程。
* **未来的研究**：可执行模型将允许实验规则引擎的新特性，而不需要以drl格式对它们进行编码，并修改drl解析器以支持它们。

##### [2.1.1.2。可执行模型DSL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_executable_model_dsls)

在为可执行模型设计DSL的第一次迭代时，一个目标是摆脱模式的概念，并将规则视为表达式（约束）和动作（后果）的流程。出于这个原因，我们称之为Flow DSL。这个DSL的一些例子可以在这里找到。

然而，在实施了流程DSL之后，很明显避免明确使用模式的决定迫使我们实施一些兼具复杂性和性能成本的额外逻辑，因为为了正确地重新创建预期的数据结构Drools编译器有必要将那些明显不相关的表达式放在一起。

由于这个原因，我们决定在第二个DSL中重新引入这些模式，我们称之为Pattern DSL。这允许绕过算法分组表达式，这些表达式必须填补人为的语义差距，并且在运行时也很耗时。我们相信这两种DSL都适用于不同的使用情况，然后我们决定保留和支持这两种情况。特别是模式DSL更安全更快（即使更详细），所以这将是在通过kie-maven插件创建kjar时自动生成的DSL。相反，Flow DSL更简洁，更接近用户想要以Java编程定义规则的方式，并且我们计划通过以自动方式通过后处理器生成定义该规则的模型部分来减少冗余索引和属性反应。

##### [2.1.1.3。程序化构建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_programmatic_build)

正如前一节中链接的测试用例所证明的那样，可以通过编程方式在Java中定义一个或多个规则，然后使用流畅的API将它们添加到模型中

Model model = **new** ModelImpl().addRule( rule );

一旦你有了这个模型，就像所说的完全独立于Drools算法和数据结构，可以创建一个KieBase，因为它遵循

**KieBase** kieBase = KieBaseBuilder.createKieBaseFromModel( model );

或者，也可以通过从普通的drl文件开始创建基于kieproject的可执行模型，像平常一样将它们添加到KieFileSystem

**KieServices** ks = KieServices.Factory.get();

**KieFileSystem** kfs = ks.newKieFileSystem()

.write( "src/main/resources/r1.drl", createDrl( "R1" ) );

**KieBuilder** kieBuilder = ks.newKieBuilder( kfs );

然后使用buildAll（）方法的新重载构建项目，该方法接受指定要构建哪种类型的项目的类

kieBuilder.buildAll( ExecutableModelProject.class );

这样做KieBuilder将生成可执行模型（基于模式DSL），然后生成KieSession

**KieSession** ksession = ks.newKieContainer(ks.getRepository()

.getDefaultReleaseId())

.newKieSession();

将使用本文档第一部分描述的基于lambda表达式的约束。同样，也可以通过将不同的项目类传递给KieBuilder来从Flow DSL生成可执行模型

kieBuilder.buildAll( ExecutableModelFlowProject.class );

但是，在讨论2种不同的DSL时，最好使用基于模式的DSL来达到此目的。

##### [2.1.1.4。Kie Maven插件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_maven_plugin)

为了使用kie-maven-plugin生成嵌入可执行模型的kjar，需要在pom.xml文件中添加与之前提到的两个项目实现模型及其编译器相关的依赖关系：

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-model-compiler</artifactId>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-canonical-model</artifactId>

</dependency>

</dependencies>

也将插件添加到插件部分

<build>

<plugins>

<plugin>

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<version>${project.version}</version>

<extensions>true</extensions>

</plugin>

</plugins>

</build>

这里有一个已经准备好生成可执行模型的pom.xml文件的例子。默认情况下，kie-maven-plugin仍然会生成一个基于drl的kjar，因此需要使用以下参数运行插件：

-DgenerateModel=<VALUE>

其中<VALUE>可以是以下三个值之一：

YES

NO

WITHDRL

YES和WITHDRL都会生成并添加到kjar中，使用实现与原始项目中的drl文件相对应的可执行模型的Java类，区别在于第一个将从生成的kjar中排除drl文件，而第二个将添加它们。然而，在第二种情况下，drl文件将仅扮演文档角色，因为KieBase将从可执行模型构建而成，无论如何。

### [2.2。KIE Workbench 7.7.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.7.0.final)

#### [2.2.1。面向项目的工作台](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_oriented_workbench)

工作台变成项目导向。意味着每个项目现在都在一个专门的存储库中 以前每个项目都在一个文件夹中，一个存储库可以存放多个项目。这些更改合并了Project和Repository的概念。

一个项目一个存储库方法使得UI更简单并且改进了释放或冻结单个项目。以前，如果某个项目被冻结，分支或标记，则存储库仍与其他项目共享，从而使版本控制变得更加困难和复杂。

旧版Workbench版本中的项目需要迁移到新设置。为此，我们提供了一个命令行迁移工具。

目前，Project Orientated并不支持多模块设置。此功能是计划的，但不是在此版本中。现在弃用的资产管理功能取决于多模块支持，并且无法迁移到此Workbench版本。

#### [2.2.2。连接到远程Kie服务器控制器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_connecting_to_a_remote_kie_server_controller)

运行Workbench实例时，现在可以更灵活地决定如何管理Kie Server实例。以前，无论何时启动Workbench，它都会启动嵌入式Kie服务器控制器。此设置仍然可用，但现在也有一个选项可以不启动此服务，而是连接到远程Kie服务器控制器。这允许更精细的粒度部署模型，可以为特定场景确定最佳方法。

要在这些模式之间切换，可以使用一个关键的系统属性：**org.kie.workbench.controller**。默认情况下，如果缺少此系统属性，Workbench将继续启动嵌入式服务。否则，它将尝试连接到远程服务，并确保没有任何嵌入式服务启动。

请注意，连接到远程Kie服务器控制器时只有Web Socket连接协议可用。

有关所有可能的系统配置的更多详细信息，请参阅[Workbench系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)，这些系统配置关于用户名，密码，令牌和通过密钥存储的安全密码。

#### [2.2.3。内容管理增强](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_content_management_enhancements)

##### [2.2.3.1。属性面板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_properties_panel)

可以编辑不同页面元素的属性，包括页面本身或其任何行和组件。选择一个元素后，通过将元素悬停在元素上并单击编辑器区域或在“ 属性”面板下拉列表中选择元素 ，其属性将显示在左侧停靠的面板中。看截图：

*图1.内容管理属性面板*

这个版本中可用的属性基本上是与元素样式相关的属性，例如宽度，高度或边距等。每种类型可用的属性可能有所不同。注意，例如，HTML组件提供了一组额外的与文本样式相关的属性。

一旦属性发生变化，它的值将反映在编辑器的区域中，包括设计和预览模式。

##### [2.2.3.2。屏幕组件已移除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_screen_component_removed)

位于右侧边栏“ 组件”面板的“ 核心”组下的“ 屏幕”组件已被删除。原因是，这个组件不适合生产环境。

但是，在未来的版本中，域相关组件（如BPM的任务列表）将可供最终用户轻松使用。

#### [2.2.4。新的迁移工具](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_migration_tool)

已经提供了支持Linux和Windows的新命令行迁移工具，以将不同的资源移动到最新版本。它可以执行不同的迁移：

* **项目迁移**：将KIE项目从旧项目布局（7.4.x和之前）迁移到新的面向项目的结构。
* **表单迁移**：将旧的jBPM表单建模器表单迁移到新的表单格式。

*图2.命令行迁移工具*

你可以[在这里](https://github.com/kiegroup/kie-wb-common/blob/master/kie-wb-common-cli/kie-wb-common-cli-migration-tool/README.md)找到更多信息。

### [2.3。KIE Workbench 7.6.0中的新增值得关注](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.6.0.final)

#### [2.3.1。内容管理增强](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_content_management_enhancements_2)

内容管理工具（又名页面创作）中引入了一些额外的改变，以改善用户体验。以下屏幕截图显示了自最新版本以来所做的更改。

*图3.内容管理新的外观和感觉*

##### [2.3.1.1。流体/页面编辑器模式选择](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fluid_page_editor_mode_selection)

弹出的新页面允许选择两种编辑模式：

*图4.内容管理的新页面弹出*

* **流体**：它更像一个传统的网页，当页面超过可用高度时显示一个垂直滚动条。
* **页面**：这是以前版本的默认设置。选中时，它会强制页面始终适合窗口的100％高度。

##### [2.3.1.2。预览功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_preview_feature)

页面作者可以去从来回编辑器的预览需要模式多次。在预览模式下，所有编辑器的控件都将被删除，从而允许显示页面，因为最终用户在页面发布后会看到该页面。

##### [2.3.1.3。使用停靠面板来增加编辑器内容区域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_use_dock_panels_to_increase_the_editor_content_area)

为了增加编辑器中心区域的可用空间，页面列表，导航配置和组件面板都已作为停靠面板移动到左边栏中。

##### [2.3.1.4。组件面板重组](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_component_palette_reorganization)

在以前的版本中，调色板中只存在一个单一类别的组件。从版本7.6开始，组件现在分为三个主要类别：

* 核心：HTML和页面组件
* 导航：菜单栏，树，瓷砖，选项卡列表，传送带和目标分区
* 报告：Bar，Pie，Line，Area，Map，Bubble，Metric，Meter，Table和Filter

这些更改的目标是使调色板更具吸引力，并减轻拖动目标组件的选择。

### [2.4。新的Kie服务器控制器客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieserver.releasenoteskie.7.5.1)

为了便于Kie Server Controller相关任务的管理，如创建服务器模板，启动和停止容器等，我们开发了一个新的Java客户端API，可在kie-server-controller-client Maven模块下使用。使用此API，您可以使用REST或Web Socket协议连接到Kie服务器控制器。有关更多详细信息，请参阅[Kie Server Controller客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#ControllerJavaAPI-section.adoc)一章。

### [2.5。从7.0开始改变Kie Server 7.5.1的变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_breaking_changes_in_kie_server_7_5_1_from_7_0)

#### [2.5.1。Kie服务器控制器API更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_controller_api_changes)

SpecManagementService界面的变化：

* 包含的新方法getContainerInfo允许检索单个ContainerSpec定义的ServerTemplate。
* 更改方法listContainerSpec，listServerTemplateKeys以及listServerTemplates从泛型集合返回类型特定的域列表类型（ContainerSpecList，ServerTemplateKeyList，并ServerTemplateList以正确的序列化和反序列化使用JAXB和JSON返回值）。

RuleCapabilitiesService界面的变化：

* startScanner方法现在采用的是java.lang.Long时间，而不是原始long的interval参数，以避免JSON和JAXB序列化问题。

RuntimeManagementService界面的变化：

* 为了使用JAXB和JSON正确地序列化和反序列化返回的值，更改了方法getContainers并将getServerInstances泛型集合中的类型返回到特定的域列表类型（ContainerList和ServerInstanceKeyList）。

有关更多详细信息，请参阅[JBPM-6243](https://issues.jboss.org/browse/JBPM-6243)。

#### [2.5.2。Kie服务器API更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_api_changes)

对ServiceResponse包装的更改：

* 将ResponseType枚举和常用方法移动到一个新的调用接口KieServiceResponse，从而将其扩展为多个实现。

### [2.6。KIE Workbench 7.5.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.5.0.final)

#### [2.6.1。内容管理增强](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_content_management_enhancements_3)

为了改善用户体验，内容管理（aka Dashboards）工具中引入了显着的变化。

##### [2.6.1.1。对页面重命名的透视](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_perspective_to_page_renaming)

用户对“页面”术语更加熟悉。请注意，“透视图”是一个从Eclipse开发工具中借鉴而来的概念，因此它是开发者圈子中最常用的概念。网页与网页内容更相关，更容易被普通人理解。出于这个原因，透视图已被重新命名为遍布整个工具。

##### [2.6.1.2。左侧边栏新的外观和感觉](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_left_sidebar_new_look_and_feel)

这是引入的最显着的变化。为了提供更好的外观和感觉，左侧栏已经完全重写。

*图5.内容管理工具的左侧栏*

页面和导航这两个部分已合并为一个视图。从这个视图中，用户可以创建新页面或更改导航配置。

在导航部分列出了导航树。具体来说，默认情况下名为Workbench的树是可用的，它不能被删除，并且它包含在工作台的顶级超级菜单中显示的条目。对其应用的任何更改都将导致超级菜单修改其条目。这实际上是用户必须使用的机制来扩展新页面的工作台。

##### [2.6.1.3。标签按钮被禁用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_tags_button_disabled)

标签功能或在编辑期间将一组标签附加到页面的功能已被删除。此功能与前“应用程序”视角相结合，用于允许用户以分类方式发布其动态页面。此功能不再需要。在页面编辑期间，用于显示在编辑器右上方按钮栏的“ 标签”按钮已被删除。

作为替代，用户可以利用现有的导航功能创建新页面并将其附加到工作台的大型菜单中。

#### [2.6.2。额外的导航组件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_extra_navigation_components)

内容管理工具中现有的导航组件已经扩展了一些额外的类型。迄今为止，存在以下组件：

* 瓷砖导航
* 选项卡列表
* 圆盘传送带

最近增加了以下内容：

##### [2.6.2.1。菜单栏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_menu_bar)

正如其名称所示，此组件以手动条形状显示导航树的条目。支持的级别数量没有限制。当单击页面项目时，页面内容将显示在 菜单栏配置中指定的目标div中（请参阅下面的详细信息）组件。

*图6.菜单栏导航组件*

##### [2.6.2.2。树导航](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_tree_navigator)

与菜单栏相同，但条目显示为垂直树结构。

*图7.菜单栏导航组件*

##### [2.6.2.3。目标分区](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_target_div)

两个传送带和瓷砖导航组件可以通过自身处理用户点击该项目的显示。其他人喜欢选项卡列表，菜单栏和树形导航器需要一个目标div组件作为它的显示输出，因为它们的条目显示与最后点击的内容之间有明确的分隔。

因此，每次将基于目标div的导航组件放入页面中时，Target div组件也必须被删除，以便第一个组件可以链接到它。以下屏幕显示每次将基于目标div的组件放入页面时显示的配置面板。

*图8.基于目标div的导航组件配置*

导航组对于所有导航组件都是必需的，因为它指示要显示的导航结构，而目标div设置**不适**用于非目标div组件，如Carousel或Tile导航器。

### [2.7。KIE Workbench 7.4.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.4.0.final)

#### [2.7.1。指导决策表的改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_table_improvements)

除了修复大量错误之外，向导用于创建和编辑列的功能已得到改进，以显示不同列类型所需的不同步骤的描述。

*图9.导向决策表向导示例1*

*图10.导向决策表向导示例2*

#### [2.7.2。禁用实验编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_disable_experimental_editors)

以下功能被认为是实验性的，可以使用安全/用户管理管理屏幕禁用： -

* 指导决策树编辑器
* 导游计分卡编辑器
* XLS记分卡编辑器
* （新）BPMN2流程编辑器
* 部署/服务器配置透视

这些功能默认启用。

*图11.禁用编辑器*

### [2.8。KIE Workbench 7.3.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.3.0.final)

#### [2.8.1。新家和菜单栏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_home_and_menu_bar)

主页，菜单栏和关于弹出窗口现在有一个新的设计。菜单项目被分成代表主要功能区域的组。

*图12.展开Menu的Home视图*

*图13.关于弹出窗口*

#### [2.8.2。管理页面更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_admin_page_changes)

Admin页面现在可以通过菜单栏上的齿轮图标进行访问，并且有更多的项目。访问视角工件，数据集，数据源和语言选项已移至此处。

*图14.管理页面*

### [2.9。什么是Drools中的新特性和值得注意的7.1](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.7.1.0)

### [2.10。KIE Workbench 7.1.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.1.0.final)

#### [2.10.1。项目指标仪表板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_metrics_dashboard)

一个全新的仪表板现在可用于创作库中列出的每个项目。打开项目详细信息页面后，屏幕右侧会显示一张指标卡片。

*图15.项目度量卡*

该卡片显示了随着时间的推移对该特定项目作出的贡献（提交）的历史记录。点击查看全部链接访问完整的仪表板，该仪表板显示关于项目贡献的所有指标。

*图16.项目指标仪表板*

请注意，不同的过滤器控件可用于选择具体用户或特定时间范围内的贡献。

#### [2.10.2。团队指标仪表板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_teams_metrics_dashboard)

一个全新的仪表板也已添加到团队页面。右侧的指标卡显示所有贡献的历史记录（提交）。

*图17.团队度量卡片*

点击查看全部链接访问显示总体贡献度量标准的完整仪表板。

*图18.团队度量标准仪表板*

请注意，不同的过滤器控件可用于通过不同的标准选择贡献：

* 由具体的用户，
* 在特定的时间范围内，
* 或者，
* 按项目。

该仪表板取代了以前的Authoring> Contributors仪表板，该仪表板在工作台的顶部菜单栏中不再可用。

### [2.11。Drools 7.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.7.0.0)

#### [2.11.1。核心引擎](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_core_engine)

##### [2.11.1.1。DMN运行时支持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dmn_runtime_support)

Drools现在对DMN（Decision Model and Notation）有完整的运行时支持。DMN文件现在是可以添加到任何kjar执行的资产。 有关如何构建和执行DMN模型的详细信息，请参阅[DMN部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#drools.DMN)。

目前，不支持DMN创作，但将在未来的版本中支持。

##### [2.11.1.2。多线程规则引擎](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_multithreaded_rule_engine)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |

从历史上看，Rete一直是一个连续的算法，即使在Drools 6中引入了它的进化，Phreak也允许一定程度的并行化，但这种可能性还没有具体地利用。在Drools 7中，最终有可能使引擎同时评估更多规则。这是通过将Rete / Phreak划分为独立分区并对它们进行并行评估来实现的。

此功能默认为关闭状态，但可以通过KieBaseConfiguration如下所示启用并行KieBase ：

KieServices ks = KieServices.Factory.**get**();

KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();

kieBaseConf.setOption(MultithreadEvaluationOption.YES);

KieBase kieBase = kieContainer.newKieBase(kieBaseConf);

或者通过系统属性：

drools.multithreadEvaluation = true

|  |  |
| --- | --- |
|  | 并行引擎目前不支持使用查询，显着性或议程组的规则。如果它们存在于规则库中，编译器会发出警告并使用单线程实现自动切换回来。在其他情况下，使用多线程规则引擎不正确并导致错误结果，即使引擎可能无法自动检测它们，然后也不会回退到单线程模式，例如规则依赖于由规则给出的隐式显着性在drl文件中排序。出于这个原因，启用此选项时必须特别注意。最后，目前不支持会话序列化和增量编译。 |

##### [2.11.1.3。OOPath改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_oopath_improvements)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |

OOPath已经在Drools 6.3.0中引入，但是在Drools 7.0.0中，语法已经稍微改变了，以使其更接近标准xpath。这意味着约束条件必须放在方括号内而不是卷曲的内容之间，而内联表达式必须用约束表达。例如，以6.x语法表示的以下oopath：

/list{#SubClass, prop == 0}

在Drools 7.x中变成：

/list#SubClass[prop == 0]

此外Drools的7.0.0提高了标准的Java Collection中的支持，有专门的实施List，并Set作为专业ReactiveList和ReactiveSet; 一个ReactiveCollection也是可用的。这也包括开箱无功支持，通过他们的表演可变操作时，Iterator和ListIterator。

*例：*

**public** **class** **School** **extends** **AbstractReactiveObject** {

**private** String name;

**private** **final** List<Child> children = **new** ReactiveList<Child>();

**public** **void** **setName**(String name) {

**this**.name = name;

notifyModification();

}

**public** **void** **addChild**(Child child) {

children.add(child);

*// no need to call notifyModification() here.*

}

|  |  |  |
| --- | --- | --- |
|  | 使用专用ReactiveList于标准Java的反应支持List。 | |
|  | 通常notifyModification()在被动支持更改字段时需要调用 | |
|  | 但在这种情况下ReactiveList会自动处理，就像在场上执行的每个其他变异操作一样children。 | |
|  | | 作为最佳做法，建议final按照所示示例声明被动收集字段。 |

###### [OOPath Maven插件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_oopath_maven_plug_in)

Kie Maven插件injectreactive为测试字节码提供了一个新的目标，并为标准情况自动注入反应性支持。

该injectreactive目标是默认禁用，并可以通过的Maven插件配置中启用instrument-enabled的设置。

*例：*

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<extensions>true</extensions>

<configuration>

<instrument-enabled>true</instrument-enabled>

</configuration>

|  |  |
| --- | --- |
|  | 实现injectreactive目标。 |

该injectreactive有关的Maven项目生成的输出目录目标将仪器的字节码${project.build.outputDirectory}。

可以通过Maven插件配置instrument-packages设置列表将目标的范围限制为特定的包或软件包层次结构。。例：

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<extensions>true</extensions>

<configuration>

<instrument-enabled>true</instrument-enabled>

<instrument-packages>

<instrumentPackage>to.instrument</instrumentPackage>

<instrumentPackage>org.drools.compiler.xpath.tobeinstrumented.\*</instrumentPackage>

</instrument-packages>

</configuration>

|  |  |
| --- | --- |
|  | 仪器的范围to.instrument仅限于包装。 |
|  | 限制仪器的范围org.drools.compiler.xpath.tobeinstrumented和其子包。 |

在以下标准情况下，插件将为每个字段分配测试字节码：

* 字段分配也会触发 notifyModification()
* 将任何定义为List的字段用a包裹 ReactiveList
* 将任何定义为Set的字段换成一个 ReactiveSet
* 用a包装任何定义为Collection的字段 ReactiveCollection

|  |  |
| --- | --- |
|  | 为了类型列表/集被正确地缠绕的场，该Java类的字段部件必须专门使用要么被声明 java.util.Collection，java.util.List或java.util.Set （声明例如一个字段java.util.ArrayList将不与反应性专门集合被仪器）。 |
|  | 不建议混合用于反应性的手动支持（手动实现）和字节码工具Maven插件; 最好设想保持两个范围不同，例如通过使用插件配置来仅测试上述文件中的特定软件包。 |

以下部分介绍了插件检测的详细示例。

[仪器的字段分配](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_instrumentation_of_field_assignments)

字段分配如下例所示：

*原版的：*

**public** **class** **Toy** {

**private** String owner;

...

**public** **void** **setOwner**(String owner) {

**this**.owner = owner;

}

}

将通过拦截字段分配和触发以下内容进行检测notifyModification()：

*结果：*

**public** **class** **Toy** **implements** **ReactiveObject** {

**private** String owner;

...

**public** **void** **setOwner**(**final** String owner) {

**this**.$$\_drools\_write\_owner(owner);

}

**public** **void** $$\_drools\_write\_owner(**final** String owner) {

**this**.owner = owner;

ReactiveObjectUtil.notifyModification((ReactiveObject) **this**);

}

}

请注意，这种仪器只适用于该领域不是Collection。

在字段分配引用a List或a 的情况下Set，检测将用a ReactiveList或`ReactiveSet相应地包装该分配; 例如：

*原版的：*

**public** **class** **School** {

**private** **final** String name;

**private** **final** List<Child> children = **new** ArrayList<Child>();

...

**public** **School**(String name) {

**this**.name = name;

}

**public** List<Child> **getChildren**() {

**return** children;

}

}

将通过拦截和包装ReactiveList：

*结果：*

**public** **class** **School** **implements** **ReactiveObject** {

**private** **final** String name;

**private** **final** List<Child> children;

**public** **School**(**final** String name) {

**this**.$$\_drools\_write\_children(**new** ArrayList());

**this**.name = name;

}

**public** List<Child> **getChildren**() {

**return** **this**.children;

}

**public** **void** $$\_drools\_write\_children(**final** List list) {

**this**.children = (List<Child>) **new** ReactiveList(list);

}

##### [2.11.1.4。PMML支持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pmml_support)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |
|  | 此功能使用[规则单元](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_units_2) |

Drools现在支持符合Predictive Modeling Markup Language（PMML）的子集的资产。现在支持以下预测模型类型：

* 回归
* 记分卡
* 树

此外，挖掘模型类型有部分支持; 目前提供以下模式：

* 模型链
* 全选
* 选择第一个

进一步的操作模式将在可用时得到支持。

##### [2.11.1.5。事件的软到期](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_soft_expiration_for_events)

在Drools 6.x中明确定义事件过期时，它总是被认为是硬过期，意味着它总是优先于在涉及事件的时间窗口和约束条件下隐式计算的任何其他过期。Drools 7还允许为可以使用的事件指定软过期，如果推断的过期偏移量是无限的。通过这种方式，可以保证有效期限是推断的或者指定的期满，如果另一方缺失的话。此外，这意味着规则制定者不需要在所有规则中包括时间约束，然后即使尚未知道规则，也可以设计事件类别。

默认情况下，事件过期被认为是困难的，但可以更改过期策略并定义一个软过期，如下所示注释事件的类：

@Role(Role.Type.EVENT)

@Expires( value = "30s", policy = TIME\_SOFT )

**public** **class** **MyEvent** { ... }

或使用类型声明：

**declare** MyEvent

@**role**( **event** )

@expires(**value** = 30s, **policy** = TIME\_SOFT)

**end**

##### [2.11.1.6。规则单元](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_units)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |

规则单元表示将规则集划分为更小单元的纯粹声明式方法，将不同的数据源绑定到这些单元并编排单个单元的执行。规则单元是数据源，全局变量和规则的集合。

#### [2.11.2。工作台](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_workbench)

除了工作台的一般改进（下面在一个单独的章节中列出）之外，在工作台中还有一些Drools特定的增强功能。

##### [2.11.2.1。决策表的DMN风格命中策略](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dmn_style_hit_policies_for_decision_tables)

对于每个命中策略，默认情况下，一行在其下面的每一行都具有优先权。

* 独特的命中有了独特的命中策略，每行必须是唯一的，这意味着不能有重叠。如果验证功能在开发时发出警告，则永远不会出现两行可触发的情况。
* 首先击中第一击只有一行，首先是自上而下满足的一击。
* 已解决的命中与第一次命中类似，但您可以例如为第5行提供第10行的优先级。这意味着您可以保持为了视觉可读性而希望的行的顺序，但指定优先级异常。
* 规则顺序多行可以触发，并且Verification不会报告行之间的冲突，因为它们预计会发生。
* 无这是正常的命中模式。旧的决策表默认使用这个，但由于7.0使用了PHREAK，现在行顺序很重要。旧表格不需要迁移工具。多行可以触发。验证会对冲突的行发出警告。

##### [2.11.2.2。引导规则编辑器：支持复合字段约束中的公式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_rule_editor_support_formulae_in_composite_field_constraints)

复合字段约束现在支持使用公式。

向Pattern添加约束时，“多个字段约束”选项（“所有（和）”和“任何（或）”）除了表达式之外，还支持使用公式。

*图19.复合字段约束 - 选择公式*

*图20.复合字段约束 - 公式编辑器*

##### [2.11.2.3。指导决策表编辑器：新编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_table_editor_new_editor)

指导决策表编辑器已被广泛重写，以支持在同一编辑器中编辑多个表。共享关联的表格可视化地链接在一起，使关系更易于可视化。关联是从Actions创建或更新另一个表的Conditions条件消耗的事件。

亮点包括： -

* 新的外观和感觉
* 可调整大小的列
* 通过拖放“在表格中”对列进行重新排序
* 通过拖放“在表格中”对行进行重新排序
* 通过拖放重新定位表格
* 平移视图以滚动内容
* 可缩放视图，因此您可以放大“出”以一次查看更多内容
* 每个决策表的文件锁定和版本历史记录

*图21.新的编辑器*

*图22.新的编辑器 - 多个打开的表格*

##### [2.11.2.4。指导决策表编辑器：枚举查找的缓存](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_table_editor_caching_of_enumeration_lookups)

指导性决策表编辑器一直能够使用枚举定义。但是，由于表格可以包含许多单元，因此如果定义需要服务器往返来从辅助类中检索查找，则枚举的性能有时可能会不理想。

服务器往返的结果现在缓存在客户端中，因此在修改单元格时不需要连续的网络调用。当编辑器被打开并按需填充时，缓存被初始化。

##### [2.11.2.5。指导决策表编辑器：验证和验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_table_editor_verification_and_validation)

###### [系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_system_property)

可以使用系统属性禁用验证和验证。这对于大型决策表或者用户想要忽略V＆V很有用。org.kie.verification.disable-dtable-realtime-verification

###### [范围检查](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_range_checks)

验证首先帮助您制定完整的决策表。在下一个版本中，我们添加了对检查所有范围是否覆盖布尔值，数值和日期值的支持。这意味着如果您的餐桌有检查申请是否获得批准的检查，则验证报告会提醒您确保您也处理申请未获批准的情况。

###### [独特的单击](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_unique_single_hit)

过去，如果行相互包含，验证和验证就引发了一个问题。如果一行包含另一行，那么可以用相同的事实来满足条件。这意味着同一张桌子上的两排可以同时开火。在某些情况下，包容并不重要，但在其他情况下，您希望有一个表格，当时只有一条规则会被触发。该表是一个单击命令决策表。为了帮助制作只有一行可触发的单击表格，验证会密切关注条件。报告单击时被打破的情况。

### [2.12。在6.x中打破Drools 7.0的变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_breaking_changes_in_drools_7_0_from_6_x)

#### [2.12.1。属性反应性默认启用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_property_reactivity_enabled_by_default)

在Drools 5.4中引入了属性反应性，但用户必须通过@PropertyReactive注解或者使用PropertySpecificOption.ALWAYS构建器选项在整个知识库上逐个类地明确地启用它。但是，由于在正确性和性能方面，使用此功能被认为是一种很好的做法，所以在Drools 7.0中默认启用该功能。如果需要，可以禁用属性反应性，并通过将以下配置添加到kmodule.xml文件中来重新配置Drools 7.0，使其与6.x版完全相同。

<configuration>

<property key="drools.propertySpecific" value="ALLOWED"/>

</configuration>

#### [2.12.2。保存类型保存功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_type_preserving_accumulate_functions)

在Drools 6中，当sum在累积模式中使用函数时，无论执行总和的字段类型如何，结果都始终为Double。这导致了以下3个问题：

* 精确度损失：长期的总和1881617265586265321L将错误地返回1.88161726558626534E18。的BigDecimal的总和0.09，并0.01也将是不正确的。
* 性能损失：总计为Double的总和比使用Long或Integer总和慢得多。
* 泄漏的复杂性：它强制用户在Double上进行模式匹配，或者对Number进行更一般的（建议选择）匹配，同时可以预期在Integer类型的字段上求和的结果也会是Integer。

相反，Drools 7保留执行总和的表达式的类型，因此可以直接在该类型上进行匹配，如下所示：

Long(...) from accumulate(..., sum($p.getLongWeight()))

#### [2.12.3。重命名TimedRuleExecutionOption](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_renaming_timedruleexecutionoption)

控制何时自动执行定时规则的KieSession选项已重命名为TimedRuleExecutionOption修复名称中影响以前版本的打字错误; 该物业已被列入drools.timedRuleExecution。

| *表1.名称更改* | | |
| --- | --- | --- |
|  | **以前的版本** | **版**7.0.0.Final |
| KieSession选项 | TimedRuleExectionOption | TimedRuleExecutionOption |
| 属性 | drools.timedRuleExection | drools.timedRuleExecution |

#### [2.12.4。重命名和统一配置文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_renaming_and_unification_of_configuration_files)

在Drools 6.x中，默认的Drools配置属性配置在两个不同的文件中：\* drools.default.rulebase.conf位于drools.default.packagebuilder.confdrools-compiler 的META-INF文件夹中的drools-core \* 的META-INF文件夹中

在Drools 7.0.0中，这些文件被统一为一个名为的文件kie.default.properties.conf，位于drools-core的META-INF文件夹中。如果您想覆盖这些属性的默认值或添加您自己的属性，可以将它们放入kie.properties.conf位于项目的META-INF文件夹中的文件中。

### [2.13。KIE Workbench 7.0.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.7.0.0.final)

该工作台已更新为支持Wildfly 10和EAP7。最低Java要求是JDK8。

#### [2.13.1。新创作（图书馆）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_authoring_library)

现在创作的是一个新的设计，具有更好的信息组织。现在可以管理（创建，删除和编辑）团队（组织单位），列出存储库中的项目和项目中的资产。选择资产后，您可以看到资产编辑器和项目浏览器。

|  |  |
| --- | --- |
|  | 该库使用工作台的索引。因此，**必须**删除现有的索引信息，以便Workbench可以用必要的信息重建它们。索引信息存储在.index应用程序服务器\bin文件夹内的文件夹中（或者您可能已使用org.uberfire.metadata.index.dir系统属性配置了其他文件夹）。 |

*图23.团队视图*

*图24.团队创建*

*图25.清空存储库*

*图26.包含项目的存储库*

*图27.空项目*

*图28.包含资产的项目*

*图29.资产视图*

此外，您可以通过访问主页>管理>库设置有关您的默认工作空间的一些首选项。

*图30.库首选项*

#### [2.13.2。创作 - 进口的例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_imports_of_examples)

在7.x之前，Workbench用于在启动时安装预定义的示例。

7.x版本提供了从git存储库导入示例的功能。创作视角包含“示例”菜单项，点击此按钮可启动向导来引导您完成导入。

*例子1.菜单项*

创作视角包含“示例”的菜单项。

*示例2.向导 - 输入源存储库*

向导的第1页允许用户选择预定义的示例库，或者输入他们自己的URL。

*示例3.向导 - 选择项目（S）*

向导的第2页列出了源代码库中可用的项目。

*示例4.向导 - 输入目标存储库*

向导的第3页允许用户输入目标存储库名称并将其与组织单位相关联。

#### [2.13.3。创作 - 弹出窗口的改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_pop_ups_improvements)

所有系统弹出窗口都改善了用户体验。

“评论”字段默认为隐藏。

现在，当从任何包中复制项目文件时，可以选择目标包。

#### [2.13.4。创作 - 项目编辑器 - 重新导入按钮](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_project_editor_reimport_button)

“重新导入”按钮会使所有缓存的依赖项无效，以便处理特定依赖项更新而不修改其版本的情况。

#### [2.13.5。安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_management)

在6.4版本中发布的用户和组管理透视图已统一到一个单一透视图中，该透视图提供了管理用户和组的共享视图以及授予任何应用程序角色的权限。

这个非常新的视角位于顶部菜单栏的主页部分。

*图31.链接到安全管理角度*

下一个屏幕截图显示了这个新视角的外观：

*图32.安全管理主页*

选项卡式窗格显示在左侧，允许用户选择角色，组或用户选项卡。点击角色（或组）后，会显示一个详细屏幕，允许用户配置一些安全设置。

*图33.角色安全设置*

* **家庭视角**：登录后用户定向的目标视角，这使得每个角色/组可以拥有不同的主页。
* **优先级**：用于确定哪些设置（主页透视图，权限...）具有分配了多个角色或组的多个用户的优先级。
* **权限**：完整的ACL（访问控制列表）编辑器，用于授予/拒绝平台中可用的不同资源（如透视图， 组织单位，存储库或项目）的授予/拒绝权限。在这些资源类型之上的全局权限可以通过添加个别异常来完成，这使得可以实施全部否认或拒绝全部授予一些策略。

#### [2.13.6。kie-config-cli已被删除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_config_cli_has_been_removed)

由于kie-config-cli.[sh/bat]以下原因，删除了存在于6.x中的用于管理远程存储库的命令行工具：

* 它提供的安全相关操作（add-role-repo，remove-role-repo，add-role-org-unit，remove-role-org-unit，add-role-project，remove-role-project）已被替换更全面的[安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.userandgroupmgmt)功能。
* 有关管理部署的操作（list-deployment，add-deployment，remove-deployment）不再有意义，因为jBPM的运行已经从工作台删除。部署仍然可以使用[Kie Server REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.ksrestapi)以编程方式进行管理。
* 剩余的操作（create-org-unit，remove-org-unit，list-org-units，create-repo，remove-repo，list-repo，add-repo-org-unit，remove-repo-org-unit，list-project-details）可作为一部分[工作台远程API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.workbenchremoteapi)

#### [2.13.7。用户和项目管理页面和首选项](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_and_project_admin_pages_and_preferences)

工作台现在有一个新的菜单项：“Admin”。在那里，你可以找到一些管理工具，如“用户”，“组”和“角色”管理，以及一般偏好。如果在此处更改首选项，它将影响所有依赖它的位置，但仅限于登录的用户。

*图34.用户管理页面访问*

*图35.用户管理页面，其中包含用户级别的工具和首选项*

每个项目都有自己的管理页面，包含管理工具和首选项。如果在此处更改首选项，它将仅影响该项目，并且仅影响登录的用户。

*图36.项目管理页面访问*

*图37.项目管理页面，其中包含项目级别的工具和首选项*

#### [2.13.8。GAV冲突检查和儿童GAV版本](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_gav_conflict_check_and_child_gav_edition)

现在可以为每个用户设置GAV冲突检查标志，并允许或阻止所有项目的子GAV版本，或者专门针对每个项目。

如果通过“管理”菜单项进行访问，可以在管理工具“项目”中找到该配置。也可以通过在项目管理页面上输入管理工具“常规”来找到它。

*图38.分组工件版本（GAV）首选项*

#### [2.13.9。数据源管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_source_management)

新的数据源管理系统使工作台能够定义访问外部数据库的数据源和驱动程序。

一些包括的功能是：

* 管理数据源的新视角：

*图39.数据源创作透视图*

* 用于指导数据源创建的新向导。

*图40.新的数据源向导*

* 指导驾驶员创作的新向导。

*图41.新的驱动程序向导*

浏览数据源指向的数据库的数据库信息的能力。

* 可用的架构浏览

*图42.数据库模式*

* 可用的表格浏览

*图43.模式表*

* 表内容浏览

*图44.表格信息*

### [2.14。在6.x中打破Kie Server 7.0的变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieserver.releasenoteskie.7.0.0)

#### [2.14.1。ServiceResponse XStream编组更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_serviceresponse_xstream_marshalling_changes)

|  |  |
| --- | --- |
|  | 本发行说明仅适用于直接连接Kie服务器（kie-server）API，而不适用于使用Kie Server Java Client（kie-server-client）API的情况。 |

为了更加符合JAXB编组，XStream编组经历了以下更改：

* XML ServiceResponse元素的response对象不再使用规范名称进行呈现。
* 现在XStream使用type和msg作为属性，而不是子元素。

有关更多详细信息，请参阅[DROOLS-1509](https://issues.jboss.org/browse/DROOLS-1509)。

#### [2.14.2。简化的Planner REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_simplified_planner_rest_api)

##### [2.14.2.1。ServiceResponse包装移除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_serviceresponse_wrapper_removal)

ServiceResponse包装已从KIE服务器返回的Planner服务响应中删除。这样可以更轻松地处理客户端的响应。

<solver-instance>

...

<status>SOLVING</status>

<score scoreClass="org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore">0hard/-10soft</score>

<best-solution class="curriculumcourse.curriculumcourse.CourseSchedule">

...

</best-solution>

</solver-instance>

##### [2.14.2.2。新计划器API概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_planner_api_overview)

注册求解器：

* PUT http：// $ {kie-server} / services / rest / server / containers / $ {container\_id} / solvers / $ {solver\_id}

提交解决方案：

* POST http：// $ {kie-server} / services / rest / server / containers / $ {container\_id} / solvers / $ {solver\_id} / state / solve

获得最佳解决方案：

* GET http：// $ {kie-server} / services / rest / server / containers / $ {container\_id} / solvers / $ {solver\_id} / bestsolution

终止求解器：

* POST http：// $ {kie-server} / services / rest / server / containers / $ {container\_id} / solvers / $ {solver\_id} / state / terminating-early

配置解算器：

* DELETE http：// $ {kie-server} / services / rest / server / containers / $ {container\_id} / solvers / $ {solver\_id}

### [2.15。什么是Drools 6.5.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.5.0)

#### [2.15.1。可配置的ThreadFactory](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configurable_threadfactory)

某些运行时环境（例如Google App Engine）不允许直接创建新线程。因此，现在可以通过设置系统属性**drools.threadFactory**的类名来插入自己的ThreadFactory实现。例如，如果您使用该类实现了与Google App Engine兼容的ThreadFactory，则com.user.project.GoogleAppEngineThreadFactory可以通过设置以下内容使Drools能够使用它：

drools.threadFactory = com.user.project.GoogleAppEngineThreadFactory

#### [2.15.2。使用任何表达式作为查询的输入](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_use_of_any_expressions_as_input_for_a_query)

现在可以将查询的两个字段用作查询的输入参数，如下所示：

query contains(String $s, String $c)

$s := String( this.contains( $c ) )

**end**

rule PersonNamesWithA **when**

$p : Person()

contains( $p.name, "a"; )

**then**

**end**

一般来说任何类型的有效表达式都可以用于：

query checkLength(String $s, int $l)

$s := String( length == $l )

**end**

rule CheckPersonNameLength **when**

$i : Integer()

$p : Person()

checkLength( $p.name, 1 + $i + $p.age; )

**then**

**end**

#### [2.15.3。用修改后的属性更新](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_update_with_modified_properties)

引入了属性反应性，以避免不必要的和无用的（重新）评估，并允许引擎仅对修改实际受限或绑定在给定模式内的属性做出反应。但是，此功能只能在规则结果内进行修改时自动使用。相反，程序化更新不知道已更改的对象属性，因此无法使用此功能。

为了克服这个限制，现在可以在更新语句中选择性地指定已修改对象中已更改的属性的名称，如下例所示：

Person me = **new** Person("me", 40);

FactHandle meHandle = ksession.insert( me );

me.setAge(41);

me.setAddress("California Avenue");

ksession.update( meHandle, me, "age", "address" );

#### [2.15.4。监测框架的改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_monitoring_framework_improvements)

为了提供对KieContainers的监视，已经引入了一种新的MBean，并且JMX MBeans的层次结构已被重新考虑，以反映与KieBase的相关MBean的关系。JMX对象已被标准化以反映Kie API中使用的术语。为了提供无状态KieSession的监视功能，引入了一种新的MBean，这在以前的版本中是不可用的。

| *表2. JMX对象名称更改* | | |
| --- | --- | --- |
| **的MBean** | **之前**6.5.x | **从**6.5.0.Final |
| KieContainer | N / A | org.kie:kcontainerId={kcontainerId} |
| KieBase | org.drools.kbases:type={kbaseId} | org.kie:kcontainerId={kcontainerId},kbaseId={kbaseId} |
| KieSession（有状态） | org.drools.kbases:type={kbaseId},group=Sessions,sessionId={ksessionId} | org.kie:kcontainerId={kcontainerId},kbaseId={kbaseId},ksessionType=Stateful,ksessionName={ksessionName} |
| 无状态的KieSession | N / A | org.kie:kcontainerId={kcontainerId},kbaseId={kbaseId},ksessionType=Stateless,ksessionName={ksessionName} |

KieSession MBean合并所有实例化的同名会话的统计数据。

KieSession创建通过JPAKnowledgeService，将根据KieSession的MBean被监控具有恒定{ksessionName}增值的到persistent; 此MBean不直接由KieContainer管理，因此需要在不再需要监视的情况下从JMX手动注销。

新的JMX objectnaming方案现在强制执行正确的JMX引用ID，例如： org.kie:kcontainerId="2cb55f40-f220-432a-aba8-7940c18bf108",kbaseId="KBase1"

旧的DroolsManagementAgent（已在JMX上注册org.drools:type=DroolsManagementAgent）不再需要，因此不再在JMX上注册。

如果适用，KieSession MBean现在具有适当的JMX CompositeData和TabularData支持。KieSession MBean继续支持所有与流程相关的聚合统计信息监视，但不再显示每个流程实例的开始/结束日期：jBPM中提供了该范围的审计和日志记录支持。

Drools RHQ / JON插件已更改，以反映上述所有更改，以及旨在正确显示分层嵌套的特定错误修复。

### [2.16。什么是Drools 6.4.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.4.0)

#### [2.16.1。更好的Java 8兼容性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_better_java_8_compatibility)

现在可以在规则的右手边（当时）部分使用Java 8语法（lambdas和方法引用）。

#### [2.16.2。更强大的增量编译](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_more_robust_incremental_compilation)

增量编译（动态规则库更新）在涉及子网（带有复杂存在性模式的规则）的一个或多个规则被涉及时存在一些相关缺陷，特别是当不同规则共享相同的子网时。此问题需要对现有渐进式编译算法进行部分重写，然后进行完整的审计，并且仅在此区域通过超过20,000个测试用例制作的全新测试套件进行验证。

#### [2.16.3。改进了多线程行为](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_multi_threading_behaviour)

处理多线程的引擎代码已被部分重写，以消除大量同步点并提高稳定性和可预测性。特别是，这个新的实现允许用户线程（在会话上执行插入/更新/删除操作），引擎线程（执行适当的规则评估）和Timer 1（执行基于时间的操作）之间更清晰的分离和更好的交互像事件到期）。

由于Drools 6引入了新的phreak算法，使得这种改进成为可能。事实上，在ReteOO算法中，网络评估是在用户插入/更新/删除操作期间执行的，这意味着每个用户操作都会锁定整个引擎。与phreak相反，插入/更新/删除是分开的，并且在调用fireAllRules或fireUntilHalt时进行网络评估。

更详细地说，这个改进是由2部分组成的。首先添加一个新的线程安全队列，将所有用户操作存储为命令。该队列由用户线程填充，而其条目在规则评估阶段期间由引擎线程刷新和处理。第二部分介绍了一个协调用户，计时器和引擎线程的状态机，然后提供了一种更清晰和自我记录的方式来模拟他们的交互。

#### [2.16.4。OOPath改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_oopath_improvements_2)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |

OOPath已经与Drools 6.3.0一起引入。在Drools 6.4.0中，它已被增强以支持以下功能：

* 约束还可以与当前迭代之前遍历的图的对象有关联。例如下面的OOPath：

Student( $grade: /plan/exams/grades{ result > ../averageResult } )

只会匹配高于平均水平的成绩。

* 一个约束也可以递归地成为另一个OOPath，如下所示：

Student( $exam: /plan/exams{ /grades{ result > 20 } } )

* 也可以使用?/分隔符而不是/一个分隔符。如下例所示：

Student( $grade: /plan/exams{ course == "Big Data" }?/grades )

通过这样做，引擎将对考试作出的更改作出反应，或者如果将考试添加到计划中，而不是在现有考试中添加新成绩。当然，如果一个OOPath块没有响应，那么从那里直到表达式结尾的OOPath的所有剩余部分也不会被反应。例如下面的OOPath

Student( $grade: ?/plan/exams{ course == "Big Data" }/grades )

将完全没有反应。出于这个原因，不允许?/在相同的OOPath中多次使用分隔符，因此表达式如下所示：

Student( $grade: /plan?/exams{ course == "Big Data" }?/grades )

会导致编译时错误。

### [2.17。KIE Workbench 6.4.0中的新增值得关注](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.4.0.final)

#### [2.17.1。新的外观和感觉](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_look_and_feel)

整个工作台的整体外观已经更新为采用[PatternFly](https://www.patternfly.org/)。该更新为每个屏幕带来更清晰，轻量且更一致的用户体验。通过删除所有不必要的视觉元素，允许用户专注于数据和任务。交互和行为大部分保持不变，将此更改的范围限制为视觉更新。

*图45.工作台 - 新的外观*

#### [2.17.2。各种UI改进](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_various_ui_improvements)

除了上述针对一般外观的PatternFly更新之外，工作台中的许多单独组件都得到了改进，以创建更好的用户体验。这涉及确保模式弹出窗口的默认大小适合于适合相应的内容，调整文本字段的大小以及对齐标签，并改善在较小屏幕上使用时各种组件的调整行为。

*图46. Workbench - 适当大小的弹出窗口*

*图47.工作台 - 适当大小的文本字段和对齐的标签*

*图48.工作台 - 调整后的编辑器窗口的水平空间有限*

#### [2.17.3。新的区域设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_locales)

现在已添加语言环境ru（俄语）和zh\_TW（中文繁体）。

现在支持的语言环境是：

* 默认英语。
* es （西班牙语）
* fr （法文）
* de （德语）
* ja （日本）
* pt\_BR （葡萄牙语 - 巴西）
* zh\_CN （简体中文）
* zh\_TW （中国传统的）
* ru （俄语）

#### [2.17.4。创作 - 导入 - 一致的术语](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_imports_consistent_terminology)

工作台曾在项目编辑器中为“导入建议”提供了一个部分，这是用户注册Java运行时环境提供的类以供规则创作使用的真正方法。此外，编辑有一个“配置”选项卡，用户可以从其他包中导入类到规则所在的类。

这两个术语都不明确，两者都与Workbench的其他方面不一致。

我们已经改变了这些术语（希望）其含义更加清晰，并与在Workbench中创作Java类时使用的“数据对象”术语保持一致。

*图49.项目编辑器 - 外部数据对象*

*图50.项目编辑器 - 定义外部数据对象*

*图51.资产编辑器 - 数据对象*

“数据对象”屏幕列出与资产相同包中的所有数据对象，并允许导入其他包中的其他数据对象。

*图52.资产编辑器 - 定义可用于创作的数据对象*

#### [2.17.5。禁用自动构建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_disable_automatic_build)

使用Project Explorer浏览项目时，工作台会自动构建所选项目，并在消息控制台中显示构建消息。虽然这是有益的，但它在创作大型项目时可能会对工作台的性能产生不利影响。org.kie.build.disable-project-explorer系统属性现在可以禁用自动构建。将该值设置true为禁用。默认值是false。

#### [2.17.6。支持SCP样式git存储库URL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_support_for_code_scp_code_style_code_git_code_repository_urls)

例如，克隆git存储库时，现在可以使用SCP样式URLS git@github.com:user/repository.git。如果您的操作系统的公共密钥库受密码保护，则密码可以与org.uberfire.nio.git.ssh.passphrase系统属性一起提供。

#### [2.17.7。创作 - 重复的GAV检测](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authoring_duplicate_gav_detection)

在执行以下任何操作时，现在针对项目已解决的所有Maven存储库进行检查，以确定项目的GroupId，ArtifactId和Version是否预先存在。如果发现冲突，则防止操作; 尽管这可以由admin角色的用户覆盖。

|  |  |
| --- | --- |
|  | 该功能可以通过系统属性设置来禁用org.guvnor.project.gav.check.disabled到true。 |

已解决的存储库是在以下位置发现的存储库：

* 项目的POM <repositories>部分（或任何父母POM）。
* 该项目的POM <distributionManagement>部分。
* Maven的全局settings.xml配置文件。

受影响的操作：

* 创建新的管理存储库。
* 使用项目编辑器保存项目定义。
* 将新模块添加到托管多模块存储库。
* 保存pom.xml文件。
* 使用项目编辑器构建和安装项目。
* 使用项目编辑器构建和部署项目。
* 资产管理操作构建，安装或部署项目。
* REST 创建，安装或部署项目的操作。

具有该Admin角色的用户可以覆盖使用项目编辑器中的“存储库”设置检查的存储库列表。

*图53.项目编辑器 - 查看已解析的存储库*

*图54.项目编辑器 - 已解析的存储库列表*

*图55.检测到重复的GAV*

#### [2.17.8。新的执行服务器管理用户界面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_execution_server_management_user_interface)

KIE执行服务器管理界面已完全重新设计，以适应最近推出的重大改进。除了从头开始构建新UI以及遵循PatternFly提供的最佳实践之外，新界面扩展了先前的功能，使用户可以更好地控制其服务器。

*图56. KIE执行服务器 - 新的用户界面*

#### [2.17.9。用户和组管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_and_group_management)

提供后端服务和直观友好的用户界面，使工作台管理员可以管理应用程序的用户和组。

该界面为工作台管理员提供了执行领域相关操作的能力，如创建用户，创建组，给指定用户分配组或角色等。

默认情况下，它具有用于管理Wildfly，EAP和Tomcat默认领域的内置实现，并且其设计具有可扩展性 - 任何第三方领域管理系统均可轻松集成到工作台中。

### [2.18。什么是Drools 6.3.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.3.0)

#### [2.18.1。使用OOPath浏览对象的图形](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_browsing_graphs_of_objects_with_oopath)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |

当事实的字段是集合时，可以使用from关键字将集合中的所有项目绑定并推理。尽管如此，当需要浏览对象的图形时，from条件元素的广泛使用可能会导致冗长的cubersome语法，如下例所示：

*示例5.使用from浏览对象的图形*

rule "Find all grades for Big Data exam" **when**

$student: Student( $plan: plan )

$exam: Exam( course == "Big Data" ) from $plan.exams

$grade: Grade() from $exam.grades

**then** /\* RHS \*/ end

在这个例子中，已经假设使用一个领域模型，该模型由一个Student具有Plan研究的人组成：a Plan可以具有零个或多个Exams和Exam零个或多个Grades。请注意，只有图形的根对象（Student在这种情况下）需要在工作内存中才能生效。

通过从XPath中借鉴想法，可以使这种语法更加简洁，因为XPath在处理集合和过滤约束时具有导航相关元素的简洁表示法。这个XPath启发的符号被称为OOPath是明确地打算浏览对象的图形。使用这个符号，前面的例子可以被重写，如下所示：

*示例6.使用OOPath浏览对象图*

rule "Find all grades for Big Data exam" **when**

Student( $grade: /plan/exams[course == "Big Data"]/grades )

**then** /\* RHS \*/ end

形式上，OOPath表达式的核心语法可以用EBNF表示法以这种方式定义。

OOPExpr = "/" OOPSegment { ( "/" | "." ) OOPSegment } ;

OOPSegment = [ID ( ":" | ":=" )] ID ["[" Number "]"] ["{" Constraints "}"];

实际上，OOPath表达式具有以下特征。

* 它必须从开始/。
* 它可以用.操作符取消引用对象的单个属性
* 它可以使用/操作符取消引用对象的多个属性。如果返回一个集合，它将迭代集合中的值
* 在遍历引用对象时，它可以过滤那些不满足一个或多个约束的对象，并将其写为大括号之间的谓词表达式，如下所示：

Student( $grade: /plan/exams[course == "Big Data"]/grades )

* 项目也可以通过将其索引放在方括号之间来访问，如下所示：

Student( $grade: /plan/exams[0]/grades )

遵守Java约定与基于XPath 1的版本相比，OOPath索引是基于0的

##### [2.18.1.1。反应性OOPath](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reactive_oopath)

目前Drools无法对OOPath表达式评估过程中涉及深度嵌套的更新做出反应。为了使这些对象能够立即响应变化，有必要让这些对象扩展类org.drools.core.phreak.ReactiveObject。计划通过实施一种机制来克服属于特定域模型的类的机制，从而克服这个限制。

扩展该类后，域对象可以通过调用继承方法更新其某个字段的某个字段时通知引擎notifyModification，如下例所示：

*示例7.通知引擎某个考试已移至其他课程*

**public** **void** **setCourse**(String course) {

**this**.course = course;

notifyModification(**this**);

}

通过这种方式，如果考试移至不同的课程，则重新触发该规则，并重新计算匹配该规则的成绩列表。

#### [2.18.2。Eclipse的Kie Navigator视图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_shared.releasenotes.kienavigator)

Eclipse工具中添加了新的查看器。这个Kie Navigator View用于管理Kie Server的安装和项目。

请阅读[Kie Navigator View](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/Eclipe-KieNavigator-section.html#Kie_Navigator_View)一章 了解更多关于这项新功能的信息

### [2.19。KIE Workbench 6.3.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.3.0.final)

#### [2.19.1。实时验证和确认决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_real_time_validation_and_verification_for_the_decision_tables)

决策表用于验证表的验证按钮。现在将其删除，并在每个单元格值更改后验证表格。验证和验证检查包括：

* 冗余
* 涵摄
* 冲突
* 缺少列

这些检查在工作台文档中有详细说明。

#### [2.19.2。改进的DRL编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_drl_editor)

DRL编辑已经经历了一次彻底的改变。从纯文本区域移动到使用ACE编辑器和自定义DRL语法突出显示。

*图57. ACE编辑器*

#### [2.19.3。资产锁定](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_asset_locking)

为了避免编辑资产时发生冲突，引入了新的锁定机制，确保一次只有一个用户可以编辑资产。当用户开始编辑资产时，会自动获取锁定。这由资产标题栏以及项目资源管理器视图中出现的锁定符号表示。如果用户开始编辑已锁定的资产，则会弹出通知，通知用户该资产目前无法编辑，因为其他用户正在处理该资产。只要编辑用户持有锁定，其他用户的更改将被阻止。当编辑用户保存或关闭资产或从工作台注销时，锁将自动释放。如果需要，每个用户还可以选择在元数据标签中强制使用锁定版本。

*图58.编辑资产会自动获取一个锁*

*图59.锁定的资产不能被其他用户编辑*

#### [2.19.4。数据建模器工具窗口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_modeller_tool_windows)

Drools和jBPM配置，持久性（请参阅生成启用JPA的数据模型）和高级配置已移入“工具窗口”。“工具Windows”是在最新的Uberfire版本中引入的一个新概念，它支持开发情境感知屏幕。每个“工具窗口”将包含一个域编辑器，它将管理一组相关的数据对象参数。

*图60. Drools和jBPM域工具窗口*

*图61.持久性工具窗口*

*图62.高级配置工具窗口*

#### [2.19.5。生成启用JPA的数据模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_generation_of_jpa_enabled_data_models)

数据建模器已扩展为支持生成可持久数据对象。持久数据对象基于JPA规范，并且所有基础元数据都是自动生成的。

* “新建→数据对象”数据对象可以在创建时标记为可持久化。

*图63.新数据对象*

* 持久工具窗口包含数据对象和字段的JPA域编辑器。每个编辑器都将管理默认生成的JPA元数据

*图64.数据对象级JPA域编辑器*

*图65.字段级JPA域编辑器*

* 持久性配置屏幕已添加到项目编辑器中。

*图66.持久性配置*

#### [2.19.6。数据集创作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_set_authoring)

增加了创作数据集定义的新视角。数据集定义可以从外部系统（如数据库，CSV / Excel文件）检索数据，甚至可以使用Java类生成数据。一旦数据可用，它就可以用于从Perspective Editor中创建图表和仪表板，只需从任何可用的数据集中提供图表即可。

*图67.数据集创作视角*

### [2.20。什么是Drools 6.2.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.2.0)

#### [2.20.1。传播模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_propagation_modes)

作为Drools引擎的默认算法引入了PHREAK，这使得规则的评估懒惰。这种新的Drools懒惰行为允许相关的性能提升，但在某些特定情况下，打破了一些Drools功能的语义。

更准确地说，在某些情况下，有必要立即将新的事实传播到会议中。例如，Drools允许通过预先添加一个'？'来在只有拉（或被动）模式下执行查询。符号调用，如下例所示：

*示例8.一个被动查询*

query **Q** (Integer i)

**String**( **this** == i.toString() )

end

rule R when

$i : **Integer**()

?**Q**( $i; )

then

System.out.**println**( $i );

end

在这种情况下，由于查询是被动的，因此它不应该在查询本身中插入匹配连接条件的字符串。换句话说，这个命令序列

KieSession ksession = ...

ksession.insert(1);

ksession.insert("1");

ksession.fireAllRules();

应该不会导致规则R被触发，因为满足查询条件的字符串已经在Integer之后插入，并且被动查询不应对此插入作出反应。相反，如果插入序列被反转，则规则应该触发，因为插入整数（当被动查询可以通过存在的字符串来满足时）将触发它。

不幸的是，PHREAK的懒惰性质不允许引擎对两个事实的插入顺序作出任何区分，所以规则将在两种情况下触发。在这种情况下，有必要像以前的基于RETEOO的引擎一样热切地评估规则。

在其他情况下，要求传播是渴望的，这意味着它不是直接的，但无论如何必须在引擎/议程开始计划评估之前发生。例如，当规则具有无回路或锁定激活属性时，这是必要的，事实上，当这种情况发生时，传播模式由引擎自动执行。

为了涵盖这些用例以及需要立即或积极规则评估的所有其他情况，可以通过使用**@Propagation（Propagation.Type）**注释规则本身来进行声明性指定，其中Propagation.Type是一个枚举3个可能的值：

* **IMMEDIATE**表示传播立即执行。
* **EAGER**意味着繁殖是在计划评估之前懒惰地执行但热切地评估的。
* **LAZY**意味着传播是完全懒惰的，这是默认的PHREAK行为

这意味着以下drl：

*示例9.使用被动查询的数据驱动规则*

query **Q** (Integer i)

**String**( **this** == i.toString() )

end

rule R @**Propagation**(IMMEDIATE) when

$i : **Integer**()

?**Q**( $i; )

then

System.out.**println**( $i );

end

当且仅当Integer被插入到String之后时，才会使规则R被触发，从而按照被动查询的语义行事。

### [2.21。KIE Workbench 6.2.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.2.0.final)

#### [2.21.1。将版本库或版本库的一部分下载为ZIP](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_download_repository_or_part_of_the_repository_as_a_zip)

该功能可以从存储库下载一个存储库或文件夹作为ZIP文件。

*图68.下载当前的存储库或项目*

*图69.下载文件夹*

#### [2.21.2。项目编辑器权限](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_editor_permissions)

已添加为项目编辑器配置基于角色的权限的功能。

权限可以使用该WEB-INF/classes/workbench-policy.properties文件进行配置。

支持以下权限：

* 保存按钮

feature.wb\_project\_authoring\_save

* 删除按钮

feature.wb\_project\_authoring\_delete

* 复制按钮

feature.wb\_project\_authoring\_copy

* 重命名按钮

feature.wb\_project\_authoring\_rename

* 构建和部署按钮

feature.wb\_project\_authoring\_buildAndDeploy

#### [2.21.3。在指导决策表向导中统一验证样式。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_unify_validation_style_in_guided_decision_table_wizard)

我们所有的新屏幕都使用GWT-Bootstrap小部件，并提醒用户以一致的方式输入错误。

最明显的差异之一是导向决策表向导以与我们使用GWT-Bootstrap不一致的方式提醒错误。

此向导已更新为使用新外观。

*图70.新的指导决策表向导验证*

#### [2.21.4。改进的奇才](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_wizards)

在指导决策表向导的重新工作过程中，为了使其验证与应用程序的其他领域保持一致，我们借此机会将向导框架移至GWT-Bootstrap。

由此产生的外观更令人愉快。我们希望随着时间和优先权的允许，将更多传统编辑人员迁移到GWT-Bootstrap。

*图71.新建向导框架*

#### [2.21.5。XLS，导向决策表和指导模板的一致行为](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_consistent_behaviour_of_xls_guided_decision_tables_and_guided_templates)

一致性对每个人都是好事。用户可以期望不同的创作隐喻产生相同的规则行为（并且开发人员知道什么时候出现错误！）。

在XLS决策表，指导性规则模板和方法规则模板的方式中产生了空单元的基本规则中的一些不一致之处。这些已被淘汰，使他们的操作一致。

* 如果所有约束都有空值（空单元），则不会创建模式。

你是否需要该模式，但没有限制; 你将需要包含约束this != null。

该操作与XLS和指导决策表一直如何工作一致。

* 您可以通过用双引号标记来为空字符串或空白字符串字段定义约束。在生成规则时，封闭的引号将从该值中移除。

对其他字符串值使用引号不是必需的，它们可以省略。然而，它们的使用对于区分空字符串的约束和空字符是非常重要的 - 在这种情况下，约束被忽略。

#### [2.21.6。改进的元数据选项卡](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_metadata_tab)

之前版本中提供的“元数据”选项卡进行了重新设计，以提供更好的资产版本信息浏览和恢复。现在，每个工作台编辑器都会提供一个“概览选项卡”，使用户能够管理以下信息。

*图72.改进的元数据选项卡*

* 版本历史

版本历史记录显示资产版本的表格视图，并提供一个“选择”按钮，使用户能够加载以前创建的版本。

*图73.版本历史*

* 元数据

元数据部分可以访问其他文件属性。

*图74.元数据部分*

* 评论区

重新设计的评论区域可以更清晰地讨论文件。

* 版本选择下拉菜单

位于菜单栏的“版本选择器下拉菜单”提供了从“编辑器”选项卡加载和恢复以前版本的功能，而无需打开“概览选项卡”以加载“版本历史记录”。

*图75.版本选择下拉菜单*

#### [2.21.7。改进的数据对象编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_data_objects_editor)

Java编辑器被统一到了标准工作台编辑器的功能上。这意味着现在每个数据对象都在自己的编辑器窗口中编辑。

*图76.改进的数据对象编辑器*

* 添加了“新建→数据对象”选项来创建数据对象。
* “概览”选项卡是为每个文件添加的，用于管理文件元数据并可以访问文件版本历史记录。
* 添加了可编辑的“源选项卡”选项卡。现在，Java代码可以由管理员使用工作台进行修改。
* 提供“编辑器” - “源选项卡”往返。这将使管理员可以对生成的Java代码进行手动更改，并返回到编辑器选项卡继续工作。
* 类使用检测。每当一个数据对象即将被删除或重命名，该项目将被扫描的类用法。如果找到了用法（例如，在drl文件，决策表等中），则用户将收到警报。这将阻止用户打破项目构建。

*图77.使用情况检测*

#### [2.21.8。执行服务器管理UI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_execution_server_management_ui)

在**服务器**顶层菜单下添加了一个名为**Management**的新视图。这种观点为用户提供了使用多个容器来管理多个执行服务器的能力。可用功能包括连接到已部署的执行服务器; 创建新的，启动，停止，删除或升级容器。

*图78.管理角度*

|  |  |
| --- | --- |
|  | 当前版本的Execution Server仅支持基于规则的执行。 |

#### [2.21.9。社交活动](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_social_activities)

在一个全新的功能，**社会活动**被称为一个新的顶级菜单项的组中添加**活动**。

这个新功能分为两个不同的角度：**时间线透视** 和**人员透视**。

在**时间线透视图**显示在左侧最近登录的用户创建或编辑的资产。在主窗口中有“最新更改”屏幕，显示所有最近更新的资产以及通过资源库过滤最近更新的选项。

*图79.时间轴透视图*

在**人们视野**是用户的主页。显示他的信息（包括来自用户电子邮件的gravatar图片），用户连接（用户关注的人）以及用户最近的活动。还有一种编辑用户信息的方法。搜索建议可以用于导航到用户配置文件，跟随他并在您的时间线上查看他的更新。

*图80.人员视角*

*图81.编辑用户信息*

#### [10年2月21日。贡献者仪表板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_contributors_dashboard)

名为“ **贡献者”的**全新视角已添加到名为“ **活动”**的新顶级菜单项组下。这个观点本身就是一个仪表板，它显示了有关在工作台内对管理的组织/存储库作出的贡献的几个指标。每当组织/存储库从工作台添加/删除时，仪表板本身都会相应更新。

这种新的视角允许监控管理存储库上的基础活动。

*图82.贡献者视角*

#### [11年2月21日。包选择器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_package_selector)

项目资源管理器的上下文驱动了创作时新资产的位置。

这已被New Resource Popup中的Package Selector所取代。

该位置默认为Project Explorer上下文，但现在可以更轻松地选择不同的包。

*图83.程序包选择器*

#### [12年2月21日。改善视觉一致性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_improved_visual_consistency)

所有Popups都被重构为使用GWT-Bootstrap小部件。

虽然简单的改变，但它为应用程序整体带来更大的视觉一致性。

*图84.示例引导式决策表编辑器弹出窗口*

*图85.示例引导规则编辑器弹出*

#### [13年2月21日。指导决策树编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guided_decision_tree_editor)

添加了一个新的编辑器来支持简单决策树的建模。

有关使用的更多信息，请参阅用户指南中的适用部分。

*图86.示例指导决策树*

#### [14年2月21日。创建存储库向导](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_create_repository_wizard)

已经创建了一个向导来指导存储库创建过程。现在，用户可以在存储库创建时决定它是否应该是托管或非托管存储库并配置所有相关参数。

*图87.创建存储库向导1/2*

*图88.创建存储库向导2/2*

#### [15年2月21日。存储库结构屏幕](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_repository_structure_screen)

新的存储库结构屏幕将允许用户管理指定存储库的项目，以及与托管存储库相关的其他操作，例如：分支创建，资产升级和项目发布。

*图89.托管存储库的存储库结构屏幕*

*图90.非托管存储库的存储库结构屏幕*

### [2.22。集成中的新增值得关注6.2.0](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.releasenotesintegration.6.2.0)

#### [2.22.1。KIE执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_execution_server)

创建了一个新的KIE执行服务器，其目的是支持部署kjars和为远程规则执行自动创建REST端点。这个初始实现支持通过REST提供和执行kjars，而不需要任何粘合代码。

用户界面也集成到工作台中进行远程供应。有关详细信息，请参阅工作台的新增＆值得注意的内容。

*Kie服务器界面*

@Path("/server")

**public** **interface** **KieServer** {

@GET

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **getInfo**();

@POST

@Consumes({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **execute**( CommandScript command );

@GET

@Path("containers")

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **listContainers**();

@GET

@Path("containers/{id}")

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **getContainerInfo**( @PathParam("id") String id );

@PUT

@Path("containers/{id}")

@Consumes({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **createContainer**( @PathParam("id") String id, KieContainerResource container );

@DELETE

@Path("containers/{id}")

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **disposeContainer**( @PathParam("id") String id );

@POST

@Path("containers/{id}")

@Consumes({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **execute**( @PathParam("id") String id, String cmdPayload );

@GET

@Path("containers/{id}/release-id")

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **getReleaseId**( @PathParam("id") String id);

@POST

@Path("containers/{id}/release-id")

@Consumes({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **updateReleaseId**( @PathParam("id") String id, ReleaseId releaseId );

@GET

@Path("containers/{id}/scanner")

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **getScannerInfo**( @PathParam("id") String id );

@POST

@Path("containers/{id}/scanner")

@Consumes({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

@Produces({MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON})

**public** Response **updateScanner**( @PathParam("id") String id, KieScannerResource resource );

}

### [2.23。什么是Drools 6.1.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.1.0)

#### [2.23.1。JMX支持KieScanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jmx_support_for_kiescanner)

增加了对KieScanner和KieContainer的JMX监控和管理的支持。要启用，请将该属性设置kie.scanner.mbeans为enabled，例如通过Java命令行： -Dkie.scanner.mbeans=enabled 。

KieScannerMBean将以下列名称注册：

它公开了以下属性：

* **扫描仪版本ID：**扫描仪配置的版本ID。可能包括Maven范围版本和特殊关键字，如LATEST，SNAPSHOT等。
* **当前版本ID：**解析为工件的实际版本ID。
* **状态：**启动，扫描，更新，运行，停止，关闭

它还暴露了以下操作：

* **scanNow（）：**强制立即扫描Maven存储库寻找工件更新
* **start（）：**根据轮询间隔参数开始轮询Maven存储库中的工件更新
* **stop（）：**停止自动轮询Maven仓库

### [2.24。KIE Workbench 6.1.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.1.0)

#### [2.24.1。Data Modeler - 往返和源代码保存](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_modeler_round_trip_and_source_code_preservation)

现在支持Data modeler和Java源代码之间的完整往返。无论Java代码是在哪里生成的（例如Eclipse，Data建模器），数据建模人员只会更新必要的代码块以保持模型的更新。

#### [2.24.2。Data Modeler - 改进了注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_modeler_improved_annotations)

新增注释@TypeSafe，@ClassReactive，@PropertyReactive，@Timestamp，@Duration和@Expires是为了丰富当前由数据建模人员管理的Drools注释。

#### [2.24.3。标准化表格数据的显示](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_standardization_of_the_display_of_tabular_data)

我们用一个新的表格小部件来标准化表格数据的显示。

新表格支持以下功能：

* 选择可见的列
* 可调整大小的列
* 可移动的列

*图91.新表*

该表用于以下情况：

* 收件箱（传入更改）
* 收件箱（最近编辑）
* 收件箱（最近打开）
* 项目问题总结
* 工件存储库浏览器
* 项目编辑器依赖关系网格
* 项目编辑器KSession网格
* 项目编辑器工作项处理程序配置网格
* 项目编辑器监听器配置网格
* 搜索结果网格

#### [2.24.4。生成modify(x) {…​}块](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_generation_of_code_modify_x_code_blocks)

引导规则编辑器，引导模板编辑器和引导决策表编辑器已更改为生成 modify(x){…​}

从历史上看，这些编辑器支持较旧的update(x)语法，因此在Workbench内创建的规则不会对@PropertyReactive模型内的相关注释做出正确响应。现在已经通过使用modify(x){…​}块来纠正这一问题。

### [2.25。KIE A​​PI 6.0.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.releasenoteskie.6.0.0)

#### [2.25.1。新的KIE名字](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_new_kie_name)

KIE是用于将我们的相关项目组合在一起的新伞名称; 随着家庭的不断成长。KIE也用于统一API的通用部分; 如建设，部署和加载。这取代了以前使用的kiegroup和知识关键字。

*图92. KIE解剖学*

#### [2.25.2。Maven对齐项目和模块以及Maven部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_aligned_projects_and_modules_and_maven_deployment)

5.x系列最大的抱怨之一是缺乏定义的部署方法。Drools和jBPM使用的机制非常灵活，但它太灵活了。6.0的重点是精简系统的构建，部署和加载（利用）方面。构建和部署活动现在与Maven和Maven存储库保持一致。加载规则和进程的利用率现在是约定和面向配置的，而不是程序化的，并且具有理智的默认值以最小化配置。

项目可以使用Maven构建并安装到本地M2\_REPO或远程Maven存储库。然后使用Maven声明和构建依赖关系的类路径，供KIE访问。

#### [2.25.3。基于配置和约定的项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configuration_and_convention_based_projects)

'kmodule.xml'为KIE项目提供声明性配置。约定和默认值用于减少所需配置的数量。

*例10.声明KieBases和KieSessions*

<kmodule xmlns="http://www.drools.org/xsd/kmodule">

<kbase name="kbase1" packages="org.mypackages">

<ksession name="ksession1"/>

</kbase>

</kmodule>

*例11.使用KieSession*

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

#### [2.25.4。KieBase包容](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_inclusion)

可以将属于KieBase的所有KIE工件包含到第二个KieBase中。这意味着第二个KieBase除了直接定义在其中的所有规则，函数和过程之外，还将包含在所包含的KieBase中创建的那些规则，功能和过程。这个包含可以在kmodule.xml文件中声明地完成

*例12.将KieBase包含到另一个声明中*

<kmodule xmlns="http://www.drools.org/xsd/kmodule">

<kbase name="kbase2" includes="kbase1">

<ksession name="ksession2"/>

</kbase>

</kmodule>

或以编程方式使用KieModuleModel。

*示例13.以编程方式将KieBase包含到另一个中*

KieModuleModel kmodule = KieServices.Factory.get().newKieModuleModel();

KieBaseModel kieBaseModel1 = kmodule.newKieBaseModel("KBase2").addInclude("KBase1");

#### [2.25.5。KieModules，KieContainer和KIE-CI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodules_kiecontainer_and_kie_ci)

任何Maven在其中生成带有“kmodule.xml”的JAR都被认为是KieModule。这可以从类路径中加载，也可以在运行时从资源位置加载。如果kie-ci依赖项位于类路径上，它将嵌入Maven，所有解析操作都将使用Maven自动完成，并且可以访问本地或远程存储库。Maven配置遵守Settings.xml。

KieContainer提供了一个使用KieModule的运行时，版本控制贯穿整个Maven。Kie-ci将动态地从所有Maven声明的依赖项中为待加载的工件创建一个类路径。支持Maven LATEST，SNAPSHOT，RELEASE和版本范围。

*示例14.使用并运行Java*

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.newKieContainer(

ks.newReleaseId("org.mygroup", "myartefact", "1.0") );

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

KieContainers可以动态更新为特定版本，如果KIE-CI位于类路径中，则可以通过Maven解决。对于有状态的KieSessions，现有会话将逐步更新。

*示例15.动态更新 - Java*

KieContainer kContainer.updateToVersion(

ks.newReleaseId("org.mygroup", "myartefact", "1.1") );

#### [2.25.6。KieScanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiescanner)

这KieScanner是一个面向Maven的版本，替代了Drools 5中的KnowledgeAgent。它持续监视您的Maven存储库，以检查是否已安装Kie项目的新版本，如果是，则将其部署到KieContainer该项目的包装中。KieScanner要求使用kie-ci.jar在类路径上。

A KieScanner可以KieContainer在下面的例子中注册。

*示例16.在KieContainer上注册并启动KieScanner*

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "myartifact", "1.0-SNAPSHOT" );

KieContainer kContainer = kieServices.newKieContainer( releaseId );

KieScanner kScanner = kieServices.newKieScanner( kContainer );

*// Start the KieScanner polling the Maven repository every 10 seconds*

kScanner.start( 10000L );

在这个例子中，它KieScanner被配置为以固定的时间间隔运行，但是也可以通过调用scanNow()它的方法按需运行它。如果KieScanner在Maven存储库中找到Kie项目所使用的更新版本，KieContainer它将自动下载新版本并触发新项目的增量构建。从这一刻起所有新KieBaseS和KieSession从创建小号KieContainer将使用新的项目版本。

#### [2.25.7。分层类加载器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hierarchical_classloader)

CompositeClassLoader不再使用; 因为它是性能问题和错误的持续来源。传统的分层类加载器现在被使用。根类加载器位于KieContext级别，每个名称空间只有一个子类ClassLoader。这使得添加和删除规则变得更加简洁，但是现在在DRL文件中的名称空间之间不会引用; 即函数只能由声明它们的名称空间使用。建议在项目中使用静态Java方法，这对所有名称空间都是可见的; 但是那些不能（像根KieContainer ClassLoader上的其他类）被动态更新。

#### [2.25.8。旧版API适配器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_legacy_api_adapter)

用于构建和运行Drools和jBPM的5.x API仍然可以通过Maven依赖项“knowledge-api-legacy5-adapter”获得。由于6.0版本的部署性质发生了重大变化，因此无法为KnowledgeAgent提供适配器桥接。如果其他方法丢失或存在问题，请打开JIRA，我们将修复6.1

#### [2.25.9。KIE文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_documentation)

虽然已经添加了许多新的文档来处理新的KIE A​​PI，但是整个文档还没有被提出来。出于这个原因，将继续引用旧的术语。提前道歉，并感谢您的耐心。我们希望社区中的人员能够与我们合作，为6.1版本全面更新文档

### [2.26。什么是Drools 6.0.0中的新增功能和值得注意的功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.releasenotesdrools.6.0.0)

#### [2.26.1。PHREAK - 惰性规则匹配算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_phreak_lazy_rule_matching_algorithm)

在6.0中为Drools完成的主要工作涉及新的PREAK算法。这是一个懒惰的算法，应该使Drools能够处理更多的规则和事实。AngendaGroups现在可以帮助提高性能，因为规则在尝试触发它们之前不会被评估。

顺序模式继续支持PHREAK，但现在允许“修改”。虽然顺序配置没有“推论”，但由于规则是懒散评估的，任何尚未评估的规则都会因'修改'而看到更新的数据。这更符合人们直觉地认为顺序工作的方式。

冲突解决令已经调整为PHREAK，现在按显着顺序排列，然后排序; 基于文件中的规则位置。在Drools 6.0.0之前，突出之后，它被认为是任意的。当KieModules和updateToVersion用于动态部署时，文件中的规则顺序通过差异处理保留。

#### [2.26.2。自动开启被动模式下的定时规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_automatically_firing_timed_rule_in_passive_mode)

当规则引擎默认运行在被动模式（即：使用fireAllRules）时，它不会触发定时规则的后果，除非fireAllRules不再被调用。现在可以通过使用a配置KieSession来更改此默认行为，TimedRuleExectionOption如下例所示。

*例17.配置KieSession自动执行定时规则*

KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();

ksconf.setOption( TimedRuleExectionOption.YES );

KSession ksession = kbase.newKieSession(ksconf, **null**);

也可以对必须自动执行的定时规则进行更细粒度的控制。要做到这一点，有必要设置一个'FILTERED' `TimedRuleExectionOption来允许定义一个回调来过滤这些规则，就像下一个例子中所做的那样。

*示例18.配置筛选器以选择应自动执行哪些定时规则*

KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();

conf.setOption( **new** TimedRuleExectionOption.FILTERED(**new** TimedRuleExecutionFilter() {

**public** **boolean** **accept**(Rule[] rules) {

**return** rules[0].getName().equals("MyRule");

}

}) );

#### [2.26.3。表达式定时器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_expression_timers)

现在可以将间隔定时器的延迟和间隔定义为表达式而不是固定值。为此，有必要将计时器声明为表达式1（由“expr：”指示），如下例所示：

*例子19.表达式计时器例子*

declare Bean

delay : String = "30s"

period : **long** = 60000

end

rule "Expression timer"

timer( expr: $d, $p )

when

**Bean**( $d : delay, $p : period )

then

end

在这种情况下，表达式$ d和$ p可以使用规则的模式匹配部分中定义的任何变量，并且可以是任何可以在持续时间内解析的字符串或任何将在内部转换为长整型的数值表示以毫秒表示的持续时间。

间隔和表达式定时器都可以有3个可选参数，名称分别为“start”，“end”和“repeat-limit”。当使用这些参数中的一个或多个参数时，定时器定义的第一部分必须跟有分号';' 并且参数必须用逗号“，”分隔，如下例所示：

*例20.一个带有开始和结束的间隔定时器*

timer (**int**: 30s 10s; start=3-JAN-2010, end=5-JAN-2010)

开始和结束参数的值可以是Date，一个表示Date或Long的字符串，或者更一般的任何Number，它将在Java日期中进行转换，应用以下转换：

**new** Date( ((Number) n).longValue() )

相反，repeat-limit只能是一个整数，它定义了定时器允许的最大重复次数。如果同时设置了结束和重复限制参数，则当两个参数中的第一个匹配时，计时器将停止。

启动参数的使用意味着定时器阶段的定义，其中阶段的开始由启动本身加上最终的延迟给出。换句话说，在这种情况下，定时规则将在以下时间安排：

start + delay + n\*period

达到重复限制时间并且不迟于结束时间戳（以先到者为准）。例如，具有以下间隔计时器的规则

timer ( **int**: 30s 1m; start="3-JAN-2010" )

将在2010年1月3日午夜后的每分钟30秒安排。这也意味着，例如，如果您在2010年2月3日午夜打开系统，则不会立即安排，但会保留定时器定义的阶段，因此将在第一次30秒后安排午夜。如果由于某种原因系统处于暂停状态（例如，会话被序列化，并在一段时间后进行反序列化），规则将只被安排一次以从缺失的激活中恢复（不管我们错过了多少次激活），随后将再次计划与计时器相位。

#### [2.26.4。RuleFlowGroups和AgendaGroups被合并](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleflowgroups_and_agendagroups_are_merged)

这两个组已经合并，现在RuleFlowGroups的行为与AgendaGroups相同。由于弃用的原因，get方法一直存在，但都返回相同的基础数据。当jBPM激活一个组时，它现在只是调用setFocus。RuleFlowGroups和AgendaGroups一起使用时是错误的持续来源。它还使代码库对齐，朝向PHREAK和未来计划的多核心扩展。

### [2.27。KIE Workbench 6.0.0中新的和值得注意的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.releasenotesworkbench.6.0.0)

该工作台使用名为UberFire的新基础项目进行了大规模改造。UberFire受Eclipse的启发，为工作台提供了一个干净，可扩展且灵活的框架。最终结果不仅为我们的最终用户带来更丰富的体验，而且我们现在可以通过基于清洁组件的体系结构更快地开发。如果你喜欢他的Workbench体验，可以立即使用UberFire构建自己的基于Web的仪表板和控制台。

除了向UberFire转型之外，另一个最大的变化是从JCR转移到Git; 有一个实用项目来帮助迁移。Git是最具扩展性和强大的源代码仓库。JGit为Git提供了坚实的OSS实现。这解决了各种JCR实现的持续性能问题，一旦文件数量和版本数量变得过高，这些问题会减慢。有一个大的“低科技”驱动器，以消除复杂性。现在一切都被存储为一个文件，包括元数据。数据库仅用于提供快速索引和搜索。因此，导入和导出都是标准的Git，外部站点（如GitHub）可用于交换存储库。

在5.x开发人员可以使用他们自己的源代码库，然后通过团队提供者推送JCR。这个团队提供者不是全功能的，在Eclipse之外不可用。Git使我们的存储库可以使用任何现有的Git工具或团队提供者。虽然用户界面还不支持，但随着时间的推移，这将被添加，有可能连接到回购和标签，并分支和恢复的东西。

*图93.工作台*

Guvnor品牌从其预定角色泄漏过多; 如创作隐喻，如决策表，被视为Guvnor组件而不是Drools组件。这对于Guvnor在5.x中使用的庞大的项目结构没有帮助。在6.0中，Guvnor的重点已经缩小到封装UberFire插件集，这些插件为构建基于Web的IDE提供了基础。例如用于构建和部署的Maven集成，Maven存储库的管理以及通过收件箱进行的活动通知。Drools和jBPM使用Uberfire作为基础构建工作台发行版，并包含一系列插件，如Guvnor，以及自己的插件，如决策表，引导编辑器，BPMN2设计器，人工任务等。

“模型结构”图概述了新的项目解剖。Drools工作台被称为KIE-Drools-WB。KIE-WB是结合了所有Guvnor，Drools和jBPM插件的超级工作台。jBPM-WB被虚拟出来，因为它实际上并不存在，被KIE-WB重复使用。

*图94.模块结构*

|  |  |
| --- | --- |
|  | KIE Drools Workbench和KIE Workbench共享一套通用的工作台功能组件，例如项目导航，项目定义，基于Maven的项目，Maven Artifact Repository。本文档中详细介绍了这些常用功能。 |

这两个主要分布包括：

* KIE Drools Workbench
  + Drools编辑，了解规则和配套资产。
  + jBPM Designer，用于规则流和支持资产。
* KIE工作台
  + Drools编辑，了解规则和配套资产。
  + jBPM Designer，用于BPMN2和支持资产。
  + jBPM控制台，运行时和人工任务支持。
  + jBPM Form Builder。
  + BAM。

工作台亮点：

* 全新灵活的Workbench环境，带有透视图和面板。
* 遵循KIE A​​PI的新包装和生产系统。
  + 基于Maven的项目。
  + Maven Artifact Repository用全面的依赖性支持取代了Global Area。
* 新数据建模器替换声明性的事实模型编辑器; 将Java类的创作带入创作环境。Java类被打包到项目中，可以在规则，流程等中使用，也可以在您自己的应用程序的外部使用。
* 虚拟文件系统用默认的基于Git的实现替换JCR。
  + 默认的基于Git的实现支持远程操作。
  + 在Workbench中出现外部修改。
* 增量构建系统显示您的项目和资产的近实时验证结果。

编辑本身基本上没有改变; 但是注释导入已经从包装定义转移到单个编辑器，因此您只需要导入用于资产的类型，而不是整个包装。

### [2.28。集成6.0.0中新增值得注意的一点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.releasenotesintegration.6.0.0)

#### [2.28.1。CDI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cdi)

*并行版本加载“jar1.KBase1”KieBase*

@Inject

@KSession("kbase1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.0")

**private** KieBase kbase1v10;

@Inject

@KBase("kbase1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.1")

**private** KieBase kbase1v10;

*并行版本加载“jar1.KBase1”KieBase*

@Inject

@KSession("ksession1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.0")

**private** KieSession ksessionv10;

@Inject

@KSession("ksession1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.1")

**private** KieSession ksessionv11;

CDI现在紧密集成到KIE A​​PI中。它可以用于注入版本化的KieSession和KieBases。

#### [2.28.2。弹簧](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spring)

春季已经改版，现在已经与KIE合并。Spring可以用更强大的spring版本替换'kmodule.xml'。目的是为了与kmodule.xml保持一致

#### [2.28.3。白羊座蓝图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_aries_blueprints)

白羊座蓝图现在也得到支持，并且遵循春季所做的工作。目标是与spring和kmodule.xml保持一致

#### [2.28.4。OSGi就绪](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_osgi_ready)

所有模块都进行了重构以避免软件包分割，这是5.x中的一个问题。测试已转移到PAX。

## [3.兼容性矩阵](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_compatibilitymatrix)

从KIE 6.0开始，Drools（包括工作台），jBPM（包括设计器和控制台）和OptaPlanner都遵循相同的版本编号。

# KIE

KIE是Drools和jBPM的共享核心。它为构建，部署和利用资源提供了统一的方法和编程模型。

## [4. KIE](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiechapter)

### [4.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieoverviewsection)

#### [4.1.1。项目剖析](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieanatomysection)

研究Drools和jBPM集成知识解决方案的过程仅使用“kiegroup”组名称。这个名字渗透到GitHub帐户和Maven POM中。随着示波器范围的扩大和新项目的推出，KIE（知识就是一切的缩写）被选为新组名。KIE名称也用于系统的共享方面; 如统一建设，部署和利用。

KIE目前由以下子项目组成：

*图95. KIE解剖学*

OptaPlanner是一种本地搜索和优化工具，已从Drools Planner中分离出来，现在已成为Drools和jBPM的顶级项目。这是Optaplanner的自然演进，虽然Drools的整合力强劲，但它一直独立于Drools。

从Polymita收购以及其他方面来看，功能强大的Dashboard Builder提供了强大的报告功能。Dashboard Builder目前是一个临时名称，6.0版本之后会选择一个新名称。Dashboard Builder完全独立于Drools和jBPM，并将在JBoss的许多项目中使用，并希望在JBoss之外:)

UberFire是新的基础工作台项目，从头开始重写。UberFire提供类似Eclipse的工作台功能，以及来自插件的面板和页面。该项目独立于Drools和jBPM，任何人都可以使用它作为构建灵活和强大工作台的基础。UberFire将用于整个JBoss的控制台和工作台开发。

确定Guvnor品牌从其预定角色泄漏过多; 如创作隐喻，如决策表，被视为Guvnor组件而不是Drools组件。这对于Guvnor在5.x中使用的庞大的项目结构没有帮助。在6.0中，Guvnor的重点已经缩小到封装一组UberFire插件，这些插件为构建基于Web的IDE提供了基础。例如用于构建和部署的Maven集成，Maven存储库的管理以及通过收件箱进行的活动通知。Drools和jBPM使用Uberfire作为基础构建工作台发行版，并包含一系列插件，如Guvnor，以及自己的插件，如决策表，引导编辑器，BPMN2设计器，人工任务等。Drools工作台被称为Drools-WB。KIE-WB是结合了所有Guvnor，Drools和jBPM插件的超级工作台。jBPM-WB被虚拟出来，因为它实际上并不存在，被KIE-WB重复使用。

#### [4.1.2。生命周期](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kielifecyclessection)

与KIE系统一起工作的不同方面或生命周期，无论是Drools还是jBPM，通常可以细分为以下几种：

* **作者**
  + 使用UI隐喻编写知识，例如：DRL，BPMN2，决策表，类模型。
* **建立**
  + 将创作的知识构建到可部署的单元中。
  + 对于KIE这个单位是一个JAR。
* **测试**
  + 测试KIE在将其部署到应用程序之前知晓。
* **部署**
  + 将单元部署到应用程序可能使用（消耗）它们的位置。
  + KIE使用Maven样式库。
* **利用**
  + 加载JAR以提供应用程序可以与之交互的KIE会话（KieSession）。
  + KIE在运行时通过KIE容器（KieContainer）公开JAR。
  + KieSessions是运行时与KieContainer交互的产物。
* **跑**
  + 通过API与KieSession进行系统交互。
* **工作**
  + 用户通过命令行或UI与KieSession进行交互。
* **管理**
  + 管理任何KieSession或KieContainer。

### [4.2。构建，部署，利用和运行](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_builddeployutilizeandrunsection)

#### [4.2.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemoduleintroductionbuildingintroductionsection)

6.0引入了一种新的配置和约定方法来构建知识库，而不是在5.x中使用编程构建器方法。建筑商仍然可以重新使用，因为它用于工具集成。

Building现在使用Maven，并与Maven实践保持一致。KIE项目或模块只是一个Maven Java项目或模块; 还有一个额外的元数据文件META-INF / kmodule.xml。kmodule.xml文件是选择资源到知识库并配置这些知识库和会话的描述符。还有通过Spring和OSGi BluePrints提供的替代XML支持。

虽然标准Maven可以构建和打包KIE资源，但它不会在构建时提供验证。有一个Maven插件，推荐使用它来获取构建时间验证。该插件还生成许多类，使得运行时加载速度也更快。

截图中显示了示例项目布局和Maven POM描述符

*图96.示例项目布局和Maven POM*

KIE使用默认值来减少配置的数量。空kmodule.xml是最简单的配置。必须始终有一个kmodule.xml文件，即使它是空的，因为它用于发现JAR及其内容。

Maven可以'mvn install'将KieModule部署到本地机器上，本地机器上的所有其他应用程序都使用它。或者可以'部署'将KieModule推送到远程Maven存储库。构建应用程序将引入KieModule并在此过程中填充本地Maven存储库。

*图97.示例项目布局和Maven POM*

JAR可以通过两种方式进行部署。可以像Maven依赖列表中的任何其他JAR一样添加到类路径中，也可以在运行时动态加载它们。KIE将扫描类路径以找到所有带有kmodule.xml的JAR。每个找到的JAR都由KieModule接口表示。术语classpath KieModule和动态KieModule用于指代两种加载方法。虽然动态模块支持并排版本控制，但classpath模块不支持。进一步，一旦一个模块在类路径上，就不会动态加载其他版本。

接下来的部分将包括API的详细参考资料，不耐烦的人可以直接跳到示例部分，这在不同的用例中是非常明显的。

#### [4.2.2。建造](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebuildingsection)

*图98. org.kie.api.core.builder*

##### [4.2.2.1。创建和建立一个Kie项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creatingandbuildingakieproject)

Kie项目具有普通Maven项目的结构，唯一的特点是包含一个kmodule.xml文件，以声明的方式定义可以从中创建的KieBases和KieSessions。该文件必须放置在Maven项目的resources / META-INF文件夹中，而其他所有Kie工件（如DRL或Excel文件）必须存储在资源文件夹或其下的任何其他子文件夹中。

由于已经为所有配置方面提供了有意义的默认值，最简单的kmodule.xml文件可以包含一个空的kmodule标签，如下所示：

*例21.一个空的kmodule.xml文件*

**<?xml version="1.0" encoding="UTF-8"?>**

<kmodule xmlns="http://www.drools.org/xsd/kmodule"/>

这样kmodule将包含一个默认值KieBase。存储在资源文件夹或其任何子文件夹下的所有Kie资产将被编译并添加到其中。要触发这些构件的构建，就足以KieContainer为它们创建一个。

*图99. KieContainer*

对于这种简单的情况，创建一个KieContainer从classpath中读取要构建的文件就足够了：

*例22.从classpath创建一个KieContainer*

KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();

`KieServices`是可以访问Kie所有建筑和运行时设施的界面：

*图100. KieServices*

通过这种方式，所有的Java源代码和Kie资源都被编译并部署到KieContainer中，这使得它的内容可以在运行时使用。

##### [4.2.2.2。kmodule.xml文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_thekmodulexmlfile)

正如前一节所述，kmodule.xml文件是可以声明式配置可以从KIE项目创建的KieBase（s）和KieSession（s）的地方。

特别是a KieBase是所有应用程序知识定义的存储库。它将包含规则，流程，函数和类型模型。在KieBase本身不包含数据; 相反，会话是根据KieBase数据可以插入的以及可以从哪些流程实例开始创建的。创建KieBase可能很重，而会话创建非常轻，因此建议KieBase尽可能缓存以允许重复创建会话。但是最终用户通常不应该担心，因为这个缓存机制已经被自动提供了KieContainer。

*图101. KieBase*

相反，KieSession存储并在运行时数据上执行。它是从所生成的KieBase或更容易地能够直接创建从KieContainer它是否已在kmodule.xml文件中定义

*图102. KieSession*

kmodule.xml允许定义和配置一个或多个KieBases，以及可以从中创建的KieBase所有不同的KieSessions，如以下示例所示：

*示例23.一个示例kmodule.xml文件*

<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.drools.org/xsd/kmodule">

<configuration>

<property key="drools.evaluator.supersetOf" value="org.mycompany.SupersetOfEvaluatorDefinition"/>

</configuration>

<kbase name="KBase1" default="true" eventProcessingMode="cloud" equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">

<ksession name="KSession2\_1" type="stateful" default="true"/>

<ksession name="KSession2\_2" type="stateless" default="false" beliefSystem="jtms"/>

</kbase>

<kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">

<ksession name="KSession3\_1" type="stateful" default="false" clockType="realtime">

<fileLogger file="drools.log" threaded="true" interval="10"/>

<workItemHandlers>

<workItemHandler name="name" type="org.domain.WorkItemHandler"/>

</workItemHandlers>

<listeners>

<ruleRuntimeEventListener type="org.domain.RuleRuntimeListener"/>

<agendaEventListener type="org.domain.FirstAgendaListener"/>

<agendaEventListener type="org.domain.SecondAgendaListener"/>

<processEventListener type="org.domain.ProcessListener"/>

</listeners>

</ksession>

</kbase>

</kmodule>

这里标签包含一系列键值对，它们是用于配置KieBases构建过程的可选属性。例如，这个示例kmodule.xml文件定义了一个名为supersetOf并由org.mycompany.SupersetOfEvaluatorDefinition类实现的额外自定义运算符。

在这2 KieBases被定义之后，可以KieSession从第一个实例中获得2个不同类型的s，而从第二个中只有一个。可以在kbase标签上定义的属性列表及其含义和默认值如下所示：

| *表3. kbase属性* | | | |
| --- | --- | --- | --- |
| **属性名称** | **默认值** | **已接受的值** | **含义** |
| 名称 | 没有 | 任何 | 从KieContainer中检索这个KieBase的名字。这是唯一的强制性属性。 |
| 包括 | 没有 | 任何以逗号分隔的列表 | 此kmodule中包含的其他KieBase的逗号分隔列表。所有这些KieBases的文物也将包含在这一个中。 |
| 包 | 所有 | 任何以逗号分隔的列表 | 默认情况下，资源文件夹下的任何级别的所有Drools工件都包含在KieBase中。这个属性允许将在这个KieBase中编译的工件限制为只属于包列表的工件。 |
| 默认 | 假 | 真假 | 定义此KieBase是否为该模块的默认值，因此可以通过KieContainer创建而不需要传递任何名称。每个模块中最多可以有一个默认的KieBase。 |
| equalsBehavior | 身分 | 身份，平等 | 将新事实插入工作内存时定义Drools的行为。有了身份，它总是创建一个新的FactHandle，除非相同的对象不在工作内存中，而只有当新插入的对象与已有事实不相等（根据其等同的方法）时才相等。 |
| eventProcessingMode | 云 | 云，流 | 在云模式下编译时，KieBase将事件视为正常事实，而在流模式下允许对它们进行时间推理。 |
| declarativeAgenda | 残 | 禁用启用 | 定义是否启用声明式议程。 |

同样，ksession标签的所有属性（当然除名称）都具有有意义的默认值。它们在下表中列出并描述：

| *表4. ksession属性* | | | |
| --- | --- | --- | --- |
| **属性名称** | **默认值** | **已接受的值** | **含义** |
| 名称 | 没有 | 任何 | 这个KieSession的唯一名称。用于从KieContainer中获取KieSession。这是唯一的强制性属性。 |
| 类型 | 状态 | 有状态的，无状态的 | 有状态会话允许迭代地使用工作内存，而无状态会话则是使用提供的数据集一次性执行工作内存。 |
| 默认 | 假 | 真假 | 定义此KieSession是否为该模块的默认值，因此可以从KieContainer创建，而不需要传递任何名称。在每个模块中，每种类型最多只能有一个默认的KieSession。 |
| clockType | 即时的 | 实时，伪 | 定义事件时间戳是由系统时钟还是由应用程序控制的伪随机时钟确定的。这个时钟对单元测试时间规则特别有用。 |
| 信仰体系 | 简单 | 简单，jtms，可以废止 | 定义KieSession使用的信念系统的类型。 |

正如前面的kmodule.xml示例所述，还可以声明性地在每个KieSession文件（或控制台）记录器上创建一个或多个WorkItemHandlers和一些可以具有3种不同类型的侦听器：ruleRuntimeEventListener，agendaEventListener和processEventListener

在定义了一个kmodule.xml之后，现在可以使用它们的名称简单地从KieContainer中检索KieBases和KieSessions。

*例24.从KieContainer中回收KieBases和KieSessions*

KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();

KieBase kBase1 = kContainer.getKieBase("KBase1");

KieSession kieSession1 = kContainer.newKieSession("KSession2\_1");

StatelessKieSession kieSession2 = kContainer.newStatelessKieSession("KSession2\_2");

必须注意的是，由于KSession2\_1和KSession2\_2有两种不同的类型（第一种是有状态的，而第二种是无状态的），KieContainer根据它们的声明类型，有必要调用2种不同的方法。如果KieSession请求的类型KieContainer与在kmodule.xml文件中声明的类型不一致，KieContainer则会抛出一个RuntimeException。另外，由于a KieBase和a KieSession已被标记为默认值，因此可以在KieContainer不通过任何名称的情况下从中获取它们。

*例25.从KieContainer中恢复默认的KieBases和KieSessions*

KieContainer kContainer = ...

KieBase kBase1 = kContainer.getKieBase(); *// returns KBase1*

KieSession kieSession1 = kContainer.newKieSession(); *// returns KSession2\_1*

由于Kie项目也是Maven项目，因此在pom.xml文件中声明的groupId，artifactId和版本用于生成一个ReleaseId在应用程序内唯一标识此项目的项目。这允许通过简单地将其传递ReleaseId给项目来从项目中创建新的KieContainer KieServices。

*示例26.通过ReleaseId创建现有项目的KieContainer*

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "myartifact", "1.0" );

KieContainer kieContainer = kieServices.newKieContainer( releaseId );

##### [4.2.2.3。用Maven构建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_buildingwithmaven)

Maven的KIE插件可确保工件资源得到验证和预编译，建议始终使用此工具。要使用插件，只需将其添加到Maven pom.xml的构建部分，然后使用包装激活它kjar。

*示例27.将KIE插件添加到Maven pom.xml并激活它*

<packaging>kjar</packaging>

...

<build>

<plugins>

<plugin>

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<version>7.7.0.Final</version>

<extensions>true</extensions>

</plugin>

</plugins>

</build>

该插件支持所有Drools / jBPM知识资源。但是，如果您在Java类中使用特定的KIE批注，例如@kie.api.Position，您需要将编译时间依赖项添加kie-api到您的项目中。我们建议为所有其他KIE依赖项使用提供的范围。这样kjar保持尽可能的轻量级，而不依赖于任何特定的KIE版本。

不使用Maven插件构建KIE模块将会将所有资源原样复制到生成的JAR中。当JAR由运行时加载时，它将尝试构建所有资源。如果有编译问题，它将返回一个空KieContainer。它还会将编译开销推向运行时。一般来说这不被推荐，并且总是应该使用Maven插件。

##### [4.2.2.4。以编程方式定义KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_definingakiemoduleprogrammatically)

也可以以编程方式定义属于KieModule 的KieBases和KieSessions，而不是在kmodule.xml文件中声明性定义。相同的编程API也允许显式添加包含Kie工件的文件，而不是从项目的资源文件夹中自动读取它们。为此，有必要创建一种KieFileSystem虚拟文件系统，并将项目中包含的所有资源添加到其中。

*图103. KieFileSystem*

像所有其他的纪伊核心部件，你可以得到的一个实例KieFileSystem从KieServices。kmodule.xml配置文件必须添加到文件系统。这是一个强制性步骤。Kie还提供了一个方便流畅的API，由其执行KieModuleModel，以编程方式创建该文件。

*图104. KieModuleModel*

要在实践中做到这一点，有必要创建一个KieModuleModel从KieServices，配置所需的KieBases和KieSessions，将其转换为XML并将XML添加到KieFileSystem。以下示例显示了此过程：

*例28.以编程方式创建kmodule.xml并将其添加到KieFileSystem*

KieServices kieServices = KieServices.Factory.get();

KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel( "KBase1 ")

.setDefault( **true** )

.setEqualsBehavior( EqualityBehaviorOption.EQUALITY )

.setEventProcessingMode( EventProcessingOption.STREAM );

KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel( "KSession1" )

.setDefault( **true** )

.setType( KieSessionModel.KieSessionType.STATEFUL )

.setClockType( ClockTypeOption.get("realtime") );

KieFileSystem kfs = kieServices.newKieFileSystem();

kfs.writeKModuleXML(kieModuleModel.toXML());

此时，还需要KieFileSystem通过其流畅的API 添加组成项目的所有其他Kie工件。这些工件必须添加到相应的常用Maven项目的相同位置。

*例子29.将Kie工件添加到KieFileSystem*

KieFileSystem kfs = ...

kfs.write( "src/main/resources/KBase1/ruleSet1.drl", stringContainingAValidDRL )

.write( "src/main/resources/dtable.xls",

kieServices.getResources().newInputStreamResource( dtableFileStream ) );

这个例子表明可以将Kie工件作为普通字符串和Resources一起添加。在后一种情况下，Resource可以由KieResources工厂创建，也可以由工厂提供KieServices。在KieResources提供了许多便捷的工厂方法，将一个InputStream，一个URL，一个File，或String代表你的文件系统的路径Resource可以由管理KieFileSystem。

*图105. KieResources*

通常，Resource可以从用于将其添加到名称的扩展名中推断出a的类型KieFileSystem。但是，也可以不遵循关于文件扩展名的Kie约定，并明确指定一个具体ResourceType的a Resource，如下所示：

*示例30.使用显式类型创建和添加资源*

KieFileSystem kfs = ...

kfs.write( "src/main/resources/myDrl.txt",

kieServices.getResources().newInputStreamResource( drlStream )

.setResourceType(ResourceType.DRL) );

将所有资源添加到KieFileSystem并通过传递KieFileSystem给a来构建它KieBuilder

*图106. KieBuilder*

当KieFileSystem成功构建a的内容时，结果KieModule会自动添加到KieRepository。这KieRepository是一个单身作为所有可用KieModules 的存储库。

*图107. KieRepository*

在此之后，可以通过创建KieServices一个新的KieContainer为KieModule使用它的ReleaseId。但是，由于在这种情况下，KieFileSystem不包含任何pom.xml文件（可以使用该KieFileSystem.writePomXML方法添加一个文件），Kie无法确定ReleaseId该文件KieModule并将其分配为默认文件。这个默认值ReleaseId可以从中获得KieRepository并用来标识其KieModule内部KieRepository。以下示例显示了整个过程。

*示例31.构建KieFileSystem的内容并创建KieContainer*

KieServices kieServices = KieServices.Factory.get();

KieFileSystem kfs = ...

kieServices.newKieBuilder( kfs ).buildAll();

KieContainer kieContainer = kieServices.newKieContainer(kieServices.getRepository().getDefaultReleaseId());

在这一点上，可以KieBase通过与从类路径直接创建的方式完全相同的方式获取s并从中创建新的KieSessions 。KieContainerKieContainer

检查编译结果是一个最佳实践。在KieBuilder3个不同的严重性报道编译结果：ERROR，WARNING和INFO。ERROR表示编译项目失败，并且在没有KieModule产生的情况下，没有任何东西被添加到KieRepository。警告和INFO结果可以忽略，但可供检查。

*例32.检查一个编译没有产生任何错误*

KieBuilder kieBuilder = kieServices.newKieBuilder( kfs ).buildAll();

assertEquals( 0, kieBuilder.getResults().getMessages( Message.Level.ERROR ).size() );

##### [4.2.2.5。更改默认构建结果严重性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_changingthedefaultbuildresultseverity)

在某些情况下，可以更改构建结果类型的默认严重性。例如，如果将新规则与现有规则的名称相同添加到程序包中，则默认行为是将旧规则替换为新规则并将其报告为INFO。这对于大多数使用情况来说可能是理想的，但在某些部署中，用户可能想要防止规则更新并将其报告为错误。

可以通过API调用，系统属性或配置文件来更改结果类型的默认严重性，这与Drools中的其他选项一样。从此版本开始，Drools支持规则更新和功能更新的可配置结果严重性。要使用系统属性或配置文件进行配置，用户必须使用以下属性：

*示例33.使用属性设置严重性*

*// sets the severity of rule updates*

drools.kbuilder.severity.duplicateRule = <INFO|WARNING|ERROR>

*// sets the severity of function updates*

drools.kbuilder.severity.duplicateFunction = <INFO|WARNING|ERROR>

#### [4.2.3。部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiedeployingsection)

##### [4.2.3.1。KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase)

这KieBase是所有应用程序知识定义的存储库。它将包含规则，流程，函数和类型模型。在KieBase本身不包含数据; 相反，会话是根据KieBase数据可以插入的以及可以从哪些流程实例开始创建的。在KieBase可以从能够得到KieContainer含有KieModule的其中KieBase已被定义。

*图108. KieBase*

有时，例如在OSGi环境中，KieBase需要解析不在默认类加载器中的类型。在这种情况下，有必要创建一个KieBaseConfiguration附加的类加载器，并KieContainer在创建新的类时将其传递给KieBase它。

*例34.用一个自定义ClassLoader创建一个新的KieBase*

KieServices kieServices = KieServices.Factory.get();

KieBaseConfiguration kbaseConf = kieServices.newKieBaseConfiguration( **null**, MyType.class.getClassLoader() );

KieBase kbase = kieContainer.newKieBase( kbaseConf );

##### [4.2.3.2。KieSessions和KieBase修改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesessions_and_kiebase_modifications)

KieSessions将在“Running”一节中更详细地讨论。在KieBase创建并返回KieSession的对象，它可以选择保留这些引用。当KieBase修饰发生的那些修改应用于针对在会话中的数据。该引用是一个弱引用，它也是可选的，由布尔标志控制。

##### [4.2.3.3。KieScanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiescanner_2)

将KieScanner让你的Maven仓库的连续监测，检查项目乃纪伊的新版本是否已安装。在KieContainer包装该项目中部署新版本。KieScanner要求使用kie-ci.jar在类路径上。

*图109. KieScanner*

A KieScanner可以KieContainer在下面的例子中注册。

*示例35.在KieContainer上注册并启动KieScanner*

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "myartifact", "1.0-SNAPSHOT" );

KieContainer kContainer = kieServices.newKieContainer( releaseId );

KieScanner kScanner = kieServices.newKieScanner( kContainer );

*// Start the KieScanner polling the Maven repository every 10 seconds*

kScanner.start( 10000L );

在这个例子中，它KieScanner被配置为以固定的时间间隔运行，但是也可以通过调用scanNow()它的方法按需运行它。如果KieScanner在Maven存储库中找到Kie项目所使用的更新版本，KieContainer它将自动下载新版本并触发新项目的增量构建。在这一点上，现有的KieBaseS和KieSession控制项下的KieContainer会得到它自动升级-特别是那些KieBase具有s的粘度getKieBase()与它们相关的一起KieSession秒，任何KieSession直接获得KieContainer.newKieSession()这样引用默认KieBase。此外，从这一刻起，所有新KieBase的和KieSession创造出来的KieContainer将使用新的项目版本。但请注意KieBase，通过newKieBase()KieScanner升级之前获得的任何现有内容及其任何相关KieSession内容不会自动升级; 这是因为KieBase经由newKieBase()过程获得的不是直接控制的KieContainer。

将KieScanner只会如果使用快照，版本范围，最新的，或松开设置拾音器更改已部署的罐子。固定版本不会在运行时自动更新。

##### [4.2.3.4。Maven版本和依赖项](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_versions_and_dependencies)

Maven支持许多机制来管理应用程序中的版本控制和依赖关系。可以使用特定版本号发布模块，也可以使用SNAPSHOT后缀。依赖关系可以指定要使用的版本范围，或利用SNAPSHOT机制。

StackOverflow为此提供了一个非常好的描述，这在下面进行了重现。

<http://stackoverflow.com/questions/30571/how-do-i-tell-maven-to-use-the-latest-version-of-a-dependency>

|  |  |
| --- | --- |
|  | 由于Maven 3.x metaversions RELEASE和LATEST不再支持可重复构建。 |

有关更多详细信息，请参阅Maven书籍的POM语法部分。

<http://books.sonatype.com/mvnref-book/reference/pom-relationships-sect-pom-syntax.html>

<http://books.sonatype.com/mvnref-book/reference/pom-relationships-sect-project-dependencies.html>

以下是一个说明各种选项的例子。在Maven仓库中，com.foo：my-foo具有以下元数据：

<metadata>

<groupId>com.foo</groupId>

<artifactId>my-foo</artifactId>

<version>2.0.0</version>

<versioning>

<release>1.1.1</release>

<versions>

<version>1.0</version>

<version>1.0.1</version>

<version>1.1</version>

<version>1.1.1</version>

<version>2.0.0</version>

</versions>

<lastUpdated>20090722140000</lastUpdated>

</versioning>

</metadata>

如果需要对该工件的依赖关系，则可以使用以下选项（当然，也可以指定其他版本范围，只是在此显示相关的版本）：声明确切版本（将始终解析为1.0.1）：

<version>[1.0.1]</version>

声明一个明确的版本（除非发生冲突，否则当Maven将选择一个匹配的版本时，将永远解析为1.0.1）：

<version>1.0.1</version>

声明所有1.x的版本范围（目前将解析为1.1.1）：

<version>[1.0.0,2.0.0)</version>

声明开放式版本范围（将解析为2.0.0）：

<version>[1.0.0,)</version>

声明版本为LATEST（将解析为2.0.0）：

<version>LATEST</version>

声明版本为RELEASE（将解析为1.1.1）：

<version>RELEASE</version>

请注意，默认情况下，您自己的部署将更新Maven元数据中的“latest”条目，但要更新“release”条目，您需要激活Maven super POM中的“release-profile”。您可以使用“-Prelease-profile”或“-DperformRelease = true”

##### [4.2.3.5。Settings.xml和远程存储库设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_settings_xml_and_remote_repository_setup)

maven settings.xml用于配置Maven执行。详细的说明可以在Maven网站上找到：

<http://maven.apache.org/settings.html>

settings.xml文件可以位于3个位置，实际使用的设置是这3个位置的合并。

* Maven安装： $M2\_HOME/conf/settings.xml
* 用户的安装： ${user.home}/.m2/settings.xml
* 由系统属性指定的文件夹位置 kie.maven.settings.custom

settings.xml用于指定远程存储库的位置。激活指定远程存储库的配置文件非常重要，通常可以使用“activeByDefault”来完成：

<profiles>

<profile>

<id>profile-1</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

...

</profile>

</profiles>

Maven提供了有关使用多个远程存储库的详细文档：

<http://maven.apache.org/guides/mini/guide-multiple-repositories.html>

#### [4.2.4。运行](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kierunningsection)

##### [4.2.4.1。KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_2)

这KieBase是所有应用程序知识定义的存储库。它将包含规则，流程，函数和类型模型。在KieBase本身不包含数据; 相反，会话是根据KieBase数据可以插入的以及可以从哪些流程实例开始创建的。在KieBase可以从能够得到KieContainer含有KieModule的其中KieBase已被定义。

*示例36.从KieContainer获取KieBase*

KieBase kBase = kContainer.getKieBase();

##### [4.2.4.2。KieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesession)

在KieSession对运行时数据存储和执行。它是从创建KieBase。

*图110. KieSession*

*例子37.从KieBase创建一个KieSession*

KieSession ksession = kbase.newKieSession();

##### [4.2.4.3。KieRuntime](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieruntime)

###### [KieRuntime](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieruntime_2)

的KieRuntime规定，均适用于规则和过程的方法，如设置全局变量和注册信道。（“退出点”是“频道”的一个过时的同义词。）

*图111. KieRuntime*

[全局](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_globals)

全局变量是规则引擎可见的命名对象，但其方式与事实根本不同：支持全局的对象中的更改不会触发对规则的重新评估。尽管如此，全局变量对于提供静态信息很有用，作为提供在规则的RHS中使用的服务的对象，或者作为从规则引擎返回对象的手段。当您在规则的LHS中使用全局时，请确保它是不可变的，或者至少不要期望更改对规则的行为有任何影响。

全局必须在规则文件中声明，然后它需要用Java对象进行备份。

**global** java.util.**List** **list**

知识库现在知道全局标识符及其类型，现在可以ksession.setGlobal()使用全局名称和对象来调用任何会话，以将该对象与全局关联。未能在DRL代码中声明全局类型和标识符将导致此调用引发异常。

List list = **new** ArrayList();

ksession.setGlobal("list", list);

确保在用于评估规则之前设置全局。不这样做会导致a NullPointerException。

##### [4.2.4.4。事件模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_model)

事件包提供了通知规则引擎事件的方法，包括规则触发，被声明的对象等。这允许将日志记录和审计活动从应用程序的主体部分（和规则）中分离出来。

该KieRuntimeEventManager接口由KieRuntime提供两个接口实现，RuleRuntimeEventManager并且ProcessEventManager。我们只会在RuleRuntimeEventManager这里覆盖。

*图112. KieRuntimeEventManager*

在RuleRuntimeEventManager允许添加和删除侦听程序，因此，对于工作记忆和议程事件就可以听了。

*图113. RuleRuntimeEventManager*

以下代码片段显示了一个简单的日程监听器是如何声明并附加到会话中的。它会在发射后打印匹配。

*例子38.添加一个AgendaEventListener*

ksession.addEventListener( **new** DefaultAgendaEventListener() {

**public** **void** **afterMatchFired**(AfterMatchFiredEvent event) {

**super**.afterMatchFired( event );

System.out.println( event );

}

});

Drools还提供DebugRuleRuntimeEventListener并DebugAgendaEventListener使用调试打印语句实现每种方法。要打印所有工作记忆事件，请添加如下所示的侦听器：

*例子39.添加一个DebugRuleRuntimeEventListener*

ksession.addEventListener( **new** DebugRuleRuntimeEventListener() );

所有发出的事件都实现了KieRuntimeEvent可用于检索KnowlegeRuntime源自事件的实际接口。

*图114. KieRuntimeEvent*

目前支持的事件有：

* MatchCreatedEvent
* MatchCancelledEvent
* BeforeMatchFiredEvent
* AfterMatchFiredEvent
* AgendaGroupPushedEvent
* AgendaGroupPoppedEvent
* ObjectInsertEvent
* ObjectDeletedEvent
* ObjectUpdatedEvent
* ProcessCompletedEvent
* ProcessNodeLeftEvent
* ProcessNodeTriggeredEvent
* ProcessStartEvent

##### [4.2.4.5。KieRuntimeLogger](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieruntimelogger)

KieRuntimeLogger使用Drools中的综合事件系统创建一个审计日志，可以使用诸如Eclipse审计查看器之类的工具来记录应用程序的执行情况以供日后检查。

*图115. KieLoggers*

*例子40. FileLogger*

KieRuntimeLogger logger =

KieServices.Factory.get().getLoggers().newFileLogger(ksession, "logdir/mylogfile");

...

logger.close();

##### [4.2.4.6。命令和CommandExecutor](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commands_and_the_commandexecutor)

KIE具有有状态或无状态会话的概念。已经涵盖了有状态的会话，它们使用标准的KieRuntime，并且可以随着时间的推移迭代工作。无状态是通过提供的数据集一次性执行KieRuntime。它可能会返回一些结果，会话被放置在最后，禁止进一步的迭代交互。您可以将无状态视为可视返回结果的函数调用引擎。

其基础是CommandExecutor有状态和无状态接口扩展的接口。这返回一个ExecutionResults：

*图116. CommandExecutor*

*图117.执行结果*

在CommandExecutor允许在这些会话要被执行的命令，唯一的区别在于StatelessKieSession执行fireAllRules()在端设置会话之前。这些命令可以使用CommandExecutor。创建。Javadocs 使用。提供允许的命令的完整列表CommandExecutor。

setGlobal和getGlobal是与Drools和jBPM相关的两个命令。

在下面设置全局调用setGlobal。可选的布尔值指示命令是否应该返回全局值作为的一部分ExecutionResults。如果为true，则使用与全局名称相同的名称。如果需要替代名称，则可以使用字符串代替布尔值。

*例子41.设置全局命令*

StatelessKieSession ksession = kbase.newStatelessKieSession();

ExecutionResults bresults =

ksession.execute( CommandFactory.newSetGlobal( "stilton", **new** Cheese( "stilton" ), **true**);

Cheese stilton = bresults.getValue( "stilton" );

允许返回现有的全局。第二个可选的String参数允许使用另一个返回名称。

*例子42.获取全局命令*

StatelessKieSession ksession = kbase.newStatelessKieSession();

ExecutionResults bresults =

ksession.execute( CommandFactory.getGlobal( "stilton" );

Cheese stilton = bresults.getValue( "stilton" );

以上所有示例都执行单个命令。该BatchExecution代表一个复合命令，由一系列命令创建。它将遍历列表并依次执行每个命令。这意味着你可以插入一些对象，启动一个进程，调用fireAllRules并执行一个查询，所有这些都是在一次execute(…​)调用中完成的，这个功能非常强大。

StatelessKieSession将fireAllRules()在最后自动执行。然而，敏锐的读者可能已经注意到了这个FireAllRules命令，并想知道如何与无状态的KieSession一起工作。该FireAllRules命令是允许的，使用它将在最后禁用自动执行; 可以考虑使用它作为一种手动覆盖功能。

批处理中的任何具有设置了标识符的命令都会将其结果添加到返回的ExecutionResults实例中。我们来看一个简单的例子，看看它是如何工作的。为了说明起见，提供的示例包括来自Drools和jBPM的命令。它们在Drool和jBPM特定章节中有更详细的介绍。

*例43.批量执行命令*

StatelessKieSession ksession = kbase.newStatelessKieSession();

List cmds = **new** ArrayList();

cmds.add( CommandFactory.newInsertObject( **new** Cheese( "stilton", 1), "stilton") );

cmds.add( CommandFactory.newStartProcess( "process cheeses" ) );

cmds.add( CommandFactory.newQuery( "cheeses" ) );

ExecutionResults bresults = ksession.execute( CommandFactory.newBatchExecution( cmds ) );

Cheese stilton = ( Cheese ) bresults.getValue( "stilton" );

QueryResults qresults = ( QueryResults ) bresults.getValue( "cheeses" );

在上面的例子中，执行了多个命令，其中两个填充了ExecutionResults。查询命令默认使用与查询名称相同的标识符，但它也可以映射到不同的标识符。

所有命令都支持使用XStream的XML和jSON编组以及JAXB编组。这部分在[命令API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commandsapisection)部分中介绍。

##### [4.2.4.7。StatelessKieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_statelesskiesession)

该StatelessKieSession包装的KieSession，而不是扩展它。其主要重点在于决策服务类型场景。它避免了需要打电话dispose()。无状态会话不支持迭代插入和fireAllRules()来自Java代码的方法调用; 调用行为execute()是一种单步方法，它将在内部实例化一个KieSession，添加所有用户数据并执行用户命令，调用fireAllRules()，然后调用dispose()。尽管使用这个类的主要方法是通过BatchExecution接口Command支持的（子接口）CommandExecutor，但是当需要简单对象插入时提供了两种便利方法。该CommandExecutor和BatchExecution在自己的部分谈到细节。

*图118.无状态KieSession*

我们简单的例子展示了一个使用便利API执行给定的Java对象集合的无状态会话。它将迭代集合，依次插入每个元素。

*示例44.使用集合执行简单的StatelessKieSession*

StatelessKieSession ksession = kbase.newStatelessKieSession();

ksession.execute( collection );

如果这是作为一个命令完成的，那么它将如下所示：

*示例45.使用InsertElements命令执行简单的StatelessKieSession*

ksession.execute( CommandFactory.newInsertElements( collection ) );

如果你想插入集合本身和集合的个别元素，那么CommandFactory.newInsert(collection)就可以完成这项工作。

的方法CommandFactory创建支持的命令，所有这些都可以使用XStream的和被整理BatchExecutionHelper。BatchExecutionHelper提供了有关XML格式的详细信息，以及如何使用Drools的管道自动化的编组BatchExecution和ExecutionResults。

StatelessKieSession支持全局变量，范围有很多种。我们首先覆盖了非命令方式，因为命令被限定为特定的执行调用。全局可以通过三种方式解决。

* StatelessKieSession方法getGlobals()返回一个Globals实例，它提供对会话全局的访问。这些是所有执行调用共享的。请谨慎使用可变全局变量，因为执行调用可以在不同的线程中同时执行。

*示例46.会话范围全局*

StatelessKieSession ksession = kbase.newStatelessKieSession();

*// Set a global hbnSession, that can be used for DB interactions in the rules.*

ksession.setGlobal( "hbnSession", hibernateSession );

*// Execute while being able to resolve the "hbnSession" identifier.*

ksession.execute( collection );

* 使用代表是另一种全球解决方式。为全局值（赋值）赋值setGlobal(String, Object)，将值存储在内部集合中，将标识符映射到值。此内部集合中的标识符将优先于任何提供的代表。只有在此内部集合中找不到标识符时，才会使用全局代理（如果有）。
* 解决全局性的第三种方法是执行范围全局。在这里，一个Command设置全局是传递给CommandExecutor。

该CommandExecutor界面还提供了通过“输出”参数输出数据的功能。插入的事实，全局变量和查询结果都可以返回。

*例子47.输出标识符*

*// Set up a list of commands*

List cmds = **new** ArrayList();

cmds.add( CommandFactory.newSetGlobal( "list1", **new** ArrayList(), **true** ) );

cmds.add( CommandFactory.newInsert( **new** Person( "jon", 102 ), "person" ) );

cmds.add( CommandFactory.newQuery( "Get People" "getPeople" );

*// Execute the list*

ExecutionResults results =

ksession.execute( CommandFactory.newBatchExecution( cmds ) );

*// Retrieve the ArrayList*

results.getValue( "list1" );

*// Retrieve the inserted Person fact*

results.getValue( "person" );

*// Retrieve the query as a QueryResults instance.*

results.getValue( "Get People" );

##### [4.2.4.8。编组](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_marshalling)

将KieMarshallers用于编组和取消编组KieSessions。

*图119. KieMarshallers*

KieMarshallers可以从中检索一个实例KieServices。一个简单的例子如下所示：

*例子48.简单的Marshaller例子*

*// ksession is the KieSession*

*// kbase is the KieBase*

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

Marshaller marshaller = KieServices.Factory.get().getMarshallers().newMarshaller( kbase );

marshaller.marshall( baos, ksession );

baos.close();

但是，使用编组时，在处理引用的用户数据时需要更多的灵活性。要实现这个使用ObjectMarshallingStrategy界面。提供了两个实现，但用户可以实现自己的实现。这两个提供的策略是IdentityMarshallingStrategy和SerializeMarshallingStrategy。SerializeMarshallingStrategy是默认的，如上例所示，它只是调用用户实例上的Serializableor Externalizable方法。IdentityMarshallingStrategy为每个用户对象创建一个整数ID并将它们存储在Map中，同时将该ID写入流中。当解组时它访问IdentityMarshallingStrategy地图以检索实例。这意味着如果你使用IdentityMarshallingStrategy它对Marshaller实例的生命是有状态的，并且将创建id并保持对它尝试编组的所有对象的引用。以下是使用身份编组策略的代码。

*例子49. IdentityMarshallingStrategy*

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

KieMarshallers kMarshallers = KieServices.Factory.get().getMarshallers()

ObjectMarshallingStrategy oms = kMarshallers.newIdentityMarshallingStrategy()

Marshaller marshaller =

kMarshallers.newMarshaller( kbase, **new** ObjectMarshallingStrategy[]{ oms } );

marshaller.marshall( baos, ksession );

baos.close();

在大多数情况下，单一策略是不够的。为了增加灵活性，ObjectMarshallingStrategyAcceptor可以使用该接口。这个Marshaller有一系列的策略，在阅读或编写用户对象时，会迭代策略，询问他们是否接受编组用户对象的责任。提供的实现之一是ClassFilterAcceptor。这允许字符串和通配符用于匹配类名称。默认值是“ **。** ”，所以在上面的例子中，使用了身份编组策略，它有一个默认的“ **。** ”接受器。

假设我们想要序列化除一个给定包之外的所有类，我们将使用标识查找，我们可以执行以下操作：

*示例50.使用Acceptor的IdentityMarshallingStrategy*

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

KieMarshallers kMarshallers = KieServices.Factory.get().getMarshallers()

ObjectMarshallingStrategyAcceptor identityAcceptor =

kMarshallers.newClassFilterAcceptor( **new** String[] { "org.domain.pkg1.\*" } );

ObjectMarshallingStrategy identityStrategy =

kMarshallers.newIdentityMarshallingStrategy( identityAcceptor );

ObjectMarshallingStrategy sms = kMarshallers.newSerializeMarshallingStrategy();

Marshaller marshaller =

kMarshallers.newMarshaller( kbase,

**new** ObjectMarshallingStrategy[]{ identityStrategy, sms } );

marshaller.marshall( baos, ksession );

baos.close();

请注意，接受检查顺序是按提供的元素的自然顺序排列的。

另外请注意，如果您使用的是预定匹配（即您的某些规则使用计时器或日历），则只有在使用它之前，您可以将KieSession配置为使用可跟踪计时器作业工厂管理器，如下所示：

*示例51.配置可追踪的定时器作业工厂管理器*

KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();

ksconf.setOption(TimerJobFactoryOption.get("trackable"));

KSession ksession = kbase.newKieSession(ksconf, **null**);

##### [4.2.4.9。持久性和交易](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_persistence_and_transactions)

使用Drools可以实现Java Persistence API（JPA）的持久性。有必要安装一些Java事务API（JTA）的实现。出于开发目的，建议使用Bitronix事务管理器，因为建立和运行嵌入式系统很简单，但建议使用JBoss事务处理。

*例52.使用事务的简单例子*

KieServices kieServices = KieServices.Factory.get();

Environment env = kieServices.newEnvironment();

env.set( EnvironmentName.ENTITY\_MANAGER\_FACTORY,

Persistence.createEntityManagerFactory( "emf-name" ) );

env.set( EnvironmentName.TRANSACTION\_MANAGER,

TransactionManagerServices.getTransactionManager() );

*// KieSessionConfiguration may be null, and a default will be used*

KieSession ksession =

kieServices.getStoreServices().newKieSession( kbase, **null**, env );

**int** sessionId = ksession.getId();

UserTransaction ut =

(UserTransaction) **new** InitialContext().lookup( "java:comp/UserTransaction" );

ut.begin();

ksession.insert( data1 );

ksession.insert( data2 );

ksession.startProcess( "process1" );

ut.commit();

要使用JPA，环境必须同时设置EntityManagerFactory和TransactionManager。如果发生回滚，则ksession状态也会回滚，因此可能会在回滚后继续使用它。要加载先前持久化的KieSession，你需要这个id，如下所示：

*例子53.加载KieSession*

KieSession ksession =

kieServices.getStoreServices().loadKieSession( sessionId, kbase, **null**, env );

要启用持久性，必须将几个类添加到persistence.xml中，如下例所示：

*示例54.配置JPA*

<persistence-unit name="org.drools.persistence.jpa" transaction-type="JTA">

<provider>org.hibernate.ejb.HibernatePersistence</provider>

<jta-data-source>jdbc/BitronixJTADataSource</jta-data-source>

<class>org.drools.persistence.info.SessionInfo</class>

<class>org.drools.persistence.info.WorkItemInfo</class>

<properties>

<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>

<property name="hibernate.max\_fetch\_depth" value="3"/>

<property name="hibernate.hbm2ddl.auto" value="update" />

<property name="hibernate.show\_sql" value="true" />

<property name="hibernate.transaction.manager\_lookup\_class"

value="org.hibernate.transaction.BTMTransactionManagerLookup" />

</properties>

</persistence-unit>

jdbc JTA数据源必须先配置好。Bitronix提供了多种方法来完成此操作，并且应该查阅其文档以获取详细信息。为了快速入门，以下是程序化方法：

*示例55.配置JTA数据源*

PoolingDataSource ds = **new** PoolingDataSource();

ds.setUniqueName( "jdbc/BitronixJTADataSource" );

ds.setClassName( "org.h2.jdbcx.JdbcDataSource" );

ds.setMaxPoolSize( 3 );

ds.setAllowLocalTransactions( **true** );

ds.getDriverProperties().put( "user", "sa" );

ds.getDriverProperties().put( "password", "sasa" );

ds.getDriverProperties().put( "URL", "jdbc:h2:mem:mydb" );

ds.init();

Bitronix还提供了一个简单的嵌入式JNDI服务，非常适合测试。要使用它，请在您的META-INF文件夹中添加一个jndi.properties文件，并在其中添加以下行：

*例子56. JNDI属性*

java.naming.factory.initial=bitronix.tm.jndi.BitronixInitialContextFactory

#### [4.2.5。安装和部署备忘单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiedeploycheatsheets)

*图120.安装概述*

*图121.部署概述*

#### [4.2.6。构建，部署和利用示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieexamplessection)

学习新构建系统的最好方法就是举例。源项目“drools-examples-api”包含许多示例，可以在GitHub上找到：

<https://github.com/kiegroup/drools/tree/6.0.x/drools-examples-api>

下面介绍每个示例，顺序以最简单的方式（大多数选项都是默认的）开始，然后逐步实现更复杂的用例。

下面显示的部署用例都涉及到mvn install。在Maven中远程部署JAR在Maven文献中有很好的介绍。利用指的是加载资源并提供对KIE运行时的访问的初始行为。“运行”指的是与这些运行时进行交互的行为。

##### [4.2.6.1。默认KieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_default_kiesession)

* 项目：default-kesession。
* 摘要：在包含单个默认KieBase中的所有资源的类路径中，清空kmodule.xml KieModule。该示例显示从类路径中检索默认的KieSession。

一个空的kmodule.xml将生成一个包含在资源路径下找到的所有文件的单个KieBase，无论是DRL，BPMN2，XLS等。该单个KieBase是默认的，并且还包括一个默认的KieSession。默认意味着可以在不知道名称的情况下创建它们。

*例57. Author - kmodule.xml*

<kmodule xmlns="http://www.drools.org/xsd/kmodule"> </kmodule>

*例58.构建和安装 - Maven*

**mvn** install

ks.getKieClasspathContainer（）返回包含部署到环境类路径上的KieBase的KieContainer。kContainer.newKieSession（）创建默认的KieSession。请注意，您不再需要查找KieBase，以创建KieSession。KieSession知道它与哪个KieBase相关联，并使用它，在这种情况下，它是默认的KieBase。

*示例59.使用并运行Java*

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession();

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

##### [4.2.6.2。命名为KieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_named_kiesession)

* 项目：named-kiesession。
* 总结：kmodule.xml有一个名为KieBase和一个名为KieSession。这些示例显示了从类路径中检索指定的KieSession。

kmodule.xml将生成一个名为KieBase的'kbase1'，其中包含在资源路径下找到的所有文件，例如DRL，BPMN2，XLS等。KieSession的'ksession1'与该KieBase相关联，可以通过名称创建。

*例子60.作者 - kmodule.xml*

<kmodule xmlns="http://www.drools.org/xsd/kmodule">

<kbase name="kbase1">

<ksession name="ksession1"/>

</kbase>

</kmodule>

*例子61.构建和安装 - Maven*

**mvn** install

ks.getKieClasspathContainer（）返回包含部署到环境类路径上的KieBase的KieContainer。这次KieSession使用名称'ksession1'。你不需要首先查找KieBase，因为它知道哪个KieBase的ksession1与之相关联。

*例62.使用和运行 - Java*

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

##### [4.2.6.3。KieBase继承](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_inheritence)

* 项目：基地包容。
* 总结：'kmodule.xml'演示了一个KieBase可以包含来自另一个KieModule的另一个KieBase的资源。在这种情况下，它从'name-kiesession'示例中继承了名为KieBase。包含的KieBase可以来自当前的KieModule或pom.xml依赖列表中的任何其他KieModule。

kmodule.xml将生成一个名为KieBase的'kbase2'，其中包含在资源路径下找到的所有文件，无论是DRL，BPMN2，XLS等。此外，它将包括从KieBase'kbase1'中找到的所有资源，由于使用'includes'属性。KieSession'ksession2'与该KieBase相关联，可以通过名称创建。

*例63. Author - kmodule.xml*

<kbase name="kbase2" includes="kbase1">

<ksession name="ksession2"/>

</kbase>

这个例子要求前面的例子'named-kiesession'被构建并首先安装到本地Maven仓库。使用标准的Maven <dependencies>元素安装后，可以将其作为依赖项包含在内。

*例子64. Author - pom.xml*

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.drools</groupId>

<artifactId>drools-examples-api</artifactId>

<version>6.0.0/version>

</parent>

<artifactId>kiebase-inclusion</artifactId>

<name>Drools API examples - KieBase Inclusion</name>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-compiler</artifactId>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>named-kiesession</artifactId>

<version>6.0.0</version>

</dependency>

</dependencies>

</project>

一旦建立并安装'named-kiesession'，这个例子就可以像平常一样建立和安装。再次安装，将强制单元测试运行，展示用例。

*例65.构建和安装 - Maven*

**mvn** install

ks.getKieClasspathContainer（）返回包含部署到环境类路径上的KieBase的KieContainer。这次KieSession使用名称'ksession2'。你不需要首先查找KieBase，因为它知道哪个KieBase的ksession1与之相关联。注意这次有两条规则被激发，表明KieBase'kbase2'包含了来自依赖KieBase'kbase1'的资源。

*例66.利用和运行 - Java*

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession2");

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

kSession.insert(**new** Message("Dave", "Open the pod bay doors, HAL."));

kSession.fireAllRules();

##### [4.2.6.4。多个KieBases](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_multiple_kiebases)

* 项目：'多重-koases。
* 简介：演示'kmodule.xml'可以包含任意数量的KieBase或KieSession声明。引入'packages'属性来选择要包含在KieBase中的资源的文件夹。

kmodule.xml生成6个不同的名为KieBases。'kbase1'包含来自KieModule的所有资源。其他KieBases通过'packages'属性包含来自其他选定文件夹的资源。请注意使用通配符'\*'，以选择此包和它下面的所有包。

*例67. Author - kmodule.xml*

<kmodule xmlns="http://www.drools.org/xsd/kmodule">

<kbase name="kbase1">

<ksession name="ksession1"/>

</kbase>

<kbase name="kbase2" packages="org.some.pkg">

<ksession name="ksession2"/>

</kbase>

<kbase name="kbase3" includes="kbase2" packages="org.some.pkg2">

<ksession name="ksession3"/>

</kbase>

<kbase name="kbase4" packages="org.some.pkg, org.other.pkg">

<ksession name="ksession4"/>

</kbase>

<kbase name="kbase5" packages="org.\*">

<ksession name="ksession5"/>

</kbase>

<kbase name="kbase6" packages="org.some.\*">

<ksession name="ksession6"/>

</kbase>

</kmodule>

*例68.构建和安装 - Maven*

**mvn** install

下面只包含了一部分示例，因为每个KieSession都有一个测试方法，但每个都是另一个的重复，具有不同的列表期望。

*例69.使用和运行 - Java*

@Test

**public** **void** **testSimpleKieBase**() {

List<Integer> list = useKieSession("ksession1");

*// no packages imported means import everything*

assertEquals(4, list.size());

assertTrue( list.containsAll( asList(0, 1, 2, 3) ) );

}

*//.. other tests for ksession2 to ksession6 here*

**private** List<Integer> **useKieSession**(String name) {

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession(name);

List<Integer> list = **new** ArrayList<Integer>();

kSession.setGlobal("list", list);

kSession.insert(1);

kSession.fireAllRules();

**return** list;

}

##### [4.2.6.5。来自KieRepository的KieContainer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiecontainer_from_kierepository)

* 项目：kcontainer-from-repository
* 总结：该项目不包含kmodule.xml，pom.xml也没有其他KieModules的依赖关系。相反，Java代码演示了从Maven存储库加载动态KieModule。

pom.xml必须包含kie-ci作为depression，以确保Maven在运行时可用。因为这在底层使用了Maven，所以你也可以使用标准的Maven settings.xml文件。

*例子70.作者 - pom.xml*

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.drools</groupId>

<artifactId>drools-examples-api</artifactId>

<version>6.0.0</version>

</parent>

<artifactId>kiecontainer-from-kierepo</artifactId>

<name>Drools API examples - KieContainer from KieRepo</name>

<dependencies>

<dependency>

<groupId>org.kie</groupId>

<artifactId>kie-ci</artifactId>

</dependency>

</dependencies>

</project>

*例71.构建和安装 - Maven*

**mvn** install

在前面的例子中使用了类路径KieContainer。这个例子创建了一个由ReleaseId指定的动态KieContainer。ReleaseId对组标识，工件标识和版本使用Maven约定。它也服从版本的LATEST和SNAPSHOT。

*例72.使用和运行 - Java*

KieServices ks = KieServices.Factory.get();

*// Install example1 in the local Maven repo before to do this*

KieContainer kContainer = ks.newKieContainer(ks.newReleaseId("org.drools", "named-kiesession", "6.0.0-SNAPSHOT"));

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

##### [4.2.6.6。来自File的默认KieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_default_kiesession_from_file)

* 项目：default-kiesession-from-file
* 简介：动态KieModules也可以从任何资源位置加载。加载的KieModule提供了默认的KieBase和KieSession定义。

不存在kmodue.xml文件。必须首先构建项目“default-kiesession”，以便目标文件夹中生成的JAR可以作为文件引用。

*例73.构建和安装 - Maven*

**mvn** install

任何KieModule都可以从资源位置加载并添加到KieRepository中。一旦部署在KieRepository中，它可以通过它的ReleaseId来解决。请注意在这里不需要Maven或kie-ci。它不会设置传递依赖父类加载器。

*例74.利用并运行 - Java*

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

KieModule kModule = kr.addKieModule(ks.getResources().newFileSystemResource(getFile("default-kiesession")));

KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());

KieSession kSession = kContainer.newKieSession();

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

##### [4.2.6.7。从File命名KieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_named_kiesession_from_file)

* 项目：named-kiesession-from-file
* 简介：动态KieModules也可以从任何资源位置加载。加载的KieModule提供了命名的KieBase和KieSession定义。

不存在kmodue.xml文件。必须首先构建项目'named-kiesession'，以便将目标文件夹中生成的JAR引用为文件。

*例子75.构建和安装 - Maven*

**mvn** install

任何KieModule都可以从资源位置加载并添加到KieRepository中。一旦进入KieRepository，它可以通过它的ReleaseId来解决。请注意在这里不需要Maven或kie-ci。它不会设置传递依赖父类加载器。

*例76.利用和运行 - Java*

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

KieModule kModule = kr.addKieModule(ks.getResources().newFileSystemResource(getFile("named-kiesession")));

KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

##### [4.2.6.8。具有独立KieModule的KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodule_with_dependent_kiemodule)

* 项目：kie-module-form-multiple-files
* 简介：以编程方式提供依赖KieModules的列表，而不使用Maven来解决任何问题。

不存在kmodue.xml文件。必须首先构建项目'named-kiesession'和'kiebase-include'，以便将目标文件夹中生成的JAR引用为文件。

*例77.构建和安装 - Maven*

**mvn** install

创建两个资源。一个是主要的KieModule'exRes1'，另一个是依赖'exRes2'。即使kie-ci不存在，因此Maven无法解决依赖关系，这显示了如何为可变参数手动指定相关的KieModules。

*例78.利用和运行 - Java*

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

Resource ex1Res = ks.getResources().newFileSystemResource(getFile("kiebase-inclusion"));

Resource ex2Res = ks.getResources().newFileSystemResource(getFile("named-kiesession"));

KieModule kModule = kr.addKieModule(ex1Res, ex2Res);

KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());

KieSession kSession = kContainer.newKieSession("ksession2");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

Object msg2 = createMessage(kContainer, "Dave", "Open the pod bay doors, HAL.");

kSession.insert(msg2);

kSession.fireAllRules();

##### [4.2.6.9。Programmaticaly用默认构建一个简单的KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_programmaticaly_build_a_simple_kiemodule_with_defaults)

* 项目：kiemoduelmodel-example
* 简介：Programmaticaly仅从一个文件构建KieModule。POM和型号都是默认的。这是最快捷的方式，但不应该添加到Maven存储库。

*例子79.构建和安装 - Maven*

**mvn** install

这个以编程方式构建一个KieModule。它填充表示ReleaseId和kmodule.xml的模型，并添加相关资源。一个pom.xml从ReleaseId生成。

*示例80.利用并运行Java*

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

KieFileSystem kfs = ks.newKieFileSystem();

kfs.write("src/main/resources/org/kie/example5/HAL5.drl", getRule());

KieBuilder kb = ks.newKieBuilder(kfs);

kb.buildAll(); *// kieModule is automatically deployed to KieRepository if successfully built.*

**if** (kb.getResults().hasMessages(Level.ERROR)) {

**throw** **new** RuntimeException("Build Errors:\n" + kb.getResults().toString());

}

KieContainer kContainer = ks.newKieContainer(kr.getDefaultReleaseId());

KieSession kSession = kContainer.newKieSession();

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

##### [4.2.6.10。Programmaticaly使用元模型构建KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_programmaticaly_build_a_kiemodule_using_meta_models)

* 项目：kiemoduelmodel-example
* 简介：通过创建kmodule.xml元模型资源，Programmaticaly构建KieModule。

*例81.构建和安装 - Maven*

**mvn** install

这个以编程方式构建一个KieModule。它填充表示ReleaseId和kmodule.xml的模型，并添加相关资源。一个pom.xml从ReleaseId生成。

*例82.使用和运行 - Java*

KieServices ks = KieServices.Factory.get();

KieFileSystem kfs = ks.newKieFileSystem();

Resource ex1Res = ks.getResources().newFileSystemResource(getFile("named-kiesession"));

Resource ex2Res = ks.getResources().newFileSystemResource(getFile("kiebase-inclusion"));

ReleaseId rid = ks.newReleaseId("org.drools", "kiemodulemodel-example", "6.0.0-SNAPSHOT");

kfs.generateAndWritePomXML(rid);

KieModuleModel kModuleModel = ks.newKieModuleModel();

kModuleModel.newKieBaseModel("kiemodulemodel")

.addInclude("kiebase1")

.addInclude("kiebase2")

.newKieSessionModel("ksession6");

kfs.writeKModuleXML(kModuleModel.toXML());

kfs.write("src/main/resources/kiemodulemodel/HAL6.drl", getRule());

KieBuilder kb = ks.newKieBuilder(kfs);

kb.setDependencies(ex1Res, ex2Res);

kb.buildAll(); *// kieModule is automatically deployed to KieRepository if successfully built.*

**if** (kb.getResults().hasMessages(Level.ERROR)) {

**throw** **new** RuntimeException("Build Errors:\n" + kb.getResults().toString());

}

KieContainer kContainer = ks.newKieContainer(rid);

KieSession kSession = kContainer.newKieSession("ksession6");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

Object msg2 = createMessage(kContainer, "Dave", "Open the pod bay doors, HAL.");

kSession.insert(msg2);

kSession.fireAllRules();

Object msg3 = createMessage(kContainer, "Dave", "What's the problem?");

kSession.insert(msg3);

kSession.fireAllRules();

### [4.3。安全](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesecuritysection)

#### [4.3.1。安全经理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securitymanager)

KIE引擎是一个建模和执行商业行为的平台，使用大量的声明式抽象和隐喻，如规则，流程，决策表等。

许多时候，这些隐喻的创作都是由第三方团体完成的，无论是同一家公司内的不同团体，合作伙伴公司的团体，还是互联网上的匿名第三方。

规则和过程旨在执行任意代码以完成他们的工作，但在这种情况下，可能需要限制他们可以执行的操作。例如，不应该允许规则创建类加载器（可能会使系统受到攻击），当然也不应该允许调用它System.exit()。

Java平台提供了一个非常全面且定义明确的安全框架，允许用户为系统的功能定义策略。KIE平台利用该框架并允许应用程序开发人员定义特定的策略，以应用于任何用户提供代码的执行，无论是在规则，流程，工作项目处理程序等中。

##### [4.3.1.1。如何定义KIE政策](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_howtodefinekiepolicy)

规则和进程可以以非常有限的权限运行，但引擎本身需要执行许多复杂的操作才能工作。例如：它需要创建类加载器，读取系统属性，访问文件系统等。

但是，一旦安装了安全管理器，它将根据定义的策略对JVM中执行的所有代码应用限制。因此，KIE允许用户定义两个不同的策略文件：一个用于引擎本身，另一个用于部署到引擎并由引擎执行的资产。

设置环境的一个简单方法是给引擎本身一个非常宽松的策略，同时为规则和流程提供一个受限制的策略。

策略文件遵循Java文档中描述的标准策略文件语法。有关更多详情，请参阅：

<https://docs.oracle.com/javase/6/docs/technotes/guides/security/PolicyFiles.html#FileSyntax>

引擎的许可策略文件可能如下所示：

*例83.一个示例engine.policy文件*

**grant** {

**permission** java.security.AllPermission;

}

规则的示例安全策略可以是：

*例84.一个样例rules.policy文件*

**grant** {

**permission** java.util.PropertyPermission "\*", "read";

**permission** java.lang.RuntimePermission "accessDeclaredMembers";

}

请注意，根据规则和流程应该执行的操作，可能需要授予更多权限，例如访问文件系统，数据库等中的文件。

为了使用这些策略文件，所有必需的是使用这些文件作为JVM参数执行应用程序。需要三个参数：

| *表5.参数* | |
| --- | --- |
| **参数** | **含义** |
| -Djava.security.manager | 启用安全管理器 |
| -Djava.security.policy = <jvm\_policy\_file> | 定义要应用于整个应用程序（包括引擎）的全局策略文件 |
| -Dkie.security.policy = <kie\_policy\_file> | 定义要应用于规则和进程的策略文件 |

例如：

java -Djava.security.manager -Djava.security.policy=global.policy -Dkie.security.policy=rules.policy foo.bar.MyApp

|  |  |
| --- | --- |
|  | 在容器内执行引擎时，请使用容器的文档来了解如何配置安全管理器以及如何定义全局安全策略。如上所述定义kie安全策略并设置kie.security.policy系统属性以便配置引擎以使用它。 |
|  | 请注意，除非配置了安全管理器，否则kie.security.policy将被忽略。 |

|  |  |
| --- | --- |
|  | 安全管理器在JVM中具有很高的性能影响。严格的性能要求的应用程序强烈建议不要使用安全管理器。另一种方法是在测试和部署之前使用其他安全程序，例如审计规则/进程，以防止将恶意代码部署到环境中。 |

# Drools运行时和语言

Drools是一个强大的混合推理系统。

## [5.混合推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hybridreasoningchapter)

### [5.1。人工智能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_artificial_intelligence)

#### [5.1.1。一点历史](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_little_history)

在过去的几十年中，人工智能 （AI）成为一个不受欢迎的术语，其中着名的[“AI冬季”就是其中之一](http://en.wikipedia.org/wiki/AI_winter)。科学家和工程师在寻找资金时引起了巨大的争议，这从未达到预期，导致许多失败的项目。[Thinking Machines公司](http://en.wikipedia.org/wiki/Thinking_Machines_Corporation)和[第五代计算机](http://en.wikipedia.org/wiki/Fifth-generation_computer)（5GP）项目可能是当时最好的例子。

Thinking Machines公司是1990年的领先人工智能公司之一，它的销售额近6500万美元。以下是其小册子的一句话：

“有一天我们会建立一个思维机器。这将是一台真正的智能机器。一个可以看到，听到并说话的人。一台会为我们感到自豪的机器。“

然而，5年后，它根据第11章申请破产保护。该网站inc.com有一篇引人入胜的文章，题为[“思维机器的兴衰”](http://www.inc.com/magazine/19950915/2622.html)。文章涵盖了行业的发展，以及与Thinking Machines和[DARPA的合作](http://en.wikipedia.org/wiki/DARPA)关系如何使市场过热，直至崩溃。它解释了商业如何以及为何离开人工智能，转向更实际的数字计算超级计算机。

第五代计算机项目在日本是一个4亿美元的项目，用于构建下一代计算机。阀门（或管）是第一代，晶体管第二，集成电路第三，最后微处理器是第四。第五个旨在成为一个能够有效的人工智能的机器。这个项目激发了与英国和美国的“武器”竞赛，这引发了大量的人工智能泡沫。5GP将通过Prolog提供大量的多CPU并行处理硬件以及强大的知识表示和推理软件 ; 一种[术语] \_专家系统\_。到1992年，该项目被认为是失败并被取消。这是Prolog最大和最明显的商业冒险，许多失败都是针对试图在多CPU硬件上同时运行基于逻辑的编程语言并获得有效结果的问题。有人认为，5GP项目的失败使Prolog受到了损害，并将其降级到学术界，请参阅John C. Dvorak的[“](http://www.dvorak.org/blog/whatever-happened-to-prolog/)随机[序言”一](http://www.dvorak.org/blog/whatever-happened-to-prolog/)书。

然而，尽管研究经费枯竭，人工智能这个术语变得不那么用了，但许多绿色的嫩枝在纪律特定的名称下被更加安静地继续使用：认知系统 ，机器学习 ，智能系统 ，[术语] \_知识表示和推理。这些分支随后进入了商业系统，如业务规则管理系统 （BRMS）市场中的专家系统。

势在必行 ，基于系统的语言，语言，如C，C ++，Java和C＃/。NET已经占据了近20年来，通过语言和与商品硬件性能良好运行的能力的实用性功能。然而，许多人认为，由于硬件功能和人工智能研究的进步，人工智能领域正在复兴。2005年，Heather Havenstein撰写了[“春天来到AI冬天”](http://www.computerworld.com/s/article/99691/Spring_comes_to_AI_winter)，其中概述了这场复苏的一个案例。Norvig和Russel专门撰写了几页文章，讨论了哪些因素使该行业克服了其问题以及由此产生的研究成果：

*近年来，人工智能工作的内容和方法都发生了革命。现在建立在现有理论之上而不是建议全新理论，基于严格定理或强实验证据而非直觉的要求，以及与真实世界的应用而不是玩具实例相关的现象更为常见。*

*- 人工智能：现代方法*

计算机视觉 ，神经网络 ，机器学习 和知识表示和推理 （KRR）已经在商业环境中实现了很大的进步。例如，基于视觉的系统现在可以通过强大的识别技巧完全映射并导航他们的环境。因此，我们现在有自动驾驶汽车即将进入商业市场。本体论 基于描述逻辑的研究提供了非常丰富的语义来表示我们的世界。诸如tableaux算法之类的算法使得可以在大型复杂本体中有效地使用这些丰富的语义。早期的KRR系统，如5GP中的Prolog，受限于这些本体规模的有限语义能力和内存限制。

#### [5.1.2。知识表示与推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_representation_and_reasoning)

在“小历史”中，人工智能作为一个更广泛的主题，涉及知识表示和推理（KRR）以及专家系统，我将在稍后回到专家系统。

KRR是关于我们如何用符号形式表示我们的知识，即我们如何描述某些东西。推理是关于我们如何利用这些知识去思考行为。基于系统的面向对象语言（如C ++，Java和C＃）具有称为类的数据定义，用于描述建模实体的组成和行为。在Java中，我们称这些描述事物为bean或实例的范例。然而，这些分类系统是有限的，以确保计算效率。多年来，研究人员开发出了越来越复杂的方式来代表我们的世界 你们中许多人可能已经听说过OWL（网络本体语言）。在理论上可以表示的内容和以实际及时的方式计算使用的内容之间总是存在差距，这就是为什么OWL从Lite到Full具有不同的子语言。不相信任何推理系统都可以支持OWL Full。然而，算法的进步继续缩小这种差距，并提高推理引擎的表现力。

这些系统如何进行思考还有很多方法。您可能听说过将反向链接和数据驱动的正向链接与反向链接的优点进行比较，反向链接是被动和查询驱动的。存在许多其他类型的推理技术，每种推理技术都会扩大我们可以用声明方式处理的问题的范围。仅列出几个：不完全推理（模糊逻辑，确定性因素），不可行逻辑，信念系统，时间推理和关联。你不需要理解所有这些术语来理解和使用Drools。他们只是想了解一系列研究课题的范围，这些研究课题实际上要广泛得多，并且随着研究人员推动新的边界而不断增长。

KRR通常被称为人工智能的核心。即使使用像神经网络这样的生物学方法，它们模拟大脑，更多地关注模式识别而不是思维，它们仍然基于KRR理论。我对Drools的第一次尝试是以工程为导向的，因为我没有正式的KRR培训或理解。学习KRR让我得到了更广泛的理论背景，并更好地理解我所做的和我要去的地方，因为它支撑我们Drools研发的几乎所有理论方面。它确实是一个巨大而迷人的主题，将为那些花时间学习的人们带来红利。我知道它做到了，对我仍然有效。Bracham和Levesque撰写了一篇名为“知识表示与推理”的开创性着作，对于想要建立坚实基础的人来说，这是必读的。我还会推荐Russel和Norvig的书“人工智能，一种现代方法”，它也涵盖了KRR。

#### [5.1.3。规则引擎和生产规则系统（PRS）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_engines_and_production_rule_systems_prs)

我们现在已经介绍了AI的简要历史，并了解到AI的核心是围绕KRR形成的。我们已经证明，KRR是一个巨大而迷人的主题，它构成了驱动Drools研发的大部分理论。

规则引擎是为开发人员提供KRR功能的计算机程序。在较高的层面上，它有三个组成部分：

* 本体论
* 规则
* 数据

如前所述，本体是我们用于“事物”的表示模型。它可以使用记录或Java类或基于OWL的成熟本体。规则执行推理，即便于“思考”。规则和本体之间的区别模糊了OWL本体，其丰富性是基于规则的。

术语“规则引擎”是相当含糊的，因为它可以是任何使用规则的系统，可以以任何形式应用于数据以产生结果。这包括简单的系统，如表单验证和动态表达式引擎。Malcolm Chisholm撰写的“如何构建业务规则引擎”（2004）是这种含糊不清的例子。本书实际上是关于如何构建和更改数据库模式以保存验证规则。然后本书展示了如何从这些验证规则生成Visual Basic代码来验证数据输入。虽然完全有效，但这与我们所谈论的完全不同。

Drools起初是一种称为生产规则系统（PRS）的特定类型的规则引擎，基于Rete算法（通常被称为两个音节，例如REH-te或RAY-tay）。由Charles Forgy于1974年开发的Rete算法形成了生产规则系统的大脑，并能够扩展到大量的规则和事实。生产规则是一个两部分的结构：引擎将事实和数据与生产规则（也称为生产或仅仅是规则）相匹配，以推断导致行动的结论。

when

<conditions>

then

<actions>;

将新的或现有的事实与生产规则相匹配的过程称为模式匹配 ，由推理引擎执行 。响应数据更改而执行动作，如数据库触发器; 我们说这是一种数据驱动的 推理方法。这些行为本身可以改变数据，而这些数据又可以与其他规则相匹配，从而导致它们被触发; 这被称为前向链

Drools 5.x实现并扩展了Rete 算法。这种扩展的Rete算法被命名为ReteOO ，表示Drools对面向对象系统的Rete算法进行了增强和优化实现。其他基于Rete的发动机也有专门针对Rete的增强功能的营销条款，如RetePlus和Rete III。最常见的增强功能在Robert B Doorenbos的论文“大型学习系统的产品匹配”（1995）中介绍，其中介绍了Rete / UL。Drools 6.x引入了一种名为PHREAK的新惰性算法 ; 这在PHREAK算法部分有更详细的介绍。

规则存储在生产内存中，推理引擎匹配的事实保存在工作存储器中。事实被置入工作记忆，然后他们可能会被修改或收回。具有大量规则和事实的系统可能导致对于相同的事实断言许多规则是正确的; 据说这些规则有冲突。议程使用冲突解决策略管理这些冲突规则的执行顺序。

*图122.生产规则系统的高级视图*

#### [5.1.4。混合推理系统（HRS）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hybrid_reasoning_systems_hrs)

您可能已经阅读过讨论比较正向链接（反应式和数据驱动）或反向链接（被动查询）的优点。这是对这两种主要推理类型的快速解释。

正向链接是“数据驱动的”，因此是反动的，事实被声明为工作记忆，这导致一个或多个规则同时成立并被安排由议程执行。总之，我们从一个事实开始，它通过规则传播，最后得出结论。

*图123.正向链接*

反向链接是“目标驱动的”，这意味着我们从引擎试图满足的结论开始。如果它不能，那么它搜索它可以满足的结论。这些被称为子目标，这将有助于满足当前目标的某些未知部分。它继续这个过程，直到最初的结论被证明或者没有更多的子目标。Prolog是Backward Chaining引擎的一个例子。Drools也可以做后向链接，我们称之为派生查询。

*图124.向后链接*

从历史上看，你必须在诸如OPS5（向前）或Prolog（向后）之类的系统之间进行选择。现在许多现代系统提供这两种类型的推理能力。还有许多其他类型的推理技巧，每种都会扩大我们可以用声明处理的问题的范围。仅列出几个：不完全推理（模糊逻辑，确定性因素），不可行逻辑，信念系统，时间推理和关联。现代系统正在合并这些功能，而其他未列出的功能则用于创建混合推理系统（HRS）。

当Drools开始作为PRS时，5.x引入了Prolog风格的反向链接推理以及一些函数式编程风格。出于这个原因，我们现在更喜欢在描述Drools时使用混合推理系统。

Drools目前提供清晰的推理，但是不完善的推理已经准备就绪。最初，这将是模糊逻辑的不完美推理; 稍后我们将增加对其他类型不确定性的支持。目前正在开展基于OWL本体论推理的工作，这将与我们的特征 系统相结合。我们也继续改进我们的函数式编程能力。

#### [5.1.5。专家系统](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_expert_systems)

您经常会听到术语专家系统 用于指生产规则系统 或Prolog 类系统。虽然这通常是可以接受的，但在技术上是不正确的，因为这些都是构建专家系统的框架，而不是专家系统本身。一旦有代表领域的本体论模型并且有知识获取和解释的设施，它就成为专家系统。

Mycin 是最着名的专家系统，建于70年代。它在学术文献中仍然很重要，比如Peter Jackson推荐的书“专家系统”。

*图125.专家系统的早期历史*

#### [5.1.6。推荐阅读](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_recommended_reading)

\*一般AI，KRR和专家系统书籍\*

对于那些想在KRR和专家系统中获得强大理论背景的人，我强烈推荐以下书籍。对于任何人的书架来说，“人工智能：现代方法”是必须的。

* 专家系统介绍
  + 彼得杰克逊
* 专家系统：原理和编程
  + 约瑟夫C. Giarratano，加里D.赖利
* 知识表示与推理
  + Ronald J. Brachman，Hector J. Levesque
* 人工智能：一种现代方法。
  + 斯图尔特罗素和彼得诺维格

*图126.推荐阅读*

\*论文\*

以下是一些推荐的论文，涵盖规则引擎研究中的有趣领域：

* 大型学习系统的生产匹配：Rete / UL（1993）
  + 罗伯特B. Doorenbos
* Rete模式匹配中的进展
  + Marshall Schor，Timothy P. Daly，Ho Soo Lee，Beth R. Tibbitts（AAAI 1986）
* 面向集合的匹配
  + Anurag Acharya和Milind Tambe（1993）
* 跳跃算法
  + Don Batery（1990）
* Gator：针对主动数据库规则条件测试的优化辨别网络
  + 埃里克汉森，穆罕默德S.哈桑（1993）

\* Drools书籍\*

目前有三本Drools书籍，全部来自Packt Publishing。

* JBoss Drools业务规则
  + 保罗布朗
* Drools JBoss Rules 5.0开发人员指南
  + Michal巴厘岛
* Drools开发人员的食谱
  + 卢卡斯阿马多尔

*图127.推荐阅读*

### [5.2。Rete算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reteoo)

该的Rete算法是由查尔斯·福尔热博士发明的，并在1978 - 1979年在他的博士论文记录。该论文的简化版本于1982年发布（<http://citeseer.ist.psu.edu/context/505087/0>）。拉丁词“rete”意思是“网络”或“网络”。Rete算法可以分为两部分：规则编译和运行时执行。

编译算法描述了如何处理生产存储器中的规则以生成有效的歧视网络。用非技术术语来说，识别网络用于在数据通过网络传播时对数据进行过滤。网络顶端的节点会有很多匹配，而当我们沿着网络走时，会有更少的匹配。网络最底层是终端节点。在Forgy博士1982年的论文中，他描述了4个基本节点：根，1输入，2输入和终端。

*图128. Rete节点*

根节点是所有对象进入网络的地方。从那里，它立即进入ObjectTypeNode。ObjectTypeNode的目的是确保引擎不需要做更多的工作。例如，假设我们有2个对象：Account和Order。如果规则引擎试图针对每个对象评估每个节点，则会浪费很多周期。为了使事情有效，引擎只应将对象传递给匹配对象类型的节点。最简单的方法是创建一个ObjectTypeNode，并让所有1输入和2输入节点从它下降。这样，如果应用程序声明一个新的Account，它将不会传播到Order对象的节点。在Drools中，当一个对象被声明时，它会通过HashMap中对象的Class中的查找来检索一个有效的ObjectTypesNode列表; 如果此列表不存在，它将扫描所有ObjectTypeNode，查找它在列表中缓存的有效匹配项。这使得Drools能够匹配任何与类型匹配的类类型instanceof 检查。

*图129. ObjectTypeNodes*

ObjectTypeNodes可以传播到AlphaNodes，LeftInputAdapterNodes和BetaNodes。AlphaNodes用于评估文字条件。尽管1982年的论文只涉及平等条件，但许多RETE实现支持其他操作。例如，Account.name == "Mr Trout"是一个字面条件。当规则对于单个对象类型具有多个文字条件时，它们链接在一起。这意味着如果一个应用程序声明一个Account对象，它必须首先满足第一个文字条件，然后才能进入下一个AlphaNode。在Forgy博士的论文中，他将这些称为IntraElement条件。下图显示了Cheese的AlphaNode组合（name ==“cheddar”，strength ==“strong”）：

*图130. AlphaNodes*

Drools通过使用哈希来优化从ObjectTypeNode到AlphaNode的传播来扩展Rete。每次将AlphaNode添加到ObjectTypeNode时，它都会将字面值作为关键字添加到AlphaNode作为值的HashMap中。当一个新实例进入ObjectType节点，而不是传播到每个AlphaNode时，它可以从HashMap中检索正确的AlphaNode，从而避免不必要的字面检查。

有两个双输入节点，JoinNode和NotNode，都是BetaNode的类型。BetaNodes用于将2个对象及其字段彼此进行比较。对象可以是相同或不同的类型。按照惯例，我们将这两个输入作为左侧和右侧。BetaNode的左侧输入通常是对象列表; 在Drools这是一个Tuple。正确的输入是一个单一的对象。两个节点可以用来实现'存在'检查。BetaNodes也有内存。左边的输入被称为Beta内存，并记住所有传入的元组。正确的输入称为Alpha Memory并记住所有传入的对象。Drools通过在BetaNodes上执行索引来扩展Rete。例如，如果我们知道一个BetaNode正在对字符串字段执行检查，则当每个对象进入时，我们可以对该字符串值执行哈希查找。这意味着当事实从对立面进入时，我们会进行查找，返回可能有效的候选人，而不是遍历所有事实以找到有效的连接。在任何时候都找到有效的联接，元组与对象联合; 这被称为部分匹配; 然后传播到下一个节点。

*图131. JoinNode*

为了启用第一个对象，在上面的例子Cheese中，我们使用LeftInputNodeAdapter来进入网络 - 这需要一个Object作为输入并传播一个Object Tuple。

终端节点用于指示一个匹配其所有条件的单个规则; 在这一点上，我们说这个规则完全匹配。具有“或”条件分离连接的规则会导致每个可能的逻辑分支产生子规则; 因此一条规则可以有多个终端节点。

Drools也执行节点共享。许多规则重复相同的模式，并且节点共享允许我们折叠这些模式，以便它们不必为每个实例重新评估。以下两条规则分享第一种模式，但不是最后一种：

rule

**when**

Cheese( $cheddar : name == "cheddar" )

$person : Person( favouriteCheese == $cheddar )

**then**

System.out.println( $person.getName() + " likes cheddar" );

**end**

rule

**when**

Cheese( $cheddar : name == "cheddar" )

$person : Person( favouriteCheese != $cheddar )

**then**

System.out.println( $person.getName() + " does not like cheddar" );

**end**

正如你在下面看到的，编译的Rete网络显示alpha节点是共享的，但beta节点不是。每个beta节点都有自己的终端节点。如果第二种模式是相同的，那么它也将被共享。

*图132.节点共享*

### [5.3。ReteOO算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reteoosection)

该ReteOO在整个3，4和5系列版本开发的。它采用RETE算法并应用众所周知的增强功能，所有这些都由现有的学术文献涵盖：

节点共享

* 共享适用于alpha和beta网络。测试版网络共享始终来自根模式。

Alpha索引

* 带有许多孩子的Alpha节点使用散列查找机制来避免测试每个结果。

Beta索引

* Join，Not和Exist节点使用散列索引他们的记忆。这减少了等待检查的加入尝试。最近的范围索引被添加到Not和Exists。

基于树的图

* 加入比赛不包含任何对其父母或子女比赛的引用。删除操作将不得不重新计算所有连接匹配，这涉及到重新创建所有这些连接匹配对象，以便能够找到应删除元组的网络部分。这被称为对称传播。树形图提供父级和子级引用，因此删除只是遵循这些引用的问题。这是不对称的传播。结果更快，对GC的影响更小。它也更加健壮，因为值的更改不会导致内存泄漏，如果它们在没有通知引擎的情况下发生。

修改就地

* 传统的RETE将修改实现为删除+插入。这将导致所有连接元组都被GC'd，其中许多元素将作为插入的一部分重新创建。相反，就地修改传播为一次传递; 每个节点都被检查。

属性反应性

* 也称为“新的触发条件”。允许更新的更细粒度的反应性。模式可以对特定属性的更改作出反应并忽略其他属性。这缓解了递归问题，也有助于提高性能。

子网络

* 不，Exists和Accumulate每个都可以有嵌套的条件元素，它们构成了子网络。

后向链接

* 支持反向链接的Prolog风格派生树。该实现是基于堆栈的，因此对于大型图不存在方法递归问题。

懒惰的真相维护

* 无论是否使用TMS，真实维护都会产生运行时间成本。懒惰TMS仅在第一次使用时启用; 并且每个对象类型都会激活它，所以不相关的对象类型不会产生运行时成本。

基于堆的议程

* 该议程使用二进制堆队列通过显着性排序规则匹配，而不是使用任何线性搜索或维护方法。

动态规则

* 规则可以在运行时添加和删除，而引擎仍然填充数据。

### [5.4。PHREAK算法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_phreak)

Drools 6引入了一种新算法，试图解决RETE的一些核心问题。该算法不是从头开始重写：它结合了ReteOO的所有现有代码和增强功能。虽然PHREAK是RETE算法的演变，但它不再被归类为RETE实现。一旦动物超出某个特定点并且关键特征发生变化，动物就会被归类为新物种; 以同样的方式，PHREAK不能被认为是RETE。有两个关键的RETE特征强烈识别任何派生菌株，无论优化如何：

* 它是一种热切的，面向数据的算法。
* 所有的工作都是使用插入，更新或删除操作完成的，他们热切地为所有规则生成所有部分匹配。

相比之下，PHREAK被描述为一种懒惰的，面向目标的算法，其中部分匹配是积极推迟的。

RETE的这种渴望可能导致大型系统中的大量流失，以及大量浪费的工作，其中“浪费的工作”被定义为匹配工作，而不会导致规则解雇。

PHREAK深受许多算法的启发，包括（但不限于）LEAPS，RETE / UL和面向集合的匹配。PHREAK具有ReteOO部分列出的所有增强功能。此外，它还添加了以下几组增强功能，这些增强功能在以下段落中有更详细的介绍。

* 三层情境记忆; 节点，段和规则存储器。
* 规则，段和基于节点的链接。
* 懒惰（延迟）规则评估。
* 孤立的规则评估。
* 设置定向传播。
* 基于堆栈的评估，暂停和恢复。

当PHREAK引擎启动时，所有规则都被认为是不相关的; 规则未链接时，不会发生规则评估。进入beta网络之前，插入，更新和删除操作已排队。基于最有可能导致触发的规则的简单启发式被用于选择下一个评估规则; 这延误了其他规则的评估和解雇。只有一条规则拥有所有权利投入，规则才会被考虑链接，尽管尚未完成任何工作。而是创建一个代表规则的目标，并将其放入优先队列; 这是由显着性排序的。每个队列本身都与一个AgendaGroup相关联。只有积极的AgendaGroup才会检查队列，突出显示最高规则的目标并提交评估。所以完成的工作从插入转移，更新，删除阶段到fireAllRules阶段。只评估制定目标的规则，其他可能的规则评估则从这些事实中推迟。虽然对各个规则进行评估，但节点共享仍然通过分段过程实现，这将在后面解释。

RETE中每次成功的连接尝试都会生成一个将传播到子节点的元组（或标记或部分匹配）。出于这个原因，它的特点是一个面向元组的算法。对于它到达的每个子节点，它都会尝试加入节点的另一侧; 每次成功的加入尝试都会立即传播。这产生了下降递归效果，当节点网络从进入beta网络的点到上下左右摇摆到所有可到达的叶节点时，抖动节点网络。

PHREAK传播是面向（或面向集合）设置的，而不是面向元组的。对于正在评估的规则，它将访问第一个节点并处理所有排队的插入，更新和删除操作。结果被添加到一个集合中，集合被传播到子节点。在子节点中处理所有排队的插入，更新和删除操作，并将结果添加到同一个集合中。一旦完成，该集被传播到下一个子节点，依此类推，直到到达终端节点。这将创建一个单独的通道，管道类型效果与当前正在评估的规则相隔离。这创建了一个批处理效果，可以为某些规则结构（例如具有累积的子网络）提供性能优势。在未来，它将有助于以多种方式利用多核机器。

链接和取消链接使用基于网络分段的分层位掩码系统。规则网络构建时，会为由同一组规则共享的节点创建分段。规则本身是由段的路径构成的，但如果没有共享将是单个段。位掩码偏移量被分配给段中的每个节点。另外一个位掩码（分层）被分配给规则路径中的每个段。当至少有一个输入（数据传播）时，该节点的位被设置为打开。当每个节点的位设置为该段的位时也设置为打开。相反，如果任何节点的位被设置为关闭，则该段也被设置为关闭。如果规则路径中的每个细分受众群都设置为开启，则说该规则已链接，并且会创建一个目标来计划评估规则。使用相同的位掩码技术来跟踪脏节点，段和规则; 这允许已经链接的规则被安排进行评估，如果它自上次评估以来被认为是脏的。

这可以确保没有规则会评估部分匹配，如果它不可能导致规则实例，因为其中一个连接没有数据。这在RETE中是可能的，并且它将快乐地流失，为所有节点产生部分匹配尝试，即使最后的连接是空的。

尽管增量规则评估总是从根节点开始，但脏位掩码用于允许跳过不脏的节点和段。

使用每个节点至少一项数据的存在是相当基本的启发式。未来的工作将尝试使用诸如弧一致性之类的技术来进一步延迟链接，以确定匹配是否会导致规则实例激发。

由于RETE只有一个内存单元，因此节点内存PHREAK有3级内存。这允许在评估规则期间获得更多的上下文理解。

*图133.PHREAK 3分层存储系统*

示例1显示了具有三种模式的单个规则：A，B和C.它形成单个段，其中节点的位1,2和4。单个段的位偏移量为1。

*图134.示例1：单个规则，不共享*

示例2演示了在添加共享模式A的另一个规则时会发生什么.A被放置在其自己的段中，从而导致每个规则有两个段。这两个部分构成了各自规则的路径。第一部分由两个路径共享。链接A时，链接会变为链接。然后迭代段被共享的每个路径，将第1位设置为打开。如果B和C稍后打开，则路径R1的第二个段连接到; 这会导致R1的位2被打开。对于R1，位1和位2设置为开启，现在将规则链接起来并创建一个目标，以便为稍后评估和触发安排规则。

评估规则时，它是允许共享匹配结果的段。每个段都有一个暂存内存，用于对该段的所有插入，更新和删除进行排队。如果要评估R1，它将处理A并产生一组元组。该算法检测到存在分段拆分并将为每个插入创建对等元组，并在集合中进行更新和删除，并将它们添加到R2的分段存储器中。那些元组将与任何现有的阶段元组合并，并等待R2最终被评估。

*图135.例2：两条规则，共享*

示例3添加了第三条规则，并演示了共享A和B时发生的情况。这次只显示段的位。证明R4有3段，R3有3段，R1有2段。A和B由R1，R3和R4共享。而D由R3和R4共享。

*图136.例3：三条规则，共享*

子网络在Not，Exists或Accumulate节点包含多个元素时形成。在示例4中，“B不是（C）”形成子网络，请注意“not（C）”是单个元素，它不需要子网络，因此会合并到Not节点内部。

子网络获得自己的细分。R1仍然有两段的路径。子网络形成另一个“内部”路径。当子网链接时，它将链接到外部网段。

*图137.示例4：单个规则，具有子网络且不共享*

示例5显示子网络节点可以由没有子网络的规则共享。这导致子网段被分成两部分。

*图138.示例5：两个规则，一个具有子网络和共享*

约束Not节点和Accumulate节点具有特殊行为：这些节点不能断开段的链接，并始终认为它们的位已打开。

所有规则评估都是增量式的，不会浪费工作重新计算已经生成的匹配。

评估算法是基于堆栈而不是方法递归。评估可以随时暂停和恢复，通过使用StackEntry来表示当前正在评估的节点。

当规则评估到达子网时，为外部路径段和子网段创建一个StackEntry。子网段首先被评估; 当该集合到达子网络路径的末尾时，它将被合并到它所馈送到的外部节点的分级列表中。之前的StackEntry被恢复，现在可以处理子网络的结果。这具有额外的好处，即在传播到子节点之前，所有工作都是批处理的; 这对于Accumulate节点来说效率更高。

相同的堆栈系统可用于高效的反向链接。当规则评估到达查询节点时，它通过将其放置在堆栈上再次暂停当前评估。然后对该查询进行评估，该查询产生结果集，该结果集保存在用于恢复的StackEntry的存储器位置中以拾取并传播到子节点。如果查询本身称为其他查询，则过程将会重复，当前查询被暂停并为当前查询节点提供新的评估设置。

关于性能的最后一点：一般来说，单一规则不会使用PHREAK评估速度快于使用RETE。对于使用根上下文对象来启用和禁用匹配的给定规则和相同数据集，都尝试相同数量的匹配，产生相同数量的规则实例并且花费大致相同的时间量，除了用例子网和积累。

但是，对于编写不佳的规则库，并且随着规则数量和复杂性的增加，性能会更优雅地降低，可以认为PHREAK更容易忽略RETE。

对于在所有连接中没有数据的规则，RETE也会产生部分匹配，而PHREAK将避免这种情况。

所以并不是说PHREAK比RETE快; 它不会像你的系统增长那样慢下来:)

议程小组对RETE表现没有帮助，因为所有规则都在任何时候进行评估，而不管小组如何。显着性也是如此，这就是为什么根上下文对象经常被用来限制匹配尝试的原因。PHREAK仅评估活动议程组的规则，并且该组内的人员将尝试避免评估（通过显着性）不会导致规则实例解除的规则。

随着PHREAK议程小组和突出现在成为有用的表现工具。根上下文对象不再需要，并且可能会对性能产生不利影响，因为它们强制冲洗和重新匹配规则。

## [6.用户指南](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_guide)

### [6.1。基础](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_basics)

#### [6.1.1。无状态知识会话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stateless_knowledge_session)

那么我们从哪里开始？在诸如Drools这样的规则引擎中，有太多的用例以及如此多的功能，使得它变得诡计多端。不要害怕我的勇敢的冒险家，复杂性是分层次的，你可以用简单的用例来放松自己。

无状态会话，不使用推理，形成最简单的用例。无状态会话可以像调用一些数据然后接收一些结果的函数一样调用。无状态会话的一些常见用例包括但不限于：

* 验证
  + 这个人是否有资格获得抵押贷款？
* 计算
  + 计算按揭保费。
* 路由和过滤
  + 将传入的邮件（如电子邮件）过滤到文件夹中。
  + 将传入的消息发送到目标。

所以让我们从一个使用驾驶执照申请的简单例子开始。

**public** **class** **Applicant** {

**private** String name;

**private** **int** age;

**private** **boolean** valid;

*// getter and setter methods here*

}

现在我们有了我们的数据模型，我们可以编写第一条规则。我们假设应用程序使用规则来拒绝无效的应用程序。由于这是一个简单的验证用例，我们将添加一条规则，以取消任何年龄小于18岁的申请人的资格。

package com.company.license

rule "Is of valid age"

**when**

$a : Applicant( age < 18 )

**then**

$a.setValid( **false** );

**end**

为了让引擎知道数据，所以可以根据规则处理它，我们必须插入数据，就像数据库一样。当申请人实例插入引擎时，将根据规则的约束条件进行评估，在这种情况下，对于一个规则只需要两个约束条件。我们说二，因为申请人类型是第一个对象类型约束，并且age < 18是第二个字段约束。对象类型约束加上其零个或多个字段约束被称为模式。当插入的实例同时满足对象类型约束和所有字段约束时，就说它是匹配的。该$a是一个绑定变量，它允许我们在结果中引用匹配的对象。其属性可以更新。美元字符（'$'）是可选的，但它有助于区分字段名称中的变量名称。毫不奇怪，将模式与插入数据进行匹配的过程通常称为模式匹配。

为了使用这个规则，有必要把它放在一个Drools文件中，只是一个带有.drl扩展名的纯文本文件，简称“Drools规则语言”。我们称这个文件为licenseApplication.drl，并将它存储在Kie项目中。Kie项目具有普通Maven项目的结构，其中有一个额外的文件（kmodule.xml），用于定义可以创建的KieBases和KieSessions。该文件必须放置在Maven项目的resources / META-INF文件夹中，而所有其他Drools工件（如包含前一个规则的licenseApplication.drl）必须存储在资源文件夹或其下的任何其他子文件夹中。

由于已经为所有配置方面提供了有意义的默认值，最简单的kmodule.xml文件可以包含一个空的kmodule标签，如下所示：

**<?xml version="1.0" encoding="UTF-8"?>**

<kmodule xmlns="http://www.drools.org/xsd/kmodule"/>

在这一点上，可以创建一个KieContainer从类路径读取要构建的文件。

KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();

上面的代码片段编译了在类路径中找到的所有DRL文件，并将该编译的结果a KieModule放入KieContainer。如果没有错误，我们现在准备从KieContainer一些数据创建会话并执行一些数据：

StatelessKieSession kSession = kContainer.newStatelessKieSession();

Applicant applicant = **new** Applicant( "Mr John Smith", 16 );

assertTrue( applicant.isValid() );

ksession.execute( applicant );

assertFalse( applicant.isValid() );

上述代码根据规则执行数据。由于申请人年龄在18岁以下，因此申请被标记为无效。

到目前为止，我们只使用了一个实例，但如果我们想要使用多个实例呢？我们可以针对任何实现Iterable的对象执行，比如集合。让我们添加另一个名为Application的应用程序，它有应用程序的日期，我们也将布尔有效的字段移动到Application类中。

**public** **class** **Applicant** {

**private** String name;

**private** **int** age;

*// getter and setter methods here*

}

**public** **class** **Application** {

**private** Date dateApplied;

**private** **boolean** valid;

*// getter and setter methods here*

}

我们还会添加另一条规则来验证申请是在一段时间内完成的。

package com.company.license

rule "Is of valid age"

**when**

Applicant( age < 18 )

$a : Application()

**then**

$a.setValid( **false** );

**end**

rule "Application was made this year"

**when**

$a : Application( dateApplied > "01-jan-2009" )

**then**

$a.setValid( **false** );

**end**

不幸的是，Java数组没有实现Iterable接口，所以我们必须使用JDK转换器方法Arrays.asList(…​)。下面显示的代码针对可迭代列表执行，其中在触发任何匹配规则之前插入所有集合元素。

StatelessKieSession kSession = kContainer.newStatelessKieSession();

Applicant applicant = **new** Applicant( "Mr John Smith", 16 );

Application application = **new** Application();

assertTrue( application.isValid() );

ksession.execute( Arrays.asList( **new** Object[] { application, applicant } ) );

assertFalse( application.isValid() );

两个执行方法execute(Object object)和execute(Iterable objects)实际上是为接口方便的方法BatchExecutor的方法execute(Command command)。

与KIE A​​PI的所有其他工厂一样KieCommands可以获得的命令工厂KieServices用于创建命令，因此以下内容相当于execute(Iterable it)：

ksession.execute( kieServices.getCommands().newInsertElements( Arrays.asList( **new** Object[] { application, applicant } ) );

批处理执行程序和命令工厂在使用多个命令和输出标识符来获取结果时特别有用。

KieCommands kieCommands = kieServices.getCommands();

List<Command> cmds = **new** ArrayList<Command>();

cmds.add( kieCommands.newInsert( **new** Person( "Mr John Smith" ), "mrSmith", **true**, **null** ) );

cmds.add( kieCommands.newInsert( **new** Person( "Mr John Doe" ), "mrDoe", **true**, **null** ) );

BatchExecutionResults results = ksession.execute( kieCommands.newBatchExecution( cmds ) );

assertEquals( **new** Person( "Mr John Smith" ), results.getValue( "mrSmith" ) );

CommandFactory支持可在使用许多其它的命令BatchExecutor一样StartProcess，Query和SetGlobal。

#### [6.1.2。有状态的知识会议](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stateful_knowledge_session)

有状态会话很长时间，并允许随着时间的推移进行迭代更改。有状态会话的一些常见用例包括但不限于：

* 监控
  + 股票市场监测和半自动购买分析。
* 诊断
  + 故障查找，医疗诊断
* 后勤
  + 包裹跟踪和交付供应
* 合规
  + 验证市场交易的合法性。

与无状态会话相反，dispose()必须事后调用该方法以确保不存在内存泄漏，因为KieBase在创建时包含有状态知识会话的引用。由于有状态知识会话是最常用的会话类型，因此它仅KieSession在KIE A​​PI中命名。KieSession也支持该BatchExecutor接口，StatelessKieSession唯一的区别在于该FireAllRules命令不会在有状态会话结束时自动调用。

我们举例说明了监控用例引发火警的例子。我们只用四堂课，代表一间房子里的房间，每个房间都有一个洒水喷头。如果一个房间发生火灾，我们用一个Fire实例代表。

**public** **class** **Room** {

**private** String name

*// getter and setter methods here*

}

**public** **class** **Sprinkler** {

**private** Room room;

**private** **boolean** on;

*// getter and setter methods here*

}

**public** **class** **Fire** {

**private** Room room;

*// getter and setter methods here*

}

**public** **class** **Alarm** {

}

在前面关于无状态会话的部分中，介绍了插入和匹配数据的概念。该示例假定每个对象类型只有一个实例被插入，因此只使用文字约束。然而，房子有很多房间，所以规则必须表达物体之间的关系，比如喷洒器在某个房间里。这最好通过使用绑定变量作为模式中的约束来完成。这种“加入”过程产生了所谓的交叉产品，下一节将对此进行介绍。

发生火灾时Fire，会为该房间创建类的实例，并将其插入会话中。该规则room在Fire对象的字段上使用绑定来约束与当前关闭的房间的喷洒器的匹配。当这个规则触发并且结果被执行时，洒水器被打开。

rule "When there is a fire turn on the sprinkler"

**when**

Fire($room : room)

$sprinkler : Sprinkler( room == $room, on == **false** )

**then**

modify( $sprinkler ) { setOn( **true** ) };

System.out.println( "Turn on the sprinkler for room " + $room.getName() );

**end**

鉴于无状态会话使用标准的Java语法来修改字段，在上述规则中，我们使用modify语句，它充当一种“with”语句。它可能包含一系列逗号分隔的Java表达式，即对通过modify语句的控制表达式选择的对象的设置者的调用。这会修改数据，并使引擎知道这些更改，以便再次推理它们。这个过程被称为推理，对于有状态会话的工作是必不可少的。无状态会话通常不会使用推理，因此引擎不需要知道数据的更改。推理也可以通过使用顺序模式显式关闭。

到目前为止，我们已经告诉我们，当规则匹配的数据存在，但什么时候什么它不存在吗？我们如何确定火灾已经熄灭，即没有Fire物体了？以前，约束条件是根据命题逻辑的句子，其中引擎限制个别情况。Drools也支持First Order Logic，允许您查看数据集。not当某个东西不存在时，关键字下的模式匹配。一旦房间里的火已经消失，下面的规则就会使喷洒器关闭。

rule "When the fire is gone turn off the sprinkler"

**when**

$room : Room( )

$sprinkler : Sprinkler( room == $room, on == **true** )

**not** Fire( room == $room )

**then**

modify( $sprinkler ) { setOn( **false** ) };

System.out.println( "Turn off the sprinkler for room " + $room.getName() );

**end**

虽然每个房间有一个喷水灭火器，但建筑物只有一个警报。Alarm发生火灾时会创建一个对象，但Alarm无论发生多少火灾，整个建筑只需要一个对象。之前not被引入以匹配缺少事实; 现在我们使用它的补码exists来匹配某个类别的一个或多个实例。

rule "Raise the alarm when we have one or more fires"

**when**

exists Fire()

**then**

insert( **new** Alarm() );

System.out.println( "Raise the alarm" );

end

同样，当没有火灾时，我们想要移除警报，因此not关键字可以再次使用。

rule "Cancel the alarm when all the fires have gone"

**when**

**not** Fire()

$alarm : Alarm()

**then**

delete( $alarm );

System.out.println( "Cancel the alarm" );

**end**

最后，当应用程序首次启动并且在警报被移除并且所有洒水喷头都关闭后，会打印一般健康状态消息。

rule "Status output when things are ok"

**when**

**not** Alarm()

**not** Sprinkler( on == true )

**then**

System.out.println( "Everything is ok" );

end

正如我们在无状态会话示例中所做的那样，上述规则应放置在单个DRL文件中并保存到Maven项目或其任何子文件夹的资源文件夹中。和以前一样，我们就可以得到一个KieSession从KieContainer。唯一的区别是，这次我们创建了一个有状态会话，而在我们创建了一个无状态会话之前。

KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();

KieSession ksession = kContainer.newKieSession();

随着会话的创建，现在可以反复使用它。Room创建并插入四个对象，以及Sprinkler每个房间的一个对象。在这一点上，引擎已经完成了所有的匹配，但是还没有任何规则被解雇。调用ksession.fireAllRules()允许匹配的规则触发，但没有会产生健康信息的火灾。

String[] names = **new** String[]{"kitchen", "bedroom", "office", "livingroom"};

Map<String,Room> name2room = **new** HashMap<String,Room>();

**for**( String name: names ){

Room room = **new** Room( name );

name2room.put( name, room );

ksession.insert( room );

Sprinkler sprinkler = **new** Sprinkler( room );

ksession.insert( sprinkler );

}

ksession.fireAllRules();

> Everything **is** ok

我们现在创建两个大火并插入它们; 这一次保留了返回的参考FactHandle。事实句柄是插入实例的内部引擎引用，并允许实例在稍后的时间点收回或修改。现在发动机发生火灾时，一旦fireAllRules()被叫，报警器就会升起，相应的洒水喷头就会打开。

Fire kitchenFire = **new** Fire( name2room.get( "kitchen" ) );

Fire officeFire = **new** Fire( name2room.get( "office" ) );

FactHandle kitchenFireHandle = ksession.insert( kitchenFire );

FactHandle officeFireHandle = ksession.insert( officeFire );

ksession.fireAllRules();

> Raise the alarm

> Turn on the sprinkler **for** room kitchen

> Turn on the sprinkler **for** room office

一段时间后，火势将被扑灭，并且这些Fire事件将被收回。这会导致喷头关闭，报警被取消，最终再次打印健康信息。

ksession.delete( kitchenFireHandle );

ksession.delete( officeFireHandle );

ksession.fireAllRules();

> Cancel the alarm

> Turn off the sprinkler **for** room office

> Turn off the sprinkler **for** room kitchen

> Everything **is** ok

每个人都还在我身边？这并不难，我希望你能开始看到声明式规则系统的价值和威力。

#### [6.1.3。方法与规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_methods_versus_rules)

人们经常会混淆方法和规则，而新规则用户经常会问：“我如何称呼规则？” 在最后一节之后，你现在感觉自己像一个规则专家，对此的回答很明显，但让我们总结一下差异。

**public** **void** **helloWorld**(Person person) {

**if** ( person.getName().equals( "Chuck" ) ) {

System.out.println( "Hello Chuck" );

}

}

* 方法直接调用。
* 通过特定的实例。
* 一次调用会导致一次执行。

rule "Hello World" **when**

**Person**( name == "Chuck" )

then

System.**out**.**println**( "Hello Chuck" );

end

* 只要插入到引擎中，规则就会通过匹配任何数据来执行。
* 规则永远不能直接调用。
* 特定的实例不能传递给规则。
* 取决于匹配，规则可能触发一次或几次，或根本不触发。

#### [6.1.4。交叉产品](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cross_products)

早些时候提到了“交叉产品”这个词，这是加入的结果。想象一下，来自火灾报警示例的数据与没有字段限制的下列规则结合使用：

rule "Show Sprinklers" **when**

$room : Room()

$sprinkler : Sprinkler()

**then**

System.out.println( "room:" + $room.getName() +

" sprinkler:" + $sprinkler.getRoom().getName() );

**end**

用SQL语言来说，这就好比做了select \* from Room, Sprinkler，而Room表中的每一行都将与Sprinkler表中的每一行相连，从而产生以下输出：

room:office sprinkler:office

room:office sprinkler:kitchen

room:office sprinkler:livingroom

room:office sprinkler:bedroom

room:kitchen sprinkler:office

room:kitchen sprinkler:kitchen

room:kitchen sprinkler:livingroom

room:kitchen sprinkler:bedroom

room:livingroom sprinkler:office

room:livingroom sprinkler:kitchen

room:livingroom sprinkler:livingroom

room:livingroom sprinkler:bedroom

room:bedroom sprinkler:office

room:bedroom sprinkler:kitchen

room:bedroom sprinkler:livingroom

room:bedroom sprinkler:bedroom

这些交叉产品显然可能变得巨大，并且它们可能包含虚假数据。交叉产品的大小往往是新规则制定者性能问题的根源。由此可以看出，限制交叉产品总是可取的，这是通过可变约束来完成的。

rule

**when**

$room : Room()

$sprinkler : Sprinkler( room == $room )

**then**

System.out.println( "room:" + $room.getName() +

" sprinkler:" + $sprinkler.getRoom().getName() );

**end**

这导致只有四行数据，每个房间都有正确的喷淋头。在SQL（实际上HQL）相应的查询将是select \* from Room, Sprinkler where Room == Sprinkler.room。

room:office sprinkler:office

room:kitchen sprinkler:kitchen

room:livingroom sprinkler:livingroom

room:bedroom sprinkler:bedroom

### [6.2。执行控制](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_execution_control)

#### [6.2.1。议程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agenda)

议程是一个Rete功能。它维护一组能够执行的规则，其工作是以确定性的顺序安排执行。

在对此采取行动期间RuleRuntime，规则可能完全匹配并有资格执行; 单个规则运行时操作可能会导致多个符合条件的规则。当规则完全匹配时，创建规则匹配，参考规则和匹配的事实，并放置到议程中。议程使用冲突解决策略控制这些匹配的执行顺序。

发动机通过两个阶段重复循环：

1. 规则运行时操作。这是大部分工作发生的地方，无论是在Consequence（RHS本身）还是主要的Java应用程序过程中。一旦结果完成或主Java应用程序进程fireAllRules()将引擎切换调用到议程评估阶段。
2. 议程评估。这会尝试选择要触发的规则。如果未找到规则，则会退出，否则会触发找到的规则，将阶段切换回规则运行时操作。

*图139.两阶段执行*

该过程重复，直到议程清晰，在这种情况下控制返回到调用应用程序。当规则运行时操作正在发生时，没有规则被触发。

#### [6.2.2。规则匹配和冲突集。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_matches_and_conflict_sets)

##### [6.2.2.1。现金流量示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cashflow_example)

到目前为止，数据和匹配过程都很简单和小巧。为了将事情混合起来，我们将探索一个新的例子来处理日期期间的现金流量计算。发动机的状态将在关键阶段示例性地示出，以帮助更好地理解发动机罩下的实际情况。将使用三个类，如下所示。这将有助于我们增加对模式匹配的理解并进一步加入。然后我们将用这个来说明执行控制的不同技术。

**public** **class** **CashFlow** {

**private** Date date;

**private** **double** amount;

**private** **int** type;

**long** accountNo;

*// getter and setter methods here*

}

**public** **class** **Account** {

**private** **long** accountNo;

**private** **double** balance;

*// getter and setter methods here*

}

**public** AccountPeriod {

**private** Date start;

**private** Date end;

*// getter and setter methods here*

}

到目前为止，您已经知道如何创建KieBases以及如何实例化事实以填充它们KieSession，因此将使用表来显示插入数据的状态，因为它使事情变得更加清晰。下面的表格显示为此添加了单个事实Account。还插入了一系列作为CashFlow该账户的对象的借方和贷方，延长了两个季度。

*图140. CashFlows和账户*

可以使用两条规则来确定该季度的借方和贷方，并更新账户余额。下面的两条规则限制了给定时间段内某个账户的现金流量。请注意使用捷径语法的“&&”，以避免重复两次字段名称。

|  |  |
| --- | --- |
| rule "increase balance for credits"  **when**  ap : AccountPeriod()  acc : Account( $accountNo : accountNo )  CashFlow( type == CREDIT,  accountNo == $accountNo,  date >= ap.start && <= ap.**end**,  $amount : amount )  **then**  acc.balance += $amount;  **end** | rule "decrease balance for debits"  **when**  ap : AccountPeriod()  acc : Account( $accountNo : accountNo )  CashFlow( type == DEBIT,  accountNo == $accountNo,  date >= ap.start && <= ap.**end**,  $amount : amount )  **then**  acc.balance -= $amount;  **end** |

之前我们展示了规则如何等同于SQL，这通常可以帮助具有SQL背景的人理解规则。上面的两条规则可以用两个视图和每个视图的触发器来表示，如下所示：

|  |  |
| --- | --- |
| **select** \* **from** **Account** **acc**,  Cashflow cf,  AccountPeriod ap  **where** **acc**.accountNo == cf.accountNo **and**  cf.**type** == CREDIT **and**  cf.date >= ap.**start** **and**  cf.date <= ap.**end** | **select** \* **from** **Account** **acc**,  Cashflow cf,  AccountPeriod ap  **where** **acc**.accountNo == cf.accountNo **and**  cf.**type** == DEBIT **and**  cf.date >= ap.**start** **and**  cf.date <= ap.**end** |
| **trigger** : acc.balance += cf.amount | **trigger** : acc.balance -= cf.amount |

如果AccountPeriod设置为第一季度，我们将限制规则“增加信用余额”以触发两行数据，并且“减少借方余额”以对一行数据进行操作。

*图141. AccountingPeriod，CashFlows和Account*

上述两张现金流量表代表两条规则的匹配数据。数据在插入阶段匹配，正如您在前一章中发现的那样，数据不会立即触发，而只会在fireAllRules()调用之后触发。同时，规则加上其匹配的数据被放置在议程中并被称为RuIe匹配或规则实例。议程是一个规则匹配表，只要fireAllRules（）被调用，它就能够触发并执行其后果。议程上的规则匹配被称为冲突集 ，其执行取决于冲突解决策略。请注意，到目前为止的执行顺序被认为是任意的。

*图142. CashFlows和账户*

在所有上述激活被解雇后，该账户的余额为-25。

*图143. CashFlows和账户*

如果AccountPeriod更新到第二季度，我们只有一行匹配的数据，因此在议程上只有一个规则匹配。

激活的结果导致了25的平衡。

*图144. CashFlows和账户*

*图145. CashFlows和账户*

##### [6.2.2.2。解决冲突](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conflict_resolution)

如果你不希望规则执行的顺序是任意的呢？当议程上有一个或多个规则匹配时，它们被认为是冲突的，并且使用冲突解决策略来确定执行顺序。Drools策略非常简单，基于显着性值，它为规则赋予优先级。每个规则的默认值为0，值越高，优先级越高。

作为一般规则，不要指望以任何特定顺序解雇规则，并且在不担心“流量”的情况下编写规则是一个好主意。然而，当需要流量时，除了显着性之外还存在许多可能性：议程组，规则流程组，激活组和控制/信号量事实。

从Drools 6.0开始，源文件中的规则定义顺序用于在突出显示后设置优先级。

##### [6.2.2.3。凸显](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_salience)

为了说明Salience，我们添加了一个规则来打印账户余额，我们希望在所有账户的所有借记和贷项都被应用后执行此规则。我们通过给这个规则分配一个负面的显着性来达到这个目的，以便在默认显着性0的所有规则之后触发。

|  |
| --- |
| rule "Print balance for AccountPeriod"  salience -50  **when**  ap : AccountPeriod()  acc : Account()  **then**  System.out.println( acc.accountNo + " : " + acc.balance );  **end** |

下表描述了由此产生的议程。这三个借记和贷记规则显示为任意顺序，而打印规则排在最后，以后执行。

*图146.现金流和账户*

##### [6.2.2.4。议程组](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agenda_groups)

议程组允许您将规则放入组中，并将这些组放入堆栈。该堆栈具有推/推动的效果。调用“setFocus”将组放入堆栈：

ksession.getAgenda().getAgendaGroup( "Group A" ).setFocus();

议程总是评估堆栈的顶部。当所有规则都为组激发时，它将从堆栈弹出，并评估下一个组。

|  |  |
| --- | --- |
| rule "increase balance for credits"  agenda-group "calculation"  when  ap : AccountPeriod()  acc : Account( $accountNo : accountNo )  CashFlow( type == CREDIT,  accountNo == $accountNo,  date >= ap.start && <= ap.end,  $amount : amount )  then  acc.balance += $amount;  end | rule "Print balance for AccountPeriod"  agenda-group "report"  when  ap : AccountPeriod()  acc : Account()  then  System.out.println( acc.accountNo +  " : " + acc.balance );  end |

首先将重点放在“报告”组，然后将重点放在“计算”上，我们确保首先评估该组。

Agenda agenda = ksession.getAgenda();

agenda.getAgendaGroup( "report" ).setFocus();

agenda.getAgendaGroup( "calculation" ).setFocus();

ksession.fireAllRules();

##### [6.2.2.5。规则流程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_flow)

Drools还具有ruleflow-group属性，允许工作流程图声明性地指定何时允许规则触发。以下屏幕截图是使用Drools插件从Eclipse获取的。它有两个ruleflow-group节点，它们确保计算规则在报告规则之前执行。

在规则中使用ruleflow-group属性如下所示。

|  |  |
| --- | --- |
| rule "increase balance for credits"  ruleflow-group "calculation"  **when**  ap : AccountPeriod()  acc : Account( $accountNo : accountNo )  CashFlow( type == CREDIT,  accountNo == $accountNo,  date >= ap.start && <= ap.**end**,  $amount : amount )  **then**  acc.balance += $amount;  **end** | rule "Print balance for AccountPeriod"  ruleflow-**group** "report"  **when**  ap : AccountPeriod()  acc : Account()  then  System.**out**.println( acc.accountNo +  " : " + acc.balance );  end |

### [6.3。推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_inference)

#### [6.3.1。巴士通行证示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_bus_pass_example)

现在推论有一个糟糕的名字，因为它与商业用例无关，而且太复杂而无用。诚然，有些人为的和复杂的例子会在推论中出现，但这不应该影响简单而有用的例子的存在。但更重要的是，正确使用推理可以提供更敏捷，更容易出错的业务规则，这些规则更容易维护。

那么推论是什么？当我们通过使用以前的知识获得某些知识时推断出某些事物。例如，给定一个年龄字段的Person事实和一个提供年龄政策控制的规则，我们可以推断出一个Person是成年人还是孩子，并根据此行为进行操作。

rule "Infer Adult"

**when**

$p : Person( age >= 18 )

**then**

insert( new IsAdult( $p ) )

**end**

由于上述规则，每个18岁或以上的人都会为他们插入一个IsAdult实例。这个事实是特殊的，因为它被称为关系。我们可以在任何规则中使用此推断关系：

$p : Person()

IsAdult( person == $p )

所以现在我们知道推论是什么，并有一个基本的例子，这是如何促进良好的规则设计和维护？

让孩子成年后负责发放身份证的政府部门，从此称为ID部门。他们可能有一张决策表，其中包含这样的逻辑，它说明一个住在伦敦的成人18岁或以上的时候，发卡：

然而，身份证部门没有制定关于成年人的政策。这是在中央政府层面完成的。如果中央政府将这个年龄改为21岁，这将启动变革管理流程。有人必须与身份证部门保持联系，并确保他们的系统得到更新，以便法律上线。

这种变更管理流程和部门之间的沟通对于敏捷环境来说并不理想，变更成本高且容易出错。此外，信用卡部门管理的信息比管理更多的信息要多，因为它的“单一”方式对规则管理是“泄漏”更好的信息。我的意思是，它并不关心明确的年龄  ▸ = 18信息决定某人是否是成年人，只是他们是成年人。

与此相反，让我们采取一种方法，将创作职责分开（解除耦合），以便中央政府和身份证部门都保持自己的规则。

确定谁是成年人是中央政府的工作。如果他们改变法律，他们只需使用其他人使用的新规则更新其中央存储库：

如前所述，IsAdult事实是从政策规则中推断出来的。它封装了似乎任意的逻辑时间  ▸ = 18，并为其含义提供了语义抽象。现在，如果有人使用上述规则，他们不再需要知道确定某人是否是成年人的明确信息。他们可以使用推断的事实：

尽管这个例子非常简单，但却很重要。我们从知识工程的单一和漏洞入手开始。我们创建了一张决策表，其中包含所有可能的信息，并从中央政府泄漏了ID部门不关心也不想管理的信息。

我们首先将知识过程分离开来，以便每个部门仅对其需要知道的内容负责。然后，我们使用推断的事实IsAdult封装了这个泄漏的知识。术语IsAdult的使用也对先前的任意逻辑时间  ▸ = 18进行了语义抽象。

因此，在进行知识工程时，一般的经验法则是：

* **坏**
  + 单片
  + 漏
* **好**
  + 除去知识责任
  + 封装知识
  + 为这些封装提供语义抽象

### [6.4。用逻辑对象维护真相](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_truth_maintenance_with_logical_objects)

#### [6.4.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview)

定期插入后，您必须明确收回事实。使用逻辑断言时，断言的事实将自动收回，因为首先断言它的条件不再成立。实际上，它更聪明，因为只有当没有任何单一条件支持逻辑断言时它才会被收回。

据说正常的插入是说明的，就像“陈述事实”所暗示的直觉意义一样。使用一个HashMap和一个计数器，我们追踪一个特定的平等被陈述了多少次; 这意味着我们计算有多少不同的实例是相等的。

当我们在RHS执行期间逻辑插入一个对象时，我们被说成是正当的，并且它被认为是通过触发规则来证明的。对于每个逻辑插入，只能有一个相等的对象，并且每个随后的相等的逻辑插入增加该逻辑断言的对齐计数器。创建规则的LHS取消了理由变得不真实，并且计数器相应地减少。只要我们没有更多的理由，逻辑对象就会自动收回。

如果我们尝试在存在相等声明的对象时逻辑上插入对象，则将失败并返回null。如果我们声明一个具有对齐的现有平等对​​象的对象，我们将覆盖该事实; 这种覆盖的工作方式取决于配置设置。当该属性设置为放弃时，我们使用现有句柄并用新对象替换现有实例，这是默认行为; 否则，我们就覆盖到规定，但我们创建一个新的。WM\_BEHAVIOR\_PRESERVEFactHandle

这可能会导致第一次读取时产生混淆，所以希望下面的流程图有所帮助。当它说它返回一个新的时候FactHandle，这也表示这Object是通过网络传播的。

*图147.规定的插入*

*图148.逻辑插入*

##### [6.4.1.1。带推理和TMS的公共汽车通行证例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_bus_pass_example_with_inference_and_tms)

前面的例子是发行身份证超过18岁，在这个例子中，我们现在发行巴士通行证，无论是儿童通行证还是成人通行证。

rule "Issue Child Bus Pass" **when**

$p : Person( age < 16 )

**then**

insert(new ChildBusPass( $p ) );

**end**

rule "Issue Adult Bus Pass" **when**

$p : Person( age >= 16 )

**then**

insert(new AdultBusPass( $p ) );

**end**

和以前一样，上面的例子被认为是单一的，泄漏的，并且提供了很差的问题分离。

像以前一样，我们可以通过使用推理提供更强大的应用程序，并分离关注点。注意，这次我们不只是插入推断的对象，我们使用“insertLogical”：

rule "Infer Child" **when**

$p : Person( age < 16 )

**then**

insertLogical( new IsChild( $p ) )

**end**

rule "Infer Adult" **when**

$p : Person( age >= 16 )

**then**

insertLogical( new IsAdult( $p ) )

**end**

“insertLogical”是Drools真相维护系统（TMS）的一部分。当一个事实被逻辑插入时，这个事实取决于“when”从句的真实性。这意味着当规则变得错误时，事实会自动收回。由于这两条规则是相互排斥的，因此这种方法特别有效 因此，在上述规则中，如果该人不满16岁，则插入IsChild事实，一旦该人为16岁或以上，IsChild事实将自动收回并插入IsAdult事实。

回到发布总线通道的代码，这两个规则可以在逻辑上插入ChildBusPass和AdultBusPass事实，因为TMS +支持链接一系列级联回收的逻辑插入。

rule "Issue Child Bus Pass" **when**

$p : Person( )

IsChild( person == $p )

**then**

insertLogical(new ChildBusPass( $p ) );

**end**

rule "Issue Adult Bus Pass" **when**

$p : Person( age >= 16 )

IsAdult( person =$p )

**then**

insertLogical(new AdultBusPass( $p ) );

**end**

现在，当一个人从15岁变为16岁时，不仅IsChild事实自动缩回，该人的ChildBusPass事实也是如此。对于奖励积分，我们可以将此与'not'条件元素结合来处理通知，在这种情况下，请求返回通行证。因此，当TMS自动收回ChildBusPass对象时，此规则触发并向该人发送请求：

rule "Return ChildBusPass Request "**when**

$p : Person( )

**not**( ChildBusPass( person == $p ) )

**then**

requestChildBusPass( $p );

**end**

##### [6.4.1.2。重要说明：Java对象的平等](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_important_note_equality_for_java_objects)

需要注意的是，对于真理维护（和逻辑断言）根本不起作用，您的Fact对象（可能是JavaBeans）必须正确地覆盖equals和hashCode方法（来自java.lang.Object）。由于事实维护系统需要知道两个不同物理对象的值是否相等，因此必须按照Java标准正确覆盖equals和hashCode。

当且仅当它们的equals方法相互返回true并且它们的hashCode方法返回相同的值时，两个对象相等。有关更多详细信息，请参阅Java API（但请记住，您必须重写equals和hashCode）。

TMS行为不受标识vs平等的时间配置的影响，TMS始终是平等的。

##### [6.4.1.3。从工作记忆中删除陈述或逻辑断言的事实](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_deleting_stated_or_logically_asserted_facts_from_the_working_memory)

默认情况下，当从工作内存中删除一个事实时，Drools尝试从既定事实集合中删除它，并且在逻辑断言的情况下也从真值维护系统中删除它。但是，使用delete方法的重载，也可以仅从2中删除它。例如，调用：

ksession.delete( factHandle, FactHandle.State.LOGICAL );

事实只有在逻辑断言的情况下才会被删除，但如果这是事实陈述的话则不会被删除。在这种情况下，如果事实已经表明其删除失败并且被忽略。

### [6.5。电子表格中的决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_decision_tables_in_spreadsheets)

决策表是一种“精确但紧凑”（参考维基百科）表示条件逻辑的方式，非常适合商业级规则。

Drools支持以电子表格格式管理规则。支持的格式是Excel（XLS）和CSV，这意味着可以使用各种电子表格程序（例如Microsoft Excel，OpenOffice.org Calc等）。预计基于网络的决策表编辑器将包含在不久的将来发布中。

决策表是一个古老的概念（用软件术语来说），但多年来已证明有用。简而言之，在Drools中，决策表是一种生成从输入到电子表格中的数据驱动的规则的方法。数据采集​​和操作电子表格的所有常用功能都可以利用。

#### [6.5.1。何时使用决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_when_to_use_decision_tables)

如果存在可以表达为规则模板和数据的规则，则将决策表视为行动的一个过程：决策表的每一行都提供与模板组合以生成规则的数据。

许多企业已经使用电子表格来管理数据，计算等。如果您很乐意继续这种方式，您还可以通过这种方式来管理业务规则。这也假定您很乐意管理.xls 或.csv 文件中的规则包。决策表不推荐用于不遵循一组模板的规则，也不建议使用少量规则（或者对Excel或OpenOffice.org等软件不喜欢）。它们是理想的，可以控制可以编辑规则的哪些参数，而不需要直接暴露规则。

决策表还提供了一定程度的基础对象模型的绝缘。

#### [6.5.2。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview_2)

以下是一些真实世界决策表的例子（稍作修改以保护无辜者）。

*图149.使用Excel编辑决策表*

*图150.规则行的多个操作*

*图151.使用OpenOffice.org*

在上面的例子中，决策表的技术方面已经崩溃了（使用标准的电子表格功能）。

规则从第17行开始，每行产生一条规则。条件在列C，D，E等中，动作在屏幕外。单元格中的值非常简单，它们的含义由行16中的标题指示。列B只是一个描述。习惯上使用颜色来明确表中不同区域的含义。

|  |  |
| --- | --- |
|  | 请注意，尽管决策表看起来像是自上而下处理，但事实并非如此。理想情况下，创建规则时不考虑行的顺序，仅仅因为这使得维护更容易，因为行不需要一直移动。 |

由于每行都是规则，所以适用相同的原则。由于规则引擎处理事实，所有匹配的规则可能会触发。（有些人对此感到困惑，当规则触发并模拟一个非常简单的决策表时，可能会清除议程，只有第一个匹配会影响某个操作。）另请注意，您可以在一个电子表格中包含多个表格。这样，可以将规则分组到他们共享公共模板的地方，然而在一天结束时，它们全部组合成一个规则包。决策表本质上是一种自动生成DRL规则的工具。

*图152.使用多个表来分组规则的真实世界示例*

#### [6.5.3。决策表如何工作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_how_decision_tables_work)

要记住的关键点是，在决策表中，每行都是一条规则，该行中的每一列都是该规则的条件或操作。

*图153.行和列*

电子表格查找RuleTable 关键字以指示规则表（起始行和列）的开始。其他关键字也用于定义其他包级别属性（稍后介绍）。将关键字保留在一列中非常重要。按照惯例，第二列（“B”）用于此目的，但它可以是任何列（约定是在左边为笔记留下空白）。在下面的图中，C实际上是它开始的列。左侧的所有内容都被忽略。

如果我们扩大隐藏的部分，它开始变得更有意义它是如何工作的; 请注意C列中的关键字。

*图154.扩展了规则模板*

现在可以看到使其工作的隐藏的魔法。RuleSet关键字指示将在包含所有规则的规则包中使用的名称。该名称是可选的，使用默认值，但必须在单元格右侧立即有RuleSet关键字。

列C中可见的其他关键字是后面将介绍的导入和顺序。RuleTable关键字很重要，因为它表示将遵循一些规则，基于一些规则模板。在RuleTable关键字后面有一个名称，用于为生成的规则的名称添加前缀。附加表单名称和行号以保证唯一的规则名称。

|  |  |
| --- | --- |
|  | 与表格名称结合的RuleTable名称在同一KieBase中的所有电子表格文件中必须是唯一的。如果情况并非如此，一些规则可能会有相同的名称，只有其中一个会被应用。要显示这种被忽略的规则，请[提高此类规则名称冲突的严重性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_changingthedefaultbuildresultseverity)。 |

RuleTable的列表示规则开始的列; 左侧的列将被忽略。

|  |  |
| --- | --- |
|  | 通常，关键字构成名称 - 值对。 |

参考第14行（RuleTable后面的行），关键字CONDITION和ACTION指示下列各列中的数据适用于规则的LHS部分或RHS部分。规则上还有其他属性，也可以通过这种方式进行选择性设置。

第15行包含ObjectType的声明 。此行中的内容是可选的，但如果此选项未使用，则该行必须留空; 然而这个选项通常被认为是非常有用的。使用此行时，下面单元格（第16行）中的值将成为该对象类型的约束。在上面的例子中，它产生Person(age=="42")并且Cheese(type=="stilton")在42和“stilton”来自第18行。在上面的例子中，“==”是隐式的; 如果只给出了一个字段名称，那么翻译器会假定它是生成完全匹配的。

|  |  |
| --- | --- |
|  | 一个ObjectType声明可以跨越列（通过合并单元格），这意味着合并范围以下的所有列将被合并为一个模式中一次匹配单个事实的一组约束，而非包含相同的ObjectType，但导致不同的模式，可能匹配不同或相同的事实。 |

第16行包含规则模板本身。他们可以使用“$ param”占位符来指示插入来自下面单元格的数据。（对于多次插入，使用“$ 1”，“$ 2”等，表示下面单元格中逗号分隔列表中的参数。）第17行被忽略; 它可能包含该栏目的文字说明。

第18行和第19行显示的数据将与第15行中的模板组合（插值），以生成规则。如果单元格不包含数据，则其模板将被忽略。（这意味着某些条件或操作不适用于该规则行。）读取规则行直到出现空行。一张表中可以存在多个RuleTables。第20行包含另一个关键字和一个值。像这样的关键字的行位置并不重要（大多数人把它们放在顶部），但是它们的列应该与RuleTable或RuleSet关键字应该出现的位置相同。在我们的案例中，列C被选为重要的，但是可以使用任何其他列。

在上面的例子中，规则会像下面一样呈现（因为它使用“ObjectType”行）：

//一行18

规则“Cheese\_fans\_18”

当

人（年龄== “42”）

奶酪（类型== “斯蒂尔顿”）

，然后

list.add（ “老人斯蒂尔顿”）;

结束

|  |  |
| --- | --- |
|  | 这些约束条件被解释为单个约束条件，age=="42"并被type=="stilton"添加到上面单元格中的相应对象类型中。如果上面的单元格被跨越，那么在一个“列”上可能会有多个约束。 |
|  | 非常大的决策表可能会有非常大的内存要求。 |

#### [6.5.4。电子表格语法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spreadsheet_syntax)

##### [6.5.4.1。电子表格结构](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spreadsheet_structure)

有两种类型的矩形区域定义用于生成DRL文件的数据。一个标有单元格的标签RuleSet定义除规则之外的所有DRL项目。另一个可能会重复出现，并且位于以内容开头的单元格的右侧和下方RuleTable。这些领域代表了实际的决策表，每个领域都产生了一套类似结构的规则。

一个规则集区域可以包含单元格对，一个RuleSet单元格在单元格的下面，并且包含一个关键字，用于指定在同一行中包含的另一个值的类型。

规则表区域的列定义了从其派生的规则左侧的模式和约束，规则后果的动作以及单个规则属性的值。因此，规则表区域应该包含一个或多个列，既适用于条件也适用于操作，并且可以任意选择规则属性的列，每个列最多一列。标记单元格的行后面的前四行标记RuleTable为标题区域，主要用于构造规则的代码定义。它是这四个标题行下面的任何额外行，它们产生了另一个规则，其数据提供了规则表标题中定义的代码中的变体。

所有关键字不区分大小写。

只有第一张工作表才会被检查用于决策表。

##### [6.5.4.2。规则集条目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_set_entries)

规则集区域中的条目可以定义DRL构造（规则除外），并指定规则属性。虽然可以重复使用结构的条目，但每个规则属性最多只能给出一次，并且它适用于所有规则，除非它被规则表区域中定义的相同属性取代。

条目必须以垂直堆叠的单元对序列给出。第一个包含关键字和右边的值，如下表所示。只要标记RuleSet为“包含关键字的列” 被支持，该单元对序列可能会被空行或甚至规则表中断。

| *表6.“规则集”区域中的条目* | | |
| --- | --- | --- |
| **关键词** | **值** | **用法** |
| 规则集 | 生成的DRL文件的包名称。可选，默认是rule\_table。 | 必须是第一次输入。 |
| 顺序 | “对或错”。如果是“真实”，那么突出显示就是用来确保规则从上到下起火。 | 可选，最多一次。如果省略，则不执行射击命令。 |
| SequentialMaxPriority | 整数数值 | 可选，最多一次。在顺序模式下，该选项用于设置突出显示的起始值。如果省略，则默认值为65535。 |
| SequentialMinPriority | 整数数值 | 可选，最多一次。在顺序模式下，此选项用于检查是否违反了最小显着性值。如果省略，则默认值为0。 |
| EscapeQuotes | “对或错”。如果是“真”，则引号会被转义，以便它们在字面上出现在DRL中。 | 可选，最多一次。如果省略，引号将被转义。 |
| NumericDisabled | “对或错”。如果为“true”，则字符串表示形式用于DRL而不是来自Numeric单元格的double值 | 可选，最多一次。如果省略，则使用double值。 |
| 进口 | 要导入的Java类的逗号分隔列表。 | 可选，可以重复使用。 |
| 变量 | DRL全局变量的声明，即一个后跟变量名称的类型。多个全局定义必须用逗号分隔。 | 可选，可以重复使用。 |
| 功能 | 根据DRL语法，一个或多个函数定义。 | 可选，可以重复使用。 |
| 查询 | 根据DRL语法，一个或多个查询定义。 | 可选，可以重复使用。 |
| 宣布 | 根据DRL语法，一个或多个声明式类型。 | 可选，可以重复使用。 |

|  |  |
| --- | --- |
|  | 在某些语言环境中，MS Office，LibreOffice和OpenOffice将对"不同的双引号进行编码，这将导致编译错误。差异往往很难看到。例如：“A”会失败，但"A"会起作用。 |

要定义适用于生成的DRL文件中所有规则的规则属性，您可以使用下表中的任何条目。但是请注意，必须使用正确的关键字。而且，每个属性只能使用一次。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 规则集区域中指定的规则属性将影响同一包中的所有规则资产（不仅仅在电子表格中）。除非您确定电子表格是包中唯一的一个规则资产，否则建议您不是在规则集区域中指定规则属性，而是在每个规则的规则表列中指定规则属性。 | | |
| *表7.规则集区域中的规则属性条目* | | | |
| **关键词** | | **初始** | **值** |
| 优先 | | P | 一个定义规则“显着性”值的整数。由“顺序”标志覆盖。 |
| 期限 | | d | 一个长整型值，用于定义规则的“持续时间”值。 |
| TIMER | | Ť | 定时器定义。请参阅“定时器和日历”。 |
| 启用 | | 乙 | 一个布尔值。“真”使规则成为可能; “false”会禁用该规则。 |
| CALENDARS | | Ë | 日历定义。请参阅“定时器和日历”。 |
| NO-LOOP | | ü | 一个布尔值。“真”禁止由于其结果所做更改而导致的规则循环。 |
| LOCK-ON-ACTIVE | | 大号 | 一个布尔值。“true”禁止在同一规则流或议程组中设置此标志的所有规则的额外激活。 |
| 自动对焦 | | F | 一个布尔值。对于议程组中的规则而言，“真实”会导致规则的激活，从而自动将重点集中到该组。 |
| 激活GROUP | | X | 识别激活（或XOR）组的字符串。激活组中只有一个规则会触发，即第一个激活组将取消同一组内其他规则的任何现有激活。 |
| 议程-GROUP | | G | 一个标识一个议程组的字符串，必须通过给它一个“焦点”来激活它，这是控制规则组之间流程的一种方式。 |
| 规则流-GROUP | | [R | 标识规则流组的字符串。 |

##### [6.5.4.3。规则表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_tables)

所有规则表都以一个包含“RuleTable”的单元格开始，可以在同一个单元格中随后选择一个字符串。该字符串用作从该规则表派生的所有规则的名称的起始部分，并附加行号以区分。（这种自动命名可以通过使用NAME列来覆盖。）定义此Rule Table规则的所有其他单元格位于该单元格的右下方。

下一行定义了列的类型，每列导致条件或结果的一部分，或者提供一些规则属性，规则名称或注释。下表显示哪些列标题可用; 根据显示上一节中给出的规则属性条目的表格，可以使用其他列。请注意，每个属性列最多只能使用一次。对于列标题，请使用关键字或以这些表格的“初始”列中给出的字母开头的任何其他单词。

| *表8.规则表中的列标题* | | | |
| --- | --- | --- | --- |
| **关键词** | **初始** | **值** | **用法** |
| 名称 | ñ | 提供从该行生成的规则的名称。默认值是根据RuleTable标记和行号后面的文本构造的。 | 最多只有一列 |
| 描述 | 一世 | 文本，导致在生成的规则中发表评论。 | 最多只有一列 |
| 条件 | C | 代码片段和插值用于在条件中的模式中构建约束。 | 每个规则表至少有一个 |
| 行动 | 一个 | 代码片断和插值用于构建规则后果的操作。 | 每个规则表至少有一个 |
| 元数据 | @ | 代码片段和插值用于构建规则的元数据条目。 | 可选，任意数量的列 |

给定一个标题为CONDITION的列，连续行中的单元格会产生一个条件元素。

* CONDITION下面的第一个单元格中的文本发展成规则条件的模式，下一行中的代码段成为约束条件。如果单元格与一个或多个邻居合并，则会形成具有多个约束的单个模式：所有约束都合并到一个带括号的列表中，并附加到此单元格中的文本。单元格可以留空，这意味着下一行中的代码片段必须自己产生有效的条件元素。

要包含没有约束的模式，您可以将模式写在文本的前面以获取其他模式。

该模式可以写有或没有一对空括号。一个“from”子句可以被附加到该模式。

如果模式以“eval”结尾，代码片段应该生成布尔表达式，以便在“eval”之后包含在一对括号中。

* CONDITION下面第二个单元格中的文本分两步处理。
  1. 该单元格中的代码片段通过插入列中较下一个单元格的值进行修改。如果你想创建一个由“==”与下面单元格的值进行比较组成的约束条件，单独的字段选择器就足够了。任何其他比较运算符必须被指定为代码片段中的最后一项，并且下面单元格的值被附加。对于所有其他约束形式，您必须标记用符号包含单元格内容的位置$param。多个插入是可能的，通过使用码元$1，$2等等，和值在下面的细胞中的逗号分隔的列表。

根据图案的文本forall(定界符){片段}通过重复膨胀片段一次为每个在下面的每个单元的值的逗号分隔的列表的值的，插入到位码元的值$，并通过由在接合这些扩展给定分隔符。请注意，该构造可能被其他文本包围。

* 1. 如果前一行中的单元格不为空，则将完成的代码片段添加到该单元格的条件元素中。一对括号会自动提供，如果将多个约束添加到合并单元格中的模式，则会自动提供分隔逗号。

如果上面的单元格为空，则插值结果将按原样使用。

* CONDITION下面第三个单元格中的文本仅用于文档。它应该用来向读者指出专栏的用途。
* 从第四行开始，非空白条目提供如上所述的插值数据。空白单元格会导致忽略此规则的条件元素或约束条件。

给定一个由ACTION组成的列，连续行中的单元格产生一个行动陈述。

* ACTION下面的ACTION文本是可选的。如果存在，它被解释为对象引用。
* ACTION下面的第二个单元格中的文本分两步处理。
  1. 该单元格中的代码片段通过插入列中较下一个单元格的值进行修改。对于单数插入，用符号标记包含单元格内容的位置$param。多个插入是可能的，通过使用码元$1，$2等等，和值在下面的细胞中的逗号分隔的列表。

没有内插的方法调用可以通过没有任何标记符号的文本来实现。在这种情况下，请使用下面一行中的任何非空白条目来包含该语句。

这个构造也可以在这里找到。

* 1. 如果第一个单元格不是空的，则其文本，后面是句点，第二个单元格中的文本和终止分号将被串联在一起，从而产生方法调用，该方法调用作为后果的操作语句添加。

如果上面的单元格为空，则插值结果将按原样使用。

* ACTION下面第三个单元格中的文本仅用于文档。它应该用于向读者指出专栏的用途。
* 从第四行开始，非空白条目提供如上所述的插值数据。空白单元格会导致省略此规则的操作语句。

|  |  |
| --- | --- |
|  | 在大多数情况下使用$1而不是$param作品，但如果替换文本包含逗号，则会失败：然后，只插入第一个逗号前面的部分。明智地使用这个“缩写”。 |

给定一个以METADATA开头的列，连续行中的单元格会为生成的规则生成元数据注释。

* METADATA下面第一个单元格中的文本被忽略。
* 如上所述，METADATA下面第二个单元格中的文本使用来自规则行单元格的值进行插值。元数据标记字符@会自动添加前缀，因此不应将其包含在此单元格的文本中。
* METADATA下面第三个单元格中的文本仅用于文档。它应该用来向读者指出专栏的用途。
* 从第四行开始，非空白条目提供如上所述的插值数据。空白单元格导致省略此规则的元数据注释。

##### [6.5.4.4。例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_examples)

以下示例说明了各种插值。

*例85.插值单元格数据*

如果模板是Foo(bar == $param)和单元格42，那么结果是Foo(bar == 42)。

如果模板Foo(bar < $1, baz == $2)和单元格包含42,43，结果将会是Foo(bar < 42, baz ==43)。

forall(&&){bar != $}包含42,43结果的单元格模板bar != 42 && bar != 43。

下一个示例演示了定义模式类型的单元格和其下面的代码片段的联合效果。

该电子表格部分显示了Person类型声明如何跨越2列，因此这两个约束将显示为Person(age == …​, type == …​)。由于只有字段名称出现在代码片段中，它们意味着平等测试。

在下面的例子中使用了标记符号$param。

这一列的结果是模式Person(age == "42"))。您可能已经注意到标记和运算符“==”是多余的。

下一个例子说明可以省略拖尾的插入标记。

在这里，隐含从单元格附加值，导致 Person(age < "42")).

您可以提供绑定变量的定义，如下例所示。

这里的结果是c: Cheese(type == "stilton").注意引号是自动提供的。实际上，任何东西都可以放在对象类型的行中。除了绑定变量的定义之外，它也可以是一个附加的模式，可以从字面上插入。

下面显示了插入单个值的操作语句的简单构造。

ACTION标题下面的单元格留空。使用这种风格，任何事情都可以放在结果中，而不仅仅是一个方法调用。（同样的技术也适用于CONDITION列。）

下面是一个综合示例，展示了各种列标题的使用。在列标题下面没有值（如NO-LOOP列中）没有错误：这里，属性不会应用于任何规则中。

*图155.导入，标题等关键字的使用示例*

最后，这里是一个导入，变量和函数的例子。

*图156.函数关键字的使用示例*

同一单元格中的多个软件包名称必须用逗号分隔。此外，类型和变量名称对必须用逗号分隔。但是，函数必须按照出现在DRL文件中的方式编写。这应该与“RuleSet”关键字出现在同一列中; 它可能在所有规则行之上，之间或之下。

|  |  |
| --- | --- |
|  | 重复使用Import，Variables，Functions和Queries可能会更方便，而不是将几个定义打包到单个单元格中。 |

#### [6.5.5。创建和集成基于电子表格的决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_and_integrating_spreadsheet_based_decision_tables)

使用基于电子表格的决策表的API位于drools-decisiontables模块中。实际上只有一个类可以看SpreadsheetCompiler。该课程将采用各种格式的电子表格，并在DRL中生成规则（然后您可以以正常方式使用该规则）。如果SpreadsheetCompiler需要，可以用它来生成部分规则文件，并在事实之后将其组装成完整的规则包（如果需要，可以将规则的技术和非技术方面分开）。

要开始，可以使用示例电子表格作为基础。或者，如果正在使用插件（Rule Workbench IDE），向导可以从模板生成电子表格（编辑它需要使用xls兼容的电子表格编辑器）。

*图157. IDE中的向导*

#### [6.5.6。管理决策表中的业务规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_business_rules_in_decision_tables)

##### [6.5.6.1。工作流程和协作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_workflow_and_collaboration)

电子表格是完善的商业工具（使用超过25年）。决策表适合密切IT和领域专家之间的合作，同时向业务分析人员明确业务规则，这是理想的分离问题。

通常，编写规则的整个过程（提出一个新的决策表）将会是这样的：

1. 业务分析师需要一个模板决策表（来自存储库或来自IT）
2. 决策表业务语言描述被输入到表格中
3. 决策表规则（行）被输入（大致）
4. 决策表提交给技术资源，他将业务语言（描述）映射到脚本（当然，这可能涉及软件开发，如果它是新的应用程序或数据模型）
5. 技术人员回传并与业务分析师一起审查修改。
6. 业务分析师可以根据需要继续编辑规则行（移动列也很好等）。
7. 同时，技术人员可以为规则开发测试用例（与业务分析师联系），因为一旦系统运行，这些测试用例可用于验证规则和规则更改。

##### [6.5.6.2。使用电子表格功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_spreadsheet_features)

像Excel这样的应用程序的功能可以用来帮助将数据输入电子表格，例如验证字段。存储在其他工作表中的列表可用于为单元格提供有效的值列表，如下图所示。

有些应用程序提供了保留更改历史记录的有限功能，但建议使用另一种版本控制方法。当随着时间的推移对规则进行更改时，旧版本会被归档（存在许多开源解决方案，例如Subversion或Git）。

#### [6.5.7。规则模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_templates)

与决策表（但不一定需要电子表格）相关的是“规则模板”（位于drools-templates模块中）。这些使用任何表格数据源作为规则数据的来源 - 填充模板以生成许多规则。这既可以实现更灵活的电子表格，也可以实现现有数据库中的规则（以预先开发模板以生成规则为代价）。

使用规则模板时，数据与规则分开，并且规则的哪一部分是数据驱动的没有限制。所以尽管你可以在决策表中做所有你可以做的事情，但你也可以做以下事情：

* 将数据存储在数据库中（或其他任何格式）
* 有条件地根据数据中的值生成规则
* 为规则的任何部分使用数据（例如条件运算符，类名称，属性名称）
* 通过相同的数据运行不同的模板

例如，显示了一个更经典的决策表，但没有规则元数据的任何隐藏行（因此电子表格仅包含生成规则的原始数据）。

*图158.模板数据*

请参阅上述电子表格示例下载中的ExampleCheese.xls。

如果这是一个常规决策表，则在行1之前以及包含规则元数据的行1和行2之间将存在隐藏行。使用规则模板，数据与规则完全分开。这有两个方便的后果 - 您可以将多个规则模板应用于相同的数据，并且您的数据根本不依赖于您的规则。那么模板是什么样的？

1 **template** header

2 age

3 type

4 log

5

6 package org.drools.examples.templates;

7

8 global java.util.List list;

9

10 **template** "cheesefans"

11

12 rule "Cheese fans\_@{row.rowNumber}"

13 when

14 Person(age == @{age})

15 Cheese(type == "@{type}")

16 then

17 list.add("@{log}");

18 end

19

20 end **template**

前面程序清单的注释：

* 第1行：所有规则模板以template header。
* 第2-4行：标题后面是按列出的数据顺序排列的列表。在这种情况下，我们称第一栏age，第二栏type和第三栏log。
* 第5行：空行表示列定义的结束。
* 第6-9行：标准规则标题文本。这是DRL的标准规则，将出现在生成的DRL的顶部。将package语句和任何导入以及全局和函数定义放入本节。
* 第10行：关键字template指示规则模板的开始。模板文件中可以有多个模板，但每个模板都应该有唯一的名称。
* 第11-18行：规则模板 - 详见下文。
* 第20行：关键字end template表示模板的结尾。

规则模板依靠MVEL使用语法@ {token\_name}进行替换。目前有一个内置表达式，即@ {row.rowNumber}，它为每行数据提供一个唯一的编号，并使您能够生成唯一的规则名称。对于每一行数据，都会生成一条规则，并将数据中的值替换为模板中的令牌。

规则模板必须包含在扩展名为.drt的文件中，并在定义kmodule.xml文件中的kbase时与相应的决策表关联，如下例所示

**<?xml version="1.0" encoding="UTF-8"?>**

<kmodule xmlns="http://drools.org/xsd/kmodule">

<kbase name="TemplatesKB" packages="org.drools.examples.templates">

<ruleTemplate dtable="org/drools/examples/templates/ExampleCheese.xls"

template="org/drools/examples/templates/Cheese.drt"

row="2" col="2"/>

<ksession name="TemplatesKS"/>

</kbase>

</kmodule>

通过上面的示例数据，将生成以下规则文件：

package org.drools.examples.templates;

global java.util.List list;

rule "Cheese fans\_1"

**when**

Person(age == 42)

Cheese(type == "stilton")

**then**

list.add("Old man stilton");

**end**

rule "Cheese fans\_2"

**when**

Person(age == 21)

Cheese(type == "cheddar")

**then**

list.add("Young man cheddar");

**end**

在这一点上，KieSession命名的“模板”和包含从模板生成的规则可以简单地创建KieContainer和使用任何其他KieSession。

KieSession ksession = kc.newKieSession( "TemplatesKS" );

*// now create some test data*

ksession.insert( **new** Cheese( "stilton", 42 ) );

ksession.insert( **new** Person( "michael", "stilton", 42 ) );

**final** List<String> list = **new** ArrayList<String>();

ksession.setGlobal( "list", list );

ksession.fireAllRules();

### [6.6。记录](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_logging)

照亮作为规则引擎的黑盒子的一种方法是玩日志级别。

一切都记录到[SLF4J](http://www.slf4j.org/)，这是一个简单的日志记录外观，可以将任何日志委托给Logback，Apache Commons Logging，Log4j或java.util.logging。将日志记录适配器的依赖项添加到您所选择的日志记录框架中。如果你还没有使用任何日志框架，你可以通过添加这个Maven依赖项来使用Logback：

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.x</version>

</dependency>

|  |  |
| --- | --- |
|  | 如果您正在开发超轻型环境，请使用slf4j-nop或slf4j-simple改为。 |

在软件包上配置日志记录级别org.drools。例如：

在Logback中，将其配置到您的logback.xml 文件中：

<configuration>

<logger name="org.drools" level="debug"/>

...

<configuration>

在Log4J中，将其配置到您的log4j.xml 文件中：

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

<category name="org.drools">

<priority value="debug" />

</category>

...

</log4j:configuration>

## [7.跑步](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_running)

这部分扩展了KIE运行部分，该部分应该首先阅读，并提供Drools运行时的细节。

### [7.1。KieRuntime](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kieruntime_3)

#### [7.1.1。入口点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_entrypoint)

该EntryPoint提供周围插入，更新和删除事实的方法。术语“入口点”与我们在工作存储器中有多个分区并且您可以选择要插入哪个分区有关。在事件处理用例中使用多个入口点更为常见，但它们也可以被纯规则应用程序使用。

所述KieRuntime接口提供了与发动机的主要相互作用。它可用于规则后果和流程操作。在本手册中，重点介绍与规则相关的方法和接口，现在将忽略有关过程的方法。但是您会注意到KieRuntime继承了WorkingMemory和的方法ProcessRuntime，从而提供了一个统一的API来处理流程和规则。当使用规则，三个接口组成KieRuntime：EntryPoint，WorkingMemory和KieRuntime本身。

*图159.入口点*

##### [7.1.1.1。插](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_insert)

为了根据a中的规则对事实进行评估KieBase，必须将其插入会话中。这是通过调用该方法完成的insert(yourObject)。当一个事实被插入到会话中时，它的一些属性可能会立即被评估（渴望评估），有些可能被延迟以供以后评估（懒惰评估）。确切的行为取决于正在使用的规则引擎算法。

|  |  |
| --- | --- |
|  | 专家系统通常使用术语断言或断言来表示系统可用的事实。但是，由于“assert”是大多数语言中的关键字，因此我们决定使用insert关键字; 在本手册中，这两个术语可以互换使用。 |

当插入一个对象时，它返回一个FactHandle。这FactHandle是用来表示插入对象的标记WorkingMemory。它也用于与WorkingMemory想要删除或修改对象的时间进行交互。

Cheese stilton = **new** Cheese("stilton");

FactHandle stiltonHandle = ksession.insert( stilton );

如KieBase部分所述，工作内存可以在两种断言模式下运行：平等或身份。身份是默认设置。

身份意味着工作内存使用一个IdentityHashMap来存储所有断言的对象。新的实例断言总是导致新的返回FactHandle，但是如果一个实例再次被声明，那么它将返回原始的事实句柄，即它忽略了同一对象的重复插入。

平等意味着工作内存使用a HashMap来存储所有已声明的对象。FactHandle如果插入的对象equal()/hashcode()与已存在的事实不相等（根据其方法），则对象实例断言只会返回一个新的事实。

##### [7.1.1.2。删除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_delete)

为了从会话中删除事实，使用了该方法delete()。当一个事实被删除时，任何被激活并取决于该事实的匹配都将被取消。请注意，有可能具有取决于事实不存在的规则，在这种情况下，删除事实可能会导致规则激活。（请参阅not和exists关键字）。

|  |  |
| --- | --- |
|  | 专家系统通常使用术语“ 收回”或“ 收回”来指代从工作存储器中删除事实的操作。Drools喜欢关键字delete与关键字对称insert; Drools也支持关键字retract，但它被弃用赞成delete。在本手册中，这两个术语可以互换使用。 |

可以使用FactHandle插入调用返回的内容来完成缩回。在规则的右侧delete使用该语句，该语句使用简单的对象引用。

Cheese stilton = **new** Cheese("stilton");

FactHandle stiltonHandle = ksession.insert( stilton );

....

ksession.delete( stiltonHandle );

##### [7.1.1.3。更新](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_update)

规则引擎必须被通知修改的事实，以便它们可以被重新处理。您必须使用该update()方法来通知已WorkingMemory更改的对象，以便那些无法通知WorkingMemory自己的对象。请注意，update()总是将修改的对象作为第二个参数，它允许您为不可变对象指定新实例。在规则的右侧，modify推荐使用该语句，因为它会进行更改并在单个语句中通知引擎。或者，在通过调用setter方法的调用更改事实对象的字段值之后，必须update立即调用事件对象，然后再更改其他事实，否则会导致规则引擎中的索引问题。修改语句避免了这个问题。

Cheese stilton = **new** Cheese("stilton");

FactHandle stiltonHandle = workingMemory.insert( stilton );

...

stilton.setPrice( 100 );

workingMemory.update( stiltonHandle, stilton );

#### [7.1.2。RuleRuntime](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleruntime)

RuleRuntime提供对议程的访问权限，允许执行查询，并允许您访问指定的入口点。

*图160. RuleRuntime*

##### [7.1.2.1。询问](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_query)

查询用于根据模式检索事实集，因为它们在规则中使用。模式可以使用可选参数。可以在知识库中定义查询，从中调用它们以返回匹配结果。在对结果集合进行迭代时，可以使用查询中绑定的任何标识符通过调用get具有绑定变量名称作为参数的方法来访问相应的事实或事实字段。如果绑定涉及事实对象，则可以通过调用getFactHandle再次将变量的名称作为参数来检索它的FactHandle 。

*图161. QueryResults*

*图162. QueryResultsRow*

*例子86.简单的查询例子*

QueryResults results =

ksession.getQueryResults( "my query", **new** Object[] { "string" } );

**for** ( QueryResultsRow row : results ) {

System.out.println( row.get( "varName" ) );

}

##### [7.1.2.2。在线查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_live_queries)

通过遍历返回的集合来调用查询和处理结果不是监视随时间变化的好方法。

为了减轻这一点，Drools提供了Live Queries，它有一个附加的监听器，而不是返回一个可迭代的结果集。这些实时查询通过创建视图并为此视图的内容发布更改事件而保持打开状态。要激活，您可以使用参数开始查询，并聆听结果视图中的更改。该dispose方法终止查询并停止这种反应情景。

*例87.实现ViewChangedEventListener*

**final** List updated = **new** ArrayList();

**final** List removed = **new** ArrayList();

**final** List added = **new** ArrayList();

ViewChangedEventListener listener = **new** ViewChangedEventListener() {

**public** **void** **rowUpdated**(Row row) {

updated.add( row.get( "$price" ) );

}

**public** **void** **rowRemoved**(Row row) {

removed.add( row.get( "$price" ) );

}

**public** **void** **rowAdded**(Row row) {

added.add( row.get( "$price" ) );

}

};

*// Open the LiveQuery*

LiveQuery query = ksession.openLiveQuery( "cheeses",

**new** Object[] { "cheddar", "stilton" },

listener );

...

...

query.dispose() *// calling dispose to terminate the live query*

Drools博客文章包含一个用于实时查询的Glazed Lists集成示例：

<http://blog.athico.com/2010/07/glazed-lists-examples-for-drools-live.html>

#### [7.1.3。StatefulRuleSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_statefulrulesession)

该StatefulRuleSession由继承KieSession，并提供了相关的规则，方法是从发动机的外相关。

*图163. StatefulRuleSession*

##### [7.1.3.1。议程过滤器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agenda_filters)

*图164.议程过滤器*

`AgendaFilter`对象是过滤器接口的可选实现，用于允许或拒绝发起匹配。你过滤的内容完全取决于实施。Drools 4.0用于提供一些开箱即用的过滤器，这些过滤器在drools 5.0 knowledge-api中没有公开，但它们易于实现并且可以引用Drools 4.0代码库。

要使用过滤器，请在调用时指定它fireAllRules()。以下示例仅允许以字符串结尾的规则"Test"。所有其他人将被过滤掉。

ksession.fireAllRules( **new** RuleNameEndsWithAgendaFilter( "Test" ) );

### [7.2。议程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agenda_2)

议程是一个Rete功能。在对此采取行动期间WorkingMemory，规则可能完全匹配并有资格执行; 单个工作记忆操作可能导致多个符合条件的规则。当规则完全匹配时，创建匹配，引用规则和匹配的事实，并放置到议程中。议程使用冲突解决策略控制这些匹配的执行顺序。

发动机通过两个阶段重复循环：

1. 工作记忆行为。这是大部分工作发生的地方，无论是在Consequence（RHS本身）还是主要的Java应用程序过程中。一旦结果完成或主Java应用程序进程fireAllRules()将引擎切换调用到议程评估阶段。
2. 议程评估。这会尝试选择要触发的规则。如果未找到规则，则会退出，否则会触发找到的规则，将阶段切换回工作内存操作。

*图165.两阶段执行*

该过程重复，直到议程清晰，在这种情况下控制返回到调用应用程序。当工作记忆行动发生时，没有规则被解雇。

*图166.议程*

#### [7.2.1。解决冲突](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conflict_resolution_2)

如果议程上有多条规则，则需要解决冲突。（有关这方面的基础知识，请参见“快速入门”一章。）由于解雇规则可能会对工作记忆产生副作用，因此规则引擎需要知道规则应以何种顺序触发（例如，解雇规则A可能导致规则B从议程中删除）。

Drools采用的默认冲突解决策略是：Salience和LIFO（后进先出）。

最明显的是显着性（或优先级），在这种情况下，用户可以指定某个规则比其他规则具有更高的优先级（通过给予更高的优先级）。在这种情况下，更重要的规则将是首选。LIFO优先级基于分配的工作内存操作计数器值，同一操作期间创建的所有规则都会接收相同的值。具有相同优先级值的一组发射的执行顺序是任意的。

作为一般规则，不要指望以任何特定顺序解雇规则，并且在不担心“流量”的情况下编写规则是一个好主意。然而，当需要流量时，存在多种可能性，包括但不限于：议程组，规则流程组，激活组，控制/信号量事实。这些在后面的章节中讨论。

Drools 4.0支持自定义冲突解决策略; 虽然这种功能仍然存在于Drools中，但它尚未通过Drools 5.0中的知识api向最终用户展示。

#### [7.2.2。AgendaGroup](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_agendagroup)

*图167.议程组*

议程小组是分配议程中的规则（实际上匹配）的一种方式。在任何时候，只有一个组有“关注”，这意味着该组中的规则的匹配只会生效。您也可以制定“自动对焦”规则，这意味着当该规则的条件成立时，将重点放在其议程组。

议程组在CLIPS术语中被称为“模块”。虽然最好设计不需要控制流量的规则，但这并非总是可行。议程组提供了一种方便的方式在分组规则之间创建“流程”。您可以从规则引擎内或通过API切换具有焦点的组。如果您的规则明确需要处理多个“阶段”或“序列”，请考虑将议程组用于此目的。

每一次setFocus()被称为它推动议程集团进入堆栈。当焦点组为空时，它将从堆栈中弹出，并且现在处于最前面的焦点组进行评估。议程组可以出现在堆栈的多个位置。默认议程组是“MAIN”，所有规则中没有指定议程组在此组中。它也始终是堆栈中的第一个组，默认情况下最初是关注焦点。

ksession.getAgenda().getAgendaGroup( "Group A" ).setFocus();

该clear()方法可用于在有机会开火之前取消属于给定议程组的规则所产生的所有激活。

ksession.getAgenda().getAgendaGroup( "Group A" ).clear();

请注意，由于Drools使用的phreak算法的惰性，激活默认仅在激发时刻实现，但可以预测评估，然后在事实发生时激活给定规则@Propagation(IMMEDIATE)按照传播模式部分中的说明通过注释插入到会话中。

#### [7.2.3。的ActivationGroup](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_activationgroup)

*图168. ActivationGroup*

激活组是由相同的“激活组”规则属性绑定在一起的一组规则。在这个组中只有一条规则可以触发，而在此规则解除后，所有其他规则都从议程中取消。该clear()方法可以在任何时候调用，在有机会开火之前取消所有的激活。

ksession.getAgenda().getActivationGroup( "Group B" ).clear();

#### [7.2.4。RuleFlowGroup](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleflowgroup)

*图169. RuleFlowGroup*

规则流组是由“ruleflow-group”规则属性关联的一组规则。这些规则只能在组激活时触发。只有当规则流图的阐述到达代表组的节点时，组本身才会变为活动状态。这里也clear()可以随时调用该方法来取消仍然保留在议程中的所有匹配。

ksession.getAgenda().getRuleFlowGroup( "Group C" ).clear();

### [7.3。事件模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_model_2)

事件包提供了通知规则引擎事件的方法，包括规则触发，被声明的对象等。例如，这允许您将日志和审计活动与应用程序的主体部分（和规则）分开。

在WorkingMemoryEventManager允许添加和删除侦听程序，因此，对于工作记忆和议程事件就可以听了。

*图170. WorkingMemoryEventManager*

以下代码片段显示了一个简单的日程监听器是如何声明并附加到会话中的。它会在发射后打印匹配。

*例88.添加一个AgendaEventListener*

ksession.addEventListener( **new** DefaultAgendaEventListener() {

**public** **void** **afterMatchFired**(AfterMatchFiredEvent event) {

**super**.afterMatchFired( event );

System.out.println( event );

}

});

Drools还提供DebugRuleRuntimeEventListener并DebugAgendaEventListener使用调试打印语句实现每种方法。要打印所有工作记忆事件，请添加如下所示的侦听器：

*例89.添加一个DebugRuleRuntimeEventListener*

ksession.addEventListener( **new** DebugRuleRuntimeEventListener() );

目前支持的事件有：

* MatchCreatedEvent
* MatchCancelledEvent
* BeforeMatchFiredEvent
* AfterMatchFiredEvent
* AgendaGroupPushedEvent
* AgendaGroupPoppedEvent
* ObjectInsertEvent
* ObjectDeletedEvent
* ObjectUpdatedEvent
* ProcessCompletedEvent
* ProcessNodeLeftEvent
* ProcessNodeTriggeredEvent
* ProcessStartEvent

### [7.4。StatelessKieSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_statelesskiesession_2)

该StatelessKieSession包装的KieSession，而不是扩展它。其主要重点在于决策服务类型场景。它避免了需要打电话dispose()。无状态会话不支持迭代插入和fireAllRules()来自Java代码的方法调用; 调用行为execute()是一种单步方法，它将在内部实例化一个KieSession，添加所有用户数据并执行用户命令，调用fireAllRules()，然后调用dispose()。尽管使用这个类的主要方法是通过BatchExecution接口Command支持的（子接口）CommandExecutor，但是当需要简单对象插入时提供了两种便利方法。该CommandExecutor和BatchExecution在自己的部分谈到细节。

*图171.无状态KieSession*

我们简单的例子展示了一个使用便利API执行给定的Java对象集合的无状态会话。它将迭代集合，依次插入每个元素。

*示例90.使用Collection执行简单的StatelessKieSession*

StatelessKieSession ksession = kbase.newStatelessKieSession();

ksession.execute( collection );

如果这是作为一个命令完成的，那么它将如下所示：

*示例91.使用InsertElements命令执行简单的StatelessKieSession*

ksession.execute( CommandFactory.newInsertElements( collection ) );

如果你想插入集合本身和集合的个别元素，那么CommandFactory.newInsert(collection)就可以完成这项工作。

的方法CommandFactory创建支持的命令，所有这些都可以使用XStream的和被整理BatchExecutionHelper。BatchExecutionHelper提供了有关XML格式的详细信息，以及如何使用Drools的管道自动化的编组BatchExecution和ExecutionResults。

StatelessKieSession支持全局变量，范围有很多种。我将首先介绍非命令方式，因为命令的范围是特定的执行调用。全局可以通过三种方式解决。

* StatelessKieSession方法getGlobals()返回一个Globals实例，它提供对会话全局的访问。这些是所有执行调用共享的。请谨慎使用可变全局变量，因为执行调用可以在不同的线程中同时执行。

*例92.会话范围全局*

StatelessKieSession ksession = kbase.newStatelessKieSession();

*// Set a global hbnSession, that can be used for DB interactions in the rules.*

ksession.setGlobal( "hbnSession", hibernateSession );

*// Execute while being able to resolve the "hbnSession" identifier.*

ksession.execute( collection );

* 使用代表是另一种全球解决方式。为全局值（赋值）赋值setGlobal(String, Object)，将值存储在内部集合中，将标识符映射到值。此内部集合中的标识符将优先于任何提供的代表。只有在此内部集合中找不到标识符时，才会使用全局代理（如果有）。
* 解决全局性的第三种方法是执行范围全局。在这里，一个Command设置全局是传递给CommandExecutor。

该CommandExecutor界面还提供了通过“输出”参数输出数据的功能。插入的事实，全局变量和查询结果都可以返回。

*例93.输出标识符*

*// Set up a list of commands*

List cmds = **new** ArrayList();

cmds.add( CommandFactory.newSetGlobal( "list1", **new** ArrayList(), **true** ) );

cmds.add( CommandFactory.newInsert( **new** Person( "jon", 102 ), "person" ) );

cmds.add( CommandFactory.newQuery( "Get People" "getPeople" );

*// Execute the list*

ExecutionResults results =

ksession.execute( CommandFactory.newBatchExecution( cmds ) );

*// Retrieve the ArrayList*

results.getValue( "list1" );

*// Retrieve the inserted Person fact*

results.getValue( "person" );

*// Retrieve the query as a QueryResults instance.*

results.getValue( "Get People" );

#### [7.4.1。顺序模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sequential_mode)

通过使用Rete，您可以进行有状态的会话，随着时间的推移可以断言和修改对象，还可以添加和删除规则。现在如果我们假设一个无状态会话会发生什么情况，在初始数据集之后没有更多数据可以被声明或修改，并且规则不能被添加或删除？当然，没有必要重新评估规则，引擎可以以简化的方式运行。

1. 通过规则集中的显着性和位置排序规则（通过在规则终端节点上设置序列属性）。
2. 为每个可能的规则匹配创建一个元素，一个元素; 元素位置表示点火顺序。
3. 关闭除了右输入对象存储器之外的所有节点存储器。
4. 断开左输入适配器节点传播，并让对象加上节点在Command对象中被引用，该对象被添加到工作存储器的列表中供以后执行。
5. 断言所有对象，并且，当所有断言完成并因此填充右输入节点存储器时，请检查命令列表并依次执行每个对象。
6. 所有得到的匹配应根据规则的确定序列号放置在元素中。记录第一个和最后一个填充元素，以减少迭代范围。
7. 迭代Matches的元素，依次执行填充元素。
8. 如果我们有最大数量的允许规则执行，我们可以及早退出我们的网络评估以激发元素中的所有规则。

将LeftInputAdapterNode不再创建一个元组，添加对象，然后传播的元组-而不是创建一个Command对象，并加入到工作记忆列表。此Command对象包含LeftInputAdapterNode对传播对象的引用。这会在插入时停止任何左输入传播，以便我们知道右输入传播永远不需要尝试与左输入进行连接（不再需要左输入内存）。所有节点关闭其内存，包括左输入元组内存，但不包括右输入对象内存，这意味着记住插入传播的唯一节点是右输入对象内存。一旦所有断言完成并且所有正确的输入内存填充完毕，我们就可以迭代列表LeftInputAdatperNode命令对象依次调用。它们会沿着网络向下传播，尝试与正确的输入对象连接，但是它们不会在左边的输入中被记住，因为我们知道将不会有进一步的对象声明并因此传播到正确的输入内存中。

不再有一个议程，有一个优先队列来安排元组; 相反，只有一些规则的元素。该序号RuleTerminalNode指示放置匹配的元素内的元素。一旦所有的Command对象都完成了，我们可以遍历我们的元素，依次检查每个元素，并且在匹配存在的情况下触发它们。为了提高性能，我们记住元素中第一个和最后一个已填充的单元格。网络被构建，每个网络都RuleTerminalNode被赋予一个基于显着编号及其被添加到网络中的顺序的序列号。

通常，右输入节点存储器是哈希映射，用于快速删除对象; 在这里，因为我们知道不会有对象删除，所以当对象的值未被编入索引时，我们可以使用列表。对于索引较大的对象哈希映射提供了性能增加; 如果我们知道一个对象类型只有少数实例，索引可能不是有利的，并且可以使用一个列表。

顺序模式只能用于无状态会话，默认情况下处于关闭状态。要打开它，请调用RuleBaseConfiguration.setSequential(true)或将rulebase配置属性设置drools.sequential为true。顺序模式可以通过调用回落到一个动态的议程setSequentialAgenda有SequentialAgenda.DYNAMIC。您也可以将“drools.sequential.agenda”属性设置为“顺序”或“动态”。

### [7.5。规则执行模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_execution_modes)

Drools为规则执行提供了两种模式 - 被动和主动。

作为一般指导原则，被动模式最适合需要明确控制引擎何时应评估和触发规则的规则引擎应用程序，或者是否适用于使用伪时钟的CEP应用程序。活动模式对规则引擎应用程序最为有效，该应用程序可以控制何时对规则进行评估并将其发送至引擎，或对使用实时时钟的典型CEP应用程序进行控制。

#### [7.5.1。被动模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_passive_mode)

在被动模式下，用户不仅负责工作内存操作，insert()而且还负责规则何时评估数据并触发生成的规则实例 - 使用fireAllRules() 。

使用被动模式的CEP应用程序的Drools代码示例大纲：

KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

config.setOption( ClockTypeOption.get("pseudo") );

KieSession session = kbase.newKieSession( conf, **null** );

SessionPseudoClock clock = session.getSessionClock();

session.insert( tick1 );

session.fireAllRules();

clock.advanceTime(1, TimeUnit.SECONDS);

session.insert( tick2 );

session.fireAllRules();

clock.advanceTime(1, TimeUnit.SECONDS);

session.insert( tick3 );

session.fireAllRules();

session.dispose();

#### [7.5.2。活动模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_active_mode)

Drools提供了一种fireUntilHalt()方法，该方法以主动模式启动引擎，这种方式在行为上是异步的，其中规则将不断评估并触发，直到进行halt()调用。

这对于需要通常称为“活动查询”的CEP方案特别有用。

请注意调用fireUntilHalt()阻塞当前线程，而引擎将启动并继续异步运行，直到halt()在KieSession中调用。因此建议fireUntilHalt()从专用线程调用，以便当前线程不会无限期地被阻塞; 这也使得当前线程能够halt()在稍后阶段调用，参考文献。下面的例子。

使用活动模式的CEP应用程序的Drools代码示例大纲：

KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

config.setOption( ClockTypeOption.get("realtime") );

KieSession session = kbase.newKieSession( conf, **null** );

**new** Thread( **new** Runnable() {

@Override

**public** **void** **run**() {

session.fireUntilHalt();

}

} ).start();

session.insert( tick1 );

... Thread.sleep( 1000L ); ...

session.insert( tick2 );

... Thread.sleep( 1000L ); ...

session.insert( tick3 );

session.halt();

session.dispose();

|  |  |
| --- | --- |
|  | 一般来说，不建议混合fireAllRules()和fireUntilHalt()，尤其是从不同的线程。然而，由于内部状态机，引擎能够安全地处理这种情况。如果fireAllRules()正在运行并发出呼叫fireUntilHalt()，引擎将等待，直到fireAllRules()完成并启动fireUntilHalt()。但是，如果fireUntilHalt()正在运行并被fireAllRules()调用，则后者将被忽略并直接返回。有关线程安全性和内部状态机的更多详细信息，请参见“改进的多线程行为”一节。 |

##### [7.5.2.1。在主动模式下自动执行KieSession操作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_performing_kiesession_operations_atomically_when_in_active_mode)

当处于主动模式时，Drools引擎控制何时评估和触发规则; 因此对KieSession的操作以线程安全的方式进行是非常重要的。此外，从客户端的角度来看，可能需要在规则评估之间在KieSession上调用多个操作，但引擎会将这些操作视为一个原子操作：例如，将多个事件插入到给定的时间，但引擎要等到所有插入完成后才再次评估规则。

Drools提供了submit()一种在主动模式下将KieSession作为线程安全原子动作分组和执行操作的方法。

Drools代码在主动模式下以原子方式执行KieSession操作的示例大纲：

KieSession session = ...;

**new** Thread( **new** Runnable() {

@Override

**public** **void** **run**() {

session.fireUntilHalt();

}

} ).start();

**final** FactHandle fh = session.insert( fact\_a );

... Thread.sleep( 1000L ); ...

session.submit( **new** KieSession.AtomicAction() {

@Override

**public** **void** **execute**( KieSession kieSession ) {

fact\_a.setField("value");

kieSession.update( fh, fact\_a );

kieSession.insert( fact\_1 );

kieSession.insert( fact\_2 );

kieSession.insert( fact\_3 );

}

} );

... Thread.sleep( 1000L ); ...

session.insert( fact\_z );

session.halt();

session.dispose();

作为提示示例，事实句柄也可以从KieSession中检索：

...

session.insert( fact\_a );

... Thread.sleep( 1000L ); ...

session.submit( **new** KieSession.AtomicAction() {

@Override

**public** **void** **execute**( KieSession kieSession ) {

**final** FactHandle fh = kieSession.getFactHandle( fact\_a );

fact\_a.setField("value");

kieSession.update( fh, fact\_a );

kieSession.insert( fact\_1 );

kieSession.insert( fact\_2 );

kieSession.insert( fact\_3 );

}

} );

...

### [7.6。传播模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_propagation_modes_2)

作为Drools引擎的默认算法引入了PHREAK，这使得规则的评估懒惰。这种新的Drools懒惰行为允许相关的性能提升，但在某些特定情况下，打破了一些Drools功能的语义。

更准确地说，在某些情况下，有必要立即将新的事实传播到会议中。例如，Drools允许通过预先添加一个'？'来在只有拉（或被动）模式下执行查询。符号调用，如下例所示：

*例子94.被动查询*

query **Q** (Integer i)

**String**( **this** == i.toString() )

end

rule R when

$i : **Integer**()

?**Q**( $i; )

then

System.out.**println**( $i );

end

在这种情况下，由于查询是被动的，因此它不应该在查询本身中插入匹配连接条件的字符串。换句话说，这个命令序列

KieSession ksession = ...

ksession.insert(1);

ksession.insert("1");

ksession.fireAllRules();

应该不会导致规则R被触发，因为满足查询条件的字符串已经在Integer之后插入，并且被动查询不应对此插入作出反应。相反，如果插入序列被反转，则规则应该触发，因为插入整数（当被动查询可以通过存在的字符串来满足时）将触发它。

不幸的是，PHREAK的懒惰性质不允许引擎对两个事实的插入顺序作出任何区分，所以规则将在两种情况下触发。在这种情况下，有必要像以前的基于RETEOO的引擎一样热切地评估规则。

在其他情况下，要求传播是渴望的，这意味着它不是直接的，但无论如何必须在引擎/议程开始计划评估之前发生。例如，当规则具有无回路或锁定激活属性时，这是必要的，事实上，当这种情况发生时，传播模式由引擎自动执行。

为了涵盖这些用例以及需要立即或积极规则评估的所有其他情况，可以通过使用**@Propagation（Propagation.Type）**注释规则本身来进行声明性指定，其中Propagation.Type是一个枚举3个可能的值：

* **IMMEDIATE**表示传播立即执行。
* **EAGER**意味着繁殖是在计划评估之前懒惰地执行但热切地评估的。
* **LAZY**意味着传播是完全懒惰的，这是默认的PHREAK行为

这意味着以下drl：

*示例95.使用被动查询的数据驱动规则*

query **Q** (Integer i)

**String**( **this** == i.toString() )

end

rule R @**Propagation**(IMMEDIATE) when

$i : **Integer**()

?**Q**( $i; )

then

System.out.**println**( $i );

end

当且仅当Integer被插入到String之后时，才会使规则R被触发，从而按照被动查询的语义行事。

### [7.7。命令和CommandExecutor](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commands_and_the_commandexecutor_2)

将CommandFactory允许在这些会议将要执行的命令，唯一的区别是无状态会话知识执行fireAllRules()在最后处置会话之前。当前支持的命令是：

* FireAllRules
* GetGlobal
* SetGlobal
* InsertObject
* InsertElements
* 询问
* StartProcess
* BatchExecution

`InsertObject`将插入一个单独的对象，并带有一个可选的“out”标识符。InsertElements将迭代一个Iterable，插入每个元素。这意味着无状态知识会话不再局限于插入对象，它现在可以启动进程或执行查询，并以任何顺序执行此操作。

*例96.插入命令*

StatelessKieSession ksession = kbase.newStatelessKieSession();

ExecutionResults bresults =

ksession.execute( CommandFactory.newInsert( **new** Cheese( "stilton" ), "stilton\_id" ) );

Stilton stilton = bresults.getValue( "stilton\_id" );

execute方法总是返回一个ExecutionResults实例，如果它们指定了一个out标识符，比如上面的“stilton\_id”，它就允许访问任何命令结果。

*例子97. InsertElements命令*

StatelessKieSession ksession = kbase.newStatelessKieSession();

Command cmd = CommandFactory.newInsertElements( Arrays.asList( Object[] {

**new** Cheese( "stilton" ),

**new** Cheese( "brie" ),

**new** Cheese( "cheddar" ),

});

ExecutionResults bresults = ksession.execute( cmd );

执行方法只允许一个命令。这就是BatchExecution进入的地方，它代表了一个从命令列表创建的复合命令。现在，执行将遍历列表并依次执行每个命令。这意味着你可以插入一些对象，启动一个进程，调用fireAllRules并执行一个查询，所有这些都是在一次execute(…​)调用中完成的，这个功能非常强大。

如前所述，StatelessKieSession将fireAllRules()在最后自动执行。然而，敏锐的读者可能已经注意到了这个FireAllRules命令，并想知道如何与无状态的KieSession一起工作。该FireAllRules命令是允许的，使用它将在最后禁用自动执行; 可以考虑使用它作为一种手动覆盖功能。

定制的XStream编组器可以与Drools Pipeline一起使用来实现XML脚本，这对于服务来说非常完美。这里有两个简单的XML示例，一个是BatchExecution，一个是BatchExecution ExecutionResults。

*例子98.简单的批量执行XML*

<batch-execution>

<insert out-identifier='outStilton'>

<org.drools.compiler.Cheese>

<type>stilton</type>

<price>25</price>

<oldPrice>0</oldPrice>

</org.drools.compiler.Cheese>

</insert>

</batch-execution>

*例99.简单执行结果XML*

<execution-results>

<result identifier='outStilton'>

<org.drools.compiler.Cheese>

<type>stilton</type>

<oldPrice>25</oldPrice>

<price>30</price>

</org.drools.compiler.Cheese>

</result>

</execution-results>

Spring和Camel涵盖在集成书中，便于声明式服务。

*示例100. BatchExecution编组为XML*

<batch-execution>

<insert out-identifier="stilton">

<org.drools.compiler.Cheese>

<type>stilton</type>

<price>1</price>

<oldPrice>0</oldPrice>

</org.drools.compiler.Cheese>

</insert>

<query out-identifier='cheeses2' name='cheesesWithParams'>

<string>stilton</string>

<string>cheddar</string>

</query>

</batch-execution>

该CommandExecutor回报的ExecutionResults，这是由管道代码片段，以及处理。上面的<批处理执行> XML示例的类似输出将是：

*示例101.将ExecutionResults编组为XML*

<execution-results>

<result identifier="stilton">

<org.drools.compiler.Cheese>

<type>stilton</type>

<price>2</price>

</org.drools.compiler.Cheese>

</result>

<result identifier='cheeses2'>

<query-results>

<identifiers>

<identifier>cheese</identifier>

</identifiers>

<row>

<org.drools.compiler.Cheese>

<type>cheddar</type>

<price>2</price>

<oldPrice>0</oldPrice>

</org.drools.compiler.Cheese>

</row>

<row>

<org.drools.compiler.Cheese>

<type>cheddar</type>

<price>1</price>

<oldPrice>0</oldPrice>

</org.drools.compiler.Cheese>

</row>

</query-results>

</result>

</execution-results>

在BatchExecutionHelper提供了一个配置XStream的实例来支持批量处理，其中，生成的XML可以用来作为消息格式，如上所示的编组。配置的转换器仅对通过Command Factory支持的命令存在。用户可以为其用户对象添加其他转换器。这对于为无状态或有状态的知识会话编写脚本非常有用，尤其是涉及到服务时。

目前没有XML模式来支持模式验证。这里概述了基本格式，drools-pipeline模块在单元测试中有一个说明性的XStreamBatchExecutionTest单元测试。根元素是<批处理执行>，它可以包含零个或多个命令元素。

*示例102.根XML元素*

<batch-execution>

...

</batch-execution>

这包含表示命令的元素列表，受支持的命令仅限于由Command Factory提供的那些命令。其中最基本的是插入对象的<insert>元素。插入元素的内容是用户对象，由XStream指定。

*例103.插入*

<batch-execution>

<insert>

...*<!-- any user object -->*

</insert>

</batch-execution>

插入元素具有“out-identifier”属性，要求插入的对象也将作为结果有效载荷的一部分返回。

*例104.使用输出标识符命令插入*

<batch-execution>

<insert out-identifier='userVar'>

...

</insert>

</batch-execution>

也可以使用<insert-elements>元素插入一组对象。该命令不支持出标识符。这org.domain.UserClass只是XStream将要序列化的说明性用户对象。

*示例105.插入元素命令*

<batch-execution>

<insert-elements>

<org.domain.UserClass>

...

</org.domain.UserClass>

<org.domain.UserClass>

...

</org.domain.UserClass>

<org.domain.UserClass>

...

</org.domain.UserClass>

</insert-elements>

</batch-execution>

虽然该out属性在返回特定实例作为结果负载方面很有用，但我们通常希望运行实际查询。支持参数和无参数查询。该name属性是要调用的查询的名称，并且out-identifier是用于<execution-results>有效负载中的查询结果的标识符。

*例106.查询命令*

<batch-execution>

<query out-identifier='cheeses' name='cheeses'/>

<query out-identifier='cheeses2' name='cheesesWithParams'>

<string>stilton</string>

<string>cheddar</string>

</query>

</batch-execution>

## [8.规则语言参考](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_droolslanguagereferencechapter)

### [8.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview_3)

Drools有一种“本地”规则语言。这种格式在标点符号方面非常轻便，并且通过“扩展器”支持自然语言和领域特定语言，这些语言允许语言变形到问题域。本章主要与这种本地规则格式一致。用于表示语法的图表被称为“铁路”图表，它们基本上是语言术语的流程图。技术上非常热衷的还可以参考DRL.g ，它是规则语言的Antlr3语法。如果您使用Rule Workbench，则可以通过内容帮助完成许多规则结构，例如，输入“ru”并按Ctrl +空格键，它将为您构建规则结构。

#### [8.1.1。规则文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_rule_file)

规则文件通常是扩展名为.drl的文件。在DRL文件中，您可以拥有多个规则，查询和函数，以及一些资源声明，如规则和查询分配并使用的导入，全局变量和属性。但是，您也可以将规则分散到多个规则文件中（在这种情况下，建议使用扩展名.rule，但不是必需的） - 跨文件传播规则可以帮助管理大量规则。DRL文件只是一个文本文件。

规则文件的整体结构是：

*例子107.规则文件*

**package** **package**-name

imports

globals

functions

queries

rules

声明元素的顺序并不重要，但声明的包名称必须是规则文件中的第一个元素。所有元素都是可选的，因此您只能使用您需要的元素。我们将在下面的章节中讨论它们中的每一个。

#### [8.1.2。什么制定规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_what_makes_a_rule)

对于不耐烦的人来说，就像早期的观点一样，一个规则具有以下粗略的结构：

rule "name"

attributes

**when**

LHS

**then**

RHS

**end**

这真的很简单。大多数标点符号是不需要的，即使是“name”的双引号也是可选的，换行符也是如此。属性很简单（总是可选的）提示规则应该如何表现。LHS是该规则的条件部分，它遵循以下涵盖的特定语法。RHS基本上是一个允许执行方言特定语义代码的块。

值得注意的是，空白区域并不重要，除了特定于领域的语言，逐行处理行和空间可能对领域语言有重要意义。

### [8.2。关键词](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_keywords)

Drools 5引入了硬关键词和软关键词的概念。

硬关键字是保留的，在命名规则文本中使用的域对象，属性，方法，函数和其他元素时，不能使用任何硬关键字。

以下是在编写规则时必须避免作为标识符的硬关键字列表：

* true
* false
* null

软关键字只在其上下文中被识别，使您可以在任何其他位置使用这些词，但仍建议避免它们，以避免混淆，如果可能的话。以下是软关键字的列表：

* lock-on-active
* date-effective
* date-expires
* no-loop
* auto-focus
* activation-group
* agenda-group
* ruleflow-group
* entry-point
* duration
* package
* import
* dialect
* salience
* enabled
* attributes
* rule
* extend
* 什么时候
* 然后
* template
* query
* declare
* function
* global
* eval
* not
* in
* or
* and
* exists
* forall
* 积累
* 搜集
* 从
* action
* reverse
* result
* end
* 过度
* init

当然，可以将这些（硬和软）单词作为驼峰案例中的方法名称的一部分，例如notSomething（）或accumulateSomething（） - 这种情况没有问题。

虽然上面的3个硬关键字不太可能用于现有的域模型，但如果您绝对需要将它们用作标识符而不是关键字，则DRL语言提供了在规则文本上转义硬关键字的功能。为了逃避一个词，只需将它放在严肃的口音中，就像这样：

Holiday( `true` == "yes" ) *// please note that Drools will resolve that reference to the method Holiday.isTrue()*

### [8.3。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_comments)

注释是规则引擎忽略的文本部分。当它们遇到时，它们被剥离，除了在语义代码块内，如规则的RHS。

#### [8.3.1。单行评论](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_single_line_comment)

要创建单行注释，可以使用'//'。解析器将在注释符号后忽略行中的任何内容。例：

rule "Testing Comments"

**when**

// **this** **is** a single line comment

eval( true ) // **this** **is** a comment **in** the same line **of** a pattern

**then**

// **this** **is** a comment inside a semantic code block

end

|  |  |
| --- | --- |
|  | 评论的'＃'已被删除。 |

#### [8.3.2。多行评论](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_multi_line_comment)

*图172.多行注释*

多行注释用于在语义代码块内外注释文本块。例：

rule "Test Multi-line Comments"

**when**

*/\* this is a multi-line comment*

*in the left hand side of a rule \*/*

eval( **true** )

then

*/\* and this is a multi-line comment*

*in the right hand side of a rule \*/*

end

### [8.4。错误消息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_error_messages)

Drools 5引入了标准化的错误消息。此标准化旨在帮助用户以更简单快捷的方式查找和解决问题。在本节中，您将学习如何识别和解释这些错误消息，并且您还将收到一些关于如何解决与它们相关的问题的提示。

#### [8.4.1。消息格式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_message_format)

标准化包括错误消息格式，为了更好地解释这种格式，我们使用下面的例子：

*图173.错误消息格式*

**1st Block：**这个区域标识错误代码。

**第二块：**行和列信息。

**第三块：**描述问题的一些文本。

**第四块：**这是第一个上下文。通常表示发生错误的规则，函数，模板或查询。该块不是强制性的。

**第5块：**标识发生错误的模式。该块不是强制性的。

#### [8.4.2。错误消息说明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_error_messages_description)

##### [8.4.2.1。101：没有可行的选择](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_101_no_viable_alternative)

表示最常见的错误，即解析器进入决策点但无法识别替代方案。这里有些例子：

1: rule one

2: when

3: exists Foo()

4: exits Bar() // "exits"

5: then

6: end

以上示例生成此消息：

* [错误101]第4行：4在规则一的输入“退出”中没有可行的选择

乍一看这似乎是有效的语法，但它不是（退出！=存在）。我们来看下一个例子：

1: package org.drools.examples;

2: rule

3: when

4: Object()

5: then

6: System.out.println("A RHS");

7: end

现在上面的代码产生这个消息：

* [ERR 101]第3行：2输入'WHEN'时没有可行的替代方案

此消息表示解析器遇到了**WHEN**标记，实际上它是一个硬关键字，但由于缺少规则名称，所以它位于错误的位置。

当你犯一个简单的词汇错误时，错误“没有可行的选择”也会出现。这是一个词汇问题的例子：

1: rule simple\_rule

2: when

3: Student( name == "Andy )

4: then

5: end

上述代码错过了关闭引号，因此解析器会生成此错误消息：

* [ERR 101]第0行：-1模式Student中规则simple\_rule的输入'<eof>'处没有可行的替代方案

|  |  |
| --- | --- |
|  | 通常行和列信息是准确的，但在某些情况下（如未封闭的引号），解析器会生成一个0：-1的位置。在这种情况下，你应该检查你是否忘记关闭引号，撇号或括号。 |

##### [8.4.2.2。102：输入不匹配](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_102_mismatched_input)

这个错误表明解析器正在寻找它在当前输入位置找不到的特定符号。以下是一些示例：

1: rule simple\_rule

2: when

3: foo3 : Bar(

以上示例生成此消息：

* 在模式Bar中，规则simple\_rule中的[ERR 102]行0：-1不匹配的输入'<eof>'期待'）'

要解决这个问题，有必要完成规则声明。

|  |  |
| --- | --- |
|  | 通常，当您获得0：-1的位置时，意味着解析器到达源代码的末尾。 |

以下代码会生成多条错误消息：

1: package org.drools.examples;

2:

3: rule "Avoid NPE on wrong syntax"

4: when

5: not( Cheese( ( type == "stilton", price == 10 ) || ( type == "brie", price == 15 ) ) from $cheeseList )

6: then

7: System.out.println("OK");

8: end

这些是与此来源相关的错误：

* 在模式奶酪中的规则“避免错误语法中的NPE”中的[ERR 102]行5:36不匹配的输入'，'期望'）'
* [错误101]行5:57在规则“输入'类型'中没有可行的替代方法避免错误语法中的NPE'
* [错误102]第5行：106不匹配输入'）'在规则中期待'then'避免错误语法中的NPE'

请注意，第二个问题与第一个问题有关。要修复它，只需用AND运算符（'&&'）替换逗号（'，'）即可。

|  |  |
| --- | --- |
|  | 在某些情况下，您可能会收到多条错误消息。尝试从第一个开始逐个修复。一些错误消息仅仅作为其他错误的后果而产生。 |

##### [8.4.2.3。103：谓词失败](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_103_failed_predicate)

验证语义谓词评估为false。通常这些语义谓词用于识别软关键字。本示例显示了这种情况：

1: package nesting;

2: dialect "mvel"

3:

4: import org.drools.compiler.Person

5: import org.drools.compiler.Address

6:

7: fdsfdsfds

8:

9: rule "test something"

10: when

11: p: Person( name=="Michael" )

12: then

13: p.name = "other";

14: System.out.println(p.name);

15: end

有了这个例子，我们得到这个错误信息：

* [ERR 103]第7行：0规则'rule\_key'失败谓词：{（validateIdentifierKey（DroolsSoftKeywords.RULE））}？在规则

该**fdsfdsfds**文字无效，分析器无法识别它作为软关键字rule。

|  |  |
| --- | --- |
|  | 这个错误非常类似于102：不匹配的输入，但通常涉及软关键字。 |

##### [8.4.2.4。104：不允许追踪分号](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_104_trailing_semi_colon_not_allowed)

该错误与该eval子句相关，其表达式可能不以分号结尾。检查这个例子：

1: rule simple\_rule

2: when

3: eval(abc();)

4: then

5: end

由于eval内的尾随分号，我们得到这个错误信息：

* [规则104]第3行：4规则simple\_rule中不允许使用尾部分号

这个问题很容易解决：只要删除分号即可。

##### [8.4.2.5。105：提前退出](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_105_early_exit)

识别者在语法上达到了一个至少与一种选择相匹配的亚规则，但该亚规则并不匹配任何东西。简而言之：解析器已经进入了一个没有出路的分支。这个例子说明了它：

1: template test\_error

2: aa s 11;

3: end

这是与上述示例相关的消息：

* [ERR 105]第2行：需要2个（...）+循环与模板test\_error中输入'aa'的任何内容不匹配

要解决此问题，需要删除数字值，因为它既不是可能开始新模板插槽的有效数据类型，也不是任何其他规则文件构造的可能启动。

#### [8.4.3。其他消息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_other_messages)

任何其他信息意味着发生了不好的事情，所以请联系开发团队。

### [8.5。包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_package)

包是规则和其他相关结构的集合，例如导入和全局变量。包成员通常是相互关联的 - 比如HR规则。一个包表示一个名称空间，对于给定的规则分组，它理想地保持唯一。包名称本身就是名称空间，并且与文件或文件夹无关。

可以从多个规则源组装规则，并且有一个顶级程序包配置，所有规则都保存在该规则下（当组装规则时）。尽管不可能合并到以不同名称声明的相同包资源中。但是，单一的规则库可能包含构建于其上的多个程序包。一个常见的结构是将包中的所有规则与包声明在同一个文件中（因此它完全是独立的）。

以下铁路图显示了可能组成包的所有组件。请注意，程序包必须具有名称空间，并使用标准Java约定来声明程序包名称; 即没有空格，不同于允许空格的规则名称。根据元素的顺序，它们可以以任何顺序出现在规则文件中，但package语句除外，该语句必须位于文件的顶部。在所有情况下，分号都是可选的。

*图174.包*

注意，任何规则属性（如规则属性部分所述）也可以在包级别编写，取代属性的默认值。修改后的默认设置仍然可以由规则中的属性设置替换。

#### [8.5.1。进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_import)

*图175.导入*

导入语句就像Java中的导入语句一样工作。您需要为要在规则中使用的任何对象指定完全限定的路径和类型名称。Drools自动从同名的Java包中导入类，也从包中导入类java.lang。

#### [8.5.2。全球](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_global)

*图176.全球*

随global你定义全局变量。它们用于使应用程序对象可用于规则。通常，它们用于提供规则使用的数据或服务，尤其是规则后果中使用的应用程序服务，以及从规则返回数据（例如规则后果中添加的日志或值），或者规则与应用程序交互，做回调。全局变量不会被插入工作内存中，因此全局决不能用于规则中的条件，除非它具有恒定的不变值。引擎不能通知全局变量的值，也不会跟踪其变化。在约束中不正确地使用全局变量可能会产生令人惊讶的结果 - 令人惊讶的是不好的方式。

如果多个包声明具有相同标识符的全局变量，则它们必须是相同的类型，并且它们都将引用相同的全局值。

为了使用全局变量，你必须：

1. 在你的规则文件中声明你的全局变量并在规则中使用它。例如：
2. global java.util.List myGlobalList;
3. rule "Using a global"
4. **when**
5. eval( true )
6. **then**
7. myGlobalList.add( "Hello World" );

end

1. 在工作记忆中设置全局值。在向工作记忆声明任何事实之前设置所有全局值是一种最佳做法。例：
2. List list = **new** ArrayList();
3. KieSession kieSession = kiebase.newKieSession();

kieSession.setGlobal( "myGlobalList", list );

请注意，这些只是您从应用程序传递到工作内存的对象的命名实例。这意味着你可以传入你想要的任何对象：你可以传入一个服务定位器，或者一个服务本身。使用新from元素，现在通常将Hibernate会话作为全局进行传递，以允许from从命名的Hibernate查询中提取数据。

一个示例可以是电子邮件服务的实例。在调用规则引擎的集成代码中，获取您的emailService对象，然后将其设置在工作内存中。在DRL中，您声明您具有EmailService类型的全局，并将其命名为“email”。那么在你的规则后果中，你可以使用诸如email.sendSMS（number，message）之类的东西。

全局并不是为了在规则之间共享数据而设计的，它们绝不应该用于此目的。规则总是对工作内存状态产生原因并做出反应，因此如果要将规则中的数据传递给规则，请将数据作为事实断言到工作内存中。

更改全局数据时必须小心，因为规则引擎不知道这些更改，因此无法对它们做出反应。

### [8.6。功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_function)

*图177.功能*

函数是将语义代码放入规则源文件中的一种方式，与普通的Java类相反。他们不能做任何事情，而不是你能用辅助类做什么。（实际上，编译器会在幕后为你生成助手类）。在规则中使用函数的主要优点是可以将逻辑全部保存在一个位置，并且可以根据需要更改函数（可以是好的或坏的东西）。函数对于在then规则的outcome（）部分调用动作最为有用，特别是如果该特定动作一遍又一遍地使用，可能每个规则只有不同的参数。

典型的函数声明如下所示：

**function** **String** **hello**(String name) {

**return** "Hello "+name+"!";

}

请注意function，即使它不是Java的一部分，也使用关键字。函数的参数定义为一个方法，如果不需要参数，则不需要参数。返回类型的定义与常规方法一样。

或者，您可以在助手类中使用静态方法，例如Foo.hello()。Drools支持使用函数导入，所以你需要做的是：

import function my.package.Foo.hello

无论定义或导入函数的方式如何，您都可以通过函数的名称，结果或语义代码块来调用函数。例：

rule "using a static function"

**when**

**eval**( **true** )

then

System.**out**.**println**( hello( "Bob" ) );

end

### [8.7。类型声明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_type_declaration)

*图178. meta\_data*

*图179. type\_declaration*

类型声明在规则引擎中有两个主要目标：允许声明新类型，并允许为类型声明元数据。

* **声明新类型：** Drools使用简单的Java对象作为事实开箱即用。然而，有时用户可能希望将模型直接定义到规则引擎，而不必担心使用Java等较低级别的语言创建模型。在其他时候，已经建立了一个领域模型，但最终用户希望或需要用在推理过程中主要使用的其他实体来补充该模型。
* **声明元数据：**事实可能具有与其相关的元信息。元信息的例子包括任何类型的数据，这些数据不是由事实属性表示的，并且在该事实类型的所有实例中是一致的。该元信息可以在运行时由引擎查询并用于推理过程。

#### [8.7.1。声明新类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_new_types)

要声明新类型，您只需使用关键字declare，然后是字段列表和关键字end。一个新事实必须有一个字段列表，否则引擎将在类路径中查找现有的事实类，如果未找到，则会引发错误。

*例108.声明一个新的事实类型：地址*

**declare** Address

number : int

streetName : **String**

city : **String**

**end**

前面的例子声明了一个新的事实类型Address。这个事实类型将有三个属性：number，streetName和city。每个属性都有一个类型，可以是任何有效的Java类型，包括由用户创建的任何其他类，甚至是之前声明的其他事实类型。

例如，我们可能想要声明另一个事实类型Person：

*例109.声明一个新的事实类型：Person*

**declare** Person

**name** : **String**

dateOfBirth : **java**.util.Date

address : Address

**end**

就像我们在前面的例子中看到的那样，它dateOfBirth是java.util.DateJava API 的类型，而且address是之前定义的事实类型Address。

import如前所述，您可以避免每次使用该子句编写完全限定名的类。

*示例110.避免使用导入来使用完全限定的类名*

import java.util.Date

**declare** Person

**name** : **String**

dateOfBirth : Date

address : Address

**end**

当你声明一个新的事实类型时，Drools将在编译时生成一个字节码来实现一个代表事实类型的Java类。生成的Java类将是类型定义的一对一Java Bean映射。因此，对于前面的例子，生成的Java类将是：

*例111.为前一个Person事实类型声明生成的Java类*

**public** **class** **Person** **implements** **Serializable** {

**private** String name;

**private** java.util.Date dateOfBirth;

**private** Address address;

*// empty constructor*

**public** **Person**() {...}

*// constructor with all fields*

**public** **Person**( String name, Date dateOfBirth, Address address ) {...}

*// if keys are defined, constructor with keys*

**public** **Person**( ...keys... ) {...}

*// getters and setters*

*// equals/hashCode*

*// toString*

}

由于生成的类是一个简单的Java类，因此可以像规则一样透明地使用它，就像其他任何事实一样。

*例112.使用规则中的声明类型*

rule "Using a declared Type"

**when**<

$p : Person( name == "Bob" )

then

*// Insert Mark, who is Bob's mate.*

Person mark = **new** Person();

mark.setName("Mark");

insert( mark );

end

##### [8.7.1.1。声明枚举类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_enumerative_types)

DRL还支持枚举类型的声明。这样的类型声明需要额外的关键字enum，后跟一个以逗号分隔的以逗号分隔的允许值列表。

rule "Using a declared Type"

**when**

$p : Person( name == "Bob" )

then

*// Insert Mark, who is Bob's mate.*

Person mark = **new** Person();

mark.setName("Mark");

insert( mark );

end

编译器将通过静态方法valueOf（）和values（）以及实例方法ordinal（），compareTo（）和name（）生成一个有效的Java枚举。

复杂的枚举也部分支持，声明内部字段与常规类型声明类似。请注意，从版本6.x开始，枚举字段不支持其他声明的类型或枚举

**declare** enum DaysOfWeek

SUN("Sunday"),MON("Monday"),TUE("Tuesday"),WED("Wednesday"),THU("Thursday"),FRI("Friday"),SAT("Saturday");

fullName : String

**end**

枚举可以在规则中使用

*例113.在规则中使用声明性枚举*

rule "Using a declared Enum"

**when**

$p : Employee( dayOff == DaysOfWeek.MONDAY )

**then**

...

**end**

#### [8.7.2。声明元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_metadata)

元数据可以分配给Drools中的几种不同的结构：事实类型，事实属性和规则。Drools使用at符号（'@'）来引入元数据，并且它始终使用以下形式：

@metadata\_key( metadata\_value )

括号内的metadata\_value是可选的。

举例来说，如果你要声明一个元数据属性一样author，它的值是鲍勃，你可以简单地写：

*例114.声明一个元数据属性*

@author( Bob )

Drools允许声明任何任意的元数据属性，但有些对引擎有特殊意义，而另一些只是在运行时查询。Drools允许为事实类型和事实属性声明元数据。在事实类型的属性之前声明的任何元数据被分配给事实类型，而在属性之后声明的元数据被分配给该特定属性。

*例115.为事实类型和属性声明元数据属性*

import java.util.Date

**declare** Person

@author( Bob )

@dateOfCreation( 01-Feb-2009 )

**name** : **String** @**key** @maxLength( 30 )

dateOfBirth : Date

address : Address

**end**

在前面的示例中，为事实类型（@author和@dateOfCreation）声明了两个元数据项，并为name属性（@key和@maxLength）定义了两个元数据项。请注意，@key元数据没有要求的值，所以括号和值被忽略。

##### [8.7.2.1。预定义的课程级别注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_predefined_class_level_annotations)

一些注释具有由引擎解释的预定义语义。以下是其中一些预定义注释及其含义的列表。

###### [@role（<fact | event>）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__role_fact_event)

@role注释定义了引擎应如何处理该类型的实例：既可以是常规事实，也可以是事件。它接受两个可能的值：

* 事实：这是默认的，声明类型是作为一个常规事实来处理的。
* 事件：声明该类型将作为事件处理。

以下示例声明股票经纪人应用程序中的事实类型StockTick将作为事件处理。

*例116.将事实类型声明为一个事件*

import some.package.StockTick

**declare** StockTick

@**role**( **event** )

**end**

这同样适用于在线声明的事实。如果StockTick是在DRL本身中声明的事实类型，而不是先前存在的类，则代码将为：

*例117.声明一个事实类型并为其分配事件角色*

**declare** StockTick

@**role**( **event** )

datetime : **java**.util.Date

symbol : **String**

price : **double**

**end**

###### [@typesafe（<boolean>）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__typesafe_boolean)

默认情况下，所有类型声明都是在启用类型安全性的情况 @typesafe（false）提供了一种方法来通过允许回退来覆盖此行为，以便在所有约束生成为MVEL约束并动态执行时键入不安全评估。处理没有任何泛型或混合类型集合的集合时，这可能很重要。

###### [@timestamp（<属性名称）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__timestamp_attribute_name)

每个事件都有一个分配给它的关联时间戳。默认情况下，会话时钟读取给定事件的时间戳，并在事件插入工作存储器时将其分配给该事件。尽管有时该事件具有时间戳作为其自身属性之一。在这种情况下，用户可以告诉引擎使用事件属性中的时间戳，而不是从会话时钟中读取它。

@timestamp( <attributeName> )

要告诉引擎什么属性用作事件时间戳的来源，只需将属性名称列为@timestamp标记的参数即可。

*例118.声明VoiceCall时间戳属性*

**declare** VoiceCall

@**role**( **event** )

@**timestamp**( callDateTime )

**end**

###### [@duration（<属性名称）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__duration_attribute_name)

Drools支持事件语义：时间点事件和基于时间间隔的事件。时间点事件表示为持续时间为零的基于时间间隔的事件。默认情况下，所有事件的持续时间为零。用户可以通过声明事件类型中的哪个属性包含事件的持续时间来为事件赋予不同的持续时间。

@duration( <attributeName> )

因此，对于我们的VoiceCall事实类型，声明将是：

*例119.声明VoiceCall持续时间属性*

**declare** VoiceCall

@**role**( **event** )

@**timestamp**( callDateTime )

@**duration**( callDuration )

**end**

###### [@expires（<时间间隔>）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__expires_time_interval)

|  |  |
| --- | --- |
|  | 只有在STREAM模式下运行引擎时才会考虑该标签。另外，关于在内存管理部分使用这个标签的效果的补充讨论。这里包括了完整性。 |

工作记忆中的一段时间后，事件可能会自动失效。通常情况下，根据知识库中的现有规则，事件不能再匹配并激活任何规则。尽管可以明确定义一个事件应该到期的时间。

@expires( <timeOffset> )

timeOffset的值是以下形式的时间间隔：

[#d][#h][#m][#s][#[ms]]

其中[]表示可选参数，\＃表示数值。

因此，为了声明VoiceCall事实在插入工作存储器1小时35分钟后应该过期，用户将写入：

*例120.为VoiceCall事件声明到期偏移量*

**declare** VoiceCall

@**role**( **event** )

@**timestamp**( callDateTime )

@**duration**( callDuration )

@expires( 1h35m )

**end**

@expires策略将优先并覆盖从时间约束和知识库中的滑动窗口计算出的隐式到期偏移量。

###### [@propertyChangeSupport](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__propertychangesupport)

实现对Javabean（tm）规范中定义的属性更改的支持的事实现在可以注释，以便引擎注册自己以侦听事实属性的更改。Drools 4 API中insert（）方法中使用的布尔参数已弃用，并且不存在于drools-api模块中。

*示例121. @propertyChangeSupport*

**declare** Person

@propertyChangeSupport

end

###### [@propertyReactive](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__propertyreactive)

使此类型属性为反应。有关详细信息，请参阅细粒度属性更改侦听器部分。

##### [8.7.2.2。预定义的属性级别注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_predefined_attribute_level_annotations)

如前所述，Drools还支持类型属性中的注释。以下是预定义属性注释的列表。

###### [@键](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__key)

将属性声明为关键属性对生成的类型有两个主要影响：

1. 该属性将用作类型的关键标识符，因此，在比较此类型的实例时，生成的类将实现equals（）和hashCode（）方法，并考虑该属性。
2. Drools将生成一个使用所有关键属性作为参数的构造函数。

例如：

*示例122.类型的@key声明的示例*

**declare** Person

firstName : **String** @**key**

lastName : **String** @**key**

age : int

**end**

对于前面的示例，Drools将生成equals（）和hashCode（）方法，它们将检查firstName和lastName属性，以确定Person的两个实例是否彼此相等，但不会检查age属性。它还会生成一个以firstName和lastName作为参数的构造函数，从而允许使用如下代码创建实例：

*例123.使用键构造函数创建一个实例*

Person person = **new** Person( "John", "Doe" );

###### [@位置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__position)

模式支持类型声明的位置参数。

位置参数是您不需要指定字段名称的位置参数，因为位置映射到已知的命名字段。即Person（name ==“mark”）可以被重写为Person（“mark”;）。分号';' 是非常重要的，以便引擎知道它之前的一切都是位置参数。否则，我们可能会认为它是一个布尔表达式，它是如何在分号后解释的。您可以通过使用分号';'来混合模式上的位置参数和命名参数 将它们分开。位置中尚未绑定的任何变量都将绑定到映射到该位置的字段。

**declare** Cheese

**name** : **String**

shop : **String**

price : int

**end**

默认顺序是声明的顺序，但可以使用@position覆盖

**declare** Cheese

**name** : **String** @**position**(1)

shop : **String** @**position**(2)

price : int @**position**(0)

**end**

org.drools.definition.type包中的@Position注释可用于注释类路径上的原始pojos。目前只有类上的字段可以注释。支持类的继承，但不支持方法的接口。

示例模式，带有两个约束和绑定。记住分号';' 用于区分名称参数部分的位置部分。变量和文字以及仅使用文字的表达式在位置参数中受支持，但不支持变量。

**Cheese**( "stilton", "Cheese Shop", p; )

**Cheese**( "stilton", "Cheese Shop"; p : price )

**Cheese**( "stilton"; shop == "Cheese Shop", p : price )

**Cheese**( name == "stilton"; shop == "Cheese Shop", p : price )

@Position在bean彼此延伸时被继承; 虽然不推荐，但两个字段可能具有相同的@position值，并且并非所有连续值都需要声明。如果重复@位置，则使用继承（超类中的字段具有优先级）和声明顺序来解决冲突。如果缺少@position值，则选择没有显式@position（如果有）的第一个字段以填补空白。与往常一样，冲突通过继承和声明顺序来解决。

**declare** Cheese

**name** : **String**

shop : **String** @**position**(2)

price : int @**position**(0)

**end**

**declare** SeasonedCheese extends Cheese

**year** : Date @**position**(0)

origin : **String** @**position**(6)

country : **String**

**end**

在示例中，字段顺序为：price（超类中的@position 0），year（子类中的@position 0），名称（第一个没有@position的字段），shop（@position 2），country（second没有@position的字段），起源。

#### [8.7.3。为现有类型声明元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_metadata_for_existing_types)

Drools允许以与为新的事实类型声明元数据属性相同的方式为现有类型声明元数据属性。唯一的区别是该声明中没有字段。

例如，如果有一个类org.drools.examples.Person，并且想要为它声明元数据，可以写下面的代码：

*例124.为现有类型声明元数据*

import org.drools.examples.Person

declare Person

@**author(** Bob )

@dateOfCreation( 01-Feb-2009 )

end

除了使用导入之外，还可以通过完全限定名称引用类，但由于该类也将在规则中引用，所以添加导入并在任何地方使用简短类名通常较短。

*例125.使用完全限定的类名称声明元数据*

declare org.drools.examples.Person

@**author(** Bob )

@dateOfCreation( 01-Feb-2009 )

end

#### [8.7.4。声明类型的参数化构造函数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_parametrized_constructors_for_declared_types)

使用声明类型的参数生成构造函数。

例如：对于如下所示的声明类型：

**declare** Person

firstName : **String** @**key**

lastName : **String** @**key**

age : int

**end**

编译器将隐式生成3个构造函数：一个不带参数，一个带有@key字段，另一个带有所有字段。

Person() *// parameterless constructor*

Person( String firstName, String lastName )

Person( String firstName, String lastName, int age )

#### [8.7.5。非类型安全类](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_non_typesafe_classes)

@typesafe（<boolean>）已被添加到类型声明中。默认情况下，所有类型声明都是在启用类型安全性的情况 @typesafe（false）提供了一种方法来通过允许回退来覆盖此行为，以便在所有约束生成为MVEL约束并动态执行时键入不安全评估。处理没有任何泛型或混合类型集合的集合时，这可能很重要。

#### [8.7.6。从应用程序代码访问已声明的类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_accessing_declared_types_from_the_application_code)

声明的类型通常在规则文件中使用，而在规则和应用程序之间共享模型时使用Java模型。尽管有时候，应用程序可能需要访问和处理来自声明类型的事实，特别是当应用程序正在封装规则引擎并为规则管理提供更高级别，特定于域的用户界面时。

在这种情况下，生成的类可以像平常一样使用Java Reflection API进行处理，但是，正如我们所知，通常需要大量的工作才能获得小的结果。因此，Drools为应用程序可能想要处理的最常见事实提供了一个简化的API。

要认识到的第一件重要事情是，宣布的事实将属于它所宣布的包裹。因此，例如，在下面的示例中，Person将属于该org.drools.examples包，因此生成的类的完全限定名称将成为org.drools.examples.Person。

*例126.在org.drools.examples包中声明一个类型*

package org.drools.examples

import java.util.Date

**declare** Person

**name** : **String**

dateOfBirth : Date

address : Address

**end**

正如前面所讨论的，声明类型是在知识库编译时生成的，也就是说，应用程序只能在应用程序运行时访问它们。因此，这些类不能直接从应用程序中引用。

Drools然后提供一个接口，用户可以通过该接口处理来自应用程序代码的声明类型：org.drools.definition.type.FactType。通过这个接口，用户可以在已声明的事实类型中实例化，读取和写入字段。

*示例127.通过API处理声明的事实类型*

*// get a reference to a knowledge base with a declared type:*

KieBase kbase = ...

*// get the declared FactType*

FactType personType = kbase.getFactType( "org.drools.examples",

"Person" );

*// handle the type as necessary:*

*// create instances:*

Object bob = personType.newInstance();

*// set attributes values*

personType.set( bob,

"name",

"Bob" );

personType.set( bob,

"age",

42 );

*// insert fact into a session*

KieSession ksession = ...

ksession.insert( bob );

ksession.fireAllRules();

*// read attributes*

String name = personType.get( bob, "name" );

**int** age = personType.get( bob, "age" );

该API还包含其他有用的方法，例如一次设置所有属性，从Map中读取值或一次读取所有属性到Map中。

尽管API类似于Java反射（但使用起来更简单），但它并不使用底层的反射，依赖于使用生成的字节码实现的更高性能的访问器。

#### [8.7.7。类型声明'扩展'](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_type_declaration_extends)

类型声明现在支持继承的'extends'关键字

为了扩展由DRL声明的子类型在Java中声明的类型，请在不带任何字段的declare语句中重复超类型。

import org.people.Person

**declare** Person **end**

**declare** Student extends Person

school : **String**

**end**

**declare** LongTermStudent extends Student

**years** : int

course : **String**

**end**

### [8.8。规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule)

*图180.规则*

规则规定当左手侧（LHS）中指定的特定条件发生时，则执行查询被指定为右手侧（RHS）中的动作列表的内容。用户常见的问题是“为什么使用时而不是if？” 当“如果”被选择为“如果”时，因为“如果”通常是程序执行流程的一部分，则在特定时间点检查条件。

相反，“何时”表示条件评估不与特定的评估序列或时间点相关，而是在发动机的使用寿命期间的任何时间连续发生; 每当条件满足时，都会执行这些操作。

规则必须有一个名称，在其规则包中是唯一的。如果您在同一个DRL中定义了两次规则，则会在加载时产生错误。如果添加包含规则名称的DRL，它会替换以前的规则。如果规则名称有空格，则需要用双引号括起来（最好总是使用双引号）。

属性 - 如下所述 - 是可选的。他们最好每行写一个。

规则的LHS遵循when关键字（理想情况下换行），RHS同样遵循then关键字（理想情况下换行符）。该规则由关键字终止end。规则不能嵌套。

*例128.规则语法概述*

rule "<name>"

<attribute>\*

when

<conditional element>\*

then

<action>\*

end

*例129.一个简单的规则*

rule "Approve if not rejected"

salience -100

agenda-group "approval"

**when**

**not** Rejection()

p : Policy(approved == **false**, policyState:status)

exists Driver(age > 25)

Process(status == policyState)

**then**

log("APPROVED: due to no objections.");

p.setApproved(**true**);

**end**

#### [8.8.1。规则属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_attributes)

规则属性提供了一种声明方式来影响规则的行为。有些非常简单，而另一些则是复杂子系统的一部分，如ruleflow。为了充分利用Drools，你应该确保你对每个属性有一个正确的理解。

*图181.规则属性*

no-loop

默认值： false

键入：Boolean

当规则的结果修改事实时，可能会导致规则再次激活，导致无限循环。将no-loop设置为true将跳过为具有当前事实集合的规则创建另一个Activation。

ruleflow-group

默认值：N / A

键入：String

Ruleflow是一个Drools功能，可以让您控制规则的触发。仅当组的活动时，由同一个ruleflow-group标识符组装的规则才会触发。

lock-on-active

默认值： false

键入：Boolean

每当规则流组变为活动状态或议程组接收到焦点时，该组中的任何规则中的锁定活动状态设置为true的规则将不再被激活; 不管更新的起源如何，都会丢弃匹配规则的激活。这是一个更强大的版本no-loop，因为现在不仅可以由规则本身引起更改。对于计算规则而言，如果您有一些修改事实的规则，并且不希望任何规则重新匹配并再次触发，那么这是理想的计算规则。只有当规则流组不再活动或者议程组失去焦点时，那些锁定活动设置为true的规则才会再次符合条件，以便将其激活置于议程中。

salience

默认值： 0

键入：整数

每个规则都有一个整数显着属性，默认为零，可以是负值也可以是正值。显着性是一种优先级，在激活队列中排序时，具有较高显着性值的规则具有较高的优先级。

Drools还支持动态显着性，您可以使用包含绑定变量的表达式。

*例130.动态显着性*

rule "Fire in rank order 1,2,.."

salience( -$rank )

**when**

Element( $rank : rank,... )

**then**

...

**end**

agenda-group

默认值：MAIN

键入：String

议程组允许用户划分议程，提供更多的执行控制。只有获得焦点的议程小组中的规则才能被解雇。

auto-focus

默认值： false

键入：Boolean

如果在auto-focus值为真且规则的议程组尚未具有焦点的情况下激活规则，则会给予焦点，从而允许规则可能触发。

activation-group

默认值：N / A

键入：String

属于同一个激活组的规则（由该属性的字符串值标识）将仅以独占方式激发。更确切地说，激活组中的第一条规则将会取消组中所有规则的所有未决激活，即阻止它们发射。

注意：这曾经被称为异或集团，但从技术上说它不是一个异或。你仍然可以听到人们提到异或集团; 只需在激活组中使用该术语即可。

dialect

默认值：如包所指定

键入：String

可能的值：“java”或“mvel”

方言种类是在LHS或RHS代码块中用于任何代码表达式的语言。目前有两种方言可用，Java和MVEL。虽然可以在包级别指定方言，但此属性允许为规则覆盖包定义。

date-effective

默认值：N / A

键入：String，其中包含日期和时间定义

规则只能在当前日期和时间在日期有效属性之后激活。

date-expires

默认值：N / A

键入：String，其中包含日期和时间定义

如果当前日期和时间在date-expires属性之后，则无法激活规则。

duration

默认值：无默认值

键入：long

持续时间表明规则将在指定的持续时间后触发，如果仍然为真。

*例子131.一些属性例子*

rule "my rule"

salience 42

agenda-**group** "number 1"

**when** ...

#### [8.8.2。定时器和日历](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_timers_and_calendars)

规则现在支持基于时间间隔和基于cron的定时器，它们取代了现在弃用的持续时间属性。

*示例132.示例计时器属性使用*

timer ( **int**: <initial delay> <repeat interval>? )

timer ( **int**: 30s )

timer ( **int**: 30s 5m )

timer ( cron: <cron expression> )

timer ( cron:\* 0/15 \* \* \* ? )

Interval（由“int：”指示）定时器遵循java.util.Timer对象的语义，具有初始延迟和可选重复间隔。Cron（由“cron：”指示）定时器遵循标准Unix cron表达式：

*例子133.一个Cron例子*

rule "Send SMS every 15 minutes"

timer (cron:\* 0/15 \* \* \* ?)

when

$a : Alarm( on == **true** )

then

channels[ "sms" ].insert( **new** Sms( $a.mobileNumber, "The alarm is still on" );

end

由定时器控制的规则在匹配时变为激活，并且在每个单独匹配时变为一次。根据计时器的设置，其结果会重复执行。只要条件不匹配，就会停止。

即使控制从调用返回到fireUntilHalt之后，也会执行后果。而且，发动机对工作记忆的任何变化都保持反应。例如，删除涉及触发定时器规则执行的事实会导致重复执行终止，或者插入一个事实，以便某些规则匹配将导致该规则触发。但是，无论出于何种原因，只有在规则起火后，引擎才会持续活跃。因此，直到下一次执行定时器控制的规则才会发生对异步插入的反应。处理会话会终止所有计时器活动。

相反，当规则引擎默认运行在被动模式（即：使用fireAllRules而不是fireUntilHalt）时，它不会触发定时规则的后果，除非再次调用fireAllRules。但是，可以通过使用a配置KieSession来更改此默认行为，TimedRuleExecutionOption如以下示例中所示。

*示例134.配置KieSession自动执行定时规则*

KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();

ksconf.setOption( TimedRuleExecutionOption.YES );

KSession ksession = kbase.newKieSession(ksconf, **null**);

也可以对必须自动执行的定时规则进行更细粒度的控制。要做到这一点，有必要设置一个FILTEREDTimedRuleExecutionOption允许定义回调来过滤这些规则的方法，就像下一个例子中所做的那样。

*例135.配置一个过滤器来选择自动执行哪些定时规则*

KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();

conf.setOption( **new** TimedRuleExecutionOption.FILTERED(**new** TimedRuleExecutionFilter() {

**public** **boolean** **accept**(Rule[] rules) {

**return** rules[0].getName().equals("MyRule");

}

}) );

关于间隔定时器，还可以将延迟和间隔定义为表达式而不是固定值。为此，有必要使用表达式计时器（由“expr：”指示），如下例所示：

*示例136.一个表达式计时器示例*

declare Bean

delay : String = "30s"

period : **long** = 60000

end

rule "Expression timer"

timer( expr: $d, $p )

when

**Bean**( $d : delay, $p : period )

then

end

的表达，$d和$p在这种情况下，可以使用在规则的图案匹配的部分定义的任何可变的，并且可以是能够在一个时间持续时间或将被内部被转换为长表示的持续时间表示的任何数值解析任何字符串以毫秒为单位。

间隔和表达式定时器都可以有3个可选参数，名称分别为“start”，“end”和“repeat-limit”。当使用这些参数中的一个或多个参数时，定时器定义的第一部分必须跟有分号';' 并且参数必须用逗号“，”分隔，如下例所示：

*例137.一个带有开始和结束的间隔定时器*

timer (**int**: 30s 10s; start=3-JAN-2010, end=5-JAN-2010)

开始和结束参数的值可以是Date，一个表示Date或Long的字符串，或者更一般的任何Number，它将在Java日期中进行转换，应用以下转换：

**new** Date( ((Number) n).longValue() )

相反，repeat-limit只能是一个整数，它定义了定时器允许的最大重复次数。如果同时设置了结束和重复限制参数，则当两个参数中的第一个匹配时，计时器将停止。

启动参数的使用意味着定时器阶段的定义，其中阶段的开始由启动本身加上最终的延迟给出。换句话说，在这种情况下，定时规则将在以下时间安排：

start + delay + n\*period

达到重复限制时间并且不迟于结束时间戳（以先到者为准）。例如，具有以下间隔计时器的规则

timer ( **int**: 30s 1m; start="3-JAN-2010" )

将在2010年1月3日午夜后的每分钟30秒安排。这也意味着，例如，如果您在2010年2月3日午夜打开系统，则不会立即安排，但会保留定时器定义的阶段，因此将在第一次30秒后安排午夜。

如果由于某种原因系统处于暂停状态（例如，会话被序列化，并在一段时间后进行反序列化），规则将只被安排一次以从缺失的激活中恢复（不管我们错过了多少次激活），随后将再次计划与计时器相位。

日历用于控制规则何时可以触发。Calendar API基于[Quartz进行](http://www.quartz-scheduler.org/)建模 ：

*例138.修改Quartz日历*

Calendar weekDayCal = QuartzHelper.quartzCalendarAdapter(org.quartz.Calendar quartzCal)

日历以下列方式注册KieSession：

*例139.注册一个日历*

ksession.getCalendars().set( "weekday", weekDayCal );

它们可以与包括定时器在内的常规规则和规则结合使用。规则属性“日历”可能包含一个或多个以字符串文字形式书写的以逗号分隔的日历名称。

*例140.一起使用日历和定时器*

rule "weekdays are high priority"

calendars "weekday"

timer (**int**:0 1h)

when

**Alarm**()

then

**send**( "priority high - we have an alarm" );

end

rule "weekend are low priority"

calendars "weekend"

timer (**int**:0 4h)

when

**Alarm**()

then

**send**( "priority low - we have an alarm" );

end

#### [8.8.3。左手边（当）语法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_left_hand_side_when_syntax)

##### [8.8.3.1。什么是左手边？](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_what_is_the_left_hand_side)

左手边（LHS）是规则的条件部分的通用名称。它由零个或多个条件元素组成。如果LHS为空，则会将其视为始终为true的条件元素，并在创建新的WorkingMemory会话时激活一次。

*图182.左手侧*

*例141.没有条件元素的规则*

rule "no CEs"

**when**

*// empty*

then

... *// actions (executed once)*

end

*// The above rule is internally rewritten as:*

rule "eval(true)"

**when**

**eval**( **true** )

then

... *// actions (executed once)*

end

条件元素适用于一个或多个模式（在下面介绍）。最常见的条件元素是“ and"因此，当规则的LHS中有多个模式没有以任何方式连接时，它是隐含的：

*例142.隐式和*

rule "2 unconnected patterns"

**when**

Pattern1()

Pattern2()

**then**

... // actions

end

// The above rule **is** internally rewritten as:

rule "2 and connected patterns"

**when**

Pattern1()

**and** Pattern2()

**then**

... // actions

end

|  |  |
| --- | --- |
|  | 一个“和”不能有一个领先的声明绑定（例如不同or）。这是显而易见的，因为声明一次只能引用一个事实，当“和”满足时它就会匹配这两个事实 - 声明绑定哪个事实？  *// Compile error*  $person : (Person( name == "Romeo" ) **and** Person( name == "Juliet")) |

##### [8.8.3.2。模式（条件元素）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pattern_conditional_element)

###### [什么是模式？](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_what_is_a_pattern)

模式元素是最重要的条件元素。它可以匹配插入工作存储器的每个事实。

一个模式包含零个或多个约束，并有一个可选的模式绑定。下面的铁路图显示了这个语法。

*图183.图案*

在最简单的形式中，没有约束，模式与给定类型的事实相匹配。在以下情况下，类型是Cheese，这意味着该模式将与Person工作内存中的所有对象匹配：

Person()

该类型不一定是某个事实对象的实际类。模式可能涉及超类或甚至接口，从而可能与来自许多不同类的事实匹配。

Object() *// matches all objects in the working memory*

模式括号内部是所有动作发生的地方：它定义了该模式的约束。例如，与年龄有关的限制：

Person( age == 100 )

|  |  |
| --- | --- |
|  | 出于向后兼容性的原因，它允许为;字符添加后缀。但不建议这样做。 |

###### [模式绑定](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pattern_binding)

为了引用匹配的对象，使用一个模式绑定变量，如$p。

*示例143.带有绑定变量的模式*

rule ...

**when**

$p : Person()

**then**

System.out.println( "Person " + $p );

**end**

带前缀的美元符号（$）仅仅是一个约定; 它可以用于复杂规则，它有助于轻松区分变量和字段，但它不是强制性的。

##### [8.8.3.3。约束（模式的一部分）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_constraint_part_of_a_pattern)

###### [什么是约束？](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_what_is_a_constraint)

约束是返回true或的表达式false。此示例具有状态5小于6的约束：

Person( 5 < 6 ) *// just an example, as constraints like this would be useless in a real pattern*

实质上，它是一个带有一些增强（如属性访问）和一些差异（如equals()语义==）的Java表达式。让我们深入了解一下。

###### [Java Bean（POJO's）上的属性访问](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_property_access_on_java_beans_pojo_s)

任何bean属性都可以直接使用。使用标准的Java bean getter公开一个bean属性：一个方法getMyProperty()（或者isMyProperty()一个原始布尔值），它不接受任何参数并返回一些东西。例如：age属性写成ageDRL而不是getter getAge()：

Person( age == 50 )

*// this is the same as:*

Person( getAge() == 50 )

Drools使用标准的JDK Introspector类来完成这个映射，所以它遵循标准的Java bean规范。

|  |  |
| --- | --- |
|  | 我们建议使用property access（age）来明确使用getter（getAge()），因为通过字段索引提高了性能。 |
|  | 属性访问器不得以可能影响规则的方式更改对象的状态。请记住，规则引擎在调用之间有效地缓存了其匹配结果，以使其更快。  **public** **int** **getAge**() {  age++; *// Do NOT do this*  **return** age;  }  **public** **int** **getAge**() {  Date now = DateUtil.now(); *// Do NOT do this*  **return** DateUtil.differenceInYears(now, birthday);  }  为了解决后一种情况，请插入一个事实，将当前日期包装到工作内存中，并fireAllRules根据需要更新该事实。 |

|  |  |
| --- | --- |
|  | 以下回退适用：如果无法找到属性的getter，编译器将使用属性名称作为方法名称并且不带参数：  Person( age == 50 )  *// If Person.getAge() does not exists, this falls back to:*  Person( age() == 50 ) |

嵌套属性访问也受支持：

Person( address.houseNumber == 50 )

*// this is the same as:*

Person( getAddress().getHouseNumber() == 50 )

嵌套属性也被索引。

|  |  |
| --- | --- |
|  | 在有状态会话中，当使用嵌套访问器时应该小心，因为工作内存不知道任何嵌套值，并且不知道它们何时更改。将其任何父引用插入到工作内存中时，都将它们视为不可变的。或者，如果您想修改嵌套值，则应将所有外部事实标记为已更新。在上面的例子中，当houseNumber改变，任何Person与Address必须尽可能更新标记。 |

###### [Java表达式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_java_expression)

您可以使用任何Java表达式boolean在模式括号内返回a 作为约束。Java表达式可以与其他表达式增强功能混合使用，例如属性访问：

Person( age == 50 )

可以像使用任何逻辑或数学表达式一样使用括号来更改评估优先级：

Person( age > 100 && ( age % 10 == 0 ) )

可以重用Java方法：

Person( Math.round( weight / ( height \* height ) ) < 25.0 )

|  |  |
| --- | --- |
|  | 至于属性访问器，方法不得以可能影响规则的方式更改对象的状态。对LHS中的事实执行的任何方法都应该是只读方法。  Person( incrementAndGetAge() == 10 ) // **Do** **NOT** **do** this |
|  | 规则调用之间不应改变事实的状态（除非在每次更改时将这些事实标记为更新到工作内存）：  Person( System.currentTimeMillis() % 1000 == 0 ) *// Do NOT do this* |

普通Java运算符优先级适用，请参阅下面的运算符优先级列表。

|  |  |
| --- | --- |
|  | 除了==and 之外，所有运算符都具有普通的Java语义!=。  该==运营商拥有空安全的equals()语义：  // Similar to: java.util.Objects.equals(person.getFirstName(), "John")  // so (because "John" **is** **not** null) similar to:  // "John".equals(person.getFirstName())  Person( firstName == "John" )  该!=运营商拥有空安全的!equals()语义：  // Similar to: !java.util.Objects.equals(person.getFirstName(), "John")  Person( firstName != "John" ) |

如果字段和值的类型不同，则始终尝试类型强制; 如果尝试不良强制，则会抛出异常。例如，如果在数字评估器中提供“ten”作为字符串，则引发异常，而“10”将强制为数字10.强制总是倾向于字段类型而不是值类型：

Person( age == "10" ) // "10" **is** coerced to 10

###### [逗号分隔AND](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_comma_separated_and)

逗号字符（'`，`'）用于分隔约束组。它具有隐含的AND连接语义。

*// Person is at least 50 and weighs at least 80 kg*

Person( age > 50, weight > 80 )

*// Person is at least 50, weighs at least 80 kg and is taller than 2 meter.*

Person( age > 50, weight > 80, height > 2 )

|  |  |
| --- | --- |
|  | 虽然&&和,运算符具有相同的语义，但它们将以不同的优先级解决：&&运算符在||运算符之前。运营商&&和||运营商都在运营商之前,。请参阅下面的运营商优先级列表。  逗号运算符在顶层约束条件下应该是首选，因为它使得约束更容易阅读，并且引擎通常能够更好地优化它们。 |

comma（,）运算符不能嵌入到组合约束表达式中，例如括号：

Person( ( age > 50, weight > 80 ) || height > 2 ) *// Do NOT do this: compile error*

*// Use this instead*

Person( ( age > 50 && weight > 80 ) || height > 2 )

###### [绑定变量](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_binding_variables)

属性可以绑定到一个变量：

*// 2 persons of the same age*

Person( $firstAge : age ) *// binding*

Person( age == $firstAge ) *// constraint expression*

带前缀的美元符号（$）仅仅是一个约定; 它在复杂的规则中很有用，它有助于轻松区分变量和字段。

|  |  |
| --- | --- |
|  | 出于向后兼容性的原因，允许（但不推荐）混合约束绑定和约束表达式，如：  *// Not recommended*  Person( $age : age \* 2 < 100 )  *// Recommended (separates bindings and constraint expressions)*  Person( age \* 2 < 100, $age : age ) |

使用运算符的绑定变量限制==提供了非常快的执行，因为它使用散列索引来提高性能。

###### [统一](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_unification)

Drools不允许绑定到同一个声明。然而这是派生查询统一的一个重要方面。虽然位置参数总是以统一方式处理，但是为名为参数的命名参数引入了一个特殊的统一符号'：='。以下“统一”两个人的年龄论点。

Person( $age := age )

Person( $age := age)

实质上，统一将为第一次出现声明一个绑定，并将约束到序列出现的绑定字段的相同值。

###### [为嵌套对象分组访问器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_grouped_accessors_for_nested_objects)

通常情况下，如下例所示，有必要访问嵌套对象的多个属性

**Person**( name == "mark", address.city == "london", address.country == "uk" )

这些嵌套对象的访问器可以用'。（...）'语法分组，以提供更多可读规则

**Person**( name == "mark", address.( city == "london", country == "uk") )

注意'。' 前缀，这是区分方法调用的嵌套对象约束所必需的。

###### [内联投与强制](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_inline_casts_and_coercion)

在处理嵌套对象时，也需要将其转换为子类型。可以通过＃符号来实现，如下所示：

Person( name == "mark", address**#LongAddress.country == "uk" )**

本例将地址转换为LongAddress，使其获取器可用。如果转换不可行（instanceof返回false），评估将被视为错误。还支持完全限定名称：

Person( name == "mark", address**#org.domain.LongAddress.country == "uk" )**

可以在同一个表达式中使用多个内联强制转换：

Person( name == "mark", address**#LongAddress.country#DetailedCountry.population > 10000000 )**

此外，由于我们也支持instanceof操作符，如果使用该操作符，我们将在该模式内推断其结果以进一步使用该字段：

Person( name == "mark", address **instanceof** LongAddress, address.country == "uk" )

###### [特殊文字支持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_special_literal_support)

除了普通的Java文字（包括Java 5枚举）之外，该文字也被支持：

###### [日期文字](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_date_literal)

日期格式dd-mmm-yyyy默认支持。您可以通过提供替代的日期格式掩码作为名为System的属性来自定义drools.dateformat。如果需要更多控制，请使用限制。

*例144.日期文字限制*

**Cheese**( bestBefore < "27-Oct-2009" )

###### [列表和地图访问](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_list_and_map_access)

可以List通过索引直接访问值：

*// Same as childList(0).getAge() == 18*

Person( childList[0].age == 18 )

也可以Map通过键直接访问一个值：

// Same **as** credentialMap.get("jsmith").isValid()

Person( credentialMap["jsmith"].valid )

###### [简化的组合关系条件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_abbreviated_combined_relation_condition)

这使您可以使用限制连接词&&或在一个字段上放置多个限制||。允许通过圆括号分组，导致递归语法模式。

*图184.简化的组合关系条件*

*图185.缩写联合关系条件与包含关系*

*// Simple abbreviated combined relation condition using a single &&*

Person( age > 30 && < 40 )

*// Complex abbreviated combined relation using groupings*

Person( age ( (> 30 && < 40) ||

(> 20 && < 25) ) )

*// Mixing abbreviated combined relation with constraint connectives*

Person( age > 30 && < 40 || location == "london" )

###### [特殊的DRL操作员](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_special_drl_operators)

*图186.运营商*

将尝试强制为评估者和外地的正确价值。

###### [运营商 < ⇐ > >=](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operators_code_code)

这些运算符可用于具有自然排序的属性。例如，对于日期字段，<意味着在String字段之前，意味着按字母顺序降低。

Person( firstName < $otherFirstName )

Person( birthDate < $otherBirthDate )

仅适用于Comparable属性。

###### [无效的解引用操作符](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_null_safe_dereferencing_operator)

！！运算符允许以无效的方式去除拒绝。更详细地说，匹配算法需要！左边的值。运算符不为空以便为模式匹配本身提供肯定的结果。换句话说，模式：

Person( $streetName : address!.street )

将在内部翻译为：

Person( address != **null**, $streetName : address.street )

###### [运营商 matches](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_matches_code)

匹配任何有效的Java正则表达式的字段。通常，regexp是一个字符串文字，但也可以使用解析为有效正则表达式的变量。

*例145.正则表达式约束*

Cheese( type matches "(Buffalo)?\\S\*Mozzarella" )

|  |  |
| --- | --- |
|  | 像在Java中一样，写成字符串文字的正则表达式需要转义'\\'。 |

仅适用于String属性。matches对null值使用总是评估为false。

###### [运营商 not matches](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_not_matches_code)

如果字符串与正则表达式不匹配，则该运算符返回true。matches运营商也适用相同的规则。例：

*例146.正则表达式约束*

Cheese( type not matches "(Buffalo)?\\S\*Mozzarella" )

仅适用于String属性。not matches对null值使用总是评估为真。

###### [运营商 contains](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_contains_code)

该运算符contains用于检查作为Collection或元素的字段是否包含指定的值。

*示例147.包含集合*

CheeseCounter( cheeses contains "stilton" ) *// contains with a String literal*

CheeseCounter( cheeses contains $**var** ) *// contains with a variable*

仅适用于Collection属性。

操作员contains也可以用来代替String.contains()约束检查。

*示例148.包含字符串文字*

Cheese( name contains "tilto" )

Person( fullName contains "Jr" )

String( **this** contains "foo" )

###### [运营商 not contains](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_not_contains_code)

该运算符not contains用于检查作为Collection或元素的字段是否不包含指定的值。

*示例149.与集合的文字约束*

CheeseCounter( cheeses **not** contains "cheddar" ) // **not** contains **with** a String literal

CheeseCounter( cheeses **not** contains $var ) // **not** contains **with** a variable

仅适用于Collection属性。

|  |  |
| --- | --- |
|  | 为了向后兼容，*excludes*运营商被支持为一个同义词*not contains*。 |

操作员not contains也可以用来代替String.contains()约束检查的逻辑否定- 即：! String.contains()

*示例150.包含字符串文字*

Cheese( name **not** contains "tilto" )

Person( fullName **not** contains "Jr" )

String( **this** **not** contains "foo" )

###### [运营商 memberOf](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_memberof_code)

该运算符memberOf用于检查某个字段是否为集合或元素的成员; 该集合必须是一个变量。

*例151.与集合的文字约束*

CheeseCounter( cheese memberOf $matureCheeses )

###### [运营商 not memberOf](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_not_memberof_code)

运算符not memberOf用于检查字段是否不是集合的成员或元素; 该集合必须是一个变量。

*例152.与集合的文字约束*

CheeseCounter( cheese **not** memberOf $matureCheeses )

###### [运营商 soundslike](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_soundslike_code)

这个操作符类似于matches，但它检查一个单词是否具有与给定值几乎相同的声音（使用英语发音）。这基于Soundex算法（请参阅参考资料<http://en.wikipedia.org/wiki/Soundex>）。

*例153.用声音测试*

// match cheese "fubar" **or** "foobar"

Cheese( name soundslike 'foobar' )

###### [运营商 str](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operator_code_str_code)

该运算符str用于检查一个字段是以String某个值开始还是以某个值结束。它也可以用来检查字符串的长度。

**Message**( routingValue str[startsWith] "R1" )

**Message**( routingValue str[endsWith] "R2" )

Message( routingValue str[length] 17 )

###### [运营商in和notin（复合价值限制）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_operators_code_in_code_and_code_notin_code_compound_value_restriction)

在有多个可能值匹配的地方使用复合值限制。目前，只有in和not in评估支持这一点。该运算符的第二个操作数必须是用逗号分隔的值列表，用括号括起来。值可以作为变量，文字，返回值或合格的标识符给出。两个评估者实际上都是语法糖，内部重写为使用操作符!=和多个限制的列表==。

*图187.compoundValueRestriction*

*实施例154.使用“in”的化合物限制*

Person( $cheese : favouriteCheese )

Cheese( type **in** ( "stilton", "cheddar", $cheese ) )

###### [内嵌评估运算符（不建议使用）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_inline_eval_operator_deprecated)

*图188.内联评估表达式*

内联eval约束可以使用任何有效的方言表达式，只要它返回原始布尔值即可。表达式必须随时间变化。任何以前绑定的变量，可以使用当前或之前的模式; autovivification也用于自动创建字段绑定变量。当找到一个不是当前变量的标识符时，构建器会查看标识符是否是当前对象类型的字段，如果是，则字段绑定是作为同名变量自动创建的。这被称为内联eval's内的字段变量的autovivification。

这个例子将找到男性比女性大2岁的所有男女成对; 该变量age是自动创建过程在第二个模式中自动创建的。

*例155.返回值运算符*

Person( girlAge : age, sex = "F" )

Person( eval( age == girlAge + 2 ), sex = 'M' ) // eval() is actually obsolete **in** this example

|  |  |
| --- | --- |
|  | 内联评估已经过时，因为它们的内部语法现在直接受支持。建议不要使用它们。只需编写表达式而不用在其周围包装eval（）。 |

###### [运算符优先级](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_operator_precedence)

运营商的评估优先级为：

| *表9.运算符优先级* | | |
| --- | --- | --- |
| **运营商类型** | **运营商** | **笔记** |
| （嵌套/空安全）属性访问 | .!. | 不是普通的Java语义 |
| 列表/地图访问 | [ ] | 不是普通的Java语义 |
| 约束绑定 | : | 不是普通的Java语义 |
| 乘 | \\*/% |  |
| 添加剂 | \+- |  |
| 转移 | <<>>>>> |  |
| 相关的 | <>⇐>=instanceof |  |
| 平等 | ==!= | 不使用普通的Java（不） 相同的语义：使用（不）相等于 语义。 |
| 非短路和 | & |  |
| 非短路异或 | ^ |  |
| 包含非短路的OR | | |  |
| 逻辑与 | && |  |
| 逻辑或 | || |  |
| 三元 | ? : |  |
| 逗号分隔AND | , | 不是普通的Java语义 |

##### [8.8.3.4。位置参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_positional_arguments)

现在模式支持类型声明的位置参数。

位置参数是您不需要指定字段名称的位置参数，因为位置映射到已知的命名字段。即Person（name ==“mark”）可以被重写为Person（“mark”;）。分号';' 是非常重要的，以便引擎知道它之前的一切都是位置参数。否则，我们可能会认为它是一个布尔表达式，它是如何在分号后解释的。您可以通过使用分号';'来混合模式上的位置参数和命名参数 将它们分开。位置中尚未绑定的任何变量都将绑定到映射到该位置的字段。

**declare** Cheese

**name** : **String**

shop : **String**

price : int

**end**

示例模式，带有两个约束和绑定。记住分号';' 用于区分名称参数部分的位置部分。变量和文字以及仅使用文字的表达式在位置参数中受支持，但不支持变量。位置参数总是使用统一来解决。

**Cheese**( "stilton", "Cheese Shop", p; )

**Cheese**( "stilton", "Cheese Shop"; p : price )

**Cheese**( "stilton"; shop == "Cheese Shop", p : price )

**Cheese**( name == "stilton"; shop == "Cheese Shop", p : price )

给定先前声明的绑定的位置参数将限制使用统一的参数; 这些被称为输入参数。如果绑定尚不存在，它将创建绑定到position参数所代表的字段的声明; 这些被称为输出参数。

##### [8.8.3.5。细粒度的财产变化听众](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fine_grained_property_change_listeners)

当您对给定对象调用modify（）（请参阅modify语句部分）时，它将触发知识库中匹配对象类型的所有模式的重估。这可能会导致不必要的和无用的评估，并在最坏的情况下导致无限递归。避免它的唯一解决方法是将对象拆分为与原始对象具有1对1关系的较小对象。

这已被引入以提供更简单且更一致的方式来克服此问题。事实上，它允许模式匹配只对在给定模式内实际约束或绑定的属性的修改做出反应。这将有助于性能和递归，并避免人造物体分裂。

此功能默认情况下处于启用状态，但如果您需要或想要将其在特定的bean上取消激活，则可以使用@classReactive对其进行注释。此注释适用于DRL类型声明：

**declare** Person

@classReactive

firstName : **String**

lastName : **String**

**end**

和Java类上：

@ClassReactive

**public** **static** **class** **Person** {

**private** String firstName;

**private** String lastName;

}

例如，通过使用此功能，如果您有如下规则：

rule "Every person named Mario is a male" **when**

$person : Person( firstName == "Mario" )

**then**

modify ( $person ) { setMale( **true** ) }

**end**

您不必为其添加无回路属性，以避免无限递归，因为引擎认识到模式匹配是在'firstName'属性上完成的，而规则的RHS修改'男性'属性。请注意，此功能对update（）不起作用，这是我们为什么会提升modify（）的原因之一，因为它在语句中封装了字段更改。而且，在Java类中，您还可以注释任何方法来说明其调用实际上会修改其他属性。例如在前面的Person类中，可以使用如下方法：

@Modifies( { "firstName", "lastName" } )

**public** **void** **setName**(String name) {

String[] names = name.split("\\s");

**this**.firstName = names[0];

**this**.lastName = names[1];

}

这意味着如果一条规则具有如下的RHS：

modify($person) { setName("Mario Fusco") }

它会正确地认识到，属性'firstName'和'lastName'的值可能已被修改，并相应地采取行动，而不是错过重新评估约束它们的模式。目前，@Modifies的使用不允许在字段上，但只能在方法上使用。这与最常见的情况是一致的，@Modify将用于与前一示例中的Person.setName（）中的类字段无关的方法。另外请注意，@Modifies不是传递的，这意味着如果另一个方法在内部调用Person.setName（），它不足以使用@Modifies（{“name”}）对其进行注释，但有必要使用@修改（{“firstName”，“lastName”}）即使在它上面。很可能@Modifies传递性将在下一个版本中实现。

关于嵌套访问器，引擎只会通知顶级域。换句话说，模式匹配如：

**Person** ( address.city.name == "London )

将仅针对修改Person对象的“地址”属性进行重新评估。以同样的方式，约束分析目前严格限于模式内部。另一个例子可以帮助澄清这一点。LHS如下所示：

$p : Person( )

Car( owner = $p.name )

不会听取对该人姓名的修改，而这个人会做：

Person( $name : name )

Car( owner = $name )

为了解决这个问题，可以使用@watch注释一个模式，如下所示：

$p : Person( ) @watch ( name )

Car( owner = $p.name )

事实上，使用@watch注释一个模式可以让你修改该模式将作出反应的属性的推断集合。请注意，@watch注释中指定的属性实际上会自动添加到自动推断的属性中，但也可以明确排除其中一个或多个属性，并将其名称预先加上！并使模式使用wildcrds \*和！\*分别监听模式中使用的所有类型或属性。因此，例如，您可以在规则的LHS中注释一个模式，如下所示：

// listens **for** changes on both firstName (inferred) **and** lastName

Person( firstName == $expectedFirstName ) @watch( lastName )

// listens **for** all the properties **of** the Person bean

Person( firstName == $expectedFirstName ) @watch( \* )

// listens **for** changes on lastName **and** explicitly exclude firstName

Person( firstName == $expectedFirstName ) @watch( lastName, !firstName )

// listens **for** changes on all the properties except the age one

Person( firstName == $expectedFirstName ) @watch( \*, !age )

由于在使用带@ClassReactive注释的类型的模式上使用此注释是没有意义的，所以如果您尝试这样做，规则编译器将引发编译错误。同样，@watch中相同属性的重复使用（例如：@watch（firstName，！firstName））最终会出现编译错误。在下一个版本中，我们将通过分析甚至在模式之外自动检测要更加智能的属性。

也可以仅在特定类型的模型上启用此功能，或者通过使用KnowledgeBuilderConfiguration的选项完全禁用此功能。特别是这个新的PropertySpecificOption可以具有以下3个值之一：

- DISABLED => the feature **is** turned off **and** all the other related annotations are just ignored

- ALLOWED => types are **not** property reactive **unless** they are **not** annotated with @PropertyReactive (which **is** the dual **of** @ClassReactive)

- ALWAYS => all types are property reactive. This **is** the default behavior

因此，例如，要使默认情况下禁用属性反应性的KnowledgeBuilder：

KnowledgeBuilderConfiguration config = KnowledgeBuilderFactory.newKnowledgeBuilderConfiguration();

config.setOption(PropertySpecificOption.ALLOWED);

KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder(config);

在这最后一种情况下，可以通过@PropertyReactive注释来重新启用特定类型的属性反应性特性。

重要的是要注意，只有在规则结果内执行修改时，属性反应才会自动提供。相反，程序化更新不知道已更改的对象属性，因此无法使用此功能。

要解决此限制，可以选择在更新语句中指定已修改对象中已更改属性的名称，如以下示例中所示：

Person me = **new** Person("me", 40);

FactHandle meHandle = ksession.insert( me );

me.setAge(41);

me.setAddress("California Avenue");

ksession.update( meHandle, me, "age", "address" );

##### [8.8.3.6。基本的条件元素](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_basic_conditional_elements)

###### [条件元素 and](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_and_code)

条件元素"and"用于将其他条件元素分组为逻辑连接。Drools支持前缀and和中缀and。

*图188. infixAnd*

传统中缀and支持：

*//infixAnd*

Cheese( cheeseType : type ) **and** Person( favouriteCheese == cheeseType )

还支持用圆括号显式分组：

*//infixAnd with grouping*

( Cheese( cheeseType : type ) **and**

( Person( favouriteCheese == cheeseType ) **or**

Person( favouriteCheese == cheeseType ) )

|  |  |
| --- | --- |
|  | 符号&&（作为替代and）已弃用。但是它仍然支持向后兼容的语法。 |

*图190.前缀和*

前缀and也支持：

(and Cheese( cheeseType : type )

Person( favouriteCheese == cheeseType ) )

LHS的根元素是一个隐含的前缀and，不需要指定：

*示例156.隐式根前缀和*

when

Cheese( cheeseType : type )

Person( favouriteCheese == cheeseType )

**then**

...

###### [条件元素 or](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_or_code)

条件元素or用于将其他条件元素分组成逻辑和。Drools支持前缀or和中缀or。

*图191. infixOr*

传统中缀or支持：

*//infixOr*

Cheese( cheeseType : type ) **or** Person( favouriteCheese == cheeseType )

还支持用圆括号显式分组：

*//infixOr with grouping*

( Cheese( cheeseType : type ) **or**

( Person( favouriteCheese == cheeseType ) **and**

Person( favouriteCheese == cheeseType ) )

|  |  |
| --- | --- |
|  | 符号||（作为替代or）已弃用。但是它仍然支持向后兼容的语法。 |

*图192. prefixOr*

前缀or也支持：

(or **Person**( sex == "f", age > 60 )

**Person**( sex == "m", age > 65 )

|  |  |
| --- | --- |
|  | 条件元素的行为与字段约束中的约束和限制or的连接不同||。引擎实际上并不了解条件元素or。相反，通过一些不同的逻辑转换，一条规则or被重写为多个子规则。这个过程最终会产生一个规则，or每个CE 都有一个作为根节点和一个子规则。每个子规则都可以像任何正常规则一样激活并开火; 这些子规则之间没有特殊的行为或相互作用。 - 这可能会让新规则作者感到困惑。 |

条件元素or还允许可选的模式绑定。这意味着每个生成的子规则都会将其模式绑定到模式绑定。每个模式必须分别绑定，使用同名变量：

pensioner : ( Person( sex == "f", age > 60 ) **or** Person( sex == "m", age > 65 ) )

(or pensioner : Person( sex == "f", age > 60 )

pensioner : Person( sex == "m", age > 65 ) )

由于条件元素or会生成多个子规则，因此每个可能的逻辑结果都会产生一个，所以上述示例会导致内部生成两条规则。这两条规则在工作记忆中独立工作，这意味着两者都可以匹配，激活和触发 - 没有捷径。

考虑条件元素的最好方法or是作为生成两个或更多类似规则的捷径。当你这样想时，很明显，对于单个规则来说，如果两个或更多的析取项成立，可能会有多个激活。

###### [条件元素 not](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_not_code)

*图193.不*

CE not是一阶逻辑的非存在量词，并检查工作存储器中是否存在某些东西。认为“不”意思是“不能有......”。

该关键字not后面应用了CE的括号。在最简单的情况下（如下图），您可以选择省略括号。

*例子157.没有总线*

**not** Bus()

*例子158.没有红色的总线*

*// Brackets are optional:*

not Bus(color == "red")

*// Brackets are optional:*

not ( Bus(color == "red", number == 42) )

*// "not" with nested infix and - two patterns,*

*// brackets are requires:*

not ( Bus(color == "red") **and**

Bus(color == "blue") )

###### [条件元素 exists](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_exists_code)

*图194.存在*

CE exists是一阶逻辑的存在量词，并检查工作存储器中是否存在某些东西。把“存在”看作是“至少有一个......”。它不同于仅仅拥有自己的模式，更像是说“为...的每一个......”。如果使用exists模式，则不管工作内存中与exists模式内部条件匹配的数据量多少，该规则最多只会激活一次。由于只有存在很重要，所以不会建立任何约束。

该关键字exists必须在其适用的CE周围加括号。在最简单的情况下（如下），您可以省略括号。

*例子159.至少有一个总线*

**exists** Bus()

*例160.至少有一个红色巴士*

**exists** Bus(color == "red")

// brackets are optional:

**exists** ( Bus(color == "red", number == 42) )

// "exists" with nested infix **and**,

// brackets are required:

**exists** ( Bus(color == "red") **and**

Bus(color == "blue") )

##### [8.8.3.7。高级条件元素](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_advanced_conditional_elements)

###### [条件元素 forall](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_forall_code)

*图195.福尔*

条件元素forall完成了Drools中的一阶逻辑支持。forall当匹配第一个模式的所有事实匹配所有剩余模式时，条件元素的计算结果为true。例：

rule "All English buses are red"

when

forall( $bus : Bus( type == 'english')

Bus( this == $bus, color = 'red' ) )

**then**

// all English buses are red

end

在上述规则中，我们“选择”所有类型为“english”的总线对象。然后，对于与此模式相匹配的每个事实，我们评估以下模式，如果它们匹配，则全部CE将评估为真。

为了陈述在工作记忆中给定类型的所有事实必须匹配一组约束，forall可以用简单的单一模式书写。例：

*例161.单一模式*

rule "All Buses are Red"

**when**

**forall**( Bus( color == 'red' ) )

then

*// all Bus facts are red*

end

另一个例子显示了多种模式forall：

*例162.多模式全部*

rule "all employees have health and dental care programs"

**when**

forall( $emp : Employee()

HealthCare( employee == $emp )

DentalCare( employee == $emp )

)

then

// all employees have health **and** dental care

end

Forall可以嵌套在其他CE中。例如，forall可以在notCE 内部使用。请注意，只有单个模式具有可选的括号，因此forall必须使用嵌套括号：

*例163.将Forall和Not CE结合起来*

rule "not all employees have health and dental care"

**when**

**not** ( forall( $emp : Employee()

HealthCare( employee == $emp )

DentalCare( employee == $emp ) )

)

then

// **not** all employees have health **and** dental care

end

作为一个附注，forall( p1 p2 p3…​)相当于写作：

**not**(p1 **and** **not**(**and** p2 p3...))

此外，重要的是要注意这forall是一个范围分隔符。因此，它可以使用任何先前绑定的变量，但是它内部没有绑定的变量可用于其外部。

###### [条件元素 from](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_from_code)

*图196.来自*

条件元素from使用户能够指定一个任意的数据源，以便通过LHS模式进行匹配。这使得引擎可以推理不在工作存储器中的数据。数据源可以是绑定变量的子字段或方法调用的结果。这是一个强大的结构，可以与其他应用程序组件和框架集成在一起。一个常见的例子是与使用hibernate命名查询的数据库按需检索的数据的集成。

用于定义对象源的表达式是遵循常规MVEL语法的任何表达式。因此，它允许您轻松使用对象属性导航，执行方法调用以及访问地图和集合元素。

以下是推理和绑定另一个模式子字段的简单示例：

rule "validate zipcode"

when

Person( $personAddress : address )

Address( zipcode == "23920W") from $personAddress

**then**

// zip code is ok

end

凭借Drools引擎中新的表现力的所有灵活性，您可以通过多种方式切分和解决此问题。这是一样的，但显示了如何使用'from'使用图表符号：

rule "validate zipcode"

when

$p : Person( )

$a : Address( zipcode == "23920W") from $p.address

**then**

// zip code is ok

end

先前的例子是使用单一模式的评估。CE from还支持返回对象集合的对象源。在这种情况下，from将遍历集合中的所有对象并尝试单独匹配它们中的每一个。例如，如果我们想要一个规则在订单中对每个项目应用10％的折扣，我们可以这样做：

rule "apply 10% discount to all items over US$ 100,00 in an order"

when

$order : Order()

$item : OrderItem( value > 100 ) from $order.items

**then**

// apply discount to $item

end

以上示例将针对每个给定订单的每个值大于100的商品启动规则。

但是，在使用from时请特别小心，特别是与lock-on-active规则属性一起使用时，可能会产生意外的结果。考虑前面提供的示例，但现在稍微修改如下：

rule "Assign people in North Carolina (NC) to sales region 1"

ruleflow-group "test"

**lock**-**on**-active true

**when**

$**p** : Person( )

$a : Address( state == "NC") **from** $**p**.address

**then**

**modify** ($**p**) {} // Assign person **to** sales region 1 **in** a **modify** **block**

**end**

rule "Apply a discount to people in the city of Raleigh"

ruleflow-**group** "test"

**lock**-**on**-active true

**when**

$**p** : Person( )

$a : Address( city == "Raleigh") **from** $**p**.address

**then**

**modify** ($**p**) {} // **Apply** discount **to** person **in** a **modify** **block**

**end**

在上面的例子中，北卡罗来纳州罗利市的人员应被分配到销售区域1并获得折扣; 也就是说，你会希望两条规则都可以激活并激活。相反，你会发现只有第二条规则会触发。

如果您打开审计日志，您还会看到第二个规则触发时会停用第一条规则。由于规则属性lock-on-active阻止规则在一组事实更改时创建新激活，所以第一条规则无法重新激活。虽然事实并没有改变，但from每次评估时，使用都会为所有意图和目的返回一个新的事实。

首先，回顾为什么要使用上述模式非常重要。您可能在不同的规则流组中有许多规则。当规则修改RuleFlow下游的工作内存和其他规则（在不同的规则流组中）需要重新评估时，使用modify是至关重要的。但是，您不需要在同一个规则流组中的其他规则以递归方式将激活彼此放在一起。在这种情况下，该no-loop属性是无效的，因为它只会阻止规则以递归方式自行激活。因此，你诉诸于lock-on-active。

有几种方法可以解决这个问题：

* 避免使用from可以将所有事实断言为工作内存或在约束表达式中使用嵌套对象引用（如下所示）。
* 将在修改块中使用的变量作为条件（LHS）中的最后一个句子。
* 避免使用lock-on-active可以明确管理同一规则流组中的规则如何将激活彼此放置（如下所述）的方式。

首选的解决方案是尽量减少直接使用from所有事实进入工作记忆的时间。在上面的例子中，Person和Address实例都可以被声明为工作内存。在这种情况下，因为图形非常简单，所以更简单的解决方案是修改规则，如下所示：

rule "Assign people in North Carolina (NC) to sales region 1"

ruleflow-group "test"

**lock**-**on**-active true

**when**

$**p** : Person(address.state == "NC" )

**then**

**modify** ($**p**) {} // Assign person **to** sales region 1 **in** a **modify** **block**

**end**

rule "Apply a discount to people in the city of Raleigh"

ruleflow-**group** "test"

**lock**-**on**-active true

**when**

$**p** : Person(address.city == "Raleigh" )

**then**

**modify** ($**p**) {} //**Apply** discount **to** person **in** a **modify** **block**

**end**

现在，你会发现两条规则如预期的那样起火。但是，并不总是可以像上面那样访问嵌套的事实。考虑一个人拥有一个或多个地址的例子，并且您希望使用存在量词来匹配至少有一个地址符合特定条件的人。在这种情况下，你不得不求助于from推理集合。

有几种方法可以from实现这一点，并不是所有的方法都会出现使用问题lock-on-active。例如，以下使用from会导致两个规则按预期触发：

rule "Assign people in North Carolina (NC) to sales region 1"

ruleflow-group "test"

**lock**-**on**-active true

**when**

$**p** : Person($addresses : addresses)

**exists** (Address(state == "NC") **from** $addresses)

**then**

**modify** ($**p**) {} // Assign person **to** sales region 1 **in** a **modify** **block**

**end**

rule "Apply a discount to people in the city of Raleigh"

ruleflow-**group** "test"

**lock**-**on**-active true

**when**

$**p** : Person($addresses : addresses)

**exists** (Address(city == "Raleigh") **from** $addresses)

**then**

**modify** ($**p**) {} // **Apply** discount **to** person **in** a **modify** **block**

**end**

但是，下面的稍微不同的方法显示了这个问题：

rule "Assign people in North Carolina (NC) to sales region 1"

ruleflow-group "test"

**lock**-**on**-active true

**when**

$assessment : Assessment()

$**p** : Person()

$addresses : **List**() **from** $**p**.addresses

**exists** (Address( state == "NC") **from** $addresses)

**then**

**modify** ($assessment) {} // **Modify** assessment **in** a **modify** **block**

**end**

rule "Apply a discount to people in the city of Raleigh"

ruleflow-**group** "test"

**lock**-**on**-active true

**when**

$assessment : Assessment()

$**p** : Person()

$addresses : **List**() **from** $**p**.addresses

**exists** (Address( city == "Raleigh") **from** $addresses)

**then**

**modify** ($assessment) {} // **Modify** assessment **in** a **modify** **block**

**end**

在上面的例子中，$ addresses变量是从使用中返回的from。这个例子还引入了一个新的对象，评估，以突出这种情况下的一个可能的解决方案。如果在条件（LHS）中分配的$评估变量移至每个规则中的最后一个条件，则两个规则都会按预期启动。

虽然上面的例子演示了如何将使用from与lock-on-active不发生规则激活的损失结合起来，但它们带来了依赖于LHS上的条件顺序的缺点。此外，解决方案对于规则作者在跟踪哪些条件可能产生问题方面呈现出更大的复杂性。

更好的选择是将更多的事实说成工作记忆。在这种情况下，一个人的地址可能被置入工作记忆中，并且from不需要使用。

然而，有些情况下，将所有数据断言为工作记忆并不实际，我们需要寻找其他解决方案。另一种选择是重新评估需求lock-on-active。另一种方法lock-on-active是直接管理同一规则流程组中的规则如何通过在每个规则中包含条件来防止规则在修改工作内存时以递归方式激活对方。例如，在上述向Raleigh公民申请折扣的情况下，可以将条件添加到检查折扣是否已被应用的规则中。如果是这样，则该规则不会被激活。

|  |  |
| --- | --- |
|  | 如下例所示，包含from子句的模式不能跟随以括号开头的其他模式  rule R **when**  $l : List( )  String() from $l  (String() **or** Number())  **then** **end**  这是因为在这种情况下，DRL解析器将from表达式读为“from $ l（String（）or Number（））”，并且不可能从函数调用中消除这个表达式的歧义。直接解决这个问题的方法是将括号中的from子句也包括在内，如下所示：  rule R **when**  $l : List( )  (String() from $l)  (String() **or** Number())  **then** **end** |

###### [条件元素 collect](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_collect_code)

*图197.收集*

条件元素collect允许规则推理从给定源或工作存储器获得的对象的集合。在First Oder逻辑术语中，这是基数量词。一个简单的例子：

import java.util.ArrayList

rule "Raise priority if system has more than 3 pending alarms"

**when**

$system : System()

$alarms : ArrayList( size >= 3 )

from collect( Alarm( **system** == $system, status == 'pending' ) )

then

// Raise priority, because **system** $system has

// 3 **or** more alarms pending. The pending alarms

// are $alarms.

end

在上面的例子中，规则将查找每个给定系统的工作内存中的所有未决警报，并将它们分组到ArrayLists中。如果给定系统发现3个或更多警报，则规则将会触发。

结果模式collect可以是任何实现java.util.Collection接口的具体类，并提供默认的无参数公共构造函数。这意味着您可以使用Java集合，如ArrayList，LinkedList，HashSet等，或者您自己的类，只要它实现java.util.Collection接口并提供缺省的无参数公共构造函数。

源模式和结果模式都可以被限制为任何其他模式。

在collectCE 之前绑定的变量处于源和结果模式的范围内，因此您可以使用它们来约束源模式和结果模式。但请注意，这collect是绑定的范围分隔符，因此在其内部进行的任何绑定都无法在其外部使用。

Collect收取嵌套的fromCE。以下示例是“收集”的有效用法：

import java.util.LinkedList;

rule "Send a message to all mothers"

**when**

$town : Town( name == 'Paris' )

$mothers : LinkedList()

**from** **collect**( Person( gender == 'F', children > 0 )

**from** $town.**getPeople**()

)

then

*// send a message to all mothers*

end

###### [条件元素 accumulate](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_accumulate_code)

*图198.积累*

条件元素accumulate是一种更加灵活和强大的形式collect，它可以用来做什么collect并且达到CE collect无法实现的结果。累积允许规则遍历一组对象，为每个元素执行自定义操作，最后返回一个结果对象。

累积支持使用预定义的累加函数或使用内联自定义代码。内联自定义代码应该尽量避免，因为规则制作者难以维护，并且经常导致代码重复。累积函数更容易测试和重用。

Accumulate CE还支持多种不同的语法。正如下文所述，首选语法是最高级别的累积，但所有其他语法都支持向后兼容。

###### [积累CE（首选语法）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_accumulate_ce_preferred_syntax)

顶级累加语法是最紧凑和灵活的语法。简化语法如下所示：

accumulate( <source pattern>; <functions> [;<constraints>] )

例如，计算给定传感器的最小值，最大值和平均温度读数并且如果最小温度低于20C并且平均值超过70C度则发出警报的规则可以通过以下方式使用积累来写入：

|  |  |
| --- | --- |
|  | DRL语言将“acc”定义为“accumulate”的同义词。作者可能更倾向于使用“acc”作为较少的详细关键字或使用完整关键字“accumulate”来表示可读性。 |

rule "Raise alarm"

**when**

$s : Sensor()

accumulate( Reading( sensor == $s, $temp : temperature );

$min : min( $temp ),

$max : max( $temp ),

$avg : average( $temp );

$min < 20, $avg > 70 )

then

// raise the **alarm**

end

在上面的例子中，最小值，最大值和平均值是累积函数，并将计算每个传感器的所有读数的最小值，最大值和平均值。

Drools附带一些内置的累积功能，包括：

* 平均
* 分
* 最大
* 计数
* 和
* 方差
* standardDeviation
* collectList
* collectSet

这些常用函数接受任何表达式作为输入。例如，如果有人想计算订单中所有项目的平均利润，则可以使用平均函数编写规则：

rule "Average profit"

when

$order : Order()

accumulate( OrderItem( order == $order, $cost : cost, $price : price );

$avgProfit : average( 1 - $cost / $price ) )

**then**

// average profit **for** $order is $avgProfit

end

累加功能都是可插拔的。这意味着，如果需要，可以轻松地将自定义域特定功能添加到引擎，并且规则可以开始无任何限制地使用它们。要实现一个新的累加函数，所有需要做的就是创建一个实现该org.kie.api.runtime.rule.AccumulateFunction接口的Java类。作为累积函数实现的一个例子，下面是该average函数的实现：

*/\*\**

*\* An implementation of an accumulator capable of calculating average values*

*\*/*

**public** **class** **AverageAccumulateFunction** **implements** **org**.**kie**.**api**.**runtime**.**rule**.**AccumulateFunction**<**AverageAccumulateFunction**.**AverageData**> {

**public** **void** **readExternal**(ObjectInput in) **throws** IOException, ClassNotFoundException {

}

**public** **void** **writeExternal**(ObjectOutput out) **throws** IOException {

}

**public** **static** **class** **AverageData** **implements** **Externalizable** {

**public** **int** count = 0;

**public** **double** total = 0;

**public** **AverageData**() {}

**public** **void** **readExternal**(ObjectInput in) **throws** IOException, ClassNotFoundException {

count = in.readInt();

total = in.readDouble();

}

**public** **void** **writeExternal**(ObjectOutput out) **throws** IOException {

out.writeInt(count);

out.writeDouble(total);

}

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#createContext()*

*\*/*

**public** AverageData **createContext**() {

**return** **new** AverageData();

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#init(java.io.Serializable)*

*\*/*

**public** **void** **init**(AverageData context) {

context.count = 0;

context.total = 0;

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#accumulate(java.io.Serializable, java.lang.Object)*

*\*/*

**public** **void** **accumulate**(AverageData context,

Object value) {

context.count++;

context.total += ((Number) value).doubleValue();

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#reverse(java.io.Serializable, java.lang.Object)*

*\*/*

**public** **void** **reverse**(AverageData context, Object value) {

context.count--;

context.total -= ((Number) value).doubleValue();

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#getResult(java.io.Serializable)*

*\*/*

**public** Object **getResult**(AverageData context) {

**return** **new** Double( context.count == 0 ? 0 : context.total / context.count );

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#supportsReverse()*

*\*/*

**public** **boolean** **supportsReverse**() {

**return** **true**;

}

*/\* (non-Javadoc)*

*\* @see org.kie.api.runtime.rule.AccumulateFunction#getResultType()*

*\*/*

**public** Class< ? > getResultType() {

**return** Number.class;

}

}

正如我们所期望的，该功能的代码非常简单，因为所有“脏”的整合工作都是由引擎完成的。最后，要使用规则中的函数，作者可以使用“导入累积”语句导入它：

import accumulate <class\_name> <function\_name>

例如，如果实现了some.package.VarianceFunction实现该variance函数的类函数并希望在规则中使用它，他将执行以下操作：

*示例164.导入和使用自定义“变量”累加函数的示例*

import accumulate some.package.VarianceFunction variance

rule "Calculate Variance"

when

accumulate( Test( $s : score ), $v : variance( $s ) )

**then**

// the variance of the test scores is $v

end

|  |  |
| --- | --- |
|  | 内置函数（总和，平均等）由引擎自动导入。只有用户定义的自定义累加函数需要显式导入。 |
|  | 为了向后兼容，Drools仍然支持通过配置文件和系统属性配置累加功能，但这是一个不推荐的方法。为了使用配置文件或系统属性配置上一个示例中的差异函数，用户可以像下面这样设置一个属性：  drools.accumulate.function.variance = some.**package**.VarianceFunction  请注意，“ drools.accumulate.function.”是必须始终使用的前缀，“ variance”是函数将如何在drl文件中使用，“ some.package.VarianceFunction”是实现函数行为的类的完全限定名称。 |

###### [备选语法：具有返回类型的单个函数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_alternate_syntax_single_function_with_return_type)

随着时间的推移，累积语法会随着变得更加紧凑和表达的目标而演变。尽管如此，Drools仍然支持以前的语法以实现向后兼容。

如果规则在给定累积中使用单个累加函数，则作者可以为结果对象添加一个模式，并使用“from”关键字将其链接到累加结果。例如：对超过100美元的订单应用10％折扣的规则可以通过以下方式编写：

rule "Apply 10% discount to orders over US$ 100,00"

when

$order : Order()

$total : Number( doubleValue > 100 )

from accumulate( OrderItem( order == $order, $value : value ),

sum( $value ) )

**then**

// apply discount to $order

end

在上面的例子中，accumulate元素只使用一个函数（sum），因此规则作者选择显式地为累积函数（Number）的结果类型编写一个模式，并在其中写入约束。在前面介绍的紧凑语法中使用此语法没有任何问题，除了它稍微冗长些。另请注意，不允许在同一个累计语句中使用返回类型和绑定函数。

执行编译时检查是为了确保与“ from”关键字一起使用的模式可以从所使用的累加函数的结果中分配。

|  |  |
| --- | --- |
|  | 使用这种语法，“ from”绑定到由累积函数返回的单个结果，并且它不会迭代。 |

在上面的例子中，“ $total”被绑​​定到accumulate sum（）函数返回的结果。

然而，作为另一个例子，如果accumulate函数的结果是一个集合，“ from”仍然绑定到单个结果，并且它不会迭代：

rule "Person names"

when

$x : Object() from accumulate(MyPerson( $val : name );

collectList( $val ) )

**then**

// $x is a List

end

绑定的“ $x : Object()”是List本身，由collectList accumulate函数使用返回。

这是一个突出的重要区别，因为“ from”关键字也可以单独使用累加来遍历集合的元素：

rule "Iterate the numbers"

**when**

$xs : List()

$x : Integer() from $xs

then

// $x matches **and** binds to **each** Integer in the collection

end

虽然此语法仍支持向后兼容的目的，但出于此原因和其他原因，我们鼓励规则制定者使用，而不是Accumulate CE首选语法（在前一章中描述），以避免任何潜在的陷阱，如这些示例所述。

###### [用内联自定义代码累积](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_accumulate_with_inline_custom_code)

|  |  |
| --- | --- |
|  | 使用内联自定义代码进行累积并不是一个好的做法，原因有很多，其中包括维护和测试使用它们的规则方面的困难，以及无法重用该代码。实现自己的累积函数非常简单直接，它们很容易进行单元测试和使用。这种累积形式仅支持向后兼容性。 |

累加的另一种可能语法是定义内联自定义代码，而不是使用累加函数。正如前面提到的警告所指出的那样，虽然出于陈述的原因，但这并不鼓励。

accumulate内联自定义代码的CE 的一般语法是：

<result pattern> from accumulate( <source pattern>,

init( <init code> ),

action( <action code> ),

reverse( <reverse code> ),

result( <result expression> ) )

每个元素的含义如下：

* <源模式>：源模式是引擎将尝试匹配每个源对象的常规模式。
* <init code>：这是选定方言中的代码的语义块，它将在迭代源对象之前为每个元组执行一次。
* <action code>：这是选定方言中的代码语义块，将针对每个源对象执行。
* <反向代码>：这是所选方言中可选的代码语义块，如果存在的话将针对不再与源模式匹配的每个源对象执行。该代码块的目标是撤销在<action code>块中完成的任何计算，以便在源对象被修改或删除时引擎可以执行递减计算，极大地提高了这些操作的性能。
* <结果表达式>：这是所选语言中的一个语义表达式，在所有源对象迭代后执行。
* <结果模式>：这是引擎尝试匹配从<result expression>返回的对象的常规模式。如果匹配，则accumulate条件元素的计算结果为true，并且引擎继续评估规则中的下一个CE。如果不匹配，则accumulateCE评估为假，并且引擎停止评估该规则的CE。

如果我们看一个例子，则更容易理解：

rule "Apply 10% discount to orders over US$ 100,00"

**when**

$order : Order()

$total : Number( doubleValue > 100 )

from accumulate( OrderItem( order == $order, $value : value ),

init( double total = 0; ),

action( total += $value; ),

**reverse**( total -= $value; ),

result( total ) )

then

// apply discount to $order

end

在上面的例子中，对于Order工作内存中的每一个，引擎都会执行初始化代码，初始化总变量为零。然后，它将迭代所有OrderItem对象的顺序，为每个对象执行操作（在本例中，它将所有项目的值总和到总变量中）。遍历所有OrderItem对象后，它将返回对应于结果表达式的值（在上面的例子中，是变量的值total）。最后，引擎会尝试将结果与Number模式进行匹配，如果double值大于100，则规则将触发。

该示例使用Java作为语义方言，因此请注意，在init，action和reverse代码块中必须使用分号作为语句分隔符。结果是一个表达，因此它不承认';'。如果用户使用任何其他方言，他必须遵守该方言的特定语法。

如前所述，反向代码是可选的，但强烈建议用户编写它，以便从更新和删除方面的改进性能中受益。

该accumulateCE可以被用来执行对源对象的任何行动。以下示例实例化并填充自定义对象：

rule "Accumulate using custom objects"

when

$person : Person( $likes : likes )

$cheesery : Cheesery( totalAmount > 100 )

from accumulate( $cheese : Cheese( type == $likes ),

init( Cheesery cheesery = new Cheesery(); ),

action( cheesery.addCheese( $cheese ); ),

reverse( cheesery.removeCheese( $cheese ); ),

result( cheesery ) );

**then**

// **do** something

end

##### [8.8.3.8。条件元素eval](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_element_code_eval_code)

*图199. eval*

条件元素eval本质上是一个可以执行任何语义代码（返回一个原始布尔值）的捕获。此代码可以引用规则LHS中绑定的变量以及规则包中的函数。过度使用eval会降低规则的声明性，并可能导致性能较差的引擎。虽然eval可以在模式中的任何位置使用，但最佳做法是将其添加为规则LHS中的最后一个条件元素。

Evals不能编入索引，因此效率不如字段约束。然而，这使得它们非常适合在函数返回随时间变化的值时使用，这在Field Constraints中是不允许的。

对于熟悉Drools 2.x谱系的人来说，旧的Drools参数和条件标签相当于将变量绑定到适当的类型，然后在eval节点中使用它。

p1 : Parameter()

p2 : Parameter()

eval( p1.getList().containsKey( p2.getItem() ) )

p1 : Parameter()

p2 : Parameter()

// call **function** isValid **in** the LHS

eval( isValid( p1, p2 ) )

##### [8.8.3.9。铁路图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_railroad_diagrams)

#### [8.8.4。右手边（当时）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_right_hand_side_then)

##### [8.8.4.1。用法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_usage)

右手边（RHS）是该规则的后果或行动部分的通用名称; 这部分应该包含一个要执行的动作列表。在规则的RHS中使用命令式或条件式代码是不好的做法; 作为一个规则应该是原子性的 - “当这个，然后做这个”，而不是“当这个，也许这样做”。规则的RHS部分也应该保持小，从而保持它的声明性和可读性。如果你发现你需要RHS中的命令和/或条件代码，那么你可能应该将这条规则分解成多条规则。RHS的主要目的是插入，删除或修改工作内存数据。为了解决这个问题，您可以使用一些方便的方法来修改工作记忆; 而无需首先引用工作内存实例。

update(对象，句柄);会告诉引擎一个对象已经发生了变化（一种被绑定到LHS上的东西），并且规则可能需要重新考虑。

update(对象);也可以使用; 在这里，知识助手将通过身份检查为您查找事实手段，以查找传递的对象。（请注意，如果您将Property Change Listener提供给要插入到引擎中的Java bean，则可以避免update()在对象更改时需要调用）。在事实的字段值发生了变化之后，您必须在更改其他事实之前调用update，否则将导致规则引擎中的索引出现问题。修改关键字可以避免这个问题。

insert(new有些东西());会将您创作的新物件放入工作记忆中。

insertLogical(new有些东西());与插入相似，但当没有更多事实来支持当前触发规则的真实情况时，该对象将被自动删除。

delete(句柄);从工作内存中删除一个对象。

这些便捷方法基本上是宏，它们为KnowledgeHelper实例提供了快捷方式，使您可以从规则文件访问工作内存。drools类型的预定义变量KnowledgeHelper可让您调用其他几种有用的方法。（请参阅KnowledgeHelper接口文档以获取更多高级操作）。

* 该呼叫drools.halt()立即终止规则执行。这是将控制返回到当前会话所处的位置所必需的fireUntilHalt()。
* 方法insert(Object o)，update(Object o)并且delete(Object o)可以被调用drools，但由于它们的频繁使用，可以在没有对象引用的情况下调用它们。
* drools.getWorkingMemory()返回WorkingMemory对象。
* drools.setFocus( String s) 将焦点设置到指定的议程组。
* drools.getRule().getName()，从规则的RHS调用，返回规则的名称。
* drools.getTuple()返回与当前执行的规则匹配的元组，并drools.getActivation()传递相应的激活。（这些调用对于记录和调试目的很有用。）

完整的知识运行时API通过另一个预定义变量kcontext（类型）公开KieContext。它的方法getKieRuntime()提供了一个类型的对象KieRuntime，这反过来又提供了大量方法的访问，其中许多方法对编译RHS逻辑非常有用。

* 该呼叫kcontext.getKieRuntime().halt()立即终止规则执行。
* 访问者getAgenda()返回对此会话的引用Agenda，从而访问各种规则组：激活组，议程组和规则流组。一个相当常见的范例是激活一些议程组，这可以用冗长的呼叫完成：
* *// give focus to the agenda group CleanUp*

kcontext.getKieRuntime().getAgenda().getAgendaGroup( "CleanUp" ).setFocus();

（您可以使用相同的方法drools.setFocus( "CleanUp" )。）

* 要运行查询，请致电getQueryResults(String query)，然后您可以处理结果，如[查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_querysection)一节中所述。使用kcontext.getKieRuntime().getQueryResults()或使用drools.getKieRuntime().getQueryResults()是从规则的RHS运行查询的正确方法，也是唯一受支持的方式。
* 处理事件管理的一组方法可以让您为工作内存和议程添加和删除事件侦听器。
* 方法getKieBase()返回KieBase对象，系统中所有知识的骨干，以及当前会话的发起者。
* 您可以管理与全局setGlobal(…​)，getGlobal(…​)和getGlobals()。
* 方法getEnvironment()返回的运行时间Environment与您操作系统的环境非常相似。

##### [8.8.4.2。该modify声明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_code_modify_code_statement)

这种语言扩展为事实更新提供了一种结构化方法。它将更新操作与许多setter调用结合起来，以改变对象的字段。这是该modify语句的语法模式：

modify ( <fact-expression> ) {

<expression> [ , <expression> ]\*

}

带括号的<fact-expression>必须产生一个事实对象引用。块中的表达式列表应该由给定对象的setter调用组成，而不需要通常的对象引用，编译器会自动预先编写该对象引用。

该例子说明了一个简单的事实修改。

*例子165.修改语句*

rule "modify stilton"

when

$stilton : Cheese(type == "stilton")

**then**

modify( $stilton ){

setPrice( 20 ),

setAge( "overripe" )

}

end

使用修改语句的优点在与细粒度属性更改侦听器结合使用时尤其明显。请参阅相应章节了解更多详情。

#### [8.8.5。有条件的命名后果](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conditional_named_consequences)

有时，对每条规则有单一结果的限制可能会有所限制，并导致冗长而难以保持重复，如下例所示：

rule "Give 10% discount to customers older than 60"

**when**

$customer : Customer( age > 60 )

**then**

modify($customer) { setDiscount( 0.1 ) };

**end**

rule "Give free parking to customers older than 60"

**when**

$customer : Customer( age > 60 )

$car : Car ( owner == $customer )

**then**

modify($car) { setFreeParking( **true** ) };

**end**

已经有可能通过使第二个规则扩展第一个规则来部分地克服这个问题，如下所示：

rule "Give 10% discount to customers older than 60"

**when**

$customer : Customer( age > 60 )

**then**

modify($customer) { setDiscount( 0.1 ) };

**end**

rule "Give free parking to customers older than 60"

extends "Give 10% discount to customers older than 60"

**when**

$car : Car ( owner == $customer )

**then**

modify($car) { setFreeParking( **true** ) };

**end**

无论如何，该功能可以在单个规则中定义除默认规则以外的更多标签化后果，例如，2个前规则可以按如下方式进行压缩：

rule "Give 10% discount and free parking to customers older than 60"

when

$customer : Customer( age > 60 )

**do**[giveDiscount]

$car : Car ( owner == $customer )

**then**

modify($car) { setFreeParking( true ) };

**then**[giveDiscount]

modify($customer) { setDiscount( 0.1 ) };

end

一旦在知识库中发现年龄大于60岁的客户，该最后一条规则有两个后果，即通常的默认规则，另一个名为“giveDiscount”，该规则被激活，使用关键字do，无论他拥有是否有车。命名后果的激活也可以通过附加条件来保护，例如：

rule "Give free parking to customers older than 60 and 10% discount to golden ones among them"

when

$customer : Customer( age > 60 )

**if** ( type == "Golden" ) **do**[giveDiscount]

$car : Car ( owner == $customer )

**then**

modify($car) { setFreeParking( true ) };

**then**[giveDiscount]

modify($customer) { setDiscount( 0.1 ) };

end

if语句中的条件总是在它之前的模式上进行评估。最后，最后一个更复杂的示例展示了如何使用嵌套的if / else语句切换不同的条件：

rule "Give free parking and 10% discount to over 60 Golden customer and 5% to Silver ones"

when

$customer : Customer( age > 60 )

**if** ( type == "Golden" ) **do**[giveDiscount10]

**else** **if** ( type == "Silver" ) break[giveDiscount5]

$car : Car ( owner == $customer )

**then**

modify($car) { setFreeParking( true ) };

**then**[giveDiscount10]

modify($customer) { setDiscount( 0.1 ) };

**then**[giveDiscount5]

modify($customer) { setDiscount( 0.05 ) };

end

这里的目的是给黄金客户提供10％的折扣和一个免费的停车位，但是只有5％的折扣（没有免费停车）。这个结果是通过使用关键字break而不是do来激活名为“giveDiscount5”的后果来实现的。事实上，只是在议程中安排了一个结果，允许剩余部分的LHS继续按常规进行评估，而中断也阻止了进一步的模式匹配评估。注意，当然，没有任何中断条件激活命名结果是没有意义的（并且会产生编译时错误），否则LHS部分跟在它后面永远不可达。

#### [8.8.6。关于自动装箱和原始类型的注意事项](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_note_on_auto_boxing_and_primitive_types)

Drools尝试保留原始对象或对象封装形式的数字，因此在代码块或表达式中使用时绑定到int基元的变量不再需要手动拆箱; 不像Drools 3.0那样，所有原始图片都是自动复制的，需要手动拆箱。绑定到对象包装的变量将保持为对象; 在这种情况下，现有的JDK 1.5和JDK 5规则处理自动装箱和拆箱。评估现场限制时，系统试图将其中一个值强制为可比较的格式; 所以一个原语可以和一个对象包装器相媲美。

### [8.9。询问](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_querysection)

*图200.查询*

查询是在工作记忆中搜索符合规定条件的事实的简单方法。因此，它只包含规则LHS的结构，所以既不指定“何时”也不指定“然后”。查询有一组可选的参数，每个参数都可以选择输入。如果没有给出类型，则假定类型为Object。引擎会根据需要尝试强制这些值。查询名称对于KieBase来说是全球性的; 所以不要将相同名称的查询添加到同一个RuleBase的不同包中。

要返回结果使用ksession.getQueryResults("name")，其中“name”是查询的名称。这将返回一个查询结果列表，它允许您检索与查询匹配的对象。

第一个例子为所有30岁以上的人提供了一个简单的查询。第二个例子使用参数将年龄限制与位置相结合。

*例子166.查询30岁以上的人*

query "people over the age of 30"

person : Person( age > 30 )

end

*例子167.查询x年龄以及y年的人*

query "people over the age of x" (**int** x, String y)

person : Person( age > x, location == y )

end

我们使用标准的“for”循环遍历返回的QueryResults。每个元素都是一个QueryResultsRow，我们可以使用它来访问元组中的每个列。这些列可以通过绑定的声明名称或索引位置来访问。

*例子168.查询30岁以上的人*

QueryResults results = ksession.getQueryResults( "people over the age of 30" );

System.out.println( "we have " + results.size() + " people over the age of 30" );

System.out.println( "These people are are over 30:" );

**for** ( QueryResultsRow row : results ) {

Person person = ( Person ) row.get( "person" );

System.out.println( person.getName() + "\n" );

}

为更紧凑的代码添加了对位置语法的支持。默认情况下，类型声明中声明的类型顺序与参数位置匹配。但是可以使用@position注释覆盖这些。这允许模式与位置参数一起使用，而不是更详细的命名参数。

**declare** Cheese

**name** : **String** @**position**(1)

shop : **String** @**position**(2)

price : int @**position**(0)

**end**

org.drools.definition.type包中的@Position注释可用于注释类路径上的原始pojos。目前只有类上的字段可以注释。支持类的继承，但不支持接口或方法。下面的isContainedIn查询演示了模式中位置参数的使用; Location(x, y;)代替Location( thing == x, location == y).

查询现在可以调用其他查询，这与可选查询参数结合提供派生查询风格反向链接。参数支持位置和命名语法。它也可以混合位置和名称，但位置必须先来，用分号分隔。文字表达式可以作为查询参数传递，但在此阶段，您不能将表达式与变量混合使用。这是调用另一个查询的查询的示例。请注意，这里的'z'将始终是'out'变量。'？' 符号表示查询仅为拉取，一旦结果返回，您将不会收到进一步的结果，因为基础数据发生变化。

**declare** Location

thing : **String**

location : **String**

**end**

**query** isContainedIn( **String** x, **String** y )

Location(x, y;)

or

( Location(z, y;) and ?isContainedIn(x, z;) )

**end**

如前所述，您可以使用实时“开放”查询从查询​​结果中随时间变化地接收更改，作为查询变更的基础数据。注意，“look”规则在不使用'？'的情况下调用查询。

query isContainedIn( String x, String y )

Location(x, y;)

**or**

( Location(z, y;) **and** isContainedIn(x, z;) )

**end**

rule look **when**

Person( $l : likes )

isContainedIn( $l, 'office'; )

**then**

insertLogical( $l 'is in the office' );

**end**

Drools支持派生查询的统一，简而言之，这意味着参数是可选的。可以使用静态字段org.drools.core.runtime.rule.Variable.v从Java调用未指定参数的查询 - 请注意，您必须使用'v'而不是变量的替代实例。这些被称为'out'参数。请注意，查询本身并未在编译时声明参数是处于还是不在，这可以在每次使用时纯粹在运行时定义。以下示例将返回办公室中包含的所有对象。

results = ksession.getQueryResults( "isContainedIn", **new** Object[] { Variable.v, "office" } );

l = **new** ArrayList<List<String>>();

**for** ( QueryResultsRow r : results ) {

l.add( Arrays.asList( **new** String[] { (String) r.get( "x" ), (String) r.get( "y" ) } ) );

}

该算法使用堆栈来处理递归，所以方法堆栈不会炸毁。

也可以将查询的两个字段用作查询的输入参数，如下所示：

query contains(String $s, String $c)

$s := String( this.contains( $c ) )

**end**

rule PersonNamesWithA **when**

$p : Person()

contains( $p.name, "a"; )

**then**

**end**

一般来说任何类型的有效表达式都可以用于：

query checkLength(String $s, int $l)

$s := String( length == $l )

**end**

rule CheckPersonNameLength **when**

$i : Integer()

$p : Person()

checkLength( $p.name, 1 + $i + $p.age; )

**then**

**end**

以下内容尚未支持：

* 列表和地图统一
* 表达式统一 - pred（X，X + 1，X \* Y / 7）

### [8.10。域特定语言](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_domain_specific_languages)

域特定语言（或DSL）是创建专用于问题域的规则语言的一种方式。一组DSL定义由DSL“语句”到DRL结构的转换组成，可让您使用所有基础规则语言和引擎功能。鉴于DSL，您可以在DSL规则（或DSLR）文件中编写规则，将其转换为DRL文件。

DSL和DSLR文件是纯文本文件，您可以使用任何文本编辑器来创建和修改它们。但也有DSL和DSLR编辑器，无论是在IDE中还是在基于Web的BRMS中，也可以使用这些编辑器，尽管它们可能不会提供完整的DSL功能。

#### [8.10.1。何时使用DSL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_when_to_use_a_dsl)

DSL可以作为规则创作（和规则制定者）与由域对象建模和规则引擎的本地语言和方法产生的技术复杂性之间的分离层。如果您的规则需要由不是程序员的领域专家（例如业务分析员）阅读和验证，则应考虑使用DSL; 它隐藏了实现细节并专注于规则逻辑。DSL语句还可以用作条件元素和结果动作的“模板”，这些动作在规则中重复使用，可能会有细微的变化。您可以将DSL句子定义为映射到这些重复的短语，参数提供了适应这些变化的方法。

DSL在运行时对规则引擎没有影响，它们只是一个编译时功能，需要特殊的解析器和变换器。

#### [8.10.2。DSL基础知识](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dsl_basics)

Drools DSL机制允许您自定义条件表达式和结果操作。全球替代机制（“关键字”）也可用。

*示例169.示例DSL映射*

[**when**]Something **is** {colour}=Something(colour=="{colour}")

在前面的例子中，[when]表示表达式的范围，即它对于规则的LHS或RHS是否有效。括号关键字后面的部分是您在规则中使用的表达式; 通常是一种自然语言表达，但它不一定是。等号右边的部分（“=”）是表达式到规则语言的映射。该字符串的形式取决于其目的地，RHS或LHS。如果是用于LHS，那么它应该是根据常规LHS语法的术语; 如果是RHS，那么它可能是一个Java语句。

只要DSL解析器与用DSL定义的表达式写入DSL的规则文件匹配一行，它就执行三个字符串操作步骤。首先，它提取出现在表达式包含大括号变量名称的地方的字符串值（这里：）{colour}。然后，从这些捕获获得的值随后被内插，无论该名称再次包含在大括号中，都出现在映射的右侧。最后，插入的字符串替换DSL规则文件行中整个表达式匹配的内容。

请注意，表达式（即等号左侧的字符串）在与DSL规则文件的一行模式匹配操作中用作正则表达式，匹配全部或部分行。这意味着你可以使用（例如）'？' 以表明前面的字符是可选的。使用它的一个很好的理由是克服DSL中自然语言短语的变化。但是，鉴于这些表达式是正则表达式模式，这也意味着Java模式语法的所有“魔术”字符都必须用前面的反斜杠（'\'）转义。

注意到编译器逐行转换DSL规则文件是很重要的。在上面的例子中，将“Something is”后面的所有文本作为“{color}”的替换值捕获，并将其用于内插目标字符串。这可能不是你想要的。例如，当您打算合并不同的DSL表达式来生成复合DRL图案时，您需要在几个独立的操作中转换DSLR线。实现这一目标的最佳方式是确保捕捉被特征文本 - 单词或甚至单个字符包围。结果，解析器完成的匹配操作从行内的某个地方抽取出一个子字符串。在下面的示例中，引号用作区别字符。

根据经验法则，对规则编辑器可能要输入的文本数据使用引号。您还可以用文字附上捕获文件，以确保文本正确匹配。以下示例说明了这两者。请注意，Something is "green" and another solid thing现在可以正确展开一行。

*示例170.带引号的示例*

[**when**]something **is** "{colour}"=Something(colour=="{colour}")

[**when**]another {state} thing=OtherThing(state=="{state})"

尽可能避免在DSL表达式中标点符号（除引号或撇号外）。主要原因是使用DSL的规则制作者容易忘记标点符号。另一个原因是圆括号，句点和问号是魔法字符，需要在DSL定义中转义。

在DSL映射中，大括号“{”和“}”应该仅用于包含变量定义或引用，从而产生捕获。如果它们应该在字面上出现，无论是在表达式还是在右侧的替换文本中，它们都必须使用前面的反斜杠（“\”）进行转义：

[**then**]**do** something= **if** (foo) \{ **do**Something(); \}

|  |  |
| --- | --- |
|  | 如果括号“{”和“}”应出现在DSL定义的替换字符串中，请使用反斜线（'\'）将其转义。 |

*示例171. DSL映射条目的示例*

*# This is a comment to be ignored.*

[**when**]There **is** a person with name **of** "{name}"=Person(name=="{name}")

[**when**]Person **is** at least {age} years old **and** lives **in** "{location}"=

Person(age >= {age}, location=="{location}")

[**then**]Log "{message}"=System.out.println("{message}");

[**when**]And = **and**

鉴于上述DSL示例，以下示例显示了各种数码单反相机片段的扩展：

*例子172.DSL扩展的例子*

There **is** a person with name **of** "Kitty"

==> Person(name="Kitty")

Person **is** at least 42 years old **and** lives **in** "Atlanta"

==> Person(age >= 42, location="Atlanta")

Log "boo"

==> System.out.println("boo");

There **is** a person with name **of** "Bob" And Person **is** at least 30 years old **and** lives **in** "Utah"

==> Person(name="Bob") **and** Person(age >= 30, location="Utah")

|  |  |
| --- | --- |
|  | 不要忘记，如果您从DSL规则行捕获纯文本并希望将其用作扩展中的字符串文字，则必须在映射的右侧提供引号。 |

您可以将DSL表达式链接在一行上，只要解析器清楚哪里结束，下一个开始以及表示参数的文本结束。（否则，您可能会冒险将所有文本作为参数值获取，直到行尾。）根据在DSL定义文件中的顺序，DSL表达式会一个接一个地尝试。在任何匹配之后，也调查所有剩余的DSL表达式。

生成的DRL文本可能包含多行。行结束符在替换文本中被写为\n。

#### [8.10.3。为事实添加约束](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adding_constraints_to_facts)

编写规则条件时的一个常见要求是能够为模式添加任意约束组合。鉴于事实类型可能有很多字段，必须为每个组合提供单独的DSL语句将是很愚蠢的。

DSL工具允许您通过一个简单的约定为模式添加约束条件：如果您的DSL表达式以连字符（减号，“ - ”）开头，则假定它是一个字段约束，因此将被添加到最后在它之前的图案线。

例如，让我们看看课程Cheese，其中包含以下字段：类型，价格，年龄和国家。我们可以在正常DRL中表示一些LHS条件，如下所示

Cheese(age < 5, price == 20, type=="stilton", country=="ch")

下面给出的DSL定义导致三个DSL短语可用于创建涉及这些字段的任何约束组合。

[**when**]There **is** a Cheese with=Cheese()

[**when**]- age **is** less than {age}=age<{age}

[**when**]- type **is** '{type}'=type=='{type}'

[**when**]- country equal to '{country}'=country=='{country}'

然后您可以使用以下条件编写规则：

There **is** a Cheese **with**

- age **is** less than 42

- type **is** 'stilton'

解析器将拾取以“ - ”开头的行，并将其作为约束添加到前面的模式，并在需要时插入逗号。

对于前面的例子，生成的DRL是：

Cheese(age<42, type=='stilton')

将所有数字字段与所有关系运算符相结合（根据前面示例中的“年龄小于...”的DSL表达式）会产生大量的DSL条目。但是您可以为各种运算符定义DSL短语，甚至可以为处理任何字段约束的泛型表达式定义，如下所示。（注意表达式定义除了变量名以外还包含一个正则表达式。）

[when][]is less than or equal to=<=

[when][]is less than=<

[when][]is greater than or equal to=>=

[when][]is greater than=>

[when][]is equal to===

[when][]equals===

[when][]There is a Cheese with=Cheese()

[when][]- {field:\w\*} {operator} {value:\d\*}={field} {operator} {value}

鉴于这些DSL定义，您可以编写具有以下条件的规则：

There **is** a Cheese **with**

- age **is** less than 42

- rating **is** greater than 50

- type equals 'stilton'

在这种特定情况下，诸如“小于”的短语被替换为<，然后该行与最后的DSL条目匹配。这将删除连字符，但最终结果仍作为约束添加到前面的模式中。处理完所有行后，生成的DRL文本为：

Cheese(age<42, rating > 50, type=='stilton')

|  |  |
| --- | --- |
|  | 如果单独的DSL表达式旨在匹配相同的线路，那么DSL中条目的顺序很重要。 |

#### [8.10.4。开发DSL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_developing_a_dsl)

开始的一个好方法是编写应用程序需要的规则的代表性示例，并在开发时测试它们。这将为您提供稳定的条件元素及其约束框架。DRL和DSLR中的规则是指根据数据模型的实体，该数据模型表示应遵循规则中定义的推理过程的应用程序数据。注意，如果大多数数据模型的类型都是事实，那么编写规则通常会更容易。

给定一组初始规则，应该有可能识别重复或类似的代码片段并将可变部分标记为参数。这为可能是方便的DSL条目提供了可靠的线索。此外，请确保您完全掌握领域专家正在使用的行话，并将您的DSL短语放在这个词汇表上。

在第一个设计阶段，您可以通过在特定的条件元素和操作中添加更大的符号（“>”），在DRL表单中放置某些条件元素和操作，从而推迟实施有关条件和操作的决定。（这对插入调试语句也很方便。）

在下一个开发阶段，你会发现DSL配置稳定得很快。可以通过重新使用现有的DSL定义，或者通过向现有条件或后果条目添加参数来编写新规则。

尽量减少DSL条目的数量。使用参数可让您为相似的规则模式或约束应用相同的DSL句子。但不要夸大：使用DSL的作者应该仍能够通过一些固定文本识别DSL短语。

#### [8.10.5。DSL和DSLR参考](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dsl_and_dslr_reference)

DSL文件是面向行的格式的文本文件。它的条目用于根据DRL语法将DSLR文件转换为文件。

* 一行以““或”//“（有或没有前面的空格）被视为注释。注释行以”/“将被扫描以请求调试选项，请参阅下文。
* 任何以开头括号（“[”）开头的行被假定为DSL条目定义的第一行。
* 任何其他行都会附加到前面的DSL条目定义中，并将行末尾替换为空格。

DSL条目由以下四部分组成：

* 作为范围定义，写成括号（“[”和“]”）中关键字“when”或“condition”，“then”或“consequence”，“\*”和“keyword”之一。这表明DSL条目是否对条件或规则的后果有效，或两者都有效。“关键字”的范围指示意味着该条目具有全球意义，即，它在DSLR文件中的任何位置被识别。
* 用括号括起来的类型定义，写成Java类名称。除非下一部分以开头括号开头，否则该部分是可选的。一对空括号也是有效的。
* DSL表达式由（Java）正则表达式组成，其中包含任意数量的嵌入式变量定义，并由等号（“=”）终止。变量定义包含在大括号中（“{”和“}”）。它由一个变量名和两个可选附件组成，以冒号（“：”）分隔。如果有一个附件，则它是用于匹配要分配给该变量的文本的正则表达式; 如果有两个附件，第一个是GUI编辑器的提示，第二个是正则表达式。

请注意，正则表达式中的所有“魔术”字符必须使用前面的反斜杠（“\”）进行转义，前提是它们应该在表达式中直接出现。

* 定界等号后的行剩余部分是与正则表达式匹配的任何DSLR文本的替换文本。它可能包含变量引用，即括号内的变量名称。或者，变量名后面可能带有感叹号（“！”）和转换函数，如下所示。

请注意，如果花括号（“{”和“}”）必须在字面上出现在替换字符串中，则必须使用前面的反斜杠（“\”）进行转义。

可以选择性地开启DSL扩展调试，方法是使用以“＃/”开头的注释行，该注释行可能包含以下表格中的一个或多个单词。结果输出写入标准输出。

| *表10. DSL扩展的调试选项* | |
| --- | --- |
| **字** | **描述** |
| 结果 | 用行号打印生成的DRL文本。 |
| 脚步 | 打印条件和结果行的每个扩展步骤。 |
| 关键词 | 转储范围为“关键字”的所有DSL条目的内部表示。 |
| 什么时候 | 转储范围为“when”或“\*”的所有DSL条目的内部表示形式。 |
| 然后 | 转储范围为“then”或“\*”的所有DSL条目的内部表示。 |
| 用法 | 显示所有DSL条目的使用情况统计信息。 |

以下是一些示例DSL定义，其中注释描述了他们所说明的语言功能。

# Comment: DSL examples

#/ debug: display result and usage

# keyword definition: replaces "regula" by "rule"

[keyword][]regula=rule

# conditional element: "T" or "t", "a" or "an", convert matched word

[when][][Tt]here is an? {entity:\w+}=

${entity!lc}: {entity!ucfirst} ()

# consequence statement: convert matched word, literal braces

[then][]update {entity:\w+}=modify( ${entity!lc} )\{ \}

DSLR文件的转换过程如下：

1. 文本被读入内存。
2. 每个“关键字”条目都应用于整个文本。首先，关键字定义中的正则表达式通过将白色空间序列替换为匹配任意数量的空格字符的模式并将定义的正则表达式中的变量定义替换为变量定义，或使用默认值（ “\*？”）。然后，详细搜索DSLR文本中是否存在匹配修改的正则表达式的字符串。提取对应于变量捕获的匹配字符串的子串并替换相应替换文本中的变量引用，并且该文本替换DSLR文本中的匹配字符串。
3. 如下所述，分别在“当”和“然后”，“然后”和“结束”之间的DSLR文本部分按照统一的方式逐行定位和处理。

对于一条线路，每条DSL线路段的相关条目将依次显示在DSL文件中。它的正则表达式部分被修改：空白空间被匹配任意数量的空格字符的模式替换; 具有正则表达式的变量定义由具有此正则表达式的捕获替换，其默认值为“。\*？”。如果生成的正则表达式匹配全部或部分行，匹配的部分将被替换为适当修改的替换文本。

替换文本的修改是通过用对应于正则表达式捕获的文本替换变量引用来完成的。该文本可以根据变量引用中给出的字符串变换函数进行修改; 详情见下文。

如果有一个变量引用命名一个未在同一条目中定义的变量，则该扩展器会将绑定到该名称变量的值替换为当前规则的前面一行中定义的值。

1. 如果条件中的DSLR线用连字符编写，则扩展结果将插入最后一行，该行应包含模式CE，即类型名称后跟一对括号。如果该对为空，则扩展行（应包含有效约束）只需插入，否则将事先插入逗号（“，”）。

如果后果中的DSLR线用连字符编写，则扩展结果将插入最后一行，该行应包含“修改”语句，并以一对大括号（“{”和“}”）结尾。如果该对为空，则扩展的行（应包含有效的方法调用）将被简单地插入，否则将事先插入逗号（“，”）。

|  |  |
| --- | --- |
|  | 这是目前没有可以使用符合领先的连字符插入文字等条件元素形式（例如，“积累”），也可以只对第一次插入（例如，“EVAL”）工作。 |

下表中描述了所有字符串转换函数。

| *表11.字符串转换函数* | |
| --- | --- |
| **名称** | **描述** |
| UC | 将所有字母转换为大写。 |
| LC | 将所有字母转换为小写。 |
| ucfirst | 将第一个字母转换为大写，将所有其他字母转换为小写。 |
| NUM | 从字符串中提取所有数字和“ - ”。如果原始字符串中的最后两位数字前面加“。” 或“，”，在相应的位置插入一个小数点。 |
| 一个？b / c | 将字符串与字符串a进行比较，如果它们相等，则用b替换它，否则用 c替换。但c可以是另一个三元组 a，b，c，所以整个结构实际上就是一个转换表。 |

以下DSL示例显示如何使用字符串转换函数。

# definitions for conditions

[when][]There is an? {entity}=${entity!lc}: {entity!ucfirst}()

[when][]- with an? {attr} greater than {amount}={attr} <= {amount!num}

[when][]- with a {what} {attr}={attr} {what!positive?>0/negative?%lt;0/zero?==0/ERROR}

包含DSL定义的文件必须放置在资源文件夹或其任何子文件夹下，如其他任何drools工件。它必须有扩展名.dsl，或者可以用类型标记ResourceType.DSL。以编程方式添加到KieFileSystem。对于使用DSL定义的文件，.dslr应该使用扩展名，而可以将其添加到KieFileSystem带有类型的扩展名ResourceType.DSLR。

为了解析和扩展DSLR文件，读取DSL配置并将其提供给解析器。因此，解析器可以“识别”DSL表达式并将它们转换为本地规则语言表达式。

## [9.复杂的事件处理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#DroolsComplexEventProcessingChapter)

### [9.1。复杂的事件处理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_complex_event_processing)

关于术语复杂事件处理没有广泛接受的定义。术语“事件”本身经常被重载并用于指代几个不同的事物，这取决于它使用的上下文。定义术语并不是本指南的目标，因此，我们可以采用一个松散的定义，虽然不是正式的定义，但我们可以继续达成共识。

所以，在本指南的范围内：

|  |  |
| --- | --- |
|  | **事件**，是在给定时间点在应用程序领域中状态的显着变化的记录。 |

例如，在股票经纪人应用程序中，执行销售操作时，会导致域中的状态更改。这种状态变化可以在域中的几个实体上观察到，例如，证券的价格变化与运营价值相匹配，交易资产的所有权从卖方变为买方，账户余额来自卖家和买家的贷记和借记等。取决于如何对域进行建模，状态的这种改变可以由单个事件，多个原子事件或甚至相关事件的层次来表示。无论如何，在本指南的上下文中，事件是域中特定数据段变化的记录。

事件是自发明以来由计算机系统处理的，并且在整个历史中，负责这些事件的系统被赋予不同的名称和不同的方法。直到90年代，EDA（事件驱动架构）才开始了更加集中的工作，并对事件处理的要求和目标有了更加正式的定义。旧的消息传递系统开始改变以满足这些要求，并且新的系统开始以事件处理的单一目的开发。事件流处理和复杂事件处理的名称下诞生了两种趋势。

最初，事件流处理专注于处理（近）实时事件流的能力，而复杂事件处理的主要焦点是原子事件与复杂事件（复合事件）之间的关联和组合。一个重要的（也许是最重要的）里程碑是在2002年出版了David Luckham博士的书“事件的力量”。在这本书中，Luckham博士介绍了复杂事件处理的概念以及如何用它来增强系统处理事件。多年来，两种趋势都趋于一致，今天这些系统都被称为CEP系统。

这对于一个非常复杂和肥沃的研究领域来说是一个非常简单的解释，但对本指南将介绍的概念提出了高层次的共识。

目前对复杂事件处理的理解可以简单描述为以下来自维基百科的引用：

|  |  |
| --- | --- |
|  | *“****复杂事件处理****或CEP主要是一个事件处理概念，它处理处理多个事件的任务，目标是识别事件云中的有意义的事件，CEP采用诸如检测许多事件的复杂模式，事件关联和抽象，事件层次以及事件之间的关系，如因果关系，成员关系和时间关系，以及事件驱动的过程。“*  *- http://en.wikipedia.org/wiki/Complex\_event\_processing* |

换句话说，CEP是关于从事件云中检测和选择感兴趣的事件（并且仅仅是它们），找出它们之间的关系并从它们及其关系中推断新的数据。

|  |  |
| --- | --- |
|  | 对于本指南的其余部分，我们将使用术语 **复杂事件处理**和**CEP**作为任何相关技术和技术的广泛参考，包括但不限于CEP，复杂事件处理，ESP，事件流处理和事件处理一般来说。 |

### [9.2。Drools融合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools_fusion)

通常，事件处理用例与业务规则用例共享若干要求和目标。这些重叠既发生在商业方面，也发生在技术方面。

在商业方面：

* 业务规则通常基于由事件触发的情景的发生来定义。例子可能是：
  + 在算法交易应用程序中：如果证券价格与当日开盘价格相比增加了X％，那么价格上涨通常由股票交易应用程序中的事件表示。
  + 在监控应用程序上：如果服务器机房温度在Y分钟内增加X度，则采取措施，传感器读数通常由事件表示。
* 业务规则和事件处理查询都会频繁更改，并且需要立即响应业务，以适应新的市场条件，新规定和新的企业策略。

从技术角度来看：

* 两者都需要与企业基础架构和应用程序的无缝集成，特别是自治治理，包括但不限于生命周期管理，审计，安全性等。
* 两者都具有功能要求，如模式匹配和非功能要求，如响应时间和查询/规则说明。

即使共享需求和目标，历史上，这两个领域都是天生的应用，尽管行业发展并且可以在市场上找到好产品，但它们或者专注于事件处理或业务规则管理。这不仅是由于历史原因，而且还因为即使部分重叠，用例也有一些不同的要求。

|  |  |
| --- | --- |
|  | Drools几年前也是一个规则引擎，但随着成为行为建模的单一平台的愿景，它很快意识到它只能通过对三种互补的业务建模技术的相同重要性实现这一目标：   * 业务规则管理 * 业务流程管理 * 复杂的事件处理 |

在这种情况下，Drools Fusion是负责将事件处理功能添加到平台中的模块。

但是，支持复杂事件处理不仅仅是了解事件是什么。CEP方案具有几个常见和显着的特点：

* 通常需要处理大量的事件，但只有一小部分事件是真正感兴趣的。
* 事件通常是不可变的，因为它们是状态变化的记录。
* 通常，事件的规则和查询必须以反应模式运行，即对事件模式的检测作出反应。
* 通常在相关事件之间存在很强的时间关系。
* 个别事件通常并不重要。该系统关注相关事件及其关系的模式。
* 通常，系统需要执行事件的组合和聚合。

基于这个通用的共同特征，Drools Fusion定义了一系列要实现的目标，以便适当支持复杂事件处理：

* 支持事件及其适当的语义作为一等公民。
* 允许检测，关联，聚合和事件组成。
* 支持处理事件流。
* 支持时间约束以模拟事件之间的时间关系。
* 支持有趣事件的滑动窗口。
* 支持会话范围统一时钟。
* 支持CEP用例所需的事件量。
* 支持（重新）有效的规则。
* 支持事件输入引擎（管道）的适配器。

以上目标列表基于Drools Expert本身未涉及的要求，因为在统一平台中，其他模块利用了一个模块的所有功能。通过这种方式，Drools Fusion诞生了企业级功能，如模式匹配，这对CEP产品至关重要，但已由Drools Expert提供。同样，Drools Fusion提供的所有功能都由Drools Flow提供（反之亦然），使流程管理意识到事件处理，反之亦然。

对于本指南的其余部分，我们将介绍Drools Fusion为平台添加的每个功能。所有这些功能都可以支持CEP世界中的不同用例，并且用户可以自由选择和使用那些可以帮助他建立业务用例的模型。

### [9.3。事件语义](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_semantics)

一个事件是呈现出几个显着特点是事实：

* **通常是不可变的：**因为根据前面讨论的定义，事件是应用程序域中状态变化的记录，即记录已经发生的事情，过去不能被“改变”，事件是不可变的。这个约束是开发几个优化和事件生命周期规范的重要要求。这并不意味着代表对象的Java对象必须是不可变的。恰恰相反，引擎不强制对象模型的不变性，因为规则最常见的用例之一是事件数据丰富。

|  |  |
| --- | --- |
|  | 作为最佳实践，允许应用程序填充未填充的事件属性（以使用推断的数据丰富事件），但已填充的属性不应改变。 |

* **强烈的时间约束：**涉及事件的规则通常需要多个事件的相关性，特别是时间相关性，其中事件据说发生在某个时间点相对于其他事件。
* **管理生命周期：**由于它们不可变的性质和时间限制，事件通常只会在有限的时间窗口内与其他事件和事实相匹配，从而使引擎可以自动管理事件的生命周期。换句话说，一个事件被插入到工作内存中，引擎可以找出事件何时不能再匹配其他事实并自动删除它，释放其相关资源。
* **滑动窗口的使用：**因为所有事件都有与之相关的时间戳，所以可以定义滑动窗口并在其上使用滑动窗口，从而允许在一段时间内对值的聚合创建规则。例如：超过60分钟的事件值的平均值。

Drools支持使用两种语义来声明和使用事件：**时间点**事件和**基于时间间隔的**事件。

|  |  |
| --- | --- |
|  | 理解语义统一的简单方法是将时间点事件视为持续时间为零的基于时间间隔的事件。 |

### [9.4。事件处理模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_processing_modes)

一般而言，规则引擎具有众所周知的处理数据和规则的方式，并向应用程序提供结果。另外，关于如何将事实呈现给规则引擎的要求并不多，特别是因为一般来说，处理本身是时间独立的。对大多数情况来说，这是一个很好的假设，但不是所有情况。当需求包括实时或接近实时事件的处理时，时间成为推理过程的重要变量。

以下部分将解释时间对规则推理的影响以及Drools为推理过程提供的两种模式。

#### [9.4.1。云端模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cloud_mode)

CLOUD处理模式是默认的处理模式。规则引擎的用户熟悉这种模式，因为它的行为方式与任何纯正向链接规则引擎完全相同，包括以前版本的Drools。

在CLOUD模式下运行时，引擎可以看到工作记忆中的所有事实，如果它们是常规事实或事件，则无关紧要。没有时间流逝的概念，虽然事件像往常一样有时间戳。换句话说，虽然引擎知道某个事件是在2009年1月1日的09：35：40.767创建的，但引擎不可能确定事件的“老”程度，因为存在没有“现在”的概念。

在这种模式下，引擎将应用其通常的多对多模式匹配算法，使用规则约束来查找匹配的元组，像往常一样激活和激发规则。

这种模式对事实没有任何附加要求。举例来说：

* 没有时间概念。没有要求时钟同步。
* 对事件排序没有要求。引擎将事件视为引擎试图匹配规则的无序云。

另一方面，由于没有要求，所以也没有一些好处。例如，在CLOUD模式中，不可能使用滑动窗口，因为滑动窗口基于“now”的概念，并且CLOUD模式中没有“now”的概念。

由于事件没有订购要求，因此引擎不可能确定事件何时不再匹配，因此事件没有自动生命周期管理。也就是说，应用程序必须在事件不再需要时明确地删除事件，这与应用程序对常规事实的处理方式相同。

Cloud模式是Drools的默认执行模式，但无论如何，与Drools中的其他配置一样，可以通过设置系统属性，使用配置属性文件或使用API​​来更改此行为。相应的属性是：

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption( EventProcessingOption.CLOUD );

等同的性质是：

drools.eventProcessingMode = cloud

#### [9.4.2。流模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stream_mode)

当应用程序需要处理事件流时，STREAM处理模式是选择模式。它为常规处理增加了一些常见要求，但是使得大量功能使得流事件处理变得更加简单。

使用STREAM模式的主要要求是：

* 每个流中的事件必须按时间排序。也就是说，在给定的流内，首先发生的事件必须首先插入到引擎中。
* 引擎将通过使用会话时钟强制实现流之间的同步，因此，虽然应用程序无需强制实现流之间的时间排序，但使用非时间同步的流可能会导致一些意外的结果。

鉴于满足上述要求，应用程序可以使用以下API启用STREAM模式：

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption( EventProcessingOption.STREAM );

或者，等同的财产：

drools.eventProcessingMode = stream

当使用STREAM时，引擎知道时间流的概念和“now”的概念，即引擎理解基于从会话时钟读取的当前时间戳的事件是多久。这个特性允许引擎为应用程序提供以下附加功能：

* 滑动窗口支持
* 自动事件生命周期管理
* 当使用否定模式时自动规则延迟

所有这些功能在以下各节中进行了解释。

##### [9.4.2.1。会话时钟在流模式中的作用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_role_of_session_clock_in_stream_mode)

在CLOUD模式下运行引擎时，会话时钟仅用于为没有预先定义的时间戳属性的到达事件加上时间戳。虽然在STREAM模式下，会话时钟承担了更重要的角色。

在STREAM模式下，会话时钟负责保持当前时间戳，并且基于此时间引擎会执行所有关于事件老化的时间计算，同步来自多个来源的流，计划未来任务等等。

检查会话时钟部分的文档以了解如何配置和使用不同的会话时钟实现。

##### [9.4.2.2。流模式中的负面模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_negative_patterns_in_stream_mode)

与CLOUD模式相比，负模式在STREAM模式下表现不同。在CLOUD模式中，引擎假定事先知道所有事实和事件（没有时间流动的概念），因此，立即评估负面模式。

在STREAM模式下运行时，具有时间约束的负模式可能需要引擎在激活规则之前等待一段时间。时间段由引擎自动计算，用户无需使用任何技巧即可达到预期结果。

例如：

*例173.匹配时立即激活的规则*

rule "Sound the alarm"

**when**

$f : FireDetected( )

**not**( SprinklerActivated( ) )

then

// sound the **alarm**

end

上述规则没有时间约束，需要延迟规则，因此规则立即激活。另一方面，下列规则在启动之前必须等待10秒钟，因为启动洒水喷头可能需要10秒钟的时间：

*例174.一个由于时间约束而自动延迟激活的规则*

rule "Sound the alarm"

when

$f : FireDetected( )

not( SprinklerActivated( **this** after[0s,10s] $f ) )

then

*// sound the alarm*

end

这种行为允许引擎在同时处理负面模式和时间约束时保持一致性。以上内容与编写如下规则相同，但不会让用户计算并显式编写适当的持续时间参数：

*例175.具有显式持续时间参数的相同规则*

rule "Sound the alarm"

duration( 10s )

when

$f : FireDetected( )

not( SprinklerActivated( **this** after[0s,10s] $f ) )

then

*// sound the alarm*

end

以下规则预计每10秒至少有一次“心跳”事件，如果不是规则触发。这条规则的特例是我们在第一种模式和负模式中使用相同类型的对象。负模式具有时间限制，以便在发射之前等待0到10秒，并且排除绑定到$ h的心跳。排除绑定的心跳是重要的，因为时间约束[0，...]本身不排除绑定事件$ h被再次匹配，从而防止规则触发。

*示例176.排除负数模式中的绑定事件*

rule "Sound the alarm"

**when**

$h: Heartbeat( ) **from** entry-point "MonitoringStream"

not( Heartbeat( **this** != $h, **this** after[0s,10s] $h ) **from** entry-point "MonitoringStream" )

then

*// Sound the alarm*

end

### [9.5。会话时钟](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_session_clock)

随着时间的推移需要一个参考时钟。仅举一个例子，如果一条规则导致某个特定股票在过去60分钟内的平均价格出现问题，那么引擎如何知道在过去60分钟内发生了哪些股票价格变化以计算平均价格？明显的反应是：通过比较事件的时间戳和“当前时间”。引擎如何知道**现在**几点？显然，通过查询会话时钟。

会话时钟实现策略模式，允许引擎使用不同类型的时钟。这非常重要，因为引擎可能运行在可能需要不同时钟实现的不同场景的元素中。只要提一下：

* **规则测试：**测试总是需要一个受控制的环境，并且当测试包含具有时间约束的规则时，不仅需要控制输入规则和事实，还需要控制时间流。
* **定期执行：**通常，在生产环境中运行规则时，应用程序需要一个实时时钟，以允许规则引擎立即对时间进度做出反应。
* **特殊环境：**特定环境可能对时间控制有特殊要求。群集环境可能需要通过心脏跳动进行时钟同步，或者JEE环境可能需要使用AppServer提供的时钟等。
* **规则重放或模拟：**重放场景或模拟场景时，应用程序还需要控制时间流。

#### [9.5.1。可用的时钟实现](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_available_clock_implementations)

Drools 5提供2个时钟实现开箱即用。基于系统时钟的默认实时时钟和由应用程序控制的可选伪时钟。

##### [9.5.1.1。实时时钟](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_real_time_clock)

默认情况下，Drools使用实时时钟实现，在内部使用系统时钟来确定当前时间戳。

要明确配置引擎以使用实时时钟，只需将会话配置参数设置为实时：

KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

config.setOption( ClockTypeOption.get("realtime") );

##### [9.5.1.2。伪时钟](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pseudo_clock)

Drools还提供开箱即用的时钟实现，该时钟由被称为伪时钟的应用程序控制。这个时钟对单元测试时间规则特别有用，因为它可以由应用程序控制，因此结果变得确定。

要配置伪会话时钟，请执行以下操作：

KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

config.setOption( ClockTypeOption.get("pseudo") );

作为如何控制伪会话时钟的一个例子：

KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

conf.setOption( ClockTypeOption.get( "pseudo" ) );

KieSession session = kbase.newKieSession( conf, **null** );

SessionPseudoClock clock = session.getSessionClock();

*// then, while inserting facts, advance the clock as necessary:*

FactHandle handle1 = session.insert( tick1 );

clock.advanceTime( 10, TimeUnit.SECONDS );

FactHandle handle2 = session.insert( tick2 );

clock.advanceTime( 30, TimeUnit.SECONDS );

FactHandle handle3 = session.insert( tick3 );

### [9.6。滑动窗户](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sliding_windows)

滑动窗口是一种通过定义一个不断移动的窗口来确定感兴趣事件的范围的方法。两种最常见的滑动窗口实现类型是基于时间的窗口和基于长度的窗口。

接下来的部分将详细介绍它们。

|  |  |
| --- | --- |
|  | 滑动窗口仅在以STREAM模式运行引擎时可用。检查事件处理模式部分以获取有关STREAM模式工作原理的详细信息。 |
|  | 滑动窗口立即开始匹配，并且定义滑动窗口并不意味着规则必须等待滑动窗口“满”才能匹配。例如，计算窗口上事件属性平均值的规则：length（10）将立即开始计算平均值，并且它将从0（零）开始计算无事件，并将在事件到达时更新平均值逐个。 |

#### [9.6.1。滑动时间窗口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sliding_time_windows)

滑动时间Windows允许用户编写仅匹配最后X个时间单位中发生的事件的规则。

例如，如果用户只想考虑在过去2分钟内发生的股票代码，那么该模式将如下所示：

StockTick() over window:time( 2m )

Drools使用“over”关键字将窗口关联到模式。

在更详细的示例中，如果用户想要在从传感器读取的最后10分钟内的平均温度高于阈值时发出警报，则该规则将如下所示：

*例177.在时间窗口上汇总数值*

rule "Sound the alarm in case temperature rises above threshold"

**when**

TemperatureThreshold( $max : max )

Number( doubleValue > $max ) from accumulate(

SensorReading( $temp : temperature ) over window:**time**( 10**m** ),

average( $temp ) )

then

// sound the **alarm**

end

引擎会自动忽略超过10分钟的SensorReading，并保持计算的平均值一致。

|  |  |
| --- | --- |
|  | 请注意，基于时间的窗口在计算事件在工作存储器中过期之前保留的时间间隔时会被考虑，但是从滑动窗口下落的事件本身并不意味着该事件将从工作存储器中丢弃，因为可能存在是依赖于该事件的其他规则。只有在没有其他规则依赖于该事件并且满足该事件类型的到期策略时，引擎才会放弃事件。 |

#### [9.6.2。滑动长度窗口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sliding_length_windows)

“滑动长度”窗口的工作方式与“时间Windows”的工作方式相同，但根据事件的插入顺序而不是时间流程来考虑事件。

例如，如果用户只想考虑最后10个RHT股票蜱，而不考虑它们的年龄，则模式将如下所示：

StockTick( company == "RHT" ) over window:length( 10 )

如您所见，该模式与前一节中介绍的模式类似，但不是使用window：time来定义滑动窗口，而是使用window：length。

使用与前一节中类似的示例，如果用户想要在来自传感器的最近100次读数的平均温度高于阈值时发出警报，则该规则将如下所示：

*例178.汇总长度窗口上的值*

rule "Sound the alarm in case temperature rises above threshold"

**when**

TemperatureThreshold( $max : max )

Number( doubleValue > $max ) from accumulate(

SensorReading( $temp : temperature ) over window:**length**( 100 ),

average( $temp ) )

then

// sound the **alarm**

end

发动机将只保留最后100个读数来计算平均温度。

|  |  |
| --- | --- |
|  | 请注意，基于长度的窗口不符合会话中事件到期的标准。在计算该窗口时，引擎会忽略掉窗口中的事件，但不会基于该条件从会话中删除事件，因为可能存在依赖于该事件的其他规则。 |
|  | 请注意，基于长度的窗口不会为会话中的事件到期定义时间约束，并且引擎不会考虑它们。如果事件没有其他规则来定义时间限制并且没有明确的过期策略，那么引擎将无限期地保持它们在会话中。 |

在使用滑动窗口时，在考虑窗口之前评估alpha约束，但之后评估beta（连接）约束。当涉及时间窗时，这通常没有什么区别，但使用长度窗时这很重要。例如这种模式：

StockTick( company == "RHT" ) over window:length( 10 )

定义一个（最多）10个StockTicks的窗口，所有公司都等于“RHT”，而下列一个：

$s : String()

StockTick( company == $s ) over window:**length**( 10 )

首先创建一个（至多）10 StockTicks的窗口，而不管其公司属性的值如何，然后在其中筛选仅具有公司等于从工作存储器中选择的字符串的窗口。

#### [9.6.3。窗口声明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_window_declaration)

该引擎还支持Windows的声明。这促进了应用于窗口的过滤器和应用于窗口结果的约束之间的明确分离。它还允许在多个规则中轻松重复使用窗口。

另一个好处是引擎中的基本窗口支持的新实现，增加了使用滑动窗口的规则的整体性能。

简化的EBNF申报窗口是：

windowDeclaration := **DECLARE** WINDOW **ID** annotation\* lhsPatternBind **END**

例如，只包含来自给定源的最后10个股票报价的窗口可以被定义为：

**declare** window Ticks

StockTick( **source** == "NYSE" )

**over** window:**length**( 10 )

**from** entry-point STStream

**end**

规则然后可以使用这个声明的窗口作为FROM的来源，如下所示：

rule "RHT ticks in the window"

**when**

**accumulate**( StockTick( company == "RHT" ) **from** window Ticks,

$cnt : **count**(1) )

then

*// there has been $cnt RHT ticks over the last 10 ticks*

end

请注意，这个例子还演示了窗口声明如何分离应用于窗口的约束（只有窗口中包含的10个事件中包含“NYSE”作为源的StockTicks）以及应用于窗口结果的约束选择“仅具有公司等于”RHT“的来源的”NYSE“的10个事件）。

### [9.7。流支持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_streams_support)

大多数CEP使用案例必须处理事件流。流可以以各种形式提供给应用程序，从JMS队列到平面文本文件，从数据库表到原始套接字甚至通过Web服务调用。无论如何，这些流共享一组共同的特征：

* 流中的事件按时间戳排序。时间戳对于不同的流可能具有不同的语义，但它们总是在内部排序。
* 事件的数量通常很高。
* 原子事件本身很少有用。通常意义是从流中的多个事件之间的相关性以及其他来源中提取的。
* 流可以是同质的，即包含单一类型的事件，或者异构的，即包含多种类型的事件。

Drools将流的概念概括为引擎中的“入口点”。一个切入点是流口水从事实来的门。事实可能是正常的事实或事件等特殊事实。

在Drools中，来自一个入口点（流）的事实可以与来自任何其他入口点或事件的事实结合工作记忆中的事实。尽管他们从不混合，也就是说，他们从不会失去对进入发动机的切入点的指称。这很重要，因为人们可以通过几个入口点将相同类型的事实传入引擎，但例如，通过入口点A插入引擎的一个事实将永远不匹配来自入口点B的模式。

#### [9.7.1。声明和使用入口点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declaring_and_using_entry_points)

通过在规则中直接使用它们，入口点在Drools中被隐式声明。即在规则中引用入口点将使引擎在编译时识别并创建适当的内部结构以支持该入口点。

因此，例如，可以设想一个银行应用程序，在这个应用程序中，事务是从流中传入系统的。其中一个流包含ATM机器中执行的所有交易。因此，如果其中一条规则规定：只有在帐户余额超过所请求的提款金额时才会授权提款，则规则如下所示：

*示例179.数据流使用示例*

rule "authorize withdraw"

when

WithdrawRequest( $ai : accountId, $am : amount ) from entry-point "ATM Stream"

CheckingAccount( accountId == $ai, balance > $am )

**then**

// authorize withdraw

end

在前面的例子中，引擎编译器将识别该模式与入口点“ATM Stream”相关联，并且都将为规则库创建支持“ATM Stream”的所有必要结构，并且将仅匹配来自“ATM Stream”的WithdrawRequests， ATM流“。在前面的示例中，规则还将来自流的事件与来自主工作内存（CheckingAccount）的事实结合起来。

现在，让我们想象一下第二条规则，其中规定，向银行分行提出提款申请的任何账户必须收取2美元的费用：

*例180.使用不同的Stream*

**rule** "apply fee on withdraws on branches"

when

WithdrawRequest( $ai : accountId, processed == true ) from entry-point "Branch Stream"

CheckingAccount( accountId == $ai )

then

// apply a $2 fee on the account

end

以前的规则将匹配与第一条规则（WithdrawRequest）完全相同的事件，但是来自两个不同的流，因此插入“ATM流”的事件将不会根据第二条规则的模式进行评估，因为规则状态它只对来自“分支流”的模式感兴趣。

因此，入口点除了作为流的适当抽象之外，也是将事实记录在工作记忆中的一种方式，也是减少交叉产品爆炸的宝贵工具。但这是另一次的主题。

将事件插入入口点同样简单。将事件直接插入工作存储器，而不是将其插入入口点，如下例所示：

*例181.将事实插入入口点*

*// create your rulebase and your session as usual*

KieSession session = ...

*// get a reference to the entry point*

EntryPoint atmStream = session.getEntryPoint( "ATM Stream" );

*// and start inserting your facts into the entry point*

atmStream.insert( aWithdrawRequest );

前面的例子展示了如何手动将事实插入给定的入口点。虽然通常，应用程序将使用众多适配器之一将流终点（如JMS队列）直接插入引擎入口点，而无需手动编写插入代码。Drools管道API包含几个适配器和帮助程序，以及如何执行此操作的示例。

### [9.8。内存管理事件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_memory_management_for_events)

|  |  |
| --- | --- |
|  | 只有在STREAM模式下运行引擎时，才会执行事件的自动内存管理。检查事件处理模式部分以获取有关STREAM模式工作原理的详细信息。 |

在STREAM模式下运行引擎的好处之一是引擎可以检测事件由于时间限制而不再符合任何规则。发生这种情况时，引擎可以安全地从会话中删除没有副作用的事件，并释放该事件使用的任何资源。

基本上有两种方法可以为引擎计算给定事件的匹配窗口：

* 明确地使用到期策略
* 隐含地分析事件的时间限制

#### [9.8.1。显式到期抵消](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_explicit_expiration_offset)

允许引擎计算给定事件类型的感兴趣窗口的第一种方法是明确设置它。为此，只需使用declare语句并为事实类型定义过期：

*示例182.明确定义forStockTick事件的30分钟的到期偏移量*

**declare** StockTick

@expires( 30**m** )

**end**

以上示例为StockTick事件声明了30分钟的到期偏移量。在此之后，假设没有规则仍然需要该事件，则引擎将自动失效并从会话中移除该事件。

|  |  |
| --- | --- |
|  | 给定事件类型的明确过期策略将覆盖该同一类型的任何推断过期偏移量。 |

#### [9.8.2。推断的过期抵消](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_inferred_expiration_offset)

通过分析规则中的时间限制，引擎计算给定事件的到期偏移量的另一种方式是隐含的。例如，根据以下规则：

*示例183.具有时间约束的示例规则*

rule "correlate orders"

when

$bo : BuyOrderEvent( $id : id )

$ae : AckEvent( id == $id, this after[0,10s] $bo )

**then**

// **do** something

end

通过分析上述规则，引擎会自动计算每当BuyOrderEvent匹配时，它需要将其存储长达10秒钟以等待匹配的AckEvent。因此，BuyOrderEvent的隐式到期抵消将是10秒。另一方面，AckEvent只能匹配现有的BuyOrderEvent，因此其到期抵消将为零秒。

引擎将对整个规则库进行分析，并找出每种事件类型的偏移量。

|  |  |
| --- | --- |
|  | 给定事件类型的明确过期策略将覆盖该同一类型的任何推断过期偏移量。 |

### [9.9。时间推理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_temporal_reasoning)

时间推理是任何CEP系统的另一个要求。如前所述，事件的一个显着特征是其强烈的时间关系。

时间推理是一个广泛的研究领域，从时间模态逻辑的根源到其在业务系统中更实际的应用。有数百篇论文和论文的撰写和方法描述了几种应用。Drools再次采用了基于多种来源的实用而简单的方法，但特别值得注意的是以下论文：

* [ALLEN81]艾伦，JF。时间知识的区间表示。1981年。
* [ALLEN83]艾伦，JF。保持关于时间间隔的知识。1983年。
* [Benne00] Bennet，Brandon和Galton，Antony P .. 时间和事件的统一语义。2005年。
* [YONEK05]米线，Eiko和培根，让。混合网络环境中时间和空间事件关联的统一语义。2005年。

Drools实现了Allen描述的基于时间间隔的时间事件语义，并将时间点事件表示为持续时间为0（零）的基于间隔的时间事件。

|  |  |
| --- | --- |
|  | 对于所有的时间运算符间隔，**“（星号）”符号用于表示正无穷大，“ -** ”（负星形）用于表示负无穷大。 |

#### [9.9.1。时间运算符](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_temporal_operators)

Drools实现了由Allen定义的所有13个运算符以及它们的逻辑补（否定）。本节详细介绍每个操作符及其参数。

##### [9.9.1.1。后](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_after)

当从当前事件到被关联事件的时间距离属于为操作员声明的距离范围时，事后评估器关联两个事件并匹配。

让我们看一个例子：

$eventA : EventA( **this** after[ 3m30s, 4m ] $eventB )

当且仅当$ eventB完成时间与$ eventA启动时间之间的时间距离介于（3分钟至30秒）和（4分钟）之间时，以前的模式才会匹配。换一种说法：

3m30s <= $eventA.startTimestamp - $eventB.endTimeStamp <= 4m

after运算符的时间距离间隔是可选的：

* 如果定义了两个值（如下例所示），则间隔从第一个值开始，然后在第二个值结束。
* 如果只定义了一个值，则间隔从该值开始，并在正无穷大上结束。
* 如果未定义值，则假定初始值为1ms，最终值为正无穷大。

|  |  |
| --- | --- |
|  | 可以为该操作员定义负距离。例：  $eventA : EventA( **this** after[ -3m30s, -2m ] $eventB ) |
|  | 如果第一个值大于第二个值，则引擎会自动反转它们，因为没有理由使第一个值大于第二个值。例如：以下两种模式被认为具有相同的语义：  $eventA : EventA( **this** after[ -3m30s, -2m ] $eventB )  $eventA : EventA( **this** after[ -2m, -3m30s ] $eventB ) |

|  |  |
| --- | --- |
|  | 的后，前和一致运营商可以被用于定义事件之间的约束，java.util.Date属性和属性长以任何组合（自纪元解释为时间戳）。例：  EventA( this after $someDate ) |

##### [9.9.1.2。之前](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_before)

当前评估器将两个事件关联起来，并且当与事件相关的时间距离与当前相关的时间距离属于为操作员声明的距离范围时匹配。

让我们看一个例子：

$eventA : EventA( **this** before[ 3m30s, 4m ] $eventB )

当且仅当$ eventA完成时间与$ eventB启动时间之间的时间距离介于（3分钟至30秒）和（4分钟）之间时，以前的模式才会匹配。换一种说法：

3m30s <= $eventB.startTimestamp - $eventA.endTimeStamp <= 4m

before运算符的时间距离间隔是可选的：

* 如果定义了两个值（如下例所示），则间隔从第一个值开始，然后在第二个值结束。
* 如果只定义了一个值，则间隔从该值开始，并在正无穷大上结束。
* 如果未定义值，则假定初始值为1ms，最终值为正无穷大。

|  |  |
| --- | --- |
|  | 可以为该操作员定义负距离。例：  $eventA : EventA( **this** before[ -3m30s, -2m ] $eventB ) |
|  | 如果第一个值大于第二个值，则引擎会自动反转它们，因为没有理由使第一个值大于第二个值。例如：以下两种模式被认为具有相同的语义：  $eventA : EventA( **this** before[ -3m30s, -2m ] $eventB )  $eventA : EventA( **this** before[ -2m, -3m30s ] $eventB ) |

|  |  |
| --- | --- |
|  | 的后，前和一致运营商可以被用于定义事件之间的约束，java.util.Date属性和属性长以任何组合（自纪元解释为时间戳）。例：  EventA( this after $someDate ) |

##### [9.9.1.3。适逢](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_coincides)

恰逢评估者将两个事件关联起来，并在两者同时发生时进行匹配。可选地，评估者接受事件开始和结束时间戳之间距离的阈值。

让我们看一个例子：

$eventA : EventA( this coincides $eventB )

当且仅当$ eventA和$ eventB的开始时间戳相同且$ eventA和$ eventB的结束时间戳也相同时，以前的模式才会匹配。

或者，该操作员接受一个或两个参数。这些参数是匹配时间戳之间距离的阈值。

* 如果只给出一个参数，则它用于开始和结束时间戳。
* 如果给出两个参数，则第一个用作开始时间戳的阈值，第二个用作结束时间戳的阈值。

换一种说法：

$eventA : EventA( **this** coincides[15s, 10s] $eventB )

以上模式将匹配当且仅当：

**abs**( $eventA.startTimestamp - $eventB.startTimestamp ) <= 15**s** &&

**abs**( $eventA.endTimestamp - $eventB.endTimestamp ) <= 10**s**

|  |  |
| --- | --- |
|  | 对参数使用负区间值没有意义，如果发生这种情况，引擎将引发错误。 |
|  | 的后，前和一致运营商可以被用于定义事件之间的约束，java.util.Date属性和属性长以任何组合（自纪元解释为时间戳）。例：  EventA( this after $someDate ) |

##### [9.9.1.4。中](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_during)

评估者将两个事件关联起来，并在事件发生期间发生当前事件时进行匹配。

让我们看一个例子：

$eventA : EventA( this during $eventB )

当且仅当$ eventA在$ eventB开始之后启动并在$ eventB完成之前结束时，以前的模式才会匹配。

换一种说法：

$eventB.startTimestamp < $eventA.startTimestamp <= $eventA.endTimestamp < $eventB.endTimestamp

运行期间可以接受1,2或4个可选参数，如下所示：

* 如果定义了一个值，则这将是两个事件的开始时间戳与两个事件的结束时间戳之间的最大距离之间的最大距离，以便与操作员匹配。例：

$eventA : EventA( this during[ 5s ] $eventB )

只有在以下情况下才会匹配：

0 < $eventA.startTimestamp - $eventB.startTimestamp <= 5s &&

0 < $eventB.endTimestamp - $eventA.endTimestamp <= 5s

* 如果定义了两个值，则第一个值将是两个事件的时间戳之间的最小距离，而第二个值将是两个事件的时间戳之间的最大距离。例：

$eventA : EventA( **this** during[ 5s, 10s ] $eventB )

只有在以下情况下才会匹配：

5s <= $eventA.startTimestamp - $eventB.startTimestamp <= 10s &&

5s <= $eventB.endTimestamp - $eventA.endTimestamp <= 10s

* 如果定义了四个值，则前两个值将是两个事件的开始时间戳之间的最小和最大距离，而最后两个值将是两个事件的结束时间戳之间的最小和最大距离。例：

$eventA : EventA( **this** during[ 2s, 6s, 4s, 10s ] $eventB )

只有在以下情况下才会匹配：

2s <= $eventA.startTimestamp - $eventB.startTimestamp <= 6s &&

4s <= $eventB.endTimestamp - $eventA.endTimestamp <= 10s

##### [9.9.1.5。饰面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_finishes)

完成评估器关联两个事件，并在当前事件的开始时间戳发生在关联事件的开始时间戳后发生匹配，但两个结束时间戳同时发生。

让我们看一个例子：

$eventA : EventA( this finishes $eventB )

当且仅当$ eventA在$ eventB开始之后启动并且在$ eventB完成的同时完成时，以前的模式才匹配。

换一种说法：

$eventB.startTimestamp < $eventA.startTimestamp &&

$eventA.endTimestamp == $eventB.endTimestamp

完成评估程序接受一个可选参数。如果定义了它，它将确定两个事件的结束时间戳之间的最大距离，以便操作员匹配。例：

$eventA : EventA( this finishes[ 5s ] $eventB )

只有在以下情况下才会匹配：

$eventB.startTimestamp < $eventA.startTimestamp &&

**abs**( $eventA.endTimestamp - $eventB.endTimestamp ) <= 5**s**

|  |  |
| --- | --- |
|  | 对参数使用负间隔值没有意义，如果发生这种情况，引擎将引发异常。 |

##### [9.9.1.6。完成者](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_finished_by)

完成评估者关联两个事件并匹配当前事件开始时间戳发生在关联事件开始时间戳之前，但两个结束时间戳同时发生。这是完成评估者的对称。

让我们看一个例子：

$eventA : EventA( this finishedby $eventB )

当且仅当$ eventA在$ eventB开始之前启动并且在$ eventB完成的同时完成时，以前的模式才会匹配。

换一种说法：

$eventA.startTimestamp < $eventB.startTimestamp &&

$eventA.endTimestamp == $eventB.endTimestamp

完成评估者接受一个可选参数。如果定义了它，它将确定两个事件的结束时间戳之间的最大距离，以便操作员匹配。例：

$eventA : EventA( this finishedby[ 5s ] $eventB )

只有在以下情况下才会匹配：

$eventA.startTimestamp < $eventB.startTimestamp &&

**abs**( $eventA.endTimestamp - $eventB.endTimestamp ) <= 5**s**

|  |  |
| --- | --- |
|  | 对参数使用负间隔值没有意义，如果发生这种情况，引擎将引发异常。 |

##### [9.9.1.7。包括](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_includes)

包含评估程序将两个事件关联起来，并在当前事件期间发生相关事件时进行匹配。它在评估者中是对称的。

让我们看一个例子：

$eventA : EventA( this includes $eventB )

当且仅当$ eventB在$ eventA开始之后启动并在$ eventA完成之前结束时，以前的模式才会匹配。

换一种说法：

$eventA.startTimestamp < $eventB.startTimestamp <= $eventB.endTimestamp < $eventA.endTimestamp

包含操作符可接受1,2或4个可选参数，如下所示：

* 如果定义了一个值，则这将是两个事件的开始时间戳与两个事件的结束时间戳之间的最大距离之间的最大距离，以便与操作员匹配。例：

$eventA : EventA( this includes[ 5s ] $eventB )

只有在以下情况下才会匹配：

0 < $eventB.startTimestamp - $eventA.startTimestamp <= 5s &&

0 < $eventA.endTimestamp - $eventB.endTimestamp <= 5s

* 如果定义了两个值，则第一个值将是两个事件的时间戳之间的最小距离，而第二个值将是两个事件的时间戳之间的最大距离。例：

$eventA : EventA( **this** includes[ 5s, 10s ] $eventB )

只有在以下情况下才会匹配：

5s <= $eventB.startTimestamp - $eventA.startTimestamp <= 10s &&

5s <= $eventA.endTimestamp - $eventB.endTimestamp <= 10s

* 如果定义了四个值，则前两个值将是两个事件的开始时间戳之间的最小和最大距离，而最后两个值将是两个事件的结束时间戳之间的最小和最大距离。例：

$eventA : EventA( **this** includes[ 2s, 6s, 4s, 10s ] $eventB )

只有在以下情况下才会匹配：

2s <= $eventB.startTimestamp - $eventA.startTimestamp <= 6s &&

4s <= $eventA.endTimestamp - $eventB.endTimestamp <= 10s

##### [9.9.1.8。会见](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_meets)

符合评估者将当前事件的结束时间戳与相关事件的开始时间戳同时发生时的两个事件相关联并进行匹配。

让我们看一个例子：

$eventA : EventA( this meets $eventB )

当且仅当$ eventA在$ eventB开始的同时完成时，以前的模式才会匹配。

换一种说法：

**abs**( $eventB.startTimestamp - $eventA.endTimestamp ) == 0

meet评估器接受一个可选参数。如果已定义，则确定当前事件的结束时间戳与相关事件的开始时间戳之间的最大距离，以便操作员匹配。例：

$eventA : EventA( this meets[ 5s ] $eventB )

只有在以下情况下才会匹配：

**abs**( $eventB.startTimestamp - $eventA.endTimestamp) <= 5**s**

|  |  |
| --- | --- |
|  | 对参数使用负间隔值没有意义，如果发生这种情况，引擎将引发异常。 |

##### [9.9.1.9。遇到](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_met_by)

metby评估器将两个事件关联起来，并在当前事件的开始时间戳与关联事件的结束时间戳同时发生时进行匹配。

让我们看一个例子：

$eventA : EventA( this metby $eventB )

当且仅当$ eventA在同一时间$ eventB结束时启动时，以前的模式才会匹配。

换一种说法：

**abs**( $eventA.startTimestamp - $eventB.endTimestamp ) == 0

metby评估器接受一个可选参数。如果已定义，则确定相关事件的结束时间戳与当前事件的开始时间戳之间的最大距离，以便操作员匹配。例：

$eventA : EventA( this metby[ 5s ] $eventB )

只有在以下情况下才会匹配：

**abs**( $eventA.startTimestamp - $eventB.endTimestamp) <= 5**s**

|  |  |
| --- | --- |
|  | 对参数使用负间隔值没有意义，如果发生这种情况，引擎将引发异常。 |

##### [9.9.1.10。重叠](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overlaps)

当相关事件开始之前当前事件开始并且在相关事件开始之后但在相关事件结束之前完成时，重叠评估器使两个事件相关并匹配。换句话说，这两个事件都有一个重叠的时期。

让我们看一个例子：

$eventA : EventA( this overlaps $eventB )

当且仅当：

$eventA.startTimestamp < $eventB.startTimestamp < $eventA.endTimestamp < $eventB.endTimestamp

重叠操作符接受1或2个可选参数，如下所示：

* 如果定义了一个参数，则这将是相关事件的开始时间戳与当前事件的结束时间戳之间的最大距离。例：

$eventA : EventA( this overlaps[ 5s ] $eventB )

只有在以下情况下才会匹配：

$eventA.startTimestamp < $eventB.startTimestamp < $eventA.endTimestamp < $eventB.endTimestamp &&

0 <= $eventA.endTimestamp - $eventB.startTimestamp <= 5s

* 如果定义了两个值，则第一个值将是最小距离，第二个值将是相关事件的开始时间戳与当前事件的结束时间戳之间的最大距离。例：

$eventA : EventA( **this** overlaps[ 5s, 10s ] $eventB )

只有在以下情况下才会匹配：

$eventA.startTimestamp < $eventB.startTimestamp < $eventA.endTimestamp < $eventB.endTimestamp &&

5s <= $eventA.endTimestamp - $eventB.startTimestamp <= 10s

##### [9.9.1.11。重叠在](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overlapped_by)

在当前事件开始之前，当相关事件开始并且在当前事件开始之后，但在当前事件结束之前结束时，重叠的评估器使两个事件相关并且匹配。换句话说，这两个事件都有一个重叠的时期。

让我们看一个例子：

$eventA : EventA( this overlappedby $eventB )

当且仅当：

$eventB.startTimestamp < $eventA.startTimestamp < $eventB.endTimestamp < $eventA.endTimestamp

overlappedby操作符接受1或2个可选参数，如下所示：

* 如果定义了一个参数，则这将是当前事件的开始时间戳与相关事件的结束时间戳之间的最大距离。例：

$eventA : EventA( this overlappedby[ 5s ] $eventB )

只有在以下情况下才会匹配：

$eventB.startTimestamp < $eventA.startTimestamp < $eventB.endTimestamp < $eventA.endTimestamp &&

0 <= $eventB.endTimestamp - $eventA.startTimestamp <= 5s

* 如果定义了两个值，则第一个值将是最小距离，第二个值将是当前事件的开始时间戳和相关事件的结束时间戳之间的最大距离。例：

$eventA : EventA( **this** overlappedby[ 5s, 10s ] $eventB )

只有在以下情况下才会匹配：

$eventB.startTimestamp < $eventA.startTimestamp < $eventB.endTimestamp < $eventA.endTimestamp &&

5s <= $eventB.endTimestamp - $eventA.startTimestamp <= 10s

##### [9.9.1.12。启动](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_starts)

当相关事件的结束时间戳之前发生当前事件的结束时间戳时，启动评估程序关联两个事件并进行匹配，但是两个开始时间戳同时发生。

让我们看一个例子：

$eventA : EventA( this starts $eventB )

当且仅当$ eventA在$ eventB完成之前完成并且在$ eventB开始的同时开始时，前一个模式才会匹配。

换一种说法：

$eventA.startTimestamp == $eventB.startTimestamp &&

$eventA.endTimestamp < $eventB.endTimestamp

启动评估器接受一个可选参数。如果定义了它，它将确定两个事件的开始时间戳之间的最大距离，以便操作员匹配。例：

$eventA : EventA( this starts[ 5s ] $eventB )

只有在以下情况下才会匹配：

**abs**( $eventA.startTimestamp - $eventB.startTimestamp ) <= 5**s** &&

$eventA.endTimestamp < $eventB.endTimestamp

|  |  |
| --- | --- |
|  | 对参数使用负间隔值没有意义，如果发生这种情况，引擎将引发异常。 |

##### [9.9.1.13。开始者](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_started_by)

当相关事件的结束时间戳发生在当前事件的结束时间戳之前，但开始时间戳同时发生时，开始的评估器关联两个事件并匹配。让我们看一个例子：

$eventA : EventA( this startedby $eventB )

当且仅当$ eventB在$ eventA完成之前完成并且在$ eventB开始的同时开始时，前一个模式才会匹配。

换一种说法：

$eventA.startTimestamp == $eventB.startTimestamp &&

$eventA.endTimestamp > $eventB.endTimestamp

Startedby评估程序接受一个可选参数。如果定义了它，它将确定两个事件的开始时间戳之间的最大距离，以便操作员匹配。例：

$eventA : EventA( this starts[ 5s ] $eventB )

只有在以下情况下才会匹配：

**abs**( $eventA.startTimestamp - $eventB.startTimestamp ) <= 5**s** &&

$eventA.endTimestamp > $eventB.endTimestamp

|  |  |
| --- | --- |
|  | 对参数使用负间隔值没有意义，如果发生这种情况，引擎将引发异常。 |

## [10.决策模型和表示法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#drools.DMN)

### [10.1。决策模型和表示法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#decision_model_and_notation_overview)

该[**决策模型和符号**](http://www.omg.org/spec/DMN/1.1)，或者**DMN**，是由一个相对较新的标准[**对象管理组织**](http://www.omg.org/)，或**OMG**。它通过定义一种高级语言和业务模型与可执行实现之间的差距，并提供一种通用的标记，整个业务用户很容易理解，从分析师到开发人员。

实际规范建立在定义**业务流程模型和表示法**（或**BPMN**，它的兄弟规范）的经验基础上，用于对业务流程进行建模。BPMN和DMN之间的互操作性和无缝集成也是该规范的一个目标。

#### [10.1.1。一致性级别](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#dmn_conformance_levels)

DMN规范定义了三个增量级别的一致性。有关每个一致性级别的正式定义，请参阅[**第9页**](http://www.omg.org/spec/DMN/1.1)的 [**DMN 1.1规范**](http://www.omg.org/spec/DMN/1.1)。以下是了解合规级别的非正式方式：

**一致性等级1**

需要一致性级别1的实现来支持决策需求图，决策逻辑和决策表，但该模型不需要是可执行的。任何语言都可以用来定义表达式，包括自然的，非结构化的语言。

**一致性等级2**

为了支持与一致性等级相同的一致性级别2实现，但是应该使用规范第9章中定义的**S-FEEL**（简化的FEEL）语言来编写表达式，并且模型应该是完全可执行的。

**一致性等级3**

为了支持与一致性级别2相同的一致性级别3实现，但可以使用完整的**FEEL**语言编写表达式，如规范第10章中所定义。该实现还必须支持全套盒装表达式并且完全可执行。

### [10.2。执行嵌入在应用程序中的DMN服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#executing_decision_services_from_kie_container)

本节介绍如何与Kie Container进行交互以执行DMN模型的决策服务。

*先决条件*

* 包含要执行的DMN模型的KJAR
* 需要以下依赖项：
* <dependency>
* <groupId>org.kie</groupId>
* <artifactId>kie-ci</artifactId>
* </dependency>
* <dependency>
* <groupId>org.kie</groupId>
* <artifactId>kie-dmn-core</artifactId>

</dependency>

*程序*

1. 从ReleaseId或classpath创建一个Kie容器：

以下片段提供了一个基本示例：

*从ReleaseId创建Kie容器*

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );

KieContainer kieContainer = kieServices.newKieContainer( releaseId );

或者可选地：

*从classpath创建一个Kie容器*

KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();

|  |  |
| --- | --- |
|  | 如果Kie Container是从类路径创建的，kie-ci则不需要依赖关系。 |

1. DMNRuntime通过使用模型命名空间和模型名称，从Kie Container 获取，然后引用要评估的DMN模型。
2. DMNRuntime dmnRuntime = kieContainer.newKieSession().getKieRuntime( DMNRuntime.class );
3. DMNModel dmnModel = dmnRuntime.getModel(
4. "https://github.com/kiegroup/kie-dmn/input-data-string", *// model namespace*
5. "input-data-string" *// model name*

);

1. 执行所需模型的决策服务

以下片段提供了一个基本示例：

DMNContext dmnContext = dmnRuntime.newContext();

dmnContext.set( "Full Name", "John Doe" );

DMNResult dmnResult = dmnRuntime.evaluateAll( dmnModel, dmnContext );

**for** ( DMNDecisionResult dr : dmnResult.getDecisionResults() ) {

System.out.println( "Decision '" + dr.getDecisionName() + "' : " + dr.getResult() );

}

|  |  |
| --- | --- |
|  | 实例化一个新的DMN上下文作为模型评估的输入 |
|  | 为输入DMN上下文分配输入变量 |
|  | 评估在DMN模型中定义的所有DMN决策; DMN结果包含所有逃避的结果 |

并打印下面的输出：

**Decision** 'Greeting Message' : Hello John Doe

也可以循环所有评估，以便分别得到每个决策的结果，如示例中所示

*相关信息*

* 要创建KJAR并创建Kie容器，请参考“构建，部署，利用和运行”。参考。[构建，部署，利用和运行](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_builddeployutilizeandrunsection)

### [10.3。在Kie服务器上远程执行DMN服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#executing_decision_services_on_the_kie_execution_server)

本节介绍如何与部署在Kie执行服务器上的Kie容器进行交互，以便为Kie容器内可用的DMN模型执行决策服务。

*先决条件*

* Kie执行服务器安装和配置
* 部署在Kie执行服务器中的Kie容器，用于包含要执行的DMN模型的KJAR
* 连接到相关Kie执行服务器的Kie服务器Java客户端

*程序*

1. 从连接到相关Kie执行服务器的Kie Server Java Client获取DMNServicesClient

这可以通过getServicesClient()在Kie Server Java客户端实例上调用该方法来获得，如以下代码片段所示：

DMNServicesClient dmnClient = kieServicesClient.getServicesClient( DMNServicesClient.class );

在dmnClient现在可以用来在纪伊执行服务器上执行决策服务。

1. 执行所需模型的决策服务

以下片段提供了一个基本示例：

DMNContext dmnContext = dmnClient.newContext();

dmnContext.set( "Full Name", "John Doe" );

ServiceResponse<DMNResult> serverResp = dmnClient.evaluateAll($containerId,

$modelnamespace, $modelname,

dmnContext);

DMNResult dmnResult = serverResp.getResult();

|  |  |
| --- | --- |
|  | 实例化一个新的DMN上下文作为模型评估的输入 |
|  | 为输入DMN上下文分配输入变量 |
|  | 评估在DMN模型中定义的所有DMN决策 |
|  | DMN结果对象可从服务器响应中获得 |

其中$containerId部署了包含DMN模型的kjar的容器的标识，模型 $modelnamespace的名称空间是模型 $modelname的名称。

此时，dmnResult包含所评估的DMN模型的所有决策结果。

|  |  |
| --- | --- |
|  | 通过使用替代方法，可以在模型中只执行特定的DMN决策DMNServicesClient。 |
|  | 如果Kie容器包含1个且仅有1个DMN模型，则可以省略$modelnamespace并$modelname; 为了执行决策服务，唯一可用的模型将是默认引用的模型。 |

*相关信息*

* 安装KIE服务器。参考。[安装KIE服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_the_kie_server)
* 创建Kie容器。参考。[创建Kie容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_a_kie_container)
* Kie服务器Java客户端API，也描述了所需的依赖关系。参考。[Kie服务器Java客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_java_client_api)
* Decision Server的可用客户端。参考。[Decision Server的可用客户端](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_available_clients_for_the_decision_server)

### [10.4。FEEL语言参考](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_language_reference)

友好的表达式语言（FEEL）是由DMN规范定义的表达式语言。任何能够在电子表格中创作表达式的用户都可以轻松地编写FEEL表达式。

DMN完全定义了FEEL执行的语义。任何不调用外部定义函数的表达式都应该在任何兼容的实现中产生完全相同的结果。FEEL语言是无副作用的，任何外部定义的函数都应该是无副作用的。

#### [10.4.1。FEEL语言语义](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_language_reference_semantics)

##### [10.4.1.1。变量和函数名称](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_names)

FEEL和传统语言之间最大的区别之一是FEEL中支持空格和一些特殊字符作为变量和函数名称的一部分。

感觉名称必须以开始letter，?或\_。Unicode字母字符也是允许的。在名称中的其余字符可以是任何起始字符，以及digits，空格和特殊字符，如+，-，/，\*，'和.。以下名称都是有效的FEEL名称：

* 年龄
* 生日
* 航班234预检程序

需要注意的是，变量名不能用语言关键字开始，如and，true，every，等。

|  |  |
| --- | --- |
|  | *歧义*  将空格，关键字和其他特殊字符用作名称的一部分会使语言不明确。歧义在表达式的上下文中解决，从左到右匹配名称。解析器解析变量名称作为范围内匹配的最长名称。( )如有必要，用户可以使用消除歧义名称。 |
|  | *名称中的空格*  该规范限制了FEEL名称中空格的使用。根据规范，只要没有两个连续的空格，名称就可以包含多个空格。  在名称中使用空格可以使非技术用户更容易阅读表达式，但这可能会导致难以找到的问题。例如，用户可能难以在视觉上区分一个空间和两个连续的空间。  为了使语言更易于使用并避免由于空间造成的常见错误，Drools取消了对使用连续空格的限制。相反，Drools支持具有任意数量连续空格的变量名称，但将它们归一化为一个空格。举例来说，两个变量引用First Name和First Name是等价的。  Drools还规范化了其他空白的使用，就像non-breakable white space网页（html），标签和换行符中常见的那样。从Drools FEEL引擎的角度来看，所有这些角色在处理之前都被归一化为一个单独的空白区域。 |

|  |  |
| --- | --- |
|  | *关键字 in*  关键字in是该语言中唯一**不能**用作变量名称一部分的关键字。虽然在规范允许在变量名中间的关键字的使用，使用in与的语法定义变量名的冲突for，every并some 表达结构。 |

##### [10.4.1.2。数据类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes)

FEEL显式定义了几种内置的数据类型：

* [数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesNumber-section.html#feel_semantics_datatypes_number)
* [串](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesString-section.html#feel_semantics_datatypes_string)
* [布尔](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesBoolean-section.html#feel_semantics_datatypes_boolean)
* [日期](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDate-section.html#feel_semantics_datatypes_date)
* [时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesTime-section.html#feel_semantics_datatypes_time)
* [日期和时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_datetime)
* [天数和时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_daystimeduration)
* [年和月的持续时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_yearsmonthsduration)

FEEL还隐含地定义了一些其他内置数据类型。这些类型不是在不同的实现之间达成共识，而是由Drools支持：

* [功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_function)
* [上下文](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_context)
* [范围](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_range)
* [名单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/FEELDataTypesDateTime-section.html#feel_semantics_datatypes_range)
* 一元测试

##### [10.4.1.3。数字](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_number)

FEEL中的数字基于[IEEE 754-2008](http://ieeexplore.ieee.org/document/4610935/) Decimal 128格式，具有34位精度。在内部，它们以Java表示，就像 [BigDecimals](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html)MathContext DECIMAL128一样。FEEL中只有一个“类型”数字，因此使用相同的类型来表示整数和浮点数。

FEEL号码使用'。' （点）作为其小数分隔符。没有支持-INF，+INF或NaN。FEEL null用来表示无效的数字。

|  |  |
| --- | --- |
|  | *额外支持的数字语法*  Drools扩展了规范并支持附加的数字符号：   * **科学**：科学记数法可以与sufix使用e<exp>或E<exp>。例如：1.2e3与写入表达式相同1.2\*10\*\*3，但是它是一个文字而不是表达式。 * **十六进制**：可以使用preffix编写十六进制数字0x。例如：0xff与十进制数字255相同。支持大写和小写字母。例如：0XFF和...一样0xff。 * **类型后缀**：类型的后缀f，F, `d，D，l和L支持，但被忽略。 |

##### [10.4.1.4。字符串](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_string)

FEEL中的字符串是用双引号分隔的任何字符序列。例如，"John Doe"。

##### [10.4.1.5。布尔值](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_boolean)

FEEL使用文字符号true并false表示布尔值。

|  |  |
| --- | --- |
|  | FEEL使用3值布尔逻辑，所以布尔逻辑表达式可能有结果 true，false或者null。 |

##### [10.4.1.6。日期](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_date)

FEEL没有日期文字，但内置date()函数可以用来构造日期值。例如，

**date**( "2017-06-23" )

FEEL中的日期字符串遵循[XML Schema Part 2 Datatypes](https://www.w3.org/TR/xmlschema-2/#date) 文档中定义的格式。格式"YYYY-MM-DD"，其中YYYY与4位数字的年份，MM是该月的2位数字（如一月是01，二月是02）的数量，DD是天数。

日期对象的时间部分等于"00:00:00"午夜。日期被认为是本地的，没有时区。

##### [10.4.1.7。时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_time)

FEEL没有时间文字，但内置time()函数可以用来构造时间值。例如，

time( "04:25:12" )

time( "14:10:00+02:00" )

time( "22:35:40.345-05:00" )

time( "15:00:30z" )

FEEL中的时间字符串遵循[XML Schema Part 2 Datatypes](https://www.w3.org/TR/xmlschema-2/#time) 文档中定义的格式。即，格式是"hh:mm:ss[.uuu][(+-)hh:mm]"其中hh是天（从00到24）的时刻，mm是在小时的分钟和ss是秒在分钟数。可选地，该字符串可以定义uuu第二秒内的毫秒数（）和/或包含与UTC时间的正（+或负-）偏移量以定义它的时区。而不是偏移量，字母“z”可以用来表示UTC时间，与偏移量-00：00相同。如果没有定义偏移量，则该时间被认为是本地的。

定义偏移量/时区的时间值不能与不定义偏移量/时区的本地时间进行比较。

##### [10.4.1.8。日期和时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_datetime)

FEEL没有日期和时间文字，但内置date and time()函数可用于构造 日期和时间值。例如，

date **and** **time**( "2017-10-22T23:59:00" )

date **and** **time**( "2017-06-13T14:10:00+02:00" )

date **and** **time**( "2017-02-05T22:35:40.345-05:00" )

date **and** **time**( "2017-06-13T15:00:30z" )

日期和时间

日期和时间定义的偏移/时区值不能低于本地日期和时间是没有定义的偏移/时区值。

|  |  |
| --- | --- |
|  | 关键字dateTime是一个同义词date and time。 在DMN中定义日期和时间变量时非常有用，因为XML模式不支持包含空格的类型名称。 |

##### [10.4.1.9。日期和时间持续时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_daystimeduration)

FEEL没有日期和时间duration()长度文字，但内置函数可用于构建日期和时间长度 值。例如，

**duration**( "P1DT23H12M30S" )

**duration**( "P23D" )

**duration**( "PT12H" )

**duration**( "PT35M" )

FEEL中的天数和时间持续时间字符串遵循[XML Schema Part 2 Datatypes](https://www.w3.org/TR/xmlschema-2/#duration)文档中定义的格式 ，但仅限于日，小时，分钟和秒。几个月和几年是不允许的。

|  |  |
| --- | --- |
|  | 关键字dayTimeDuration是一个同义词days and time duration。在DMN中定义日期和时间持续变量时非常有用 ，因为XML架构不支持包含空格的类型名称。 |

##### [10.4.1.10。年和月份持续时间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_yearsmonthsduration)

FEEL没有几年和几个月的文字，但内置duration()函数可以用来构造

**duration**( "P3Y5M" )

**duration**( "P2Y" )

**duration**( "P10M" )

**duration**( "P25M" )

年和月份持续时间

|  |  |
| --- | --- |
|  | 关键字yearMonthDuration是一个同义词years and months duration |

##### [10.4.1.11。功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_function)

FEEL允许通过函数文字（或匿名函数）创建函数。例如，以下FEEL表达式会创建一个函数来添加参数a和b

**function**(a, b) **a** + **b**

没有明确的方式将变量声明为a function

|  |  |
| --- | --- |
|  | 一个 function |

##### [10.4.1.12。上下文](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_context)

FEEL允许通过创建上下文context literals。contextFEEL中的A 是键/值对的列表，类似于像Java这样的语言中的地图。例如，以下文字表达式将创建一个带有两个条目（x和y）的上下文，表示图表中的坐标：

{ x : 5, y : 3 }

context在DMN规范中没有明确的方式将变量声明为a 。context出于这个原因，Drools使用类型扩展了DMN内置类型。

|  |  |
| --- | --- |
|  | 一个 context |

##### [10.4.1.13。范围（或间隔）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_range)

FEEL允许创建范围，也称为间隔range literals。rangeFEEL中的A 是一个定义下限和上限的值，其中可以打开或关闭。范围的语法定义如下：

range := interval\_start endpoint '..' endpoint interval\_end

interval\_start := open\_start | closed\_start

open\_start := '(' | ']'

closed\_start := '['

interval\_end := open\_end | closed\_end

open\_end := ')' | '['

closed\_end := ']'

endpoint := expression

该端点的表达式必须返回可比较的值，并且下限端点必须低于上限端点。

例如，以下文字表达式定义1到10之间的间隔，包括边界（即两个端点上的闭合间隔）：

[ 1 .. 10 ]

下面的文字表达式定义了1小时到12小时之间的时间间隔，包括下边界（即闭区间），但不包括上边界（开区间）：

[ duration("PT1H") .. duration("PT12H") [

在测试值的范围时，范围在决策表中特别有用，但也可用于简单文字表达式。例如，如果变量x的值介于0和100之间，则以下文字表达式将返回true：

x in [ 1 .. 100 ]

|  |  |
| --- | --- |
|  | 范围文字不是FEEL规范中的有效表达式。它必须始终包含在其他表达式中。Drools扩展了规范，允许直接定义范围值，甚至分配给变量。 |

##### [10.4.1.14。清单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#feel_semantics_datatypes_list)

FEEL中的列表由用方括号括起来的以逗号分隔的值列表表示。例如：

[ 2, 3, 4, 5 ]

FEEL中的所有列表是同类的（也就是说，列表中的所有元素都是相同类型的）并且是不可变的。列表中的元素可以通过索引访问，其中第一个元素为1.负索引可用于从列表末尾开始访问元素，因此-1是最后一个元素。例如，下面的表达式返回列表x的第二个元素：

x[2]

下面的表达式返回倒数第二个元素：

x[-2]

|  |  |
| --- | --- |
|  | DMN规范不允许将变量定义为通用列表类型。Drools扩展了规范以支持这一点。 |

### [10.5。DMN支持Drools](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#dmn_support_in_drools)

Drools 7.0包含对**符合3级的** DMN 1.1模型的完整运行时支持。但是，它不包括DMN模型的任何编辑器。在不久的将来，DMN模型的编辑人员将被添加到平台中，但同时我们建议您查看市场上可用的几种DMN兼容编辑器之一。

从平台的角度来看，DMN模型只是可以包含在Drools kjar中并部署到运行时的另一种类型的资产。这意味着Drool 7.0可以从同一个kjar中消费和执行DMN模型以及任何其他支持的资产（例如DRL文​​件和本机决策表）。这使用户能够始终使用最好的工具来完成手头的工作。实际上，BPMN模型通常从其决策任务节点直接调用DMN决策服务。

#### [10.5.1。DMN实施细节](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#dmn_support_in_drools_details)

DMN规范定义了模型的语法和语义。请注意，规范中存在一些缺陷和含糊之处。目前，没有技术兼容性套件（TCK）或DMN v1.1的官方参考实现来解决这些歧义。[这里](https://agilepro.github.io/dmn-tck/index.html)提供了一个TCK的供应商中立努力。

Drools致力于尽可能符合规范，但存在以下差异类别：\*修正了规范中的错误，\*扩展规范以更好地支持用例，而不影响严格的模型。

差异也分为FEEL语言差异和DMN模型差异。

##### [10.5.1.1。一个。修复了FEEL语言中的错误和变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_fixes_to_bugs_and_changes_in_the_feel_language)

* 空间灵敏度：这种FEEL语言的实现是空间**不敏感的**。目标是基于不可见字符（例如，空格）基于上下文和行为差异来避免非确定性行为。这意味着对于此实现，first name用一个空格命名的变量与其中的first name两个空格完全相同。
* 列出函数*or()*和*and()* ：规范定义了两个名为or()and的列表函数and()。然而，根据FEEL语法，这些都不是有效的函数名，因为and和or有保留的关键字。此实现将这些函数重新命名为any()和all()分别针对DMN 1.2进行预期。
* 关键字*in*不能在变量名称中使用：规范定义任何关键字都可以作为变量名称的一部分重用，但for …​ in …​ return循环引起的含糊不清会阻止in 关键字的重用。所有其他关键字作为变量名称的一部分被支持。
* 支持范围内的日期和时间文字：根据语法规则＃8，＃18，＃19，＃34和＃62，date and time范围内支持文字（第110-111页）。另一方面，第114页的第10.3.2.7节与文法相矛盾，并表示不支持。此实现选择遵循date and time范围上的文法和支持文字，并将规范扩展为支持任意表达式（请参见下面的扩展）。
* 无效的时间语法：第112页的章节10.3.2.3.4和time第131页的项目符号点都声明time字符串词汇表示符合XML模式数据类型规范以及ISO 8601。根据XML模式规范（[https：// www.w3.org/TR/xmlschema-2/#time](https://www.w3.org/TR/xmlschema-2/#time)），一个时间的词汇表示遵循hh:mm:ss.sss没有任何主要字符的模式。在几个例子中，DMN规范使用了一个领先的“T”，我们知道这是一个错字，而不符合标准。

##### [10.5.1.2。湾 FEEL语言规范的扩展](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_b_extensions_to_the_feel_language_specification)

* 支持科学和十六进制符号：此实现支持数字的科学和十六进制符号。例如1.2e5（科学记数法）0xD5（十六进制记数法）。
* 支持表达式作为范围中的终点：此实现支持表达式作为范围的终结点。例如，[date("2016-11-24")..date("2016-11-27")]
* 支持其他类型：规范仅将以下内容定义为该语言的基本类型：
  + 数
  + 串
  + 布尔
  + 天数和时间
  + 年和月的持续时间
  + 时间
  + 日期和时间

为了完整性和正交性，该实现还支持以下类型：

* + 上下文
  + 名单
  + 范围
  + 功能
  + 一元测试
* 支持一元测试

*UnaryTestAsFunction.feel*

{

is minor : < 18,

Bob is minor : is minor( bob.age )

}

* 支持附加的内置功能：支持以下附加功能：
  + now() ：返回当前的本地日期和时间。
  + today() ：返回当前的本地日期。
  + decision table()：虽然规范提到了一个决策表，但它返回一个决策表函数。第114页的功能不能按照定义实施。
  + string( mask, p…​ )：返回格式化为每个掩码的字符串。有关掩码语法的详细信息，请参阅Java String.format（）。例如，string( "%4.2f", 7.1298 )返回字符串"7.12"。
* 支持额外的日期和时间算术

*DateArithmetic.feel*

date( "2017-05-12" ) - date( "2017-04-25" ) = duration( "P17D" )

##### [10.5.1.3。C。修复了对DMN模型的错误和更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_c_fixes_to_bugs_and_changes_to_the_dmn_model)

* 支持名称上带空格的类型：DMN XML模式将类型引用定义为QNames。QNames不允许空格。因此，不可能使用类型般的感觉date and time， days and time duration或years and months duration。这个实现将类型引用解析为字符串，并允许带空格的类型名称。但是，为了符合XML模式，它还将下列别名添加到可以使用的类型中：
  + date and time = dateTime
  + days and time duration= duration或dayTimeDuration
  + years and months duration= duration或yearMonthDuration

需要注意的是，对于“时间”类型，用户可以简单地使用duration，发动机会推断出正确的时间，无论是days and time duration或years and months duration。

##### [10.5.1.4。d。DMN模型规范的扩展](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_d_extensions_to_the_dmn_model_specification)

* 列表支持异构元素类型：目前此实现支持具有异构元素类型的列表。这是一个实验扩展，并且限制了一些函数和过滤器的功能。这一决定将在未来重新评估。
* 决策表和输入子句之间的TypeRef链接：在决策表/输入子句中，如果未定义任何值列表，则引擎将自动检查类型引用，并应用允许的值检查它是否已定义。

## [11.实验特征](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_experimentalfeatureschapter)

### [11.1。声明性议程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_declarative_agenda)

|  |  |
| --- | --- |
|  | 声明式议程是实验性的，所有方面在未来很有可能发生变化。@Eager和@Direct是控制规则行为的临时注释，随着声明式议程的发展，规则的行为也会发生变化。注释，而不是选择属性，以反映其实验性质。 |

声明式议程允许使用规则来控制哪些其他规则可以触发以及何时触发。虽然这会比简单使用显着性增加更多的开销，但优点是它是声明性的，因此更易读和可维护，并且应该允许以更简单的方式实现更多的用例。

此功能默认为关闭状态，必须明确启用，这是因为它现在被认为是高度实验性的，并且可能会发生变化，但可以通过在相应的KieBase中添加declarativeAgenda ='enabled'属性来激活它kmodule.xml文件的kbase标记，如下例所示。

*例子184.启用声明议程*

<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.drools.org/xsd/kmodule">

<kbase name="DeclarativeKBase" declarativeAgenda="enabled">

<ksession name="KSession">

</kbase>

</kmodule>

基本的想法是：

* 所有规则的匹配都作为事实插入到WorkingMemory中。所以你现在可以对Match进行模式匹配。规则的元数据和声明可用作Match对象的字段。
* 您可以使用当前规则的kcontext.blockMatch（匹配匹配）来阻止选定的匹配。只有当规则变得不真实时，比赛才有资格被解雇。如果它已经有资格被解雇并被阻止，它将从议程中删除，直到解除阻止。
* 一场比赛可能会有多个拦截器，并保留一个计数。所有的阻挡者必须变成虚假的，让计数器达到零才能让匹配符合射击条件。
* kcontext.unblockAllMatches（匹配匹配）是一种覆盖规则，无论如何将移除所有阻止程序
* 激活也可能被取消，所以它不会用cancelMatch激发
* 议程中添加了未阻止的匹配，并遵守正常突出显示，议程组，规则流组等。
* @Direct注释允许规则在匹配时立即触发，这将用于阻止/解除阻止匹配的规则，这些规则的副作用不会影响其他位置。

*示例185.新的RuleContext方法*

**void** **blockMatch**(Match match);

**void** **unblockAllMatches**(Match match);

**void** **cancelMatch**(Match match);

这是一个基本的例子，它将阻止来自具有元数据@department（'sales'）的规则的所有匹配。它们将保持阻塞状态，直到blockerAllSalesRules规则变为false，即收回“go2”。

*示例186.基于规则元数据阻止规则*

rule rule1 @Eager @department('sales') **when**

$s : String( this == 'go1' )

**then**

list.add( kcontext.rule.name + ':' + $s );

**end**

rule rule2 @Eager @department('sales') **when**

$s : String( this == 'go1' )

**then**

list.add( kcontext.rule.name + ':' + $s );

**end**

rule blockerAllSalesRules @Direct @Eager **when**

$s : String( this == 'go2' )

$i : Match( department == 'sales' )

**then**

list.add( $i.rule.name + ':' + $s );

kcontext.blockMatch( $i );

**end**

|  |  |
| --- | --- |
|  | 除了使用@Direct注释阻止规则之外，还需要使用@Eager注释可能被其阻止的所有规则。这是因为Match必须通过阻塞规则的模式匹配进行评估，所以潜在的阻塞者不能被懒惰地评估，否则将不会被任何Match评估。 |

此示例显示如何使用活动属性来计算活动或非活动（已激活）匹配的数量。

*例187.计数激活/非激活匹配的数量*

rule rule1 @Eager @department('sales') **when**

$s : String( this == 'go1' )

**then**

list.add( kcontext.rule.name + ':' + $s );

**end**

rule rule2 @Eager @department('sales') **when**

$s : String( this == 'go1' )

**then**

list.add( kcontext.rule.name + ':' + $s );

**end**

rule rule3 @Eager @department('sales') **when**

$s : String( this == 'go1' )

**then**

list.add( kcontext.rule.name + ':' + $s );

**end**

rule countActivateInActive @Direct @Eager **when**

$s : String( this == 'go2' )

$active : Number( this == 1 ) from accumulate( $a : Match( department == 'sales', active == **true** ), count( $a ) )

$inActive : Number( this == 2 ) from accumulate( $a : Match( department == 'sales', active == **false** ), count( $a ) )

**then**

kcontext.halt( );

**end**

### [11.2。使用OOPath浏览对象的图形](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_browsing_graphs_of_objects_with_oopath_2)

当事实的字段是集合时，可以使用from关键字将集合中的所有项目绑定并推理。尽管如此，当需要浏览对象的图形时，from条件元素的广泛使用可能会导致冗长的cubersome语法，如下例所示：

*示例188.使用from浏览对象的图形*

rule "Find all grades for Big Data exam" **when**

$student: Student( $plan: plan )

$exam: Exam( course == "Big Data" ) from $plan.exams

$grade: Grade() from $exam.grades

**then** /\* RHS \*/ end

在这个例子中，已经假设使用一个领域模型，该模型由一个Student具有Plan研究的人组成：a Plan可以具有零个或多个Exams和Exam零个或多个Grades。请注意，只有图形的根对象（Student在这种情况下）需要在工作内存中才能生效。

通过从XPath中借鉴想法，可以使这种语法更加简洁，因为XPath在处理集合和过滤约束时具有导航相关元素的简洁表示法。这个XPath启发的符号被称为OOPath是明确地打算浏览对象的图形。使用这个符号，前面的例子可以被重写，如下所示：

*示例189.使用OOPath浏览对象图*

rule "Find all grades for Big Data exam" **when**

Student( $grade: /plan/exams[course == "Big Data"]/grades )

**then** /\* RHS \*/ end

形式上，OOPath表达式的核心语法可以用EBNF表示法以这种方式定义。

OOPExpr = [ID ( ":" | ":=" )] ( "/" | "?/" ) OOPSegment { ( "/" | "?/" | "." ) OOPSegment } ;

OOPSegment = ID ["#" ID] ["[" ( Number | Constraints ) "]"]

实际上，OOPath表达式具有以下特征。

* 它必须从一个完全没有反应的OOPath的情况开始（/或?/参见下文）。
* 它可以用.操作符取消引用对象的单个属性
* 它可以使用/操作符取消引用对象的多个属性。如果返回一个集合，它将迭代集合中的值
* 在遍历引用对象时，它可以过滤那些不满足一个或多个约束的对象，并将其写为方括号中的谓词表达式，如下所示：

Student( $grade: /plan/exams[ course == "Big Data" ]/grades )

* 在oopath遍历期间，也可以将遍历的对象向下转换为泛型集合中声明的类的子类。通过这种方式，后续约束也可以安全地访问仅在该子类中声明的属性，如下所示：

Student( $grade: /plan/exams**#AdvancedExam[ course == "Big Data", level > 3 ]/grades )**

对象不是在此内联转换中指定的类的实例将被自动过滤掉。

* 约束还可以与当前迭代之前遍历的图的对象有关联。例如下面的OOPath：

Student( $grade: /plan/exams/grades[ result > ../averageResult ] )

只会匹配高于平均水平的成绩。

* 一个约束也可以递归地成为另一个OOPath，如下所示：

Student( $exam: /plan/exams[ /grades[ result > 20 ] ] )

* 项目也可以通过将其索引放在方括号之间来访问，如下所示：

Student( $grade: /plan/exams[0]/grades )

遵守Java约定与基于XPath 1的版本相比，OOPath索引是基于0的

#### [11.2.1。反应性和非反应性OOPath](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reactive_and_non_reactive_oopath)

目前Drools无法对涉及评估OOPath表达式期间遍历的深度嵌套对象的更新做出反应。为了使这些对象响应变化，有必要使它们扩展类org.drools.core.phreak.ReactiveObject。计划通过实施一种机制来克服属于特定域模型的类的机制，从而克服这个限制。

扩展该类后，域对象可以通过调用继承方法更新其某个字段的某个字段时通知引擎notifyModification，如下例所示：

*例190.通知引擎一个考试已经转移到另一门课程*

**public** **void** **setCourse**(String course) {

**this**.course = course;

notifyModification(**this**);

}

以这种方式使用如下的OOPath时：

Student( $grade: /plan/exams[ course == "Big Data" ]/grades )

如果考试移至其他课程，则重新触发该规则，并重新计算匹配规则的成绩列表。

也可以仅在OOPath的一个子部分中具有反应性，如下所示：

Student( $grade: /plan/exams[ course == "Big Data" ]?/grades )

在这里，使用?/分隔符而不是/一个分隔符，引擎会响应对考试所做的更改，或者如果将考试添加到计划中，而不是在现有考试中添加新分数的情况下，则不会。当然，如果一个OOPath块没有响应，那么从那里直到表达式结尾的OOPath的所有剩余部分也不会被反应。例如下面的OOPath

Student( $grade: ?/plan/exams[ course == "Big Data" ]/grades )

将完全没有反应。出于这个原因，不允许?/在相同的OOPath中多次使用分隔符，因此表达式如下所示：

Student( $grade: /plan?/exams[ course == "Big Data" ]?/grades )

会导致编译时错误。

### [11.3。性状](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_traits)

警告：此功能仍处于实验状态，可能会有所更改

同样的事实可能有多个动态类型，它们不适合在类层次结构中自然适用。性状可以模拟这种非常常见的情况。特征是可以在运行时应用（并最终移除）到单个对象的接口。要创建一个特性而不是传统的bean，必须像下面的例子那样明确地声明它们：

**declare** trait GoldenCustomer

// **fields** will **map** **to** getters/setters

code : **String**

balance : **long**

discount : int

maxExpense : **long**

**end**

在运行时，这个声明产生一个接口，它可以用来编写模式，但不能直接实例化。为了将特征应用于对象，我们提供了新的don关键字，可以像这样简单地使用它：

**when**

$c : Customer()

**then**

GoldenCustomer gc = don( $c, GoldenCustomer.**class** );

**end**

当一个核心对象拥有一个特征时，一个代理类就会被即时创建（对于每个核心/特征类组合，将会生成一个这样的类懒惰的类）。包装核心对象并实现trait接口的代理实例会自动插入，并可能会激活其他规则。声明和使用接口，从引擎中免费获得实现代理的直接好处是，编写规则时可以利用多重继承层次结构。但是，核心类不需要静态地实现任何这些接口，也便于使用遗留类作为核心。实际上，任何对象都可以使用特征，只要它们被声明为@Traitable。请注意，此注释过去是可选的，但现在是强制性的。

import org.drools.core.factmodel.traits.Traitable;

declare Customer

@**Traitable**

code : String

balance : long

end

核心类和trait接口之间的唯一连接处于代理级别：特征并不特别与核心类相关联。这意味着相同的特征可以应用于完全不同的对象。出于这个原因，这个特质并不会透明地揭露其核心对象的领域。因此，在使用trait接口编写规则时，通常只有接口的字段可用。但是，对应于核心对象字段的接口中的任何字段都将由代理类映射：

**when**

$o: OrderItem( $p : price, $code : custCode )

$c: GoldenCustomer( code == $code, $a : balance, $d: discount )

**then**

$c.setBalance( $a - $p\*$d );

**end**

在这种情况下，代码和余额将从底层的Customer对象中读取。同样，setAccount将修改底层对象，保留对数据结构的强类型访问。核心类和所有穿戴的接口中的硬字段必须具有相同的名称和类型。该名称用于建立映射：如果两个字段具有相同的名称，则它们也必须具有相同的声明类型。注释@ org.drools.core.factmodel.traits.Alias允许放宽此限制。如果提供@Alias，则其值字符串将用于解析映射而不是原始字段名称。@Alias可以应用于特征和核心bean。

**import** org.drools.core.factmodel.traits.\*;

declare trait GoldenCustomer

balance : **long** @Alias( "org.acme.foo.accountBalance" )

end

declare Person

@Traitable

name : String

savings : **long** @Alias( "org.acme.foo.accountBalance" )

end

when

**GoldenCustomer**( balance &gt; 1000 ) *// will react to new Person( 2000 )*

then

end

目前正在做更多的工作来放宽这个限制（稍后参见关于“逻辑”特性的实验部分）。现在，人们可能会想知道当核心类不提供接口中定义的字段的实现时会发生什么。我们把这些特征字段称为硬核字段，这些字段也是核心字段，因此很容易获得，而我们将那些不是由核心类提供的字段定义为软核。相反，隐藏字段是核心类中未被接口公开的字段。

所以，虽然硬性领域管理是直观的，但仍然存在软性和隐藏领域的问题。隐藏字段通常只能直接使用核心类访问。但是，“字段”映射可用于特征界面来访问隐藏字段。如果该字段无法解析，则返回null。请注意，此功能将来可能会发生变化。

**when**

$sc : GoldenCustomer( fields[ "age" ] > 18 ) // age **is** declared **by** the underlying core **class**, **but** **not** **by** **GoldenCustomer**

**then**

相反，软字段存储在特定于每个核心对象并由代理引用的类地图数据结构中，以便即使在对象使用多种特征时也可以有效地共享。

when

$sc : GoldenCustomer( $c : code, *// hard getter*

$maxExpense : maxExpense > 1000 *// soft getter*

)

then

$sc.setDiscount( ... ); *// soft setter*

end

核心对象也包含对其所有代理的引用，以便可以使用一种动态的“instanceof”运算符（我们称其为isA）来跟踪哪些类型已添加到对象中。运算符可以接受一个String，一个类文字或一个类文字列表。在后一种情况下，只有在所有性状都被穿戴的情况下才能满足约束条件。

$sc : GoldenCustomer( $maxExpense : maxExpense > 1000,

**this** isA "SeniorCustomer", **this** isA [ NationalCustomer.**class**, OnlineCustomer.**class** ]

)

最终，业务逻辑可能需要从包装对象中删除特征。为此，我们提供了两种选择。首先是一个“合乎逻辑的东西”，这将导致从traiting操作中产生的代理的逻辑插入。TMS将确保在首先删除其逻辑支持时删除特征。

**then**

don( $x, // core object

Customer.class, // trait **class**

true // optional flag **for** logical insertion

)

第二种是使用“shed”关键字，该关键字可以删除任何作为参数传递的类型的子类型（或等效类型）。请注意，从5.5版开始，棚只允许移除一个特定的特征。

**then**

Thing t = shed( $x, GoldenCustomer.**class** ) // also removes any trait that

此操作返回另一个实现org.drools.core.factmodel.traits.Thing接口的代理，其中定义了getFields（）和getCore（）方法。在内部，实际上，所有已声明的特征都是为了扩展这个接口（除了指定的其他特性）而生成的。这允许用软田保存包装，否则这些软田会丢失。

特征及其代理也以另一种方式相关。从版本5.6开始，每当核心对象被“修改”时，它的代理也会自动“修改”，以允许基于特征的模式对硬域中的潜在变化做出反应。同样，每当特性代理（由特征模式调整）被修改时，修改就会传播到核心类和其他特征。Morover每当执行don操作时，核心对象也会自动修改，以重新评估可能触发的任何“isA”操作。

可能会导致大量修改，严重影响性能（和正确性）。所以目前实施了两种解决方案。首先，每当一个核心对象被修改时，只有最特殊的特征（在特征接口之间的继承意义上）被更新，并且内部阻塞机制已经到位，以确保每个可能匹配的模式只被评估一次。所以，在以下情况下：

**declare** **trait** GoldenCustomer end

**declare** **trait** NationalGoldenustomer extends GoldenCustomer end

**declare** **trait** SeniorGoldenCustomer extends GoldenCustomer end

对GoldenCustomer，NationalGoldenCustomer和SeniorGoldenCustomer wold这两个对象的修改只会导致后两个代理实际进行修改。第一个匹配GoldenCustomer和NationalGoldenCustomer的任何模式; 后者将被禁止重新匹配GoldenCustomer，但会被允许匹配SeniorGoldenCustomer模式。相反，修改GoldenCustomer代理并不是必需的，因为它已经至少包含一个其他更具体的特征。

第二种方法，直到usr，将特征标记为@PropertyReactive。属性反应性启用特征并考虑到特征字段映射，以阻止不必要的传播。

#### [11.3.1。级联特征](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cascading_traits)

**警告**：此功能非常实验，可能会有变化

通常情况下，一个坚硬的领域必须通过物体穿戴的所有特征暴露于其原始类型，以防止诸如此类的情况出现

**declare** Person

@Traitable

**name** : **String**

**id** : **String**

**end**

**declare** trait Customer

**id** : **String**

**end**

**declare** trait Patient

**id** : **long** // Person can't don Patient, or an exception will be thrown

end

如果某个人不顾客户和患者，硬字段ID的类型将不明确。但是，请考虑以下示例，GoldenCustomers将其最好的朋友引荐给他们，以便他们成为客户：

**declare** Person

@Traitable( **logical**=true )

bestFriend : Person

**end**

**declare** trait Customer **end**

**declare** trait GoldenCustomer extends Customer

refers : Customer @**Alias**( "bestFriend" )

**end**

除了@Alias之外，Person-as-GoldenCustomer最好的朋友可能会与GoldenCustomer特性的要求兼容，前提是他们本身就是某种客户。将一个人标记为“逻辑上可联系的” - 即添加注解@Traitable（逻辑= true） - 将指示引擎尝试并保留逻辑一致性，而不是由于具有不同类型声明的硬字段而引发异常（Person vs Customer ）。接下来的操作将会起作用：

Person p1 = **new** Person();

Person p2 = **new** Person();

p1.setBestFriend( p2 );

...

Customer c2 = don( p2, Customer.class );

...

GoldenCustomer gc1 = don( p1, GoldenCustomer.class );

...

p1.getBestFriend(); *// returns p2*

gc1.getRefers(); *// returns c2, a Customer proxy wrapping p2*

请注意，在p1变为GoldenCustomer时，p2必须已经成为客户本身，否则将抛出运行时异常，因为GoldenCustomer的定义会被违反。

然而，在某些情况下，人们可能想要推断而不是验证p2是客户，因为p1是GoldenCustomer。这种模式可以通过使用注释@ org.drools.core.factmodel.traits.Trait（logical = true）将Customer标记为“逻辑”来启用。在这种情况下，当p1成为GoldenCustomer时，如果p2不是客户，它将自动拒绝客户特性以保持系统的逻辑完整性。

请注意，核心类的注释可为其字段启用动态类型管理，而特征的注释将决定它们是作为完整性约束强制执行还是动态级联。

**import** org.drools.factmodel.traits.\*;

**declare** trait Customer

@Trait( logical = true )

end

### [11.4。规则单元](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_units_2)

|  |  |
| --- | --- |
|  | 此功能是实验性的 |

规则单元表示将规则集划分为更小单元，将不同数据源绑定到这些单元并编排单个单元执行的声明性方法。规则单元是数据源，全局变量和规则的集合。您可以通过实施RuleUnit标记界面来定义规则单元，如以下示例所示：

*RuleUnit类：*

**package** org.mypackage.myunit;

**public** **static** **class** **AdultUnit** **implements** **RuleUnit** {

**private** **int** adultAge;

**private** DataSource<Person> persons;

**public** **AdultUnit**( ) { }

**public** **AdultUnit**( DataSource<Person> persons, **int** age ) {

**this**.persons = persons;

**this**.age = age;

}

*// A DataSource of Persons for this RuleUnit*

**public** DataSource<Person> **getPersons**() {

**return** persons;

}

*// A global variable valid in this RuleUnit*

**public** **int** **getAdultAge**() {

**return** adultAge;

}

*// --- life cycle methods*

@Override

**public** **void** **onStart**() {

System.out.println("AdultUnit started.");

}

@Override

**public** **void** **onEnd**() {

System.out.println("AdultUnit ended.");

}

}

这里persons是Person类型的事实的来源。它表示那部分工作记忆，它与评估规则单元时使用的特定入口点有关。adultAge是可以从属于该规则单元的所有规则访问的全局变量。最后两种方法是规则单元生命周期的一部分，并由引擎调用。可以监视RuleUnit的生命周期，覆盖以下方法：

| *表12.规则单元生命周期方法* | |
| --- | --- |
| **方法** | **调用时** |
| 在onStart（） | 规则引擎开始评估单位 |
| 结束（） | 该单元的评估终止 |
| onSuspend（） | 单元的执行被暂停（仅用于runUntilHalt） |
| 的onResume（） | 单位的执行被恢复（仅用于runUntilHalt） |
| onYield（RuleUnit其他） | 在给定规则单元中规则的结果会触发不同单元的执行 |

所有这些方法在RuleUnit接口中都有一个空的默认实现，所以它们的实现是可选的。此时，可以将一个或多个规则添加到此规则单元。默认情况下，drl文件中的所有规则都会根据drl文件名称的命名约定自动关联到规则单元：如果drl文件位于同一个包中并且具有与实现RuleUnit接口的类相同的名称，该drl文件中的规则将隐含属于该单元。例如，包org.mypackage.myunit中名为AdultUnit.drl的drl文件中的所有规则都将自动成为规则单元的一部分org.mypackage.myunit.AdultUnit。

您可以避免使用此命名约定，并使用new unit关键字明确声明drl文件中的规则所属的单位。单位声明必须始终紧跟包声明并包含drl文件规则所属的类的名称（在该包中）。例如，

*属于规则单元的规则*

package org.mypackage.myunit

unit AdultUnit

rule Adult **when**

$p : Person(age >= adultAge) from persons

**then**

System.out.println($p.getName() + " is adult and greater than " + adultAge);

**end**

|  |  |
| --- | --- |
|  | 不允许在同一个KieBase中混合使用和不使用规则单元的规则。 |

此外，您可以使用drools 6中引入的oopath符号以更方便的方式重写相同的模式，如下所示：

*属于使用oopath表示法的规则单元的规则*

package org.mypackage.myunit

unit AdultUnit

rule Adult **when**

$p : /persons[age >= adultAge]

**then**

System.out.println($p.getName() + " is adult and greater than " + adultAge);

**end**

在此，规则左侧匹配的人员将从DataSourceRuleUnit类中包含的名称中检索。该adultAge规则的左侧和右侧都使用该变量，方式与在drl级别定义的全局相同。换句话说，personsDataSource作为一个特定的入口点，为工作内存提供资源。

创建一个最简单的方法DataSource是使用一组固定的数据，如下所示：

DataSource<Person> persons = DataSource.create( **new** Person( "Mario", 42 ),

**new** Person( "Marilena", 44 ),

**new** Person( "Sofia", 4 ) );

要执行给定中定义的一个或多个规则单元KieBase，请创建一个新的RuleUnitExecutor并将其绑定到它KieBase自己。例如，

KieBase kbase = kieContainer.getKieBase();

RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

此时，AdultUnit通过将人员传递DataSource给它并在其上运行来创建它RuleUnitExecutor。

RuleUnit adultUnit = **new** AdultUnit(persons, 18);

executor.run( adultUnit );

这会产生以下输出：

org.mypackage.myunit.AdultUnit started.

Marilena is adult and greater than 18

Mario is adult and greater than 18

org.mypackage.myunit.AdultUnit ended.

不要显式创建规则单元实例，而是更方便地将要运行的规则单元类传递给执行程序，并让执行程序创建它的实例。然后可以DataSource在运行之前设置s和其他变量。为此，请确保您之前在执行程序本身上注册了这些变量，以便下面的代码产生与前一个完全相同的结果：

executor.bindVariable( "persons", persons );

.bindVariable( "adultAge", 18 );

executor.run( AdultUnit.class );

传递给RuleUnitExecutor.bindVariable()方法的名称在运行时用于将所述变量绑定到具有相同名称的RuleUnit类的字段。例如，在前面的例子中，RuleUnitExecutor将以前绑定到“persons”名称的数据源和绑定到字符串“adultAge”的值18注入到新规则单元中，以将该数据源添加到AdultUnit类中相应名称的字段中。可以重写此默认值，并使用@UnitVar注释明确地为规则单元类的每个字段定义逻辑绑定名称。例如，以下类中的字段绑定可以用其他名称重新定义：

**package** org.mypackage.myunit;

**public** **static** **class** **AdultUnit** **implements** **RuleUnit** {

@UnitVar("minAge")

**private** **int** adultAge = 18;

@UnitVar("data")

**private** DataSource<Person> persons;

}

然后，您可以使用这些替代名称将变量绑定到执行程序并运行该单元：

executor.bindVariable( "data", persons );

.bindVariable( "minAge", 18 );

executor.run( AdultUnit.class );

规则单元可以在被动模式下执行，如上所示（对应于fireAllRules在整个会话中调用）或使用runUntilHalt与会话等效的主动模式执行fireUntilHalt。至于fireUntilHalt，runUntilHalt阻塞，因此必须在单独的线程上发布。例如，

**new** Thread( () -> executor.runUntilHalt( adultUnit ) ).start();

#### [11.4.1。数据源](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_data_sources)

A DataSource是给定规则单元处理的数据的来源。规则单元可以具有零个或多个数据源，并且在规则单元内声明的每个DataSource对应于规则单元执行程序中的不同入口点。一个DataSource可以被不同的单元共享，但是在这种情况下，会有很多不同的入口点，每个单元都有一个入口点，通过它们可以插入相同的对象。

换句话说，它DataSource代表规则单元的入口点，因此可以在其中插入新的事实：

Person mario = **new** Person( "Mario", 42 );

FactHandle marioFh = persons.insert( mario );

修改事实，可选择指定已修改的属性集以便利用属性反应性：

mario.setAge( 43 );

persons.update( marioFh, mario, "age" );

或删除它

persons.delete( marioFh );

#### [11.4.2。必须运行并声明守卫RuleUnit](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_imperatively_running_and_declaratively_guarding_a_ruleunit)

如预期的那样，您可以在同一知识库中定义多个规则单元，这些单元可以通过调用或守护彼此的执行来以协调的方式工作。为了演示这一点，让我们假设有以下两个drl文件，每个文件都包含属于不同规则单元的规则。

package org.mypackage.myunit

unit AdultUnit

rule Adult **when**

Person(age >= 18, $name : name) from persons

**then**

System.out.println($name + " is adult");

**end**

package org.mypackage.myunit

unit NotAdultUnit

rule NotAdult **when**

$p : Person(age < 18, $name : name) from persons

**then**

System.out.println($name + " is NOT adult");

modify($p) { setAge(18); }

drools.run( AdultUnit.**class** );

**end**

还假设RuleUnitExecutor从KieBase这些规则和DataSource与之相关的人员中建立了一个创建。

RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

DataSource<Person> persons = executor.newDataSource( "persons",

**new** Person( "Mario", 42 ),

**new** Person( "Marilena", 44 ),

**new** Person( "Sofia", 4 ) );

请注意，在这种情况下，我们DataSource直接创建RuleUnitExecutor并将其绑定到单个语句中的“persons”变量。

此时尝试执行NotAdultUnit单元，我们获得以下输出：

Sofia **is** NOT adult

Mario **is** adult

Marilena **is** adult

Sofia **is** adult

事实上，NotAdult规则在评估年龄低于18岁的人“Sofia”时找到匹配。然后它将她的年龄修改为18，并且该语句drools.run( AdultUnit.class )触发其他单位的执行，该单位有一个现在可以触发的规则所有3人在DataSource。这意味着结果中的drools.run()陈述是强制中断规则单元执行并将控制权交给不同规则单元的方式。

相反，drools.guard()当满足包含该语句的规则的LHS中的条件时，该语句允许声明性地安排另一个规则单元的执行。更确切地说，使用这种机制，给定规则单元中的规则充当不同单位的警卫。这意味着，当规则引擎至少为守护规则的LHS生成一个匹配时，守护的RuleUnit被视为活动的。当然，RuleUnit可以有多个守护规则。

让我们看看这是如何与另一个实际的例子。假设有一个简单的BoxOffice类

**public** **class** **BoxOffice** {

**private** **boolean** open;

**public** **BoxOffice**( **boolean** open ) {

**this**.open = open;

}

**public** **boolean** **isOpen**() {

**return** open;

}

**public** **void** **setOpen**( **boolean** open ) {

**this**.open = open;

}

}

和一个BoxOfficeUnit票房数据源。

**public** **class** **BoxOfficeUnit** **implements** **RuleUnit** {

**private** DataSource<BoxOffice> boxOffices;

**public** DataSource<BoxOffice> **getBoxOffices**() {

**return** boxOffices;

}

}

现在我们介绍只要有至少一个开放的票房，就要继续销售活动的门票。为了实现这个目标，让我们定义第二个单位与一个DataSource人和第二个门票。

**public** **class** **TicketIssuerUnit** **implements** **RuleUnit** {

**private** DataSource<Person> persons;

**private** DataSource<AdultTicket> tickets;

**private** List<String> results;

**public** **TicketIssuerUnit**() { }

**public** **TicketIssuerUnit**( DataSource<Person> persons, DataSource<AdultTicket> tickets ) {

**this**.persons = persons;

**this**.tickets = tickets;

}

**public** DataSource<Person> **getPersons**() {

**return** persons;

}

**public** DataSource<AdultTicket> **getTickets**() {

**return** tickets;

}

**public** List<String> **getResults**() {

**return** results;

}

}

然后我们可以在BoxOfficeUnit中定义一个守卫第二个单元的第一个规则。

package org.mypackage.myunit;

unit BoxOfficeUnit;

rule BoxOfficeIsOpen **when**

$box: /boxOffices[ open ]

**then**

drools.guard( TicketIssuerUnit.**class** );

**end**

通过这种方式，我们实现了我们预期的目标：通过在某些时候运行BoxOfficeUnit，它还将评估TicketIssuerUnit中的规则，该规则定义为

package org.mypackage.myunit;

unit TicketIssuerUnit;

rule IssueAdultTicket **when**

$p: /persons[ age >= 18 ]

**then**

tickets.insert(new AdultTicket($p));

**end**

rule RegisterAdultTicket **when**

$t: /tickets

**then**

results.add( $t.getPerson().getName() );

**end**

这是由BoxOfficeIsOpen规则保护的，直到至少存在一组符合该规则的LHS模式的事实。换句话说，至少有一个开放票房的存在将保持守卫规则，并且其守卫单位会保持活跃状态​​，这在以下用例中很明显。

DataSource<Person> persons = executor.newDataSource( "persons" );

DataSource<BoxOffice> boxOffices = executor.newDataSource( "boxOffices" );

DataSource<AdultTicket> tickets = executor.newDataSource( "tickets" );

List<String> list = **new** ArrayList<>();

executor.bindVariable( "results", list );

*// two open box offices*

BoxOffice office1 = **new** BoxOffice(**true**);

FactHandle officeFH1 = boxOffices.insert( office1 );

BoxOffice office2 = **new** BoxOffice(**true**);

FactHandle officeFH2 = boxOffices.insert( office2 );

persons.insert(**new** Person("Mario", 40));

*// fire BoxOfficeIsOpen -> run TicketIssuerUnit -> fire RegisterAdultTicket*

executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );

assertEquals( "Mario", list.get(0) );

list.clear();

persons.insert(**new** Person("Matteo", 30));

executor.run(BoxOfficeUnit.class); *// fire RegisterAdultTicket*

assertEquals( 1, list.size() );

assertEquals( "Matteo", list.get(0) );

list.clear();

*// close one box office, the other is still open*

office1.setOpen(**false**);

boxOffices.update(officeFH1, office1);

persons.insert(**new** Person("Mark", 35));

executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );

assertEquals( "Mark", list.get(0) );

list.clear();

*// all box offices, are now closed*

office2.setOpen(**false**);

boxOffices.update(officeFH2, office2); *// guarding rule no longer true*

persons.insert(**new** Person("Edson", 35));

executor.run(BoxOfficeUnit.class); *// no fire*

assertEquals( 0, list.size() );

#### [11.4.3。RuleUnit身份](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ruleunit_identity)

由于规则可以保护多个规则单元，同时一个单位可以被守卫，然后被多条规则激活，所以有必要明确定义给定单位的身份。默认情况下，单位的标识就是规则单位类。这是在接口的getUnitIdentity()默认方法中编码的RuleUnit

**default** Identity **getUnitIdentity**() {

**return** **new** Identity( getClass() );

}

并暗示每个单位都是由单身人士威胁RuleUnitExecutor。为了演示这个让我们假设RuleUnit只有一个DataSource接受任何类型的对象的简单类

**public** **class** **Unit0** **implements** **RuleUnit** {

**private** DataSource<Object> input;

**public** DataSource<Object> **getInput**() {

**return** input;

}

}

以及属于这个单位的一个规则，它使用两种不同的条件来保护另一个单位。

package org.mypackage.myunit

unit Unit0

rule GuardAgeCheck **when**

$i: /input*#Integer*

$s: /input*#String*

**then**

drools.guard( new AgeCheckUnit($i) );

drools.guard( new AgeCheckUnit($s.length()) );

**end**

这一秒RuleUnit旨在检查一组人的年龄。然后它有一个DataSource人要检查，一个minAge变量进行此检查和一个列表积累结果

**public** **class** **AgeCheckUnit** **implements** **RuleUnit** {

**private** **final** **int** minAge;

**private** DataSource<Person> persons;

**private** List<String> results;

**public** **AgeCheckUnit**( **int** minAge ) {

**this**.minAge = minAge;

}

**public** DataSource<Person> **getPersons**() {

**return** persons;

}

**public** **int** **getMinAge**() {

**return** minAge;

}

**public** List<String> **getResults**() {

**return** results;

}

}

而实际执行人员检查的相应规则DataSource如下：

package org.mypackage.myunit

unit AgeCheckUnit

rule CheckAge **when**

$p : /persons{ age > minAge }

**then**

results.add($p.getName() + ">" + minAge);

**end**

在这一点上，我们可以创建一个RuleUnitExecutor，将它绑定到包含这2个单位的知识库中，并创建2个DataSource以供给相同的单位。

RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

DataSource<Object> input = executor.newDataSource( "input" );

DataSource<Person> persons = executor.newDataSource( "persons",

**new** Person( "Mario", 42 ),

**new** Person( "Sofia", 4 ) );

List<String> results = **new** ArrayList<>();

executor.bindVariable( "results", results );

现在我们准备将一些对象插入输入数据源并执行Unit0。

ds.insert("test");

ds.insert(3);

ds.insert(4);

executor.run(Unit0.class);

作为执行结果，结果列表将包含以下内容：

[Sofia>3, Mario>3]

正如预期的那样，名为AgeCheckUnit的规则单元被看作是一个单例，然后只执行一次，这次minAge 等于3（但这不是确定性的）。插入输入数据源的字符串“测试”和整数4也可以触发第二次执行，minAge设置为4，但这不会发生，因为具有相同身份的另一个单元已经被评估。为了解决这个问题，重写类中的getUnitIdentity()方法以 AgeCheckUnit将变量minAge包含在其身份中就足够了。

**public** **class** **AgeCheckUnit** **implements** **RuleUnit** {

...

@Override

**public** Identity **getUnitIdentity**() {

**return** **new** Identity(getClass(), minAge);

}

}

这样做后，具有minAge 3和4的单位被认为是两个不同的单位，然后进行评估，因此试图重新运行前一个例子，结果列表将包含

[Mario>4, Sofia>3, Mario>3]

# Drools集成

集成文档

## [12. Drools命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.commands)

### [12.1。API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commandsapisection)

Drools命令的XML编组/解组需要使用特殊的类，这些类将在下面的章节中描述。

以下网址显示jaxb，xstream和json编组的示例脚本示例：

* <http://fisheye.jboss.org/browse/JBossRules/trunk/drools-camel/src/test/resources/org/drools/camel/component/jaxb.mvt?r=HEAD>
* <http://fisheye.jboss.org/browse/JBossRules/trunk/drools-camel/src/test/resources/org/drools/camel/component/jaxb.mvt?r=HEAD>
* <http://fisheye.jboss.org/browse/JBossRules/trunk/drools-camel/src/test/resources/org/drools/camel/component/xstream.mvt?r=HEAD>

#### [12.1.1。XStream的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_xstream)

要使用XStream命令编组器，您需要使用DroolsHelperProvider来获取XStream实例。我们需要使用它，因为它已经注册了命令转换器。

* 编组

。BatchExecutionHelperProviderImpl.newXStreamMarshaller（）toxml用于（命令）;

* 解组

BatchExecutionHelperProviderImpl.newXStreamMarshaller（）。fromXML（XML）

#### [12.1.2。JSON](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_json)

编组/解组的JSON API类似于XStream API：

* 编组

。BatchExecutionHelper.newJSonMarshaller（）toxml用于（命令）;

* 解组

BatchExecutionHelper.newJSonMarshaller（）。fromXML（XML）

#### [12.1.3。JAXB](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jaxb)

有两种使用JAXB的选项，你可以在XSD文件中定义你的模型，或者你可以有一个POJO模型。在这两种情况下，你都必须在JAXBContext中声明你的模型，为了做到这一点，你需要使用Drools Helper类。一旦你有了JAXBContext，你需要根据需要创建Unmarshaller / Marshaller。

##### [12.1.3.1。使用XSD文件来定义模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_an_xsd_file_to_define_the_model)

使用在XSD文件中定义的模型，您需要有一个将您的XSD模型添加为资源的知识库。

为此，必须将XSD文件作为XSD ResourceType添加到KnowledgeBuilder中。最后，您可以使用使用KnowledgeBuilder创建的知识库创建JAXBContext

Options xjcOpts = **new** Options();

xjcOpts.setSchemaLanguage(Language.XMLSCHEMA);

JaxbConfiguration jaxbConfiguration = KnowledgeBuilderFactory.newJaxbConfiguration( xjcOpts, "xsd" );

kbuilder.add(ResourceFactory.newClassPathResource("person.xsd", getClass()), ResourceType.XSD, jaxbConfiguration);

KnowledgeBase kbase = kbuilder.newKnowledgeBase();

List<String> classesName = **new** ArrayList<String>();

classesName.add("org.drools.compiler.test.Person");

JAXBContext jaxbContext = KnowledgeBuilderHelper.newJAXBContext(classesName.toArray(**new** String[classesName.size()]), kbase);

##### [12.1.3.2。使用POJO模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_a_pojo_model)

在这种情况下，您需要使用DroolsJaxbHelperProviderImpl来创建JAXBContext。这个类有两个参数：

1. classNames：带有要在编组/解组过程中使用的类的规范名称的List。
2. 属性：JAXB自定义属性

List<String> classNames = **new** ArrayList<String>();

classNames.add("org.drools.compiler.test.Person");

JAXBContext jaxbContext = DroolsJaxbHelperProviderImpl.createDroolsJaxbContext(classNames, **null**);

Marshaller marshaller = jaxbContext.createMarshaller();

### [12.2。支持的命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_commands_supported)

目前，支持以下命令：

* BatchExecutionCommand
* InsertObjectCommand
* RetractCommand
* ModifyCommand
* GetObjectCommand
* InsertElementsCommand
* FireAllRulesCommand
* StartProcessCommand
* SignalEventCommand
* CompleteWorkItemCommand
* AbortWorkItemCommand
* QueryCommand
* SetGlobalCommand
* GetGlobalCommand
* GetObjectsCommand

|  |  |
| --- | --- |
|  | 在下一个片段代码中，我们将使用一个POJO org.drools.compiler.test.Person，它有两个字段   * 名称：字符串 * 年龄：整数 |
|  | 在下面的例子中，为了编组我们使用下一个代码片段的命令：   * XStream的   String xml = BatchExecutionHelper.newXStreamMarshaller().toXML(command);   * JSON   String xml = BatchExecutionHelper.newJSonMarshaller().toXML(command);   * JAXB * Marshaller marshaller = jaxbContext.createMarshaller(); * StringWriter xml = **new** StringWriter(); * marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, **true**);   marshaller.marshal(command, xml); |

#### [12.2.1。BatchExecutionCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_batchexecutioncommand)

* 说明：包含将要发送和执行的命令列表的命令。
* 属性

| *表13. BatchExecutionCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 抬头 | 设置将要执行命令的知识会话ID | 真正 |
| 命令 | 要执行的命令列表 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* InsertObjectCommand insertObjectCommand = **new** InsertObjectCommand(**new** Person("john", 25));
* FireAllRulesCommand fireAllRulesCommand = **new** FireAllRulesCommand();
* command.getCommands().add(insertObjectCommand);

command.getCommands().add(fireAllRulesCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <insert>
  + <org.drools.compiler.test.Person>
  + <name>john</name>
  + <age>25</age>
  + </org.drools.compiler.test.Person>
  + </insert>
  + <fire-all-rules/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":[{"insert":{"object":{"org.drools.compiler.test.Person":{"name":"john","age":25}}}},{"fire-all-rules":""}]}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <insert>
  + <object xsi:type="person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  + <age>25</age>
  + <name>john</name>
  + </object>
  + </insert>
  + <fire-all-rules max="-1"/>

</batch-execution>

#### [12.2.2。InsertObjectCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_insertobjectcommand)

* 说明：在知识会话中插入一个对象。
* 属性

| *表14. InsertObjectCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 目的 | 要插入的对象 | 真正 |
| outIdentifier | Id来标识在对象插入中创建并添加到执行结果中的FactHandle | 假 |
| 返回object | Boolean确定是否必须在执行结果中返回对象。默认值：true | 假 |
| 入口点 | 插入的入口点 | 假 |

* 命令创建
* List<Command> cmds = ArrayList<Command>();
* Command insertObjectCommand = CommandFactory.newInsert(**new** Person("john", 25), "john", **false**, **null**);
* cmds.add( insertObjectCommand );

BatchExecutionCommand command = CommandFactory.createBatchExecution(cmds, "ksession1" );

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <insert out-identifier="john" entry-point="my stream" return-object="false">
  + <org.drools.compiler.test.Person>
  + <name>john</name>
  + <age>25</age>
  + </org.drools.compiler.test.Person>
  + </insert>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"insert":{"entry-point":"my stream", "out-identifier":"john","return-object":false,"object":{"org.drools.compiler.test.Person":{"name":"john","age":25}}}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <insert out-identifier="john" entry-point="my stream" >
  + <object xsi:type="person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  + <age>25</age>
  + <name>john</name>
  + </object>
  + </insert>

</batch-execution>

#### [12.2.3。RetractCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_retractcommand)

* 说明：撤回知识会话中的对象。
* 属性

| *表15. RetractCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 处理 | FactHandle与要撤回的对象相关联 | 真正 |

* 命令创建：我们有两个选项，输出结果相同：
  1. 从字符串创建事实句柄
  2. BatchExecutionCommand command = **new** BatchExecutionCommand();
  3. command.setLookup("ksession1");
  4. RetractCommand retractCommand = **new** RetractCommand();
  5. retractCommand.setFactHandleFromString("123:234:345:456:567");

command.getCommands().add(retractCommand);

* 1. 设置插入对象时收到的事实句柄
  2. BatchExecutionCommand command = **new** BatchExecutionCommand();
  3. command.setLookup("ksession1");
  4. RetractCommand retractCommand = **new** RetractCommand(factHandle);

command.getCommands().add(retractCommand);

* XML输出
  1. XStream的
  2. <batch-execution lookup="ksession1">
  3. <retract fact-handle="0:234:345:456:567"/>

</batch-execution>

* 1. JSON

{"batch-execution":{"lookup":"ksession1","commands":{"retract":{"fact-handle":"0:234:345:456:567"}}}}

* 1. JAXB
  2. **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  3. <batch-execution lookup="ksession1">
  4. <retract fact-handle="0:234:345:456:567"/>

</batch-execution>

#### [12.2.4。ModifyCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_modifycommand)

* 说明：允许您修改知识会话中以前插入的对象。
* 属性

| *表16. ModifyCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 处理 | FactHandle与要撤回的对象相关联 | 真正 |
| 制定者 | setters对象的修改列表 | 真正 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* ModifyCommand modifyCommand = **new** ModifyCommand();
* modifyCommand.setFactHandleFromString("123:234:345:456:567");
* List<Setter> setters = **new** ArrayList<Setter>();
* setters.add(**new** SetterImpl("age", "30"));
* modifyCommand.setSetters(setters);

command.getCommands().add(modifyCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <modify fact-handle="0:234:345:456:567">
  + <set accessor="age" value="30"/>
  + </modify>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"modify":{"fact-handle":"0:234:345:456:567","setters":{"accessor":"age","value":30}}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <modify fact-handle="0:234:345:456:567">
  + <set value="30" accessor="age"/>
  + </modify>

</batch-execution>

#### [12.2.5。GetObjectCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getobjectcommand)

* 说明：用于从知识会话中获取对象
* 属性

| *表17. GetObjectCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| factHandle | FactHandle与要撤回的对象相关联 | 真正 |
| outIdentifier | Id来标识在对象插入中创建并添加到执行结果中的FactHandle | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* GetObjectCommand getObjectCommand = **new** GetObjectCommand();
* getObjectCommand.setFactHandleFromString("123:234:345:456:567");
* getObjectCommand.setOutIdentifier("john");

command.getCommands().add(getObjectCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <get-object fact-handle="0:234:345:456:567" out-identifier="john"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"get-object":{"fact-handle":"0:234:345:456:567","out-identifier":"john"}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <get-object out-identifier="john" fact-handle="0:234:345:456:567"/>

</batch-execution>

#### [12.2.6。InsertElementsCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_insertelementscommand)

* 说明：用于插入对象列表。
* 属性

| *表18. InsertElementsCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 对象 | 知识会话中要插入的对象列表 | 真正 |
| outIdentifier | Id来标识在对象插入中创建并添加到执行结果中的FactHandle | 假 |
| 返回object | Boolean确定是否必须在执行结果中返回对象。默认值：true | 假 |
| 入口点 | 插入的入口点 | 假 |

* 命令创建
* List<Command> cmds = ArrayList<Command>();
* List<Object> objects = **new** ArrayList<Object>();
* objects.add(**new** Person("john", 25));
* objects.add(**new** Person("sarah", 35));
* Command insertElementsCommand = CommandFactory.newInsertElements( objects );
* cmds.add( insertElementsCommand );

BatchExecutionCommand command = CommandFactory.createBatchExecution(cmds, "ksession1" );

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <insert-elements>
  + <org.drools.compiler.test.Person>
  + <name>john</name>
  + <age>25</age>
  + </org.drools.compiler.test.Person>
  + <org.drools.compiler.test.Person>
  + <name>sarah</name>
  + <age>35</age>
  + </org.drools.compiler.test.Person>
  + </insert-elements>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"insert-elements":{"objects":[{"containedObject":{"@class":"org.drools.compiler.test.Person","name":"john","age":25}},{"containedObject":{"@class":"Person","name":"sarah","age":35}}]}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <insert-elements return-objects="true">
  + <list>
  + <element xsi:type="person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  + <age>25</age>
  + <name>john</name>
  + </element>
  + <element xsi:type="person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  + <age>35</age>
  + <name>sarah</name>
  + </element>
  + <list>
  + </insert-elements>

</batch-execution>

#### [12.2.7。FireAllRulesCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fireallrulescommand)

* 说明：允许执行创建的规则激活。
* 属性

| *表19. FireAllRulesCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 最大 | 要执行的规则激活的最大数量。默认值为-1，不会对执行造成任何限制 | 假 |
| outIdentifier | 添加在执行结果上触发的规则激活次数 | 假 |
| agendaFilter | 允许使用议程过滤器执行规则 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* FireAllRulesCommand fireAllRulesCommand = **new** FireAllRulesCommand();
* fireAllRulesCommand.setMax(10);
* fireAllRulesCommand.setOutIdentifier("firedActivations");

command.getCommands().add(fireAllRulesCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <fire-all-rules max="10" out-identifier="firedActivations"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"fire-all-rules":{"max":10,"out-identifier":"firedActivations"}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <fire-all-rules out-identifier="firedActivations" max="10"/>

</batch-execution>

#### [12.2.8。StartProcessCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_startprocesscommand)

* 说明：允许您使用ID启动进程。您也可以传递要插入的参数和初始数据。
* 属性

| *表20. StartProcessCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 的ProcessID | 要启动的进程的ID | 真正 |
| 参数 | 一个Map <String，Object>在进程启动时传递参数 | 假 |
| 数据 | 在进程启动之前要在知识会话中插入的对象的列表 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* StartProcessCommand startProcessCommand = **new** StartProcessCommand();
* startProcessCommand.setProcessId("org.drools.task.processOne");

command.getCommands().add(startProcessCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <start-process processId="org.drools.task.processOne"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"start-process":{"process-id":"org.drools.task.processOne"}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <start-process processId="org.drools.task.processOne">
  + <parameter/>
  + </start-process>

</batch-execution>

#### [12.2.9。SignalEventCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_signaleventcommand)

* 说明：发送信号事件。
* 属性

| *表21. SignalEventCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 事件类型 |  | 真正 |
| processInstanceId |  | 假 |
| 事件 |  | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* SignalEventCommand signalEventCommand = **new** SignalEventCommand();
* signalEventCommand.setProcessInstanceId(1001);
* signalEventCommand.setEventType("start");
* signalEventCommand.setEvent(**new** Person("john", 25));

command.getCommands().add(signalEventCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <signal-event process-instance-id="1001" event-type="start">
  + <org.drools.pipeline.camel.Person>
  + <name>john</name>
  + <age>25</age>
  + </org.drools.pipeline.camel.Person>
  + </signal-event>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"signal-event":{"process-instance-id":1001,"@event-type":"start","event-type":"start","object":{"org.drools.pipeline.camel.Person":{"name":"john","age":25}}}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <signal-event event-type="start" process-instance-id="1001">
  + <event xsi:type="person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  + <age>25</age>
  + <name>john</name>
  + </event>
  + </signal-event>

</batch-execution>

#### [12.2.10。CompleteWorkItemCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_completeworkitemcommand)

* 说明：允许您完成一个WorkItem。
* 属性

| *表22. CompleteWorkItemCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| workItemId | 要完成的WorkItem的ID | 真正 |
| 结果 |  | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* CompleteWorkItemCommand completeWorkItemCommand = **new** CompleteWorkItemCommand();
* completeWorkItemCommand.setWorkItemId(1001);

command.getCommands().add(completeWorkItemCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <complete-work-item id="1001"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"complete-work-item":{"id":1001}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <complete-work-item id="1001"/>

</batch-execution>

#### [12.2.11。AbortWorkItemCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_abortworkitemcommand)

* 说明：允许您中止WorkItem。与session.getWorkItemManager（）相同。abortWorkItem（workItemId）
* 属性

| *表23. AbortWorkItemCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| workItemId | 要完成的WorkItem的ID | 真正 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* AbortWorkItemCommand abortWorkItemCommand = **new** AbortWorkItemCommand();
* abortWorkItemCommand.setWorkItemId(1001);

command.getCommands().add(abortWorkItemCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <abort-work-item id="1001"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"abort-work-item":{"id":1001}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <abort-work-item id="1001"/>

</batch-execution>

#### [12.2.12。QueryCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_querycommand)

* 说明：执行知识库中定义的查询。
* 属性

| *表24. QueryCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 名称 | 查询名称 | 真正 |
| outIdentifier | 查询结果的标识符。查询结果将会用这个标识符添加到执行结果中 | 假 |
| 参数 | 作为查询参数传递的对象列表 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* QueryCommand queryCommand = **new** QueryCommand();
* queryCommand.setName("persons");
* queryCommand.setOutIdentifier("persons");

command.getCommands().add(queryCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <query out-identifier="persons" name="persons"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"query":{"out-identifier":"persons","name":"persons"}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <query name="persons" out-identifier="persons"/>

</batch-execution>

#### [12.2.13。SetGlobalCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_setglobalcommand)

* 说明：允许您设置全局。
* 属性

| *表25. SetGlobalCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 识别码 | 知识库中定义的全局标识符 | 真正 |
| 目的 | 要设置为全局的对象 | 假 |
| 出 | 一个布尔值，用于将设置的全局结果添加到执行结果中 | 假 |
| outIdentifier | 全局执行结果的标识符 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* SetGlobalCommand setGlobalCommand = **new** SetGlobalCommand();
* setGlobalCommand.setIdentifier("helper");
* setGlobalCommand.setObject(**new** Person("kyle", 30));
* setGlobalCommand.setOut(**true**);
* setGlobalCommand.setOutIdentifier("output");

command.getCommands().add(setGlobalCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <set-global identifier="helper" out-identifier="output">
  + <org.drools.compiler.test.Person>
  + <name>kyle</name>
  + <age>30</age>
  + </org.drools.compiler.test.Person>
  + </set-global>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"set-global":{"identifier":"helper","out-identifier":"output","object":{"org.drools.compiler.test.Person":{"name":"kyle","age":30}}}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <set-global out="true" out-identifier="output" identifier="helper">
  + <object xsi:type="person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  + <age>30</age>
  + <name>kyle</name>
  + </object>
  + </set-global>

</batch-execution>

#### [12.2.14。GetGlobalCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getglobalcommand)

* 说明：允许您获得先前定义的全局。
* 属性

| *表26. GetGlobalCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| 识别码 | 知识库中定义的全局标识符 | 真正 |
| outIdentifier | 要在执行结果中使用的标识符 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* GetGlobalCommand getGlobalCommand = **new** GetGlobalCommand();
* getGlobalCommand.setIdentifier("helper");
* getGlobalCommand.setOutIdentifier("helperOutput");

command.getCommands().add(getGlobalCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <get-global identifier="helper" out-identifier="helperOutput"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"get-global":{"identifier":"helper","out-identifier":"helperOutput"}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <get-global out-identifier="helperOutput" identifier="helper"/>

</batch-execution>

#### [12.2.15。GetObjectsCommand](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getobjectscommand)

* 说明：将当前会话中的所有对象作为集合返回。
* 属性

| *表27. GetObjectsCommand属性* | | |
| --- | --- | --- |
| **名称** | **描述** | **需要** |
| ObjectFilter的 | 一个ObjectFilter，用于过滤从当前会话返回的对象 | 假 |
| outIdentifier | 要在执行结果中使用的标识符 | 假 |

* 命令创建
* BatchExecutionCommand command = **new** BatchExecutionCommand();
* command.setLookup("ksession1");
* GetObjectsCommand getObjectsCommand = **new** GetObjectsCommand();
* getObjectsCommand.setOutIdentifier("objects");

command.getCommands().add(getObjectsCommand);

* XML输出
  + XStream的
  + <batch-execution lookup="ksession1">
  + <get-objects out-identifier="objects"/>

</batch-execution>

* + JSON

{"batch-execution":{"lookup":"ksession1","commands":{"get-objects":{"out-identifier":"objects"}}}}

* + JAXB
  + **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
  + <batch-execution lookup="ksession1">
  + <get-objects out-identifier="objects"/>

</batch-execution>

## [13. CDI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cdichapter)

### [13.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_2)

[CDI](http://www.cdi-spec.org/)，Contexts和Dependency Injection是Java规范，它为应用程序提供声明性控件和结构。KIE可以使用它来自动实例化和绑定事物，而不需要使用编程式API。

### [13.2。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_annotations)

@KContainer，@KBase和@KSession都支持可选的'name'属性。CDI通常在注入时会执行“getOrCreate”，所有注入都会针对同一批注释接收相同的实例。'name'注释强制每个名称的唯一实例，但该名称的所有实例都将是identity equals。

#### [13.2.1。@KReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kreleaseid)

用于将实例绑定到KieModule的特定版本。如果kie-ci位于类路径中，则会自动解析依赖关系，从远程存储库下载。

#### [13.2.2。@KContainer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kcontainer)

@KContainer是可选的，因为它可以通过使用@Inject和变量类型推断来检测和添加。

*注入Classpath KieContainer*

@Inject

**private** KieContainer kContainer;

*为动态KieModule注入KieContainer*

@Inject

@KReleaseId(groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieContainer kContainer;

*为KieModule注入名为KieContainer*

@Inject

@KContainer(name = "kc1")

@KReleaseId(groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieContainer kContainer;

#### [13.2.3。@KBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kbase)

@KBase是可选的，因为它可以通过使用@Inject和变量类型推断来检测和添加。

默认参数（如果给出）映射到value属性，并从kmodule.xml文件中指定KieBase的名称。

*从Classpath KieContainer注入默认KieBase*

@Inject

**private** KieBase kbase;

*从动态KieModule注入默认KieBase*

@Inject

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieBase kbase;

*并行版本加载“jar1.KBase1”KieBase*

@Inject

@KBase("kbase1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieBase kbase1v10;

@Inject

@KBase("kbase1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieBase kbase1v10;

*使用'name'属性为'jar1.KBase1'KieBase强制新实例*

@Inject

@KSession(value="kbase1", name="kb1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieBase kbase1kb1;

@Inject

@KSession(value="kbase1", name="kb2")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieBase kbase1kb2;

#### [13.2.4。KieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_kiesession)

@KSession是可选的，因为它可以通过使用@Inject和变量类型推断来检测和添加。

默认参数（如果给出）映射到value属性，并从kmodule.xml文件中指定KieSession的名称

*从Classpath KieContainer注入默认KieSession*

@Inject

**private** KieSession ksession;

*从动态KieModule注入默认KieSession*

@Inject

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession;

*并行版本加载“jar1.KBase1”KieBase*

@Inject

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksessionv10;

@Inject

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieSession ksessionv11;

*使用'name'属性强制'jar1.KBase1'KieSession的新实例*

@Inject

@KSession(value="ksession1", name="ks1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession1ks1

@Inject

@KSession(value="ksession1", name="ks2")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession1ks2

#### [13.2.5。用于StatelessKieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_statelesskiesession)

@KSession是可选的，因为它可以通过使用@Inject和变量类型推断来检测和添加。

默认参数（如果给出）映射到value属性，并从kmodule.xml文件中指定KieSession的名称。

*从Classpath KieContainer注入默认的StatelessKieSession*

@Inject

**private** StatelessKieSession ksession;

*从动态KieModule注入默认的StatelessKieSession*

@Inject

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** StatelessKieSession ksession;

*并行版本加载“jar1.KBase1”KieBase*

@Inject

@KSession("ksession1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.0")

**private** StatelessKieSession ksessionv10;

@Inject

@KSession("ksession1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.1")

**private** StatelessKieSession ksessionv11;

*使用'name'属性来强制'jar1.KBase1'StatelessKieSession的新实例*

@Inject

@KSession(value="ksession1", name="ks1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** StatelessKieSession ksession1ks1

@Inject

@KSession(value="ksession1", name="ks2")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** StatelessKieSession ksession1ks2

### [13.3。API示例比较](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_api_example_comparison)

CDI可以将实例注入字段，甚至可以将它们作为参数传递。在这个例子中使用了字段注入。

*一个名为KieSession的CDI示例*

@Inject

@KSession("ksession1")

KieSession kSession;

**public** **void** **go**(PrintStream out) {

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

}

这是比API方法更少的代码和更多的声明。

*指定KieSession的API等效示例*

**public** **void** **go**(PrintStream out) {

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

}

## [14.与Spring集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.spring)

### [14.1。Drools 6.0的重要变化](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_important_changes_for_drools_6_0)

Drools Spring集成经历了与Drools 6.0的更改完全一致的改造。以下是一些主要变化

* 推荐的Drools Spring的前缀已经从'drools：'改为'kie：'
* 6.0中新的顶级标签
  + 纪伊：KMODULE
  + kie：import（从版本6.2开始）
  + kie：releaseId（从版本6.2开始）
* 以下标签不再作为顶级标签有效。
  + kie：kbase - kie：kmodule标签的子项。
  + kie：ksession - kie：kbase标签的孩子。
* 从以前的版本中删除标签Drools 5.x
  + 滴料：资源
  + 滴料：资源
  + 滴料：格
  + 口水：网格节点

### [14.2。与Drools Expert整合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_drools_expert)

在本节中，我们将解释kie命名空间。

#### [14.2.1。KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodule)

所述<纪伊：KMODULE>定义KieBase的集合和相关联的KieSession的。该KMODULE标签有一个强制参数“ID”。

| *表28.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| ID | Bean的id是要从其他bean引用的名称。标准的Spring ID语义适用。 | 是 |

一个KMODULE代码只能包含以下标记为儿童。

* 纪伊：kbase

请参阅Drools Expert文档中的kmodule.xml文档，以获取有关kmodule需求的详细说明。

#### [14.2.2。KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_3)

##### [14.2.2.1。<kie：kbase>的参数作为属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_kbase_s_parameters_as_attributes)

| *表29.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| 名称 | KieBase的名称 | 是 |
| 包 | 以逗号分隔的资源包列表将包含在此kbase中 | 没有 |
| 包括 | 包含kbase名称。来自相应kbase的所有资源都包含在此kbase中。 | 没有 |
| 默认 | 布尔（TRUE / FALSE）。默认kbase，如果没有提供，它被假定为FALSE | 没有 |
| 范围 | 原型| 单身。如果未提供假定为单身（默认） | 没有 |
| eventProcessingMode | 事件处理模式。有效的选项是STREAM，CLOUD | 没有 |
| equalsBehavior | 有效选项是IDENTITY，EQUALITY | 没有 |
| declarativeAgenda | 有效的选项被启用，禁用，真，假 | 没有 |

##### [14.2.2.2。一个kbase标签只能包含followingtags儿童。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_em_kbase_em_tag_can_contain_only_the_followingtags_as_children)

* 纪伊：ksession

##### [14.2.2.3。<kie：kbase>的定义示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_kbase_s_definition_example)

kmodule可以包含多个（1..n）kbase元素。

*示例191. kbase定义示例*

<kie:kmodule id="sample\_module">

<kie:kbase name="kbase1" packages="org.drools.spring.sample">

...

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.2.4。Spring Bean范围（用于KieBase和KieSession）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spring_bean_scope_for_kiebase_and_kiesession)

定义KieBase或KieSession时，您可以选择为该bean声明范围。例如，为了强制Spring在每次需要时产生一个新的bean实例，你应该声明bean的scope属性为'prototype'。类似的方法，如果你希望Spring在每次需要的时候返回相同的bean实例，你应该声明bean的scope属性是'singleton'。

#### [14.2.3。重要的提示](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_important_note)

为了正确初始化kmodule对象（kbase / ksession），必须为bean类型定义org.kie.spring.KModuleBeanFactoryPostProcessor或者org.kie.spring.annotations.KModuleAnnotationPostProcessor定义。

*例子192.普通的kie-spring post处理器bean定义*

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

*例193.使用注释时kie-spring后处理器的bean定义*

<bean id="kiePostProcessor"

class="org.kie.spring.annotations.KModuleAnnotationPostProcessor"/>

|  |  |
| --- | --- |
|  | 没有org.kie.spring.KModuleBeanFactoryPostProcessor或org.kie.spring.annotations.KModuleAnnotationPostProcessor bean定义，kie-spring集成将失效。 |

#### [14.2.4。KieSessions](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesessions)

<kie：ksession>元素定义了KieSessions。同一个标记用于定义Stateful（org.kie.api.runtime.KieSession）和无状态（org.kie.api.runtime.StatelessKieSession）会话。

##### [14.2.4.1。<kie：ksession>的参数作为属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_ksession_s_parameters_as_attributes)

| *表30.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| 名称 | ksession的名字。 | 是 |
| 类型 | 会话有状态还是 无状态？。如果此属性为空或缺失，则会话将被视为Stateful类型。 | 没有 |
| 默认 | 这是默认会话吗？ | 没有 |
| 范围 | 原型| 单身。如果未提供假定为单身（默认） | 没有 |
| clockType | REALTIME或PSEUDO | 没有 |
| 听众-REF | 指定对事件侦听器组的引用（请参阅下面的['定义一组侦听器'](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_grouping_listeners)部分）。 | 没有 |

*例子194. ksession定义的例子*

<kie:kmodule id="sample-kmodule">

<kie:kbase name="drl\_kiesample3" packages="drl\_kiesample3">

<kie:ksession name="ksession1" type="stateless"/>

<kie:ksession name="ksession2"/>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.4.2。Spring Bean范围（用于KieBase和KieSession）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_spring_bean_scope_for_kiebase_and_kiesession_2)

定义KieBase或KieSession时，您可以选择为该bean声明范围。例如，为了强制Spring在每次需要时产生一个新的bean实例，你应该声明bean的scope属性为'prototype'。类似的方法，如果你希望Spring在每次需要的时候返回相同的bean实例，你应该声明bean的scope属性是'singleton'。

#### [14.2.5。纪伊：ReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_releaseid)

##### [14.2.5.1。<kie：releaseId>的参数作为属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_releaseid_s_parameters_as_attributes)

| *表31.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| ID | Bean的id是要从其他bean引用的名称。标准的Spring ID语义适用。 | 是 |
| 的groupId | 来自Maven GAV的groupId | 是 |
| 的artifactId | 来自Maven GAV的artifactId | 是 |
| 版 | Maven GAV版本 | 是 |

*例195. releaseId定义的例子*

<kie:releaseId id="beanId" groupId="org.kie.spring"

artifactId="named-artifactId" version="1.0.0-SNAPSHOT"/>

#### [14.2.6。纪伊：进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_import)

从版本6.2开始，kie-spring允许从类路径中的kjar中导入kie对象。目前支持导入kie对象的两种模式。

| **属性** | **描述** | **需要** |
| --- | --- | --- |
| releaseId | 引用一个Bean ID。标准的Spring ID语义适用。 | 没有 |
| enableScanner | 启用扫描仪。只有在指定'releaseId'时才使用此属性。 | 没有 |
| scannerInterval | 以毫秒为单位扫描间隔。只有在指定'releaseId'时才使用此属性。 | 没有 |

##### [14.2.6.1。全球进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_global_import)

的进口标签将迫使所有的罐子的自动扫描的类路径上，初始化纪伊对象（Kbase / KSessions）和这些对象导入到弹簧上下文。

*全球进口*

<kie:import />

##### [14.2.6.2。具体导入 - ReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_specific_import_releaseid)

在导入标签上使用releaseId-ref属性将初始化特定的Kie对象（Kbase / KSessions）并将这些对象导入到spring上下文中。

*使用releaseId导入Kie对象*

<kie:import releaseId-ref="namedKieSession"/>

<kie:releaseId id="namedKieSession" groupId="org.drools"

artifactId="named-kiesession" version="7.7.0.Final"/>

Kie扫描功能可以启用KieBase导入的特定releaseId。此功能目前不适用于全球导入。

*使用releaseId导入Kie对象 - 启用扫描器*

<kie:import releaseId-ref="namedKieSession"

enableScanner="true" scannerInterval="1000"/>

<kie:releaseId id="namedKieSession" groupId="org.drools"

artifactId="named-kiesession" version="7.7.0.Final"/>

如果扫描器已定义并启用，则会创建隐式KieScanner对象并将其插入到弹簧上下文中。它可以从春天的背景中回顾。

*从Spring上下文中恢复KieScanner*

*// the implicit name would be releaseId#scanner*

KieScanner releaseIdScanner = context.getBean("namedKieSession#scanner", KieScanner.class);

releaseIdScanner.scanNow();

|  |  |
| --- | --- |
|  | kie-ci必须在类路径中可用，才能使releaseId导入功能正常工作。 |

#### [14.2.7。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_annotations_2)

@KContainer，@KBase和@KSession都支持可选的'name'属性。Spring通常在注入时会“获取”，所有注入都会为同一组注释接收相同的实例。'name'注释强制每个名称的唯一实例，但该名称的所有实例都将是identity equals。

##### [14.2.7.1。@KReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kreleaseid_2)

用于将实例绑定到KieModule的特定版本。如果kie-ci位于类路径中，则会自动解析依赖关系，从远程存储库下载。

##### [14.2.7.2。@KContainer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kcontainer_2)

*注入Classpath KieContainer*

@KContainer

**private** KieContainer kContainer;

*为动态KieModule注入KieContainer*

@KContainer

@KReleaseId(groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieContainer kContainer;

*为KieModule注入名为KieContainer*

@KContainer(name = "kc1")

@KReleaseId(groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieContainer kContainer;

##### [14.2.7.3。@KBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kbase_2)

默认参数（如果给出）映射到value属性，并从spring xml文件中指定KieBase的名称。

*从Classpath KieContainer注入默认KieBase*

@KBase

**private** KieBase kbase;

*从动态KieModule注入默认KieBase*

@KBase

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieBase kbase;

*并行版本加载“jar1.KBase1”KieBase*

@KBase("kbase1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieBase kbase1v10;

@KBase("kbase1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieBase kbase1v11;

*并行版本加载'jar1.ksession1'KieSession*

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession11kb2;

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieSession ksession11kb2;

##### [14.2.7.4。KieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_kiesession_2)

默认参数（如果给出）映射到value属性，并从kmodule.xml或spring xml文件指定KieSession的名称

*从Classpath KieContainer注入默认KieSession*

@KSession

**private** KieSession ksession;

*从动态KieModule注入默认KieSession*

@KSession

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession;

*并行版本加载“jar1.KBase1”KieBase*

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksessionv10;

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.1")

**private** KieSession ksessionv11;

*使用'name'属性强制'jar1.KBase1'KieSession的新实例*

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession1ks1

@KSession("ksession1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** KieSession ksession1ks2

##### [14.2.7.5。用于StatelessKieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_statelesskiesession_2)

默认参数（如果给出）映射到value属性，并从kmodule.xml或spring xml文件指定KieSession的名称。

*从Classpath KieContainer注入默认的StatelessKieSession*

@KSession

**private** StatelessKieSession ksession;

*从动态KieModule注入默认的StatelessKieSession*

@KSession

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** StatelessKieSession ksession;

*并行版本加载“jar1.KBase1”KieBase*

@KSession("ksession1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.0")

**private** StatelessKieSession ksessionv10;

@KSession("ksession1")

@KReleaseId( groupId = "jar1", rtifactId = "art1", version = "1.1")

**private** StatelessKieSession ksessionv11;

@KSession(value="ksession1", name="ks1")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** StatelessKieSession ksession1ks1

@KSession(value="ksession1", name="ks2")

@KReleaseId( groupId = "jar1", artifactId = "art1", version = "1.0")

**private** StatelessKieSession ksession1ks2

##### [14.2.7.6。重要的提示](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_important_note_2)

使用注释时，为了正确初始化kmodule对象（kbase / ksession），必须为org.kie.spring.annotations.KModuleAnnotationPostProcessor定义类型的bean

*例子196. kie-spring注释处理器bean定义*

<bean id="kiePostProcessor"

class="org.kie.spring.annotations.KModuleAnnotationPostProcessor"/>

*示例197. kie-spring注释 - 组件扫描*

<context:component-scan base-package="org.kie.spring.annotations"/>

|  |  |
| --- | --- |
|  | 使用注释时后处理器不同。 |

#### [14.2.8。事件监听器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_listeners)

Drools支持将3种类型的监听器添加到KieSession - AgendaListener，WorkingMemoryListener，ProcessEventListener

kie-spring模块允许您使用XML标签将这些监听器配置为KieSessions。这些标记与实际的监听器接口名称相同，即<kie：agendaEventListener ...。>，<kie：ruleRuntimeEventListener ...。>和<kie：processEventListener ...。>。

kie-spring提供了将听众定义为独立（个人）听众并将他们定义为一组的功能。

##### [14.2.8.1。定义独立听众：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_stand_alone_listeners)

##### [14.2.8.2。属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_attributes)

| *表32.样品* | | |
| --- | --- | --- |
| **属性** | **需要** | **描述** |
| REF | 没有 | 对另一个声明bean的引用。 |

*示例198.监听器配置示例 - 使用bean：ref。*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>

<bean id="mock-process-listener" class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners\_kmodule">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession2">

<kie:agendaEventListener ref="mock-agenda-listener"/>

<kie:processEventListener ref="mock-process-listener"/>

<kie:ruleRuntimeEventListener ref="mock-rr-listener"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.8.3。嵌套元素：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_nested_elements)

* 豆
  + class = String
  + name = String（可选）

*示例199.侦听器配置示例 - 使用嵌套的Bean。*

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession1">

<kie:agendaEventListener>

<bean class="mocks.MockAgendaEventListener"/>

</kie:agendaEventListener>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.8.4。空标签：没有'ref'且没有嵌套的bean的声明](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_empty_tag_declaration_with_no_ref_and_without_a_nestedbean)

当没有引用实现bean并且不包含嵌套bean 的侦听器时，<drools：ruleRuntimeEventListener />底层实现将添加在API中定义的侦听器的Debug版本。

调试侦听器将相应的Event toString消息打印到\_System.err。\_

*示例200.侦听器配置示例 - 默认为由知识API提供的调试版本。*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>

<bean id="mock-process-listener" class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession2">

<kie:agendaEventListener />

<kie:processEventListener />

<kie:ruleRuntimeEventListener />

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.8.5。混合和匹配不同的声明样式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_mix_and_match_of_different_declaration_styles)

drools-spring模块允许您在相同的KieSession中混合和匹配不同的声明式样式。以下示例提供更多清晰度。

*示例201.监听器配置示例 - “ref”/嵌套bean /空样式的混合和匹配。*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>

<bean id="mock-process-listener" class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession1">

<kie:agendaEventListener>

<bean class="org.kie.spring.mocks.MockAgendaEventListener"/>

</kie:agendaEventListener>

</kie:ksession>

<kie:ksession name="ksession2">

<kie:agendaEventListener ref="mock-agenda-listener"/>

<kie:processEventListener ref="mock-process-listener"/>

<kie:ruleRuntimeEventListener ref="mock-rr-listener"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.8.6。定义多个相同类型的侦听器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_multiple_listeners_of_the_same_type)

为KieSession定义多个相同事件监听器类型的bean也是有效的。

*示例202.侦听器配置示例 - 同一类型的多个侦听器。*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession1">

<kie:agendaEventListener ref="mock-agenda-listener"/>

<kie:agendaEventListener>

<bean class="org.kie.spring.mocks.MockAgendaEventListener"/>

</kie:agendaEventListener>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.8.7。定义一组监听器：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_grouping_listeners)

drools-spring允许分组听众。当您定义一组侦听器并希望将它们附加到多个会话时，这非常有用。当我们为“测试”定义一组监听器，然后希望将其切换为“生产”使用时，分组功能也非常有用。

##### [14.2.8.8。属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_attributes_2)

| *表33.样品* | | |
| --- | --- | --- |
| **属性** | **需要** | **描述** |
| ID | 是 | 唯一标识符 |

##### [14.2.8.9。嵌套元素：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_nested_elements_2)

* 纪伊：agendaEventListener ...
* 纪伊：ruleRuntimeEventListener ...
* 纪伊：processEventListener ...

|  |  |
| --- | --- |
|  | 上述子元素可以按任意顺序声明。组中只允许一种类型的声明。 |

##### [14.2.8.10。例：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_example)

*例203.一群听众 - 例子*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>

<bean id="mock-process-listener" class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="statelessWithGroupedListeners" type="stateless"

listeners-ref="debugListeners"/>

</kie:kbase>

</kie:kmodule>

<kie:eventListeners id="debugListeners">

<kie:agendaEventListener ref="mock-agenda-listener"/>

<kie:processEventListener ref="mock-process-listener"/>

<kie:ruleRuntimeEventListener ref="mock-rr-listener"/>

</kie:eventListeners>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

#### [14.2.9。记录仪](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_loggers)

Drools支持将两种类型的记录器添加到KieSessions - ConsoleLogger，FileLogger。

kie-spring模块允许您使用XML标签将这些记录器配置为KieSessions。这些标签与实际的记录器界面名称相同，即<kie：consoleLogger ...。>和<kie：fileLogger ...。>。

##### [14.2.9.1。定义控制台记录器：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_console_logger)

控制台记录器可以通过使用<kie：consoleLogger />标签附加到KieSession 。该标签没有属性，必须直接出现在<kie：ksession ...。>元素下。

*例204.定义一个控制台记录器 - 例子*

<kie:kmodule id="loggers\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ConsoleLogger-statefulSession" type="stateful">

<kie:consoleLogger/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.9.2。定义文件记录器：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_file_logger)

可以使用<kie：fileLogger />标记将文件记录器附加到KieSession 。此标记具有以下属性，并且必须直接存在于<kie：ksession ...。>元素下。

| *表34.样品* | | |
| --- | --- | --- |
| **属性** | **需要** | **描述** |
| ID | 是 | 唯一标识符 |
| 文件 | 是 | 磁盘上实际文件的路径 |
| 螺纹 | 没有 | 默认为false。有效值为'true'或'false' |
| 间隔 | 没有 | 整数。指定将内容从内存刷新到磁盘的时间间隔。 |

*例205.定义一个文件记录器 - 例子*

<kie:kmodule id="loggers\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ConsoleLogger-statefulSession" type="stateful">

<kie:fileLogger id="fl\_logger" file="#{ systemProperties['java.io.tmpdir'] }/log1"/>

<kie:fileLogger id="tfl\_logger" file="#{ systemProperties['java.io.tmpdir'] }/log2"

threaded="true" interval="5"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.9.3。关闭FileLogger](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_closing_a_filelogger)

为防止泄漏，建议以编程方式关闭\_ <kie：fileLogger ...> \_。

LoggerAdaptor adaptor = (LoggerAdaptor) context.getBean("fl\_logger");

adaptor.close();

#### [14.2.10。定义批处理命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_batch_commands)

可以使用<kie：batch>元素为给定的ksession定义一组批处理命令。该标记没有属性，并且必须直接存在于<kie：ksession ...。>元素下。支持的命令是

*初始化批处理命令*

* 插入对象
  + ref = String（可选）
  + 匿名bean
* 设置全局
  + identifier = String（必需）
  + reg =字符串（可选）
  + 匿名bean
* 消防所有规则
  + 最大：n
* 火，直到-停止
* 启动过程
  + 参数
    - identifier = String（必需）
    - ref = String（可选）
    - 匿名bean
* 信号事件
  + ref = String（可选）
  + event-type = String（必需）
  + process-instance-id = n（可选）

*例206.批处理命令 - 例子*

<kie:kmodule id="batch\_commands\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksessionForCommands" type="stateful">

<kie:batch>

<kie:insert-object ref="person2"/>

<kie:set-global identifier="persons" ref="personsList"/>

<kie:fire-all-rules max="10"/>

</kie:batch>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

#### [14.2.11。坚持](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_persistence)

*持久性配置选项*

* JPA的持久性
  + 交易经理
    - ref = String
  + 实体管理器工厂
    - ref = String

*示例207. ksession JPA配置示例*

<kie:kstore id="kstore" /> *<!-- provides KnowledgeStoreService implementation -->*

<bean id="myEmf"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="dataSource" ref="ds" />

<property name="persistenceUnitName"

value="org.drools.persistence.jpa.local" />

</bean>

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="myEmf" />

</bean>

<kie:kmodule id="persistence\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="jpaSingleSessionCommandService">

<kie:configuration>

<kie:jpa-persistence>

<kie:transaction-manager ref="txManager"/>

<kie:entity-manager-factory ref="myEmf"/>

</kie:jpa-persistence>

</kie:configuration>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

#### [14.2.12。利用其他弹簧功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_leveraging_other_spring_features)

本节提供了与Drools Expert集成时利用其他标准弹簧功能的详细信息。

##### [14.2.12.1。使用Spring表达式（Spel）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_spring_expressions_spel)

<kie:kmodule id="batch\_commands\_module">

<kie:kbase name="drl\_kiesample" packages="#{packageRepository.packages}">

<kie:ksession name="ksessionForCommands" type="stateful"/>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

<bean id="packageRepository" class="sample.package.class.PackageRepo">

<property name="packages" value="drl\_kiesample3">

</bean>

<kie:kmodule id="loggers\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ConsoleLogger-statefulSession" type="stateful">

<kie:fileLogger id="fl" file="#{ systemProperties['java.io.tmpdir'] }/log1"/>

<kie:fileLogger id="tfl" file="#{ systemProperties['java.io.tmpdir'] }/log2"

threaded="true" interval="5"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

##### [14.2.12.2。使用Spring配置文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_spring_profiles)

Spring 3.1为spring-beans模式的beans元素引入了一个新的profile属性。在不同环境中启用和禁用配置文件时，此属性充当开关。该属性的一个潜在用途可以是在'dev'环境中使用调试记录器定义相同的kbase，并且在'prod'环境中没有记录器。

以下代码片段说明了“配置文件”的概念。

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:kie="http://drools.org/schema/kie-spring"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

http://drools.org/schema/kie-spring http://drools.org/schema/kie-spring.xsd">

<beans profile="development">

<kie:kmodule id="test-kmodule">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession1" type="stateless">

<kie:consoleLogger />

</kie:ksession>

</kie:kbase>

</kie:kmodule>

...

</beans>

<beans profile="production">

<kie:kmodule id="test-kmodule">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession1" type="stateless"/>

</kie:kbase>

</kie:kmodule>

...

</beans>

</beans>

如上所示，Spring XML包含配置文件的定义。在加载ApplicationContext时，你必须告诉Spring你正在加载哪个配置文件。

有几种选择你的配置文件的方法，最有用的方法是使用“spring.profiles.active”系统属性。

System.setProperty("spring.profiles.active", "development");

ApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans.xml");

显然，如上所示对事物进行硬编码并不是一个好习惯，推荐的做法是保持系统属性定义独立于应用程序。

-Dspring.profiles.active="development"

配置文件也可以通过编程方式加载和启用

...

GenericXmlApplicationContext ctx = **new** GenericXmlApplicationContext("beans.xml");

ConfigurableEnvironment env = ctx.getEnvironment();

env.setActiveProfiles("development");

ctx.refresh();

...

### [14.3。与jBPM人工任务集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_jbpm_human_task)

本章描述了在使用Spring配置人工任务服务器时使用的基础架构，以及一些关于在执行此操作时使用的基础架构的信息。

#### [14.3.1。如何使用jBPM人工任务配置Spring](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_how_to_configure_spring_with_jbpm_human_task)

jBPM人工任务服务器可以配置为使用Spring持久性。[[\_kie\_spring\_human\_task\_example]](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_spring_human_task_example) 是一个使用本地事务和Spring的线程安全的EntityManager代理的例子。

下图显示[[\_kie\_spring\_human\_task\_example]中](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_spring_human_task_example)使用的依赖关系图 。

*图201. Spring Human Task集成注入依赖关系*

一个TaskService实例依赖于另外两种bean类型：一个drools SystemEventListenerbean和一个TaskSessionSpringFactoryImplbean。这个TaskSessionSpringFactoryImplbean是没有注入到TaskServicebean中的，因为这会导致循环依赖。为了解决这个问题，当TaskServicebean被注入到TaskSessionSpringFactoryImplbean中时，所使用的setter方法秘密地将TaskSessionSpringFactoryImpl实例注入到TaskServicebean中并初始化TaskServicebean。

该TaskSessionSpringFactoryImplbean负责在人工任务中创建处理事务和持久性上下文管理的所有内部实例。除了一个TaskService实例外，这个bean还需要注入一个事务管理器和一个持久化上下文。具体来说，它需要一个HumanTaskSpringTransactionManagerbean 的实例（作为事务管理器）和一个SharedEntityManagerBeanbean 实例（作为持久化上下文实例）。

我们还使用一些标准的Spring bean来配置持久性：有一个bean用来存放EntityManagerFactory实例和SharedEntityManagerBean实例。该SharedEntityManagerBean规定的实际共享，线程安全的代理EntityManager。

这个HumanTaskSpringTransactionManagerbean作为Spring事务管理器的包装器，在这个例子中是JpaTransactionManager。一个JpaTransactionManagerbean 的实例也因此被实例化。

*例208.用Spring配置人工任务*

**<?xml version="1.0" encoding="UTF-8"?>**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:jbpm="http://drools.org/schema/drools-spring"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://drools.org/schema/drools-spring org/drools/container/spring/drools-spring-1.2.0.xsd">

*<!-- persistence & transactions-->*

<bean id="htEmf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="org.jbpm.task" />

</bean>

<bean id="htEm" class="org.springframework.orm.jpa.support.SharedEntityManagerBean">

<property name="entityManagerFactory" ref="htEmf"/>

</bean>

<bean id="jpaTxMgr" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="htEmf" />

*<!-- this must be true if using the SharedEntityManagerBean, and false otherwise -->*

<property name="nestedTransactionAllowed" value="true"/>

</bean>

<bean id="htTxMgr" class="org.drools.container.spring.beans.persistence.HumanTaskSpringTransactionManager">

<constructor-arg ref="jpaTxMgr" />

</bean>

*<!-- human-task beans -->*

<bean id="systemEventListener" class="org.drools.SystemEventListenerFactory" factory-method="getSystemEventListener" />

<bean id="taskService" class="org.jbpm.task.service.TaskService" >

<property name="systemEventListener" ref="systemEventListener" />

</bean>

<bean id="springTaskSessionFactory" class="org.jbpm.task.service.persistence.TaskSessionSpringFactoryImpl"

init-method="initialize" depends-on="taskService" >

*<!-- if using the SharedEntityManagerBean, make sure to enable nested transactions -->*

<property name="entityManager" ref="htEm" />

<property name="transactionManager" ref="htTxMgr" />

<property name="useJTA" value="false" />

<property name="taskService" ref="taskService" />

</bean>

</beans>

在使用SharedEntityManagerBean实例时，配置Spring事务管理器使用嵌套事务很重要。这是因为这SharedEntityManagerBean是一个事务持久性上下文，并且会在每次操作之后关闭持久化上下文。但是，人工任务服务器需要能够在操作之后访问（保持）实体。嵌套事务允许我们仍然可以访问实体，否则这些实体将被分离并且不再可访问，特别是在使用使用实体的延迟初始化的ORM框架时。

另外，虽然TaskSessionSpringFactoryImplbean采用“useJTA”参数，但目前与Spring的JTA事务尚未完全测试。

## [15.与Aries Blueprint集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.blueprint)

### [15.1。与Drools Expert整合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_drools_expert_2)

在本节中，我们将解释kie命名空间。

#### [15.1.1。KieModule](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiemodule_2)

所述<纪伊：KMODULE>定义KieBase的集合和相关联的KieSession的。该KMODULE标签有一个强制参数“ID”。

| *表35.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| ID | Bean的id是要从其他bean引用的名称。标准Blueprint ID语义适用。 | 是 |

一个KMODULE代码只能包含以下标记为儿童。

* 纪伊：kbase

请参阅Drools Expert文档中的kmodule.xml文档，以获取有关kmodule需求的详细说明。

#### [15.1.2。KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiebase_4)

##### [15.1.2.1。<kie：kbase>的参数作为属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_kbase_s_parameters_as_attributes_2)

| *表36.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| 名称 | KieBase的名称 | 是 |
| 包 | 以逗号分隔的资源包列表将包含在此kbase中 | 没有 |
| 包括 | 包含kbase名称。来自相应kbase的所有资源都包含在此kbase中。 | 没有 |
| 默认 | 布尔（TRUE / FALSE）。默认kbase，如果没有提供，它被假定为FALSE | 没有 |
| 范围 | 原型| 单身。如果未提供假定为单身（默认） | 没有 |
| eventProcessingMode | 事件处理模式。有效的选项是STREAM，CLOUD | 没有 |
| equalsBehavior | 有效选项是IDENTITY，EQUALITY | 没有 |

##### [15.1.2.2。一个kbase代码只能包含以下标记为儿童。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_a_em_kbase_em_tag_can_contain_only_the_following_tags_as_children)

* 纪伊：ksession

##### [15.1.2.3。<kie：kbase>的定义示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_kbase_s_definition_example_2)

kmodule可以包含多个（1..n）kbase元素。

*例子209. kbase定义的例子*

<kie:kmodule id="sample\_module">

<kie:kbase name="kbase1" packages="org.drools.blueprint.sample">

...

</kie:kbase>

</kie:kmodule>

##### [15.1.2.4。蓝图Bean范围（用于KieBase和KieSession）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_blueprint_bean_scope_for_kiebase_and_kiesession)

定义KieBase或KieSession时，您可以选择为该bean声明范围。例如，为了强制Blueprint在每次需要时都生成一个新的bean实例，应该声明bean的scope属性为'prototype'。如果您希望Blueprint每次需要时都返回相同的bean实例，您应该声明bean的scope属性为'singleton'。

#### [15.1.3。KieSessions](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiesessions_2)

<kie：ksession>元素定义了KieSessions。同一个标记用于定义Stateful（org.kie.api.runtime.KieSession）和无状态（org.kie.api.runtime.StatelessKieSession）会话。

##### [15.1.3.1。<kie：ksession>的参数作为属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_ksession_s_parameters_as_attributes_2)

| *表37.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| 名称 | ksession的名字。 | 是 |
| 类型 | 会话有状态还是 无状态？ | 没有 |
| 默认 | 这是默认会话吗？ | 没有 |
| 范围 | 原型| 单身。如果未提供假定为单身（默认） | 没有 |
| clockType | REALTIME或PSEUDO | 没有 |
| 听众-REF | 指定对事件侦听器组的引用（请参阅['定义一组侦听器'](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_grouping_listeners) | 没有 |

*示例210. ksession定义示例*

<kie:kmodule id="sample-kmodule">

<kie:kbase name="drl\_kiesample3" packages="drl\_kiesample3">

<kie:ksession name="ksession1" type="stateless"/>

<kie:ksession name="ksession2"/>

</kie:kbase>

</kie:kmodule>

#### [15.1.4。纪伊：ReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_releaseid_2)

##### [15.1.4.1。<kie：releaseId>的参数作为属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kie_releaseid_s_parameters_as_attributes_2)

| *表38.样品* | | |
| --- | --- | --- |
| **属性** | **描述** | **需要** |
| ID | Bean的id是要从其他bean引用的名称。标准Blueprint ID语义适用。 | 是 |
| 的groupId | 来自Maven GAV的groupId | 是 |
| 的artifactId | 来自Maven GAV的artifactId | 是 |
| 版 | Maven GAV版本 | 是 |

*示例211. releaseId定义示例*

<kie:releaseId id="beanId" groupId="org.kie.blueprint"

artifactId="named-artifactId" version="1.0.0-SNAPSHOT"/>

#### [15.1.5。纪伊：进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_import_2)

从6.5版开始，kie-aries-blueprint允许从类路径中的kjar中导入kie对象。目前支持导入kie对象的两种模式。

| **属性** | **描述** | **需要** |
| --- | --- | --- |
| releaseId | 引用一个Bean ID。标准Blueprint ID语义适用。 | 没有 |
| enableScanner | 启用扫描仪。只有在指定'releaseId'时才使用此属性。 | 没有 |
| scannerInterval | 以毫秒为单位扫描间隔。只有在指定'releaseId'时才使用此属性。 | 没有 |

##### [15.1.5.1。全球进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_global_import_2)

该进口标签将迫使所有的罐子的自动扫描在classpath中，初始化纪伊对象（Kbase / KSessions），这些对象导入的蓝图情境。

*全球进口*

<kie:import />

##### [15.1.5.2。具体导入 - ReleaseId](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_specific_import_releaseid_2)

在导入标签上使用releaseId-ref属性将初始化特定的Kie对象（Kbase / KSessions）并将这些对象导入到蓝图上下文中。

*使用releaseId导入Kie对象*

<kie:import releaseId-ref="namedKieSession"/>

<kie:releaseId id="namedKieSession" groupId="org.drools"

artifactId="named-kiesession" version="7.7.0.Final"/>

Kie扫描功能可以启用KieBase导入的特定releaseId。此功能目前不适用于全球导入。

*使用releaseId导入Kie对象 - 启用扫描器*

<kie:import releaseId-ref="namedKieSession"

enableScanner="true" scannerInterval="1000"/>

<kie:releaseId id="namedKieSession" groupId="org.drools"

artifactId="named-kiesession" version="7.7.0.Final"/>

如果扫描仪已定义并启用，则会创建隐式KieScanner对象并将其插入到蓝图容器中。它可以使用-scanner后缀以编程方式从蓝图容器中取回。

*从Blueprint容器中恢复KieScanner*

*// the implicit name would be releaseId-scanner*

KieScanner releaseIdScanner = (KieScanner)container.getComponentInstance("namedKieSession-scanner");

releaseIdScanner.scanNow();

|  |  |
| --- | --- |
|  | kie-ci必须在类路径中可用，才能使releaseId导入功能正常工作。 |

#### [15.1.6。事件监听器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_event_listeners_2)

Drools支持将3种类型的监听器添加到KieSession - AgendaListener，WorkingMemoryListener，ProcessEventListener

kie-aries-blueprint模块允许您使用XML标签将这些监听器配置为KieSessions。这些标记与实际的监听器接口名称相同，即<kie：agendaEventListener ...。>，<kie：ruleRuntimeEventListener ...。>和<kie：processEventListener ...。>。

kie-aries-blueprint提供了将侦听器定义为独立（个人）侦听器的功能，并将它们定义为一个组。

##### [15.1.6.1。定义独立听众：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_stand_alone_listeners_2)

##### [15.1.6.2。属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_attributes_3)

| *表39.示例* | | |
| --- | --- | --- |
| **属性** | **需要** | **描述** |
| REF | 没有 | 对另一个声明bean的引用。 |

*示例212.监听器配置示例 - 使用bean：ref。*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>

<bean id="mock-process-listener" class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners\_kmodule">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession2">

<kie:agendaEventListener ref="mock-agenda-listener"/>

<kie:processEventListener ref="mock-process-listener"/>

<kie:ruleRuntimeEventListener ref="mock-rr-listener"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

##### [15.1.6.3。定义多个相同类型的侦听器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_multiple_listeners_of_the_same_type_2)

为KieSession定义多个相同事件监听器类型的bean也是有效的。

*示例213.侦听器配置示例 - 同一类型的多个侦听器。*

<bean id="mock-agenda-listener1" class="mocks.MockAgendaEventListener"/>

<bean id="mock-agenda-listener2" class="mocks.MockAgendaEventListener"/>

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksession1">

<kie:agendaEventListener ref="mock-agenda-listener1"/>

<kie:agendaEventListener ref="mock-agenda-listener2"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

##### [15.1.6.4。定义一组监听器：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_grouping_listeners)

kie-aries-blueprinty允许对听众进行分组。当您定义一组侦听器并希望将它们附加到多个会话时，这非常有用。当我们为“测试”定义一组监听器，然后希望将其切换为“生产”使用时，分组功能也非常有用。

##### [15.1.6.5。属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_attributes_4)

| *表40.样品* | | |
| --- | --- | --- |
| **属性** | **需要** | **描述** |
| ID | 是 | 唯一标识符 |

##### [15.1.6.6。嵌套元素：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_nested_elements_3)

* 纪伊：agendaEventListener ...
* 纪伊：ruleRuntimeEventListener ...
* 纪伊：processEventListener ...

|  |  |
| --- | --- |
|  | 上述子元素可以按任意顺序声明。组中只允许一种类型的声明。 |

##### [15.1.6.7。例：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_example_2)

*例214.一群听众 - 例子*

<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>

<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>

<bean id="mock-process-listener" class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="statelessWithGroupedListeners" type="stateless"

listeners-ref="debugListeners"/>

</kie:kbase>

</kie:kmodule>

<kie:eventListeners id="debugListeners">

<kie:agendaEventListener ref="mock-agenda-listener"/>

<kie:processEventListener ref="mock-process-listener"/>

<kie:ruleRuntimeEventListener ref="mock-rr-listener"/>

</kie:eventListeners>

#### [15.1.7。记录仪](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_loggers_2)

Drools支持将两种类型的记录器添加到KieSessions - ConsoleLogger，FileLogger。

kie-aries-blueprint模块允许您使用XML标签将这些记录器配置为KieSessions。这些标签与实际的记录器界面名称相同，即<kie：consoleLogger ...。>和<kie：fileLogger ...。>。

##### [15.1.7.1。定义控制台记录器：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_console_logger_2)

控制台记录器可以通过使用<kie：consoleLogger />标签附加到KieSession 。该标签没有属性，必须直接出现在<kie：ksession ...。>元素下。

*例子215.定义一个控制台记录器 - 例子*

<kie:kmodule id="loggers\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ConsoleLogger-statefulSession" type="stateful">

<kie:consoleLogger/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

##### [15.1.7.2。定义文件记录器：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_file_logger_2)

可以使用<kie：fileLogger />标记将文件记录器附加到KieSession 。此标记具有以下属性，并且必须直接存在于<kie：ksession ...。>元素下。

| *表41.样品* | | |
| --- | --- | --- |
| **属性** | **需要** | **描述** |
| ID | 是 | 唯一标识符 |
| 文件 | 是 | 磁盘上实际文件的路径 |
| 螺纹 | 没有 | 默认为false。有效值为'true'或'false' |
| 间隔 | 没有 | 整数。指定将内容从内存刷新到磁盘的时间间隔。 |

*例216.定义一个文件记录器 - 例子*

<kie:kmodule id="loggers\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ConsoleLogger-statefulSession" type="stateful">

<kie:fileLogger id="fl\_logger" file="#{ systemProperties['java.io.tmpdir'] }/log1"/>

<kie:fileLogger id="tfl\_logger" file="#{ systemProperties['java.io.tmpdir'] }/log2"

threaded="true" interval="5"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

##### [15.1.7.3。关闭FileLogger](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_closing_a_filelogger_2)

为防止泄漏，建议以编程方式关闭<kie：fileLogger ...>。

LoggerAdaptor adaptor = (LoggerAdaptor) container.getComponentInstance("fl\_logger");

adaptor.close();

#### [15.1.8。定义批处理命令](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_batch_commands_2)

可以使用<kie：batch>元素为给定的ksession定义一组批处理命令。该标记没有属性，并且必须直接存在于<kie：ksession ...。>元素下。支持的命令是

*初始化批处理命令*

* 插入对象
  + ref = String（可选）
  + 匿名bean
* 设置全局
  + identifier = String（必需）
  + reg =字符串（可选）
  + 匿名bean
* 消防所有规则
  + 最大：n
* 火，直到-停止
* 启动过程
  + 参数
    - identifier = String（必需）
    - ref = String（可选）
    - 匿名bean
* 信号事件
  + ref = String（可选）
  + event-type = String（必需）
  + process-instance-id = n（可选）

*例子217.批量命令 - 例子*

<kie:kmodule id="batch\_commands\_module">

<kie:kbase name="drl\_kiesample" packages="drl\_kiesample">

<kie:ksession name="ksessionForCommands" type="stateful">

<kie:batch>

<kie:insert-object ref="person2"/>

<kie:set-global identifier="persons" ref="personsList"/>

<kie:fire-all-rules max="10"/>

</kie:batch>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

## [16. Android集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.android)

### [16.1。与Drools Expert整合](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_drools_expert_3)

Drools Android集成有两种风格，有或没有drools-compiler依赖。如果没有drools编译器依赖关系，知识库将在构建时使用kie-maven插件预先序列化。然后可以使用API​​对它们进行反序列化，或者使用Roboguice直接注入它们。在使用drools编译器依赖时，有两种选择：（1）标准的KieContainer API或（2）与Roboguice类似的CDI注入。

#### [16.1.1。预序列化规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pre_serialized_rules)

在没有drools编译器依赖性的情况下，可以使用Drools，这会导致较小的apk，这是通过在编译时预编序知识库，然后在运行时对它们进行反序列化来实现的。

##### [16.1.1.1。Maven配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_configuration)

必须在构建时使用kie-maven-plugin序列化KieBase 。

*例218.预序列化的KieBase Maven pom.xml*

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-bom</artifactId>

<version>7.7.0.Final</version>

<type>pom</type>

<scope>import</scope>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-android</artifactId>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</exclusions>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-core</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<version>7.7.0.Final</version>

<configuration>

<kiebases>

<kiebase>HelloKB</kiebase>

</kiebases>

<resDirectory>${basedir}/src/main/res/raw</resDirectory>

</configuration>

<executions>

<execution>

<id>touch</id>

<goals>

<goal>touch</goal>

</goals>

<phase>initialize</phase>

</execution>

<execution>

<id>compile-kbase</id>

<goals>

<goal>build</goal>

</goals>

<phase>compile</phase>

</execution>

<execution>

<id>serialize</id>

<goals>

<goal>serialize</goal>

</goals>

<phase>compile</phase>

</execution>

</executions>

</plugin>

</plugins>

</build>

##### [16.1.1.2。加载KieBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_loading_the_kiebase)

在创建会话之前，KieBase必须被反序列化。

*加载序列化的KieBase*

**private** **class** **LoadKieBaseTask** **extends** **AsyncTask**<**InputStream**, **Void**, **KieBase**> {

@Override

**protected** KieBase **doInBackground**(InputStream... params) {

**try** {

logger.debug("Loading knowledge base");

**final** KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

kbase.addKnowledgePackages((List<KnowledgePackage>) DroolsStreamUtils.streamIn(params[0]));

**return** kbase;

}**catch**(Exception e) {

logger.error("Drools exception", e);

**return** **null**;

}

}

}

#### [16.1.2。KieContainer API与drools编译器依赖关系](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiecontainer_api_with_drools_compiler_dependency)

使用drools编译器依赖标准可以使用KieContainer API。这是以更大的apk为代价的。为了避免65K的限制，可以使用multidex（或者proguard）。

##### [16.1.2.1。使用drools编译器和multidex的Maven配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_configuration_with_drools_compiler_and_multidex)

kie-maven插件必须配置为构建kiebase。Multidex必须用于允许增加的依赖性。在apk中还有一些用于合并各种Drools XML文件的设置。

*示例219.带有drools-compiler和multidex的pom.xml*

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-bom</artifactId>

<version>7.7.0.Final</version>

<type>pom</type>

<scope>import</scope>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-android</artifactId>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-compiler</artifactId>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</exclusion>

<exclusion>

<groupId>xmlpull</groupId>

<artifactId>xmlpull</artifactId>

</exclusion>

<exclusion>

<groupId>xpp3</groupId>

<artifactId>xpp3\_min</artifactId>

</exclusion>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</exclusion>

<exclusion>

<groupId>org.eclipse.jdt.core.compiler</groupId>

<artifactId>ecj</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<version>7.7.0.Final</version>

<executions>

<execution>

<id>compile-kbase</id>

<goals>

<goal>build</goal>

</goals>

<phase>compile</phase>

</execution>

</executions>

</plugin>

<plugin>

<groupId>com.simpligility.maven.plugins</groupId>

<artifactId>android-maven-plugin</artifactId>

<version>4.2.1</version>

<extensions>true</extensions>

<configuration>

<sdk>

<platform>21</platform>

</sdk>

<dex>

<coreLibrary>true</coreLibrary>

<jvmArguments><jvmArgument>-Xmx2048m</jvmArgument></jvmArguments>

<multiDex>true</multiDex>

<mainDexList>maindex.txt</mainDexList>

</dex>

<extractDuplicates>true</extractDuplicates>

<apk>

<metaInf>

<includes>

<include>services/\*\*</include>

<include>kmodule.\*</include>

<include>HelloKB/\*\*</include>

<include>drools\*\*</include>

<include>maven/${project.groupId}/${project.artifactId}/\*\*</include>

</includes>

</metaInf>

</apk>

</configuration>

</plugin>

</plugins>

</build>

### [16.2。与Roboguice集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_integration_with_roboguice)

#### [16.2.1。预先序列化规则与Roboguice](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pre_serialized_rules_with_roboguice)

使用Roboguice预先序列化的知识库可以使用@KBase注释注入。

##### [16.2.1.1。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_annotations_3)

@KBase支持可选的'name'属性。CDI通常在注入时会执行“getOrCreate”，所有注入都会针对同一批注释接收相同的实例。'name'注解强制每个名称的唯一实例，但该名称的所有实例都将是identity equals。

##### [16.2.1.2。@KBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kbase_3)

默认参数映射到value属性，并从kmodule.xml文件中指定KieBase的名称。

*通过预序列化资源的名称注入KieBase*

@KBase("kbase1")

**private** KieBase kbase;

##### [16.2.1.3。AndroidManifest.xml配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_androidmanifest_xml_configuration)

Roboguice模块需要在清单中指定。

*例220. Roboguice具有预先序列化的知识库*

<application

android:largeHeap="true"

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme">

<meta-data

android:name="roboguice.modules"

android:value="org.drools.android.roboguice.DroolsModule"/>

<activity

android:label="@string/app\_name"

android:name="org.drools.examples.android.SplashActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</intent-filter>

</activity>

</application>

#### [16.2.2。KieContainer与drools编译器依赖和Roboguice](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kiecontainer_with_drools_compiler_dependency_and_roboguice)

使用Roboguice和drools编译器，几乎可以使用完整的CDI语法来注入KieContainers，KieBases和KieSessions。

##### [16.2.2.1。注释](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_annotations_4)

@KContainer，@KBase和@KSession都支持可选的'name'属性。CDI通常在注入时会执行“getOrCreate”，所有注入都会针对同一批注释接收相同的实例。'name'注释强制每个名称的唯一实例，但该名称的所有实例都将是identity equals。

##### [16.2.2.2。@KContainer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kcontainer_3)

*注入Classpath KieContainer*

@Inject

**private** KieContainer kContainer;

##### [16.2.2.3。@KBase](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__kbase_4)

默认参数（如果给出）映射到value属性，并从kmodule.xml文件中指定KieBase的名称。

*从Classpath KieContainer注入默认KieBase*

@Inject

**private** KieBase kbase;

*从Classpath KieContainer中按名称注入KieBase*

@Inject

@KBase("kbase1")

**private** KieBase kbase;

##### [16.2.2.4。KieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_kiesession_3)

@KSession是可选的，因为它可以通过使用@Inject和变量类型推断来检测和添加。

默认参数（如果给出）映射到value属性，并从kmodule.xml文件中指定KieSession的名称

*从Classpath KieContainer注入默认KieSession*

@Inject

**private** KieSession ksession;

*从Classpath KieContainer中通过名称注入StatelessKieSession*

@Inject

@KSession("ksession1")

**private** KieSession ksession;

##### [16.2.2.5。用于StatelessKieSession的@KSession](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__ksession_for_statelesskiesession_3)

@KSession是可选的，因为它可以通过使用@Inject和变量类型推断来检测和添加。

默认参数（如果给出）映射到value属性，并从kmodule.xml文件中指定KieSession的名称。

*从Classpath KieContainer注入默认的StatelessKieSession*

@Inject

**private** StatelessKieSession ksession;

*从Classpath KieContainer中通过名称注入StatelessKieSession*

@Inject

@KSession("ksession1")

**private** StatelessKieSession ksession;

##### [16.2.2.6。AndroidManifest.xml配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_androidmanifest_xml_configuration_2)

Roboguice模块需要在清单中指定。

*例子221. Roboguice清单配置*

<application

android:largeHeap="true"

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme">

<meta-data

android:name="roboguice.modules"

android:value="org.drools.android.roboguice.DroolsContainerModule"/>

<activity

android:label="@string/app\_name"

android:name="org.drools.examples.android.SplashActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</intent-filter>

</activity>

</application>

## [17. Apache Camel集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.camel)

### [17.1。骆驼](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_camel)

骆驼提供了一个轻量级的公共汽车框架来获取进出Drools的信息。

Drools引入了两个元素来实现轻松集成。

* 流口水政策

增加任何JAXB或XStream数据加载器。对于JAXB，它添加了与contextpath相关的drools相关路径，为XStream添加了Drools类的自定义转换器和别名。它还处理将ClassLoader设置为目标ksession。

* Drools端点

根据指定的drools会话执行有效负载

Drools可以像任何普通的骆驼组件一样进行配置，但请注意包装Drools相关细分的政策。这会将所有有效载荷路由到ksession1

*示例222.使用CXFRS生产者配置的Drools EndPoint*

<bean id="kiePolicy" class="org.kie.camel.component.KiePolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

<route>

<from uri="cxfrs://bean://rsServer"/>

<policy ref="kiePolicy">

<unmarshal ref="xstream" />

<to uri="kie:ksession1" />

<marshal ref="xstream" />

</policy>

</route>

</camelContext>

可以不在drools端点uri中指定会话，而是基于属性或标题来“复用”。在这个例子中，策略将检查头部字段“DroolsLookup”以执行指定的会话，如果没有指定，它将检查传入有效负载上的“查找”属性。

*示例223.使用CXFRS生产者配置的Drools EndPoint*

<bean id="kiePolicy" class="org.kie.camel.component.KiePolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

<route>

<from uri="cxfrs://bean://rsServer"/>

<policy ref="kiePolicy">

<unmarshal ref="xstream" />

<to uri="kie:dynamic" />

<marshal ref="xstream" />

</policy>

</route>

</camelContext>

*例224.从Spring和CamelContext针对Route执行的Java代码*

**public** **class** **MyTest** **extends** **CamelSpringTestSupport** {

@Override

**protected** AbstractXmlApplicationContext **createApplicationContext**() {

**return** **new** ClassPathXmlApplicationContext("org/drools/camel/component/CxfRsSpring.xml");

}

**public** **void** **test1**() **throws** Exception {

String cmd = "";

cmd += "<batch-execution lookup=\"ksession1\">\n";

cmd += " <insert out-identifier=\"salaboy\">\n";

cmd += " <org.drools.pipeline.camel.Person>\n";

cmd += " <name>salaboy</name>\n";

cmd += " </org.drools.pipeline.camel.Person>\n";

cmd += " </insert>\n";

cmd += " <fire-all-rules/>\n";

cmd += "</batch-execution>\n";

Object object = **this**.context.createProducerTemplate().requestBody("direct://client", cmd);

System.out.println( object );

}

}

以下网址显示jaxb，xstream和json编组的示例脚本示例：

* <http://fisheye.jboss.org/browse/JBossRules/trunk/drools-camel/src/test/resources/org/drools/camel/component/jaxb.mvt?r=HEAD>
* <http://fisheye.jboss.org/browse/JBossRules/trunk/drools-camel/src/test/resources/org/drools/camel/component/jaxb.mvt?r=HEAD>
* <http://fisheye.jboss.org/browse/JBossRules/trunk/drools-camel/src/test/resources/org/drools/camel/component/xstream.mvt?r=HEAD>

## [18. Drools骆驼服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.server)

### [18.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_3)

drools骆驼服务器（drools-camel-server）模块是一种可以部署的远程执行知识库的任何类型的客户端应用程序。这不限于JVM应用程序客户端，而是任何可以通过REST接口使用HTTP的技术。此版本的执行服务器以本地方式支持无状态和有状态会话。

### [18.2。部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_deployment)

Drools Camel Server是一个war文件，可以部署在应用服务器（如JBoss AS）中。由于服务是无状态的，因此可以部署尽可能多的服务来满足客户端负载的需求。部署在JBoss AS 4.x / Tomcat 6.x上即可使用，而必须添加一些外部依赖关系，并且必须更改配置以部署在JBoss AS 5中

### [18.3。组态](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configuration)

在war文件里面你会发现一些XML配置文件。

* beans.xml中
  + 引入knowledge-services.xml和camel-server.xml的骨架XML
* 骆驼的server.xml
  + 使用骆驼路由配置CXF端点
  + 来路由管道消息到各种配置的知识服务
* 知识的services.xml
  + 各种知识库和会话
* 骆驼client.xml的
  + 示例骆驼客户端显示如何发送和接收消息
  + 由“开箱即用”test.jsp使用

#### [18.3.1。REST /骆驼服务配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rest_camel_services_configuration)

下一步是配置将通过drools-server公开的服务。您可以在camel-server.xml文件中修改此配置。

**<?xml version="1.0" encoding="UTF-8"?>**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:cxf="http://camel.apache.org/schema/cxf"

xmlns:jaxrs="http://cxf.apache.org/jaxrs"

xsi:schemaLocation="

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd

http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd

http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

<import resource="classpath:META-INF/cxf/cxf.xml" />

<import resource="classpath:META-INF/cxf/cxf-extension-jaxrs-binding.xml"/>

<import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

*<!--*

*! If you are running on JBoss you will need to copy a camel-jboss.jar into the lib and set this ClassLoader configuration*

*! http://camel.apache.org/camel-jboss.html*

*! <bean id="jbossResolver" class="org.apache.camel.jboss.JBossPackageScanClassResolver"/>*

*-->*

*<!--*

*! Define the server end point.*

*! Copy and paste this element, changing id and the address, to expose services on different urls.*

*! Different Camel routes can handle different end point paths.*

*-->*

<cxf:rsServer id="rsServer"

address="/rest"

serviceClass="org.kie.jax.rs.CommandExecutorImpl">

<cxf:providers>

<bean class="org.kie.jax.rs.CommandMessageBodyReader"/>

</cxf:providers>

</cxf:rsServer>

<cxf:cxfEndpoint id="soapServer"

address="/soap"

serviceName="ns:CommandExecutor"

endpointName="ns:CommandExecutorPort"

wsdlURL="soap.wsdl"

xmlns:ns="http://soap.jax.drools.org/" >

<cxf:properties>

<entry key="dataFormat" value="MESSAGE"/>

<entry key="defaultOperationName" value="execute"/>

</cxf:properties>

</cxf:cxfEndpoint>

*<!-- Leave this, as it's needed to make Camel "drools" aware -->*

<bean id="kiePolicy" class="org.kie.camel.component.KiePolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

*<!--*

*! Routes incoming messages from end point id="rsServer".*

*! Example route unmarshals the messages with xstream and executes against ksession1.*

*! Copy and paste this element, changing marshallers and the 'to' uri, to target different sessions, as needed.*

*!-->*

<route>

<from uri="cxfrs://bean://rsServer"/>

<policy ref="kiePolicy">

<unmarshal ref="xstream" />

<to uri="kie:ksession1" />

<marshal ref="xstream" />

</policy>

</route>

<route>

<from uri="cxf://bean://soapServer"/>

<policy ref="kiePolicy">

<unmarshal ref="xstream" />

<to uri="kie:ksession1" />

<marshal ref="xstream" />

</policy>

</route>

</camelContext>

</beans>

##### [18.3.1.1。RESTful服务端点创建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_restful_service_endpoint_creation)

在下一个XML片段代码中，我们创建一个绑定到/ kservice / rest地址的RESTful（JAX-RS）端点，并使用org.drools.jax.rs.CommandExecutorImpl作为服务实现者。此类仅用于实例化服务端点，因为所有内部实现都由Camel管理，并且您可以在源文件中看到必须永远不会调用暴露的执行服务。

还提供JAX-RS提供程序以确定是否可以在此服务端点中处理传输的消息。

<cxf:rsServer id="rsServer"

address="/rest"

serviceClass="org.kie.jax.rs.CommandExecutorImpl">

<cxf:providers>

<bean class="org.kie.jax.rs.CommandMessageBodyReader"/>

</cxf:providers>

</cxf:rsServer>

理想情况下，不需要修改此配置，至少服务类和JAX-RS提供程序，但您可以添加更多与不同地址关联的终端，以便在其他骆驼路由中使用它们。

经过所有这些初始配置后，您就可以开始配置您自己的知识服务。

##### [18.3.1.2。骆驼基耶政策和上下文创建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_camel_kie_policy_context_creation)

KiePolicy用于在骆驼中添加Drools支持，基本上它是将拦截器添加到骆驼路由中以动态创建骆驼处理器并修改内部导航路由。如果您希望获得SOAP支持，则需要创建自定义Drools策略，但它将在下一个版本中添加。

但是你不需要知道更多的内部细节，只需要实例化这个bean：

<bean id="kiePolicy" class="org.kie.camel.component.KiePolicy" />

接下来创建骆驼路由，它将负责执行通过JAX-RS发送的命令。基本上，我们创建一个与JAX-RS定义相关的路由定义作为数据输入，要使用的骆驼策略以及在“执行路径”或ProcessorDefinitions中。如您所见，我们将XStream设置为编组/解组器和drools执行路由定义

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

<route>

<from uri="cxfrs://bean://rsServer"/>

<policy ref="kiePolicy">

<unmarshal ref="xstream" />

<to uri="kie:ksession1" />

<marshal ref="xstream" />

</policy>

</route>

<route>

<from uri="cxf://bean://soapServer"/>

<policy ref="kiePolicy">

<unmarshal ref="xstream" />

<to uri="kie:ksession1" />

<marshal ref="xstream" />

</policy>

</route>

</camelContext>

drools端点创建有下一个参数

<to uri="kie:{1}/{2}" />

1. 在CamelContext中注册的执行节点标识符
2. 在标识符为{1}的执行节点中注册的知识会话标识符

这两个参数都是在knowledge-services.xml文件中配置的。

##### [18.3.1.3。知识服务配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_services_configuration)

下一步是创建您要使用的知识会话。

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:kie="http://drools.org/schema/kie-spring"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://drools.org/schema/kie-spring http://drools.org/schema/kie-spring.xsd">

<kie:kmodule id="drools-camel-server">

<kie:kbase name="kbase1" packages="org.drools.server">

<kie:ksession name="ksession1" type="stateless"/>

</kie:kbase>

</kie:kmodule>

<bean id="kiePostProcessor"

class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

</beans>

执行节点是上下文或已注册的kbases和ksessions，这里kbase1和ksession1是在node1上下文中计划的。kbase本身由两个知识定义组成，一个DRL和一个XSD。Spring文档包含更多关于配置这些知识服务的信息。

##### [18.3.1.4。测试](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_test)

将drools-server war解压后，你应该可以看到一个test.jsp并运行它。这个例子只执行一个简单的“回声”类型的应用程序。它向规则服务器发送消息，在消息前面添加单词“echo”并将其发回。默认情况下，消息是“Hello World”，可以使用url参数msg - test.jsp？msg =“My Custom Message”传递不同的消息。

在引擎盖下，jsp调用Test.java类，然后调用Camel，这是会面发生的地方。camel-client.xml仅使用几行XML定义客户端：

*<!-- Leave this, as it's needed to make Camel "drools" aware -->*

<bean id="kiePolicy" class="org.kie.camel.component.KiePolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

<route>

<from uri="direct://kservice/rest"/>

<policy ref="kiePolicy">

<to uri="cxfrs://http://localhost:8080/drools-server/kservice/rest"/>

</policy>

</route>

<route>

<from uri="direct://kservice/soap"/>

<policy ref="kiePolicy">

<to uri="cxfrs://http://localhost:8080/drools-server/kservice/soap"/>

</policy>

</route>

</camelContext>

“direct：// kservice”只是一个命名的钩子，允许Java获取引用并将数据推送到其中。在这个例子中，数据已经在XML中，所以我们不需要添加任何DataFormats来进行编组。KiePolicy为路线增加了一些智能，你也会在服务器端看到它的使用。如果使用JAXB或XStream，它将注入自定义路径和转换器，它也可以在服务器端设置ClassLoader，在客户端它会自动解开Response对象。

规则本身可以在这里找到：test.drl。注意类型Message被声明为DRL的一部分，因此不在Classpath中。

declare Message

text : String

**end**

rule "echo" dialect "mvel"

**when**

$m : Message();

**then**

$m.text = "echo:" + $m.text;

**end**

## [19.使用RHQ / JON进行JMX监控](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rhqchapter)

### [19.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_4)

Drools引擎支持通过JMX标准MBean进行运行时监控。这些MBeans将配置和指标数据从实时知识库和会话展示给内部细节，如规则执行时间。任何JMX兼容的控制台都可以用来访问这些数据。本章详细介绍如何使用RHQ / JON来完成此任务，但类似的步骤可用于任何其他控制台。

#### [19.1.1。在Drools应用程序中启用JMX监视](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_enabling_jmx_monitoring_in_a_drools_application)

要在Drools应用程序中启用JMX监视，必须在JVM中启用远程监视。有几个关于如何在互联网上做的教程，但我们建议您检查特定JVM的文档。使用Oracle / Sun JVM，它可以像运行具有几个命​​令行系统属性的引擎一样简单。

例如，要启用具有禁用身份验证的端口19988的远程监视（应仅用于测试/演示，因为应该启用生产身份验证），只需使用以下命令行参数运行应用程序：

-Dcom.sun.management.jmxremote.port=19988 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false

第二步是启用Drools MBean。与任何Drools配置一样，可以通过设置系统属性或将该属性添加到配置文件或使用API​​来完成。

要在命令行中启用它，请使用：

-Dkie.mbeans=enabled

要使用API​​启用ID，请使用：

KieBaseConfiguration conf = ...

conf.setOption( MBeansOption.ENABLED );

#### [19.1.2。安装并运行RHQ / JON插件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_and_running_the_rhq_jon_plugin)

可以使用以下步骤序列来配置JON以监视Drools应用程序：

1. 下载JON服务器和代理。
2. 下载“Drools和jBPM工具”包中包含的Drools插件（<https://www.jboss.org/drools/downloads.html>）。
3. 安装服务器，代理和插件。
4. 检查服务器是否正在运行，代理正在运行并且安装了插件。
5. 执行drools应用程序[详见上一节]。
6. 在代理控制台上，为代理键入“discovery”命令以查找在端口19988上可以找到的drools应用程序。
7. 在JON控制台上，点击自动发现队列。
8. 选择显示在那里的JMX服务器进程，运行在端口19988上。
9. 点击导入。
10. 点击资源→服务器。
11. 点击JMX服务器。
12. 在左侧的JMXServer下，您有Drools服务。

# Drools Workbench

Drools工作台使用UberFire框架构建，并使用Guvnor插件。Drools为规则创作隐喻提供了额外丰富的插件集。

## [20.工作台（一般）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbench)

### [20.1。安装](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.installation)

#### [20.1.1。战争安装](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.warinstallation)

使用war与您的应用程序服务器对应的工作台分发zip文件。这些war文件之间的差异主要是肤浅的。例如，如果应用程序服务器已经提供了一些JAR，则可能会排除这些JAR。

* eap7：针对红帽JBoss企业应用平台7量身定制
* tomcat8：为Apache Tomcat 8量身打造

|  |  |
| --- | --- |
|  | Apache Tomcat需要额外的配置才能正确安装Workbench。请咨询README.md在war为最最新的程序。 |

* wildfly10：为Red Hat JBoss Wildfly 10量身定制

#### [20.1.2。工作台数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchdata)

工作台将其数据存储，默认的目录$WORKING\_DIRECTORY/.niogit，例如wildfly-10.0.0.Final/bin/.niogit，但它可以与覆盖[系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)-Dorg.uberfire.nio.git.dir。

|  |  |
| --- | --- |
|  | 在生产中，请确保备份工作台数据目录。 |

#### [20.1.3。系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)

以下是所有系统属性的列表：

* org.uberfire.nio.git.dir：目录的位置.niogit。默认：工作目录
* org.uberfire.nio.git.dirname：git目录的名称。默认：.niogit
* org.uberfire.nio.git.daemon.enabled：启用/禁用git守护进程。默认：true
* org.uberfire.nio.git.daemon.host：如果启用git守护程序，则使用此属性作为本地主机标识符。默认：localhost
* org.uberfire.nio.git.daemon.port：如果启用git守护进程，则使用此属性作为端口号。默认：9418
* org.uberfire.nio.git.ssh.enabled：启用/禁用ssh守护进程。默认：true
* org.uberfire.nio.git.ssh.host：如果启用了ssh守护程序，则使用此属性作为本地主机标识符。默认：localhost
* org.uberfire.nio.git.ssh.port：如果启用了ssh守护程序，则使用此属性作为端口号。默认：8001
* org.uberfire.nio.git.ssh.cert.dir：.security将存储本地证书的目录的位置。默认：工作目录
* org.uberfire.nio.git.ssh.passphrase：在git使用scp样式URL 克隆存储库时通过密码访问您的操作系统公用密钥存储; 例如git@github.com:user/repository.git。
* org.uberfire.nio.git.ssh.algorithm：SSH使用的算法。默认：DSA

|  |  |
| --- | --- |
|  | 如果您打算使用RSA或DSA以外的算法，请确保正确设置您的应用程序服务器以使用Bouncy Castle JCE库。 |

* org.uberfire.metadata.index.dir：放置Lucene .index文件夹的地方。默认：工作目录
* org.uberfire.ldap.regex.role\_mapper：用于将LDAP主体名称映射到应用程序角色名称的正则表达式模式。请注意，变量role必须是模式的一部分，因为在将主体值与角色名称匹配时，该变量被应用程序角色名称所取代。默认：未使用。
* org.uberfire.sys.repo.monitor.disabled：禁用配置监视器（不要禁用，除非你知道你在做什么）。默认：false
* org.uberfire.secure.key：密码加密使用的密码。默认：org.uberfire.admin
* org.uberfire.secure.alg：密码加密使用的加密算法。默认：PBEWithMD5AndDES
* org.uberfire.domain：uberfire使用的安全域名。默认：ApplicationRealm
* org.guvnor.m2repo.dir：放置Maven存储库文件夹的地方。默认值：working-directory / repositories / kie
* org.guvnor.project.gav.check.disabled：禁用GAV检查。默认：false
* org.kie.demo：从GitHub启用演示应用程序的外部克隆。
* org.kie.build.disable-project-explorer：在Project Explorer中禁用自动构建所选项目。默认：false
* org.kie.verification.disable-dtable-realtime-verification：禁用决策表的实时验证和验证。默认：false
* org.kie.workbench.controller：用于连接Kie服务器控制器的URL，例如：ws://localhost:8080/kie-server-controller/websocket/controller。

只有Web Socket协议才支持与远程Kie服务器控制器连接。指定此比例时，Workbench将自动禁用与运行嵌入式控制器相关的所有功能。

* org.kie.workbench.controller.user：用于与Kie服务器控制器连接的用户名。默认：kieserver
* org.kie.workbench.controller.pwd：用于连接Kie服务器控制器的密码。默认：kieserver1!
* org.kie.workbench.controller.token：用于连接Kie服务器控制器的令牌字符串。

有关如何使用基于[令牌的身份验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#usingTokenBasedAuthentication)的详细信息，请参阅[使用基于](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#usingTokenBasedAuthentication)令牌的身份验证。

* kie.keystore.keyStoreURL：应该用于连接远程Kie服务器控制器的密钥库的URL。
* kie.keystore.keyStorePwd：密码到密钥库。
* kie.keystore.key.ctrl.alias：存储密码的密钥的别名。
* kie.keystore.key.ctrl.pwd：存储密码的别名密码。

请参阅[使用密钥存储保护密码以](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_password_using_key_store)获取有关如何[使用密钥存储保护密码](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_password_using_key_store)的更多详细信息。

要在WildFly或JBoss EAP集群中更改其中一个系统属性，请执行以下操作：

1. 编辑文件$JBOSS\_HOME/domain/configuration/host.xml。
2. 找到server属于该main-server-group系统的XML元素并添加系统属性，例如：
3. <system-properties>
4. <property name="org.uberfire.nio.git.dir" value="..." boot-time="false"/>
5. ...

</system-properties>

#### [20.1.4。故障排除](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.troubleshooting)

##### [20.1.4.1。Loading ..不会消失，Workbench无法显示](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.troubleshootingloadingspinner)

有报告称，服务器和浏览器之间的防火墙可能会干扰Workbench使用的服务器发送事件（SSE）。

此问题导致“正在加载...”微调器保持可见状态，工作台未能实现。

解决方法是通过将文件添加/WEB-INF/classes/ErraiService.properties到包含值的分解WAR中，禁用Workbench使用Server Sent Events errai.bus.enable\_sse\_support=false。重新打包WAR并重新部署。

某些用户也报告禁用服务器发送事件不能解决问题。发现的解决方案是将JVM配置为在Linux上使用不同的Entropy收集设备SecureRandom。这可以通过将系统属性设置java.security.egd为file:/dev/./urandom。有关 详细信息，请参阅[此](http://stackoverflow.com/questions/33166198/kie-workbench-not-loading-after-login/39110177#39110177)堆栈溢出文章。

但请注意，这会影响JVM的随机数生成，并且在需要强大的加密技术时可能会带来其他挑战。谨慎配置。

##### [20.1.4.2。无法使用ssh协议克隆KIE Workbench Git存储库。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_not_able_to_clone_kie_workbench_git_repository_using_ssh_protocol)

使用ssh与和Workbench捆绑在一起的Git服务器进行交互的Git客户端通过身份验证并被授权通过作为Uberfire后端服务器一部分的安全API执行git命令。在使用LDAP安全领域时，一些git客户端没有按预期授权。这是因为，对于非Web客户端（例如通过ssh的Git），由应用程序服务器的用户注册表分配给用户的主体（即用户或组）名称是LDAP与该主体关联的更复杂的DN。Uberfire后端服务器的逻辑查找所返回的主体名称允许的角色完全匹配，因此失败。

现在可以通过系统属性来控制角色主体匹配

org.uberfire.ldap.regex.role\_mapper

将LDAP主体与角色名称匹配时应用的正则表达式模式的值。该模式必须包含文字变量“角色”。在授权期间，变量被每个允许的应用程序角色取代。如果模式匹配，角色将添加到用户。

例如，如果LDAP中管理组的DN是

DN: cn=admin,ou=groups,dc=example,dc=com

其预期的作用是管理员，然后设定org.uberfire.ldap.regex.role\_mapper值

cn[\\ ]\*=[\\ ]\*role

会在角色'admin'上找到匹配。

### [20.2。快速开始](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstart)

这些步骤可帮助您以最小的努力开始工作。

它们不应该成为完整阅读文档的替代品。

#### [20.2.1。导入示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartimportexamples)

*导入示例 - 快速安装示例*

如果工作台为空，则显示空白页面。点击下面的“试用样品”按钮将显示可用的示例。

“试用样品”页面打开后，您可以选择一个或多个示例并单击“确定”。

如果工作台已包含项目，则可以使用菜单中的“尝试示例”按钮导入示例。

#### [20.2.2。添加项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartaddproject)

或者，要导入示例，可以使用“添加项目”从空间页面创建一个新的空项目。

*图202.新建项目按钮*

给项目一个名称和可选的描述。

*图203.给项目一个名字*

#### [20.2.3。定义数据模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartdefinedatamodel)

项目创建完成后，您需要定义规则使用的类型。

从“创建新资产”菜单中选择“数据对象”。

|  |  |
| --- | --- |
|  | 您还可以使用现有JAR中包含的类型。  有关详细信息，请参阅完整文档。 |

*图204.创建“数据对象”*

设置名称并为新类型选择一个包。

*图205.创建一个新类型*

单击“+添加字段”按钮并设置字段名称和类型，然后单击“创建”为该类型创建一个字段。

*图206.单击“创建”并添加该字段*

点击“保存”更新模型。

*图207.单击“保存”*

#### [20.2.4。定义规则](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartdefinerule)

从“创建新资产”菜单中选择“DRL文件”（例如）。

*图208.从“创建新资产”菜单中选择“DRL文件”*

输入新规则的文件名。

|  |  |
| --- | --- |
|  | 确保你选择了与规则相同的软件包。在不同的包中可以有规则和数据模型，但为了演示的目的，让我们保持简单。 |

*图209.输入规则的文件名*

输入规则的定义。

定义过程因资产类型不同而不同。

完整的文档详细介绍了不同的编辑。

*图210.定义一条规则*

一旦定义了规则，就需要以我们保存模型的相同方式进行保存。

#### [20.2.5。构建和部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.quickstartbuildanddeloy)

一旦在项目中定义了规则，该项目可以构建并部署到Workbench的Maven Artifact Repository中。

要构建项目，请从项目创作中选择“构建和部署”。

*图211.建立一个项目*

点击“Build＆Deploy”构建项目并将其部署到Workbench的Maven Artifact Repository中。

当您选择构建和部署时，工作台将部署到工作台项目中的POM的依赖管理部分中定义的任何存储库。您可以在项目浏览器的Repository视图下编辑与工作台项目关联的pom.xml文件。有关Maven中依赖关系管理的详细信息，请访问：[http](http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html) : [//maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html](http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html)

如果在构建过程中出现错误，它们将在“消息”面板中报告。

现在该项目已经建成并部署完毕; 它可以像你其他的Maven Artifact一样从你自己的项目中引用。

完整的文档包含有关将项目与自己的应用程序集成的详细信

### [20.3。组态](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.configuration)

#### [20.3.1。基本的用户管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.usermanagement)

工作台根据应用程序服务器的身份验证和授权（JAAS）对其用户进行身份验证。

在JBoss EAP和WildFly上，添加一个用户脚本$JBOSS\_HOME/bin/add-user.sh（或.bat）：

$ ./add-user.sh

*// Type: Application User*

*// Realm: empty (defaults to ApplicationRealm)*

*// Role: admin*

不需要重新启动应用程序服务器。

#### [20.3.2。角色](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.roles)

工作台使用以下角色：

* 管理
* 分析人士
* 开发人员
* 经理
* 用户

##### [20.3.2.1。管理员](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_admin)

管理BPMS系统。

* 管理用户
* 管理VFS存储库
* 有完全权限进行必要的更改

##### [20.3.2.2。开发人员](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_developer)

开发人员可以完成管理员可以执行的任何操作，但克隆存储库除外。

* 管理规则，模型，流程，表单和仪表板
* 管理资产存储库
* 可以创建，构建和部署项目
* 可以使用JBDS连接查看进程

##### [20.3.2.3。分析人士](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_analyst)

分析师是开发人员较弱的版本，无法访问资产存储库或部署项目的能力。

##### [20.3.2.4。商业用户](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_business_user)

系统的每日用户对流程继续前进所需的业务任务采取行动。主要与任务列表一起工作。

* 进程管理
* 处理任务和仪表板

##### [20.3.2.5。经理/查看器用户](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_manager_viewer_only_user)

系统的查看者对业务流程及其性能，业务指标以及与系统交互的系统和人员的其他报告相关的统计信息感兴趣。

* 只能访问仪表板

### [20.4。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.introduction)

#### [20.4.1。登录并注销](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.loginandlogout)

使用角色创建一个用户admin并使用这些凭证登录。

成功登录后，帐户用户名将显示在右上角。点击它查看当前帐户的角色。

#### [20.4.2。主屏幕](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.homescreen)

登录后，显示主屏幕。主屏幕的实际内容取决于工作台变体（Drools，jBPM，...）。

#### [20.4.3。工作台概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.administrationoverview)

工作台由空间和项目组成：

##### [20.4.3.1。空间](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.space)

空间对建模部门和部门很有用。

一个空间可以容纳多个项目。

##### [20.4.3.2。项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.project)

项目是资产存储的地方，每个项目都属于一个空间。

项目实际上是基于虚拟文件系统的存储，默认情况下使用GIT作为后端。这种设置允许工作台与多个后端一起工作，并且同时充分利用后端特定功能，如GIT案例版本控制，分支甚至外部访问。

一个新的项目可以从头开始创建或从现有仓库克隆。

使用GIT作为后端的最大优势之一是可以从外部克隆存储库并使用您首选的工具编辑和构建您的资产。

|  |  |
| --- | --- |
|  | **切勿**直接从[.niogit](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)目录中克隆存储库。 |

#### [20.4.4。Workbench用户界面概念](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchconcepts)

工作台由不同的逻辑实体组成：

* 部分

零件是用户可以与之交互以执行操作的屏幕或编辑器。

示例部分是“项目浏览器”，“项目编辑器”，“引导规则编辑器”等。

* 页

透视图是相关面板和部件的逻辑分组。透视图通常被命名为页面，因为它对于最终用户来说是一个更为熟悉的术语，而透视更多地是面向开发人员的。但是请注意，Workbench既支持开发者创建的页面，也支持来自页面构建器（又名内容管理）工具的最终用户创建的页面，但一般而言，页面用于引用两者。

用户可以通过点击其中一个顶层菜单项来切换页面; 如“家”，“创作”，“部署”等。

### [20.5。更改布局](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.changinglayout)

#### [20.5.1。调整](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.resizing)

将鼠标指针移到面板分离器上（面板之间的灰色水平或垂直线）。

光标将通过改变指示它正确地位于分离器上方。按住鼠标左键并将分离器拖动到所需的位置; 然后释放鼠标左键。

### [20.6。创作（一般）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.authoring)

#### [20.6.1。工件库](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.artifactrepository)

项目通常需要在其类路径中使用外部工件，以便构建（例如）域模型JAR。工件存储库包含这些工件。

工件库是一个完整的Maven仓库。它遵循Maven远程仓库的语义：所有快照都有时间戳。但它通常存储在本地硬盘上。

默认情况下，工件存储库存储在下$WORKING\_DIRECTORY/repositories/kie，但可以使用[系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)覆盖它-Dorg.guvnor.m2repo.dir。每次安装只有1个Maven存储库。

工件存储库屏幕显示Maven存储库中工件的列表：

要将新的工件添加到该Maven存储库，请执行以下操作之一：

* 使用上传按钮并选择一个JAR。如果JAR包含一个POM文件META-INF/maven（Maven构建的每个JAR都有），则不需要进一步的信息。否则，还需要给出groupId，artifactId和版本。
* 使用Maven，mvn deploy到该Maven仓库。刷新列表以使其显示。

|  |  |
| --- | --- |
|  | 这个远程Maven仓库相对简单。它不支持代理，镜像，...像Nexus或Archiva。 |

#### [20.6.2。资产编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.asseteditor)

资产编辑器是工作台用户界面的主要组成部分。它由两个主视图编辑器和概述组成。

* 观点

*图212.资产编辑器 - 编辑器选项卡*

* + 答：编辑区域 - 编辑器所采取的形式取决于资产类型。资产一次只能由一位用户编辑以避免冲突。当用户开始编辑资产时，会自动获取锁定。这由资产标题栏以及[项目资源管理器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorer)视图中显示的锁定符号指示（请参阅[Project Explorer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorer)了解详情）。如果用户开始编辑已锁定的资产，则会弹出通知，通知用户该资产目前无法编辑，因为其他用户正在处理该资产。在编辑用户保存或关闭资产或退出工作台之前，更改将被阻止。会话超时也会导致锁定被释放。如果需要，每个用户还可以选择强制释放锁（请参阅下面的元数据部分）。
  + B：此菜单栏包含资产的各种操作; 如保存，重命名，复制等。请注意，如果资产被其他用户锁定，则保存，重命名和删除操作将被停用。
  + C：资产内容或资产信息的不同观点。
    - 编辑器显示资产的主编辑
    - 概述包含此编辑器的元数据和对话视图。下面更详细地解释。
    - 信息来源以简单的DRL显示资产。注意：此选项卡仅在资产内容可生成DRL时才可见。
    - 数据对象包含可用于创作的模型。默认情况下，只有与资产位于同一包中的数据对象才可用于创作。可以导入此包之外的数据对象，以便可用于创建资产。

*图213.资产编辑器 - 数据对象选项卡*

* 概观
  + 答：关于资产和资产描述的一般信息。

“类型：”资产类型的格式名称。

“描述：”资产的描述。

“在项目中使用：”命名使用此规则的项目。

“上次修改时间：”谁做出了最后一次更改和何时。

“创建日期：”谁创建了资产和何时创建。

* + B：资产的版本历史记录。选择版本将选定的版本加载到此编辑器中。
  + C：元数据（来自“都柏林核心”标准）
  + D：关于资产开发的评论可以在这里记录下来。

。资产编辑器 - 概述选项卡image :: Workbench / Authoring / AssetEditor / Overview.png [align =“center”]

* 元数据
  + 答：元数据： -

“标签：”用于分组资产的标签系统。

“注意：”资产上次更新时发表的评论（即为什么进行更改）

“URI：”指向Git存储库内的资产的URI。

“主题/类型/外部链接/源”：资产的其他杂项元数据。

“锁定状态”：显示资产的锁定状态，如果锁定，则允许强制解锁资产。

* 。元数据标签图像:: Workbench / Authoring / AssetEditor / Metadata.png [align =“center”]
* 锁定

Workbench支持资产的悲观锁定。当一个用户开始编辑资产时，它被锁定以被其他用户更改。锁定一直保持到一段时间不活动，编辑器关闭或应用程序停止并重新启动。也可以在概览选项卡的元数据部分强制删除锁。

当资产被锁定时，编辑器的标题栏和项目资源管理器中的资产旁边会显示“挂锁”图标。

*图214.资产编辑器 - 锁定的资产不能被其他用户编辑*

#### [20.6.3。标签编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.tagseditor)

标签允许资产标有您定义的任意数量的标签。这些标签可用于过滤Project Explorer中的资产，以启用“标签过滤”。

##### [20.6.3.1。创建标签](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.managingtags)

要创建标签，您只需将它们写在标签输入上，然后按下“添加新标签”按钮即可。标签编辑器允许逐个创建标签，或者使用空格分隔多个标签。

*图215.创建标签*

一旦你创建了新的标签，它们将出现在编辑器上，允许你按需要删除它们。

*图216.现有标签*

#### [20.6.4。项目浏览器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorer)

Project Explorer提供了浏览当前Project内文件的功能。当资源编辑器处于打开状态时，可从左侧访问Project Explorer。

##### [20.6.4.1。初始视图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorerempty)

如果其他用户正在编辑某个文件，则锁定符号将显示在文件名称的前面。该符号为蓝色，以防该锁由当前经过身份验证的用户拥有，否则为黑色。将鼠标指针移到锁定符号上将显示一个工具提示，提供当前正在编辑文件的用户的名称（因此拥有该锁定）。要了解更多关于锁定的信息，请参阅[资产编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.asseteditor) 以获取详细信

*图217.扩大的资产组*

##### [20.6.4.2。不同的意见](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorerswitchingview)

Project Explorer支持多个视图。

* 项目视图

基础项目结构的简化视图。某些系统文件从视图中隐藏。

* 存储库视图

基础项目结构的完整视图，包括所有文件; 用户定义的或系统生成的。

单击Project Explorer中的图标可以选择视图，如下所示。

通过选择“显示为文件夹”或“显示为链接”，可以进一步改进项目和存储库视图。

*图218.切换视图*

###### [存储库查看示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorertechnicalview)

*图219.储存库视图 - 文件夹*

*图220.库视图 - 链接*

##### [20.6.4.3。下载项目或存储库](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorerdownloads)

下载下载和下载存储库可以将项目或存储库作为zip文件下载。

*图221.存储库和项目下载*

##### [20.6.4.4。按标签过滤](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorerfilteringbytag)

为了便于查看包含大量资产的包，可以启用标签过滤器，这可以让您通过标签过滤资产。

要查看如何向资产添加标签，请看：[标签编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.tagseditor)

*图222.启用按标签过滤*

*图223.按标签过滤*

*图224.按标签过滤*

##### [20.6.4.5。复制，重命名，删除和下载操作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorercopyrenamedeleteactions)

复制，重命名和删除操作在链接模式下可用，适用于项目视图中的包和存储库视图中的文件和目录。下载操作可用于目录。下载选项将选定的目录下载为zip文件。

* 复印件
* B：重命名
* C：删除
* D：下载

*图225.项目视图 - 程序包操作*

|  |  |
| --- | --- |
|  | 工作台路线图包括重构和影响分析工具，但是很少有。在提供这两种工具之前，请确保您对包，文件或目录的更改（复制/重命名/删除）不会对您的项目产生重大影响。  如果您的更改产生不可预测的影响，Workbench允许您使用[Repository编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.vfsrepositoryeditor)恢复您的存储[库](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.vfsrepositoryeditor)。 |
|  | 其他用户锁定的文件以及包含这些文件的目录不能重命名或删除，直到释放相应的锁。如果是这种情况，重命名和删除符号将被停用。要了解更多关于锁定的信息，请参阅[资产编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.asseteditor) 以获取详细信 |

#### [20.6.5。项目编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projecteditor)

项目编辑器屏幕可以从项目浏览器访问。项目编辑器显示当前活动项目的设置。

与大多数工作台编辑器不同，项目编辑器编辑多个文件。显示在一个地方配置KIE项目所需的一切。

*图226.项目屏幕和不同的视图*

##### [20.6.5.1。构建和部署](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_build_deploy)

Build＆Depoy构建当前项目并将KJAR部署到工作台内部Maven存储库中。

##### [20.6.5.2。项目设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_settings)

项目设置编辑Maven使用的pom.xml文件。

###### [项目常规设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_general_settings)

常规设置为项目名称和GAV数据（组，工件，版本）提供工具。GAV值用作区分同一项目的项目和版本的标识符。

*图227.项目设置*

###### [依赖](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dependencies)

该项目可能有任何数量的内部或外部依赖项。依赖项是一个已经构建并部署到Maven存储库的项目。内部依赖项是与项目相同的工作台构建和部署的项目。从当前工作台以外的存储库检索外部依赖关系。每个依赖项都使用GAV值来指定项目使用的项目名称和版本。

*图228.依赖关系*

[包名白名单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_package_name_white_list)

白名单软件包中的类和声明类型显示为可以导入到资产中的数据对象。完整列表存储在存储在每个项目根目录下的package-name-white-list文件中。

包白名单有三种模式：

* 所有软件包包括：此jar中定义的每个软件包都是白名单。
* 不包括的软件包：此jar中列出的软件包均未列出白名单。
* 包括一些软件包：只有部分jar包中的软件包是白名单。

###### [元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_metadata)

pom.xml文件的元数据。

##### [20.6.5.3。知识库设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_base_settings)

知识库设置编辑Drools使用的kmodule.xml文件。

*图229.知识库设置*

|  |  |
| --- | --- |
|  | 有关知识库属性的更多信息，请查看kmodule.xml的Drools Expert文档。 |

###### [知识库和会话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_bases_and_sessions)

知识库和会议列出了为项目指定的知识库和知识会议。

[知识库列表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_base_list)

按名称列出所有知识库。只有一个知识库可以设置为默认值。

[知识库属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_base_properties)

知识库可以包含其他知识库。包含的知识库中的模型，规则和任何其他内容将由当前选定的知识库可见和可用。

规则和模型存储在包中。packages属性指定包含在这个知识库中的包。

等同行为在文档的Drools Expert部分中进行了解释。

事件处理模式在文档的Drools Fusion部分进行了说明。

[知识会议](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_knowledge_sessions)

该表列出了所选知识库中的所有知识会议。每种类型只能有一个默认值。这些类型是无状态和有状态的。点击笔形图标打开一个弹出窗口，显示知识会话的更多属性。

###### [元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_metadata_2)

kmodule.xml的元数据

##### [20.6.5.4。进口](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_imports)

设置编辑工作台编辑器使用的project.imports文件。

*图230.导入*

###### [外部数据对象](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_external_data_objects)

由Java运行时环境提供的数据对象可能需要注册才能用于规则创作，其中这些数据对象不是作为在工作台或项目依赖项中定义的现有数据对象的一部分隐式可用的。例如，作者可能想要定义一个检查java.util.ArrayList工作内存的规则。如果一个域的数据对象有一个类型的字段，java.util.ArrayList则不需要创建一个注册表。

###### [元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_metadata_3)

project.imports文件的元数据。

##### [20.6.5.5。重复的GAV检测](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_duplicate_gav_detection)

在执行以下任何操作时，现在针对项目已解决的所有Maven存储库进行检查，以确定项目的GroupId，ArtifactId和Version是否预先存在。如果发现冲突，则防止操作; 尽管这可以由admin角色的用户覆盖。

|  |  |
| --- | --- |
|  | 该功能可以通过系统属性设置来禁用org.guvnor.project.gav.check.disabled到true。 |

已解决的存储库是在以下位置发现的存储库： -

* 项目的部分（或任何父母）。POM<repositories>POM
* 该项目的部分。POM<distributionManagement>
* Maven的全局settings.xml配置文件。

受影响的操作： -

* 创建新的管理存储库。
* 使用项目编辑器保存项目定义。
* 将新模块添加到托管多模块存储库。
* 保存pom.xml文件。
* 使用项目编辑器构建和安装项目。
* 使用项目编辑器构建和部署项目。
* 资产管理操作构建，安装或部署项目。
* REST 创建，安装或部署项目的操作。

具有该Admin角色的用户可以覆盖使用项目编辑器中的“存储库”设置检查的存储库列表。

*图231.项目编辑器 - 查看解析的存储库*

*图232.项目编辑器 - 已解析的存储库列表*

*图233.检测到重复的GAV*

#### [20.6.6。验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.validation)

Workbench为用户提供了一个通用且一致的服务，用于了解环境中创建的文件是否有效。

##### [20.6.6.1。问题面板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.problemspanel)

问题面板显示项目中资产的实时验证结果。

从项目浏览器中选择项目时，问题面板将刷新所选项目的验证结果。

在创建，保存或删除文件时，问题面板内容将更新，以显示新的验证错误，或删除文件时删除现有文件。

*图234.问题面板*

##### [20.6.6.2。按需验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.ondemandvalidation)

为了确定文件是否处于有效状态，并不总是需要保存文件。

所有文件编辑器都可以在保存内容之前验证内容。

点击“Validate”按钮显示验证错误，如果有的话。

#### [20.6.7。数据建模器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datamodeller)

##### [20.6.7.1。首先要创建一个数据模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sect_datamodeler_firststeps)

默认情况下，数据模型始终受限于项目的上下文。为了本教程的目的，我们将假定正确配置的项目已经存在，并且创作页面已打开。

要开始在项目中创建数据模型，请执行以下步骤：

1. 在主屏幕中，选择设计页面并选择给定的项目。

*图235.转到创作页面并选择一个项目*

1. 单击数据对象文件或使用“创建新资产→数据对象”菜单选项打开数据建模器工具。将数据对象名称设置为“PurchaseOrder”并单击确定。

*图236.点击一个数据对象*

这将启动Data Modeller工具，它具有以下一般方面：

*图237.数据建模器概述*

“编辑器”选项卡分为以下几部分：

* 新字段部分专门用于创建新字段，并在按下“添加字段”按钮时打开。

*图238.新的字段创建*

* 数据对象的“字段浏览器”部分显示包含数据对象字段的列表。

*图239.数据对象的字段浏览器*

* “数据对象/字段常规属性”部分。这是Data Modeller编辑器的最右侧部分，根据用户选择可视化“数据对象”或“字段”常规属性。

数据对象的一般属性可以通过点击数据对象选择器来选择。

*图240.数据对象选择器*

*图241.数据对象的一般属性*

可以通过单击字段来选择字段常规属性。

*图242.字段选择器*

*图243.字段常规属性*

* 在工作台的右侧，提供了一个新的“工具栏”，可以选择不同的上下文相关工具窗口，以便用户执行特定于域的配置。目前，为以下域“Drools＆jBPM”，“OptaPlanner”，“持久性”和“高级”配置提供了四个工具窗口。

*图244.数据建模器工具栏*

*图245. Drools和jBPM工具窗口*

*图246. OptaPlanner工具窗口*

|  |  |
| --- | --- |
|  | 要查看和使用OptaPlanner工具窗口，用户需要具有角色plannermgmt。 |

*图247.持久工具窗口*

*图248.高级工具窗口*

“源”选项卡显示一个编辑器，它允许生成的Java代码的可视化和修改。

* “编辑器”和“源”标签之间的往返行程是可能的，并且还提供源代码保存。这意味着无论Java代码在何处生成（例如Eclipse，Data建模器），数据建模器都将只更新必要的代码块以保持模型更新。

*图249.源代码编辑器*

“概览”选项卡将标准元数据和版本信息显示为其他工作台编辑器。

##### [20.6.7.2。数据对象](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sect_datamodeler_entities)

数据模型由数据对象组成，这些数据对象是一些真实世界数据的逻辑表示。这样的数据对象具有一组固定的建模器（或应用程序拥有的）属性，例如其内部标识符，标签，描述，包等。除此之外，数据对象还具有变量集的用户定义字段，它们是该逻辑数据对象表示的数据类型的现实世界属性的抽象。

可以使用工作台“New Item - Data Object”菜单选项来创建数据对象。

*图250.“新建数据对象”菜单选项*

资源名称和位置都是强制参数。当按下“确定”按钮时，将创建一个新的Java文件并为文件版本打开一个新的编辑器实例。可选的“Persistable”属性将在数据对象上添加默认配置，以使其成为JPA实体。如果您的jBPM项目需要将数据对象的信息存储在数据库中，请使用此选项。

##### [20.6.7.3。属性和关系](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sect_datamodeler_propertyrelationships)

一旦数据对象被创建，它现在必须通过向其定义添加用户定义的属性来完成。这可以通过按下“添加字段”按钮来实现。“新建字段”对话框将打开，新字段可通过按下“创建”按钮创建。“创建并继续”按钮也会将新字段添加到数据对象，但不会关闭对话框。通过这种方式可以创建多个字段，避免多次弹出打开。以下字段可以（或必须）填写：

* 该字段的内部标识符（必填）。该字段的值必须对每个数据对象都是唯一的，即如果建议的标识符已经存在于当前数据对象中，则会显示错误消息。
* 标签（可选）：与数据对象定义一样，用户可以为即将创建的数据对象字段定义用户友好的标签。这对于如何处理来自该数据对象的对象的字段没有进一步的影响。如果定义了标签，那么这就是该字段在整个数据建模器工具中的显示方式。
* 字段类型（必填）：每个数据对象字段需要分配一个类型。

此类型可以是以下任一种类型：

* 1. 一个'原始java对象'类型：它们包括大部分标准Java基本类型的对象等价物，比如布尔型，Short型，Float型等，以及String，Date，BigDecimal和BigInteger。

*图251.原始对象字段类型*

* 1. 一个'数据对象'类型：任何用户定义的数据对象自动成为一个候选者，被定义为另一个数据对象的一个​​字段类型，从而能够创建它们之间的关系。数据对象字段可以'单'或'多'形式创建，后者意味着字段将被定义为这种类型的集合，这将通过选择“列表”复选框来指示。

*图252.数据对象字段类型*

1. 一种'原始java'类型：这些类型包括java基本类型byte，short，int，long，float，double，char和boolean。

*图253.原始字段类型*

完成介绍新字段的初始信息后，单击“创建”按钮将新创建的字段添加到下面的数据对象字段表的末尾：

*图254.新字段已创建*

新字段也将自动在数据对象的字段列表中选择，其属性将显示在字段常规属性编辑器中。此外，字段属性将加载到不同的工具窗口中，这样该字段就可以在任何选定的工具窗口中进行编辑。

在任何时候，通过单击数据对象字段表中相应的'x'图标，可以从数据对象定义中删除任何字段（没有限制）。

##### [20.6.7.4。其他选项](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sect_datamodeler_additionaloptions)

如前所述，数据对象以及字段都需要在创建时设置它们的一些初始属性。此外，还可以为给定的数据对象配置三个属性域。域基本上是一组与特定业务领域相关的属性。当前可用的域名是“Drools＆jJBPM”，“持久性”和“高级”域名。要在给定域上工作，用户应该在右侧工具栏上选择相应的“工具窗口”（见下文）。每个工具窗口通常都会提供两个编辑器，即“数据对象”级别编辑器和“字段”级编辑器，这些编辑器将根据最后选择的项目，数据对象或字段显示。

###### [Drools＆jBPM域名](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools_jbpm_domain)

Drools＆jBPM域编辑器管理与drools应用程序相关的一组数据对象或字段属性。

[Drools＆jBPM对象编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools_jbpm_object_editor)

Drools＆jBPM对象编辑器管理对象级别的drools属性

*图255.数据对象的属性*

* TypeSafe：此属性允许启用/禁用当前类型的类型安全行为。默认情况下，所有类型声明都是在启用类型安全的情况下编译的 （有关此事的更多信息，请参阅Drools）。
* ClassReactive：此属性允许将此类型标记为Drools引擎的“Reactive类”。（有关此事的更多信息，请参阅Drools）。
* PropertyReactive：此属性允许将此类型标记为Drools引擎的“Property Reactive”。（有关此事的更多信息，请参阅Drools）。
* 作用：这个属性允许配置Drools引擎应该如何处理这种类型的实例：或者作为常规事实，或者作为事件。默认情况下，所有类型都是作为一个常规事实来处理的，所以暂时唯一可以设置的值是“Event”来声明这个类型应该作为一个事件来处理。（有关此事的更多信息，请参阅Drools Fusion）。
* 时间戳：该属性允许通过选择他的一个属性来配置事件的“时间戳”。如果设置引擎将使用来自给定属性的时间戳，而不是从会话时钟中读取它。否则，引擎会自动为事件分配一个时间戳。（有关此事的更多信息，请参阅Drools Fusion）。
* 持续时间：该属性允许通过选择他的一个属性来配置事件的“持续时间”。如果设置，引擎将使用来自给定属性的持续时间，而不是使用默认事件持续时间= 0。（有关此问题的更多信息，请参阅Drools Fusion）。
* 过期：此属性允许为事件过期配置“时间偏移”。如果设置，该值必须是以下形式的时间间隔：[d] [＃H] [＃米] [＃s]的[[ms]]其中[]表示可选参数，＃表示数值。例如：1d2h，意味着一天两小时。（有关此事的更多信息，请参阅Drools Fusion）。
* Remotable：如果选中此属性，则可以将数据对象与jBPM远程服务配合使用，如REST，JMS和WS。（有关此事的更多信息，请参阅jBPM）。

[Drools＆jJBPM现场编辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools_jjbpm_field_editor)

Drools＆jBPM对象编辑器管理字段级别的drools属性

*图256.数据对象的字段属性*

* 等于：检查数据对象字段的这个属性意味着它将在代码生成级别被考虑用于在生成的Java类中创建equals（）和hashCode（）方法。我们将在下一节更详细地解释这一点。
* 位置：该字段需要一个零或正整数。设置时，该字段将由Drools引擎解释为位置参数（请参阅下面的部分以及有关此主题的更多信息的Drools文档）。

###### [持久性域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_persistence_domain)

持久性域编辑器管理与持久性相关的一组数据对象或字段属性。

[持久性域对象编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_persistence_domain_object_editor)

持久域对象编辑器管理对象级持久性属性

*图257.数据对象的属性*

* 持久性：此属性允许将当前数据对象配置为可持久化。
* 表名称：此属性允许为当前数据对象设置用户定义的数据库表名称。

[持久性域字段编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_persistence_domain_field_editor)

持久域字段编辑器管理字段级别的持久性属性，并分为三部分。

*图258.持久性域字段编辑器部分*

[标识符：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_identifier)

持久数据对象应该只有一个字段被定义为数据对象标识符。标识符通常是一个唯一的数字，用于将给定的数据对象实例与同一类的所有其他实例进行区分。

* 是标识符：将当前字段标记为数据对象标识符。持久数据对象应该只有一个标记为标识符的字段，它应该是一个基本的java类型，如String，Integer，Long等。引用数据对象的字段或者是多字段的字段不能被标记作为标识符。此版本中不支持复合标识符。创建持久数据对象时，默认情况下会创建一个标识符字段，并强制进行初始化，强烈建议使用此标识符。
* 生成策略：生成策略确定如何在创建数据对象实例并将其存储在数据库中时自动生成标识符值。（例如，通过与jBPM处理人工任务相关的表单）。当默认标识符字段被创建时，生成策略也会自动设置，强烈建议使用此配置。
* 序列生成器：生成器表示生成策略将使用的值的种子。当默认标识符字段被创建时，序列生成器也将自动生成并被正确配置以供生成策略使用。

[列属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_column_properties)

列属性部分允许定制将存储字段值的数据库列的某些属性。

* 列名称：为给定字段设置数据库列名称的可选值。
* 唯一：如果选中，则唯一属性确定当前字段值应存储在数据库中时是唯一键。（如果未设置，默认值为false）
* 可空：当选中时确定当前字段值在存储在数据库中时可以为空。（如果没有设置默认值是true）
* 可插入：如果选中，则确定该列将包含在由持久性提供程序生成的SQL INSERT语句中。（如果没有设置默认值是true）
* 可更新：当选中时确定该列将包含由持久性提供程序生成的SQL UPDATE语句。（如果没有设置默认值是true）

[关系属性：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_relationship_properties)

当字段的类型是数据对象类型或数据对象类型列表时，应设置关系类型以便让持久性提供程序管理关系。幸运的是，当这种类型的字段被添加到已标记为可持久数据对象时，这种关系类型会自动设置。关系类型由以下弹出框设置。

*图259.关系配置弹出窗口*

* 关系类型：通过以下选项之一设置关系的类型：

一对一：通常用于1：1关系，其中“A与B的一个实例有关”，而B仅在A存在时才存在。例如PurchaseOrder→PurchaseOrderHeader（PurchaseOrderHeader仅在PurchaseOrder存在时才存在）

一对多：通常用于1：N关系，其中“A与B的N个实例有关”，并且B的相关实例仅在A存在时才存在。例如PurchaseOrder→PurchaseOrderLine（PurchaseOrderLine仅在PurchaseOrder存在时才存在）

多对一：通常用于1：1关系，其中“A与B的一个实例有关”，而B可以存在，即使没有A.例如PurchaseOrder→Client（即使没有关联的PurchaseOrder，客户端也可以存在于数据库中）

多对多：通常用于N：N关系，其中“A可以与B的N个实例相关，并且B可以同时与A的M个实例相关”，并且B和A实例都可以退出数据库独立于相关实例。例如课程→学生。（课程可以与N个学生相关，并且给定的学生可以参加M个课程）

将“数据对象”类型的字段添加到给定的持久数据对象时，缺省情况下会生成“多对一”关系类型。

当“数据对象列表”类型的字段添加到给定的持久数据对象时，缺省情况下会生成“一对多”关系。

* 级联模式：定义传播到关联实体的一系列级联操作。值cascade = ALL相当于cascade = {PERSIST，MERGE，REMOVE，REFRESH}。例如当A→B，级联“PERSIST或ALL”时，如果保存了A，则B也将被保存。

由数据建模器创建的默认级联模式是“ALL”，强烈建议当jBPM流程和表单使用数据对象时强烈建议使用此模式。

* 提取模式：定义在读取时如何从数据库提取相关数据。

EAGER：相关数据将被同时读取。例如，如果A→B，当从数据库B读取A时将同时读取。

LAZY：相关数据的阅读通常会延迟到需要的时候。例如，如果PurchaseOrder→PurchaseOrderLine行读取将被推迟，直到在PurchaseOrder实例上调用方法“getLines（）”。

数据建模器创建的默认提取模式为“EAGER”，强烈建议当jBPM流程和表单使用数据对象时强烈建议使用此模式。

* 可选：确定关系的右侧成员是否可以为空。
* 映射者：用于反向关系。

###### [高级域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_advanced_domain)

高级域允许配置由其他域设置的任何参数以及添加任意参数。如代码生成部分所示，每个“数据对象/字段”参数都由java注释表示。高级模式可用于配置此注释。

[高级域数据对象/字段编辑器。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_advanced_domain_data_object_field_editor)

高级域编辑器对数据对象和字段都具有相同的形状。

*图260.高级域编辑器。*

以下操作可用

* 删除：启用删除给定的数据对象或字段注释。
* 清除：清除给定的注释参数值。
* 编辑：启用给定注释参数值的编辑。
* 添加注释：添加注释按钮将启动一个向导，该向导将允许添加项目依赖项中可用的任何Java注释。

添加注释向导步骤＃1：向导的第一步需要输入注释的全限定类名称，并通过按下“搜索”按钮将注释定义加载到向导中。另外，当加载注解定义时，将创建不同的向导步骤，以便完成不同的注释参数。所需的参数将标有“\*”。

*图261.加载到向导中的注释定义。*

只要有可能，向导将为给定的参数提供合适的编辑器。

*图262.为Enumeration注解参数自动生成枚举值编辑器。*

当无法计算自定义编辑器时，将提供通用参数编辑器

*图263.通用注释参数编辑器*

当所有必需参数输入和验证后，完成按钮将被启用，向导可以通过将注释添加到给定的数据对象或字段来完成。

##### [20.6.7.5。生成数据模型代码。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sect_datamodeler_generatecode)

数据模型本身只是一个可视化工具，它允许用户定义高级数据结构，一方面与Drools引擎以及另一方面与jBPM平台交互。为了实现这一目标，这些高级视觉结构必须转化为可以被这些平台有效利用的低级工件。这些工件是Java POJO（普通Java对象），每次保存数据模型时都会生成这些工件，方法是按顶部“数据建模器菜单”中的“保存”按钮。此外，当用户在“编辑器”和“源代码”选项卡之间往返时，代码会自动生成以保持与编辑器视图的一致性，反之亦然。

*图264.从顶部菜单保存数据模型*

生成的代码根据以下转换规则生成：

* 数据对象的标识符属性将成为Java类的名称。因此它需要是一个有效的Java标识符。
* 数据对象的包属性成为Java类的包声明。
* 数据对象的超类属性（如果存在）成为Java类的扩展声明。
* 数据对象的标签和描述属性将分别转化为Java注释“@ org.kie.api.definition.type.Label”和“@ org.kie.api.definition.type.Description”。这些注释只是保存相关信息的一种方式，至今还没有进一步处理。
* 数据对象的角色属性（如果存在）将被翻译成“@ org.kie.api.definition.type.Role”的Java注解，即IS应用程序平台的解释，在某种意义上说，它标志着这个Java类作为流口水事件事实类型。
* 数据对象的类型安全属性（如果存在）将被转换为“@ org.kie.api.definition.type.TypeSafe Java注释（请参阅Drools）
* 数据对象的类反应属性（如果存在）将被转换为“@ org.kie.api.definition.type.ClassReactive Java注释（请参阅Drools）
* 数据对象的属性反应属性（如果存在）将被转换为“@ org.kie.api.definition.type.PropertyReactive Java注释（参见Drools）
* 数据对象的timestamp属性（如果存在）将被转换为“@ org.kie.api.definition.type.Timestamp Java注释（请参阅Drools）
* 数据对象的持续时间属性（如果存在）将被转换为“@ org.kie.api.definition.type.Duration Java注释（请参阅Drools）
* 数据对象的expires属性（如果存在）将被转换为“@ org.kie.api.definition.type.Expires Java注释（请参阅Drools）
* 数据对象的远程属性（如果存在）将被转换为“@ org.kie.api.remote.Remotable Java注释（请参阅jBPM）

将生成一个标准的Java默认（或无参数）构造函数，以及一个完整的参数构造函数，即一个构造函数，它接受每个数据对象的用户定义字段的值作为参数。

根据以下转换规则，将数据对象的用户定义字段转换为Java类字段，每个字段都有其自己的getter和setter方法：

* 数据对象字段的标识符将成为Java字段标识符。因此它需要是一个有效的Java标识符。
* 数据对象字段的类型直接转换为Java类的字段类型。如果该字段被声明为多个（即'List'），则生成的字段为“java.util.List”类型。
* equals属性：当它被设置为一个特定的字段时，这个类属性将被注释为“@ org.kie.api.definition.type.Key”注释，由Drools引擎解释，它将'参与'生成的equals（）方法，该方法将覆盖Object类的equals（）方法。后者意味着如果该字段是“原始”类型，则equals方法将简单地将其值与该类的另一个实例中相应字段的值进行比较。如果该字段是子实体或集合类型，那么equals方法将分别对相应的数据对象的Java类或java.util.List标准Java类的equals方法进行方法调用。

如果为任何数据对象的用户定义字段检查了equals属性，那么这也意味着除了默认生成的构造函数以外，还会生成另一个构造函数，并将所有用Equals标记的字段接受为参数。此外，equals（）方法的生成也意味着Object类的hashCode（）方法被覆盖，以这种方式调用相应Java类类型的hashCode（）方法（无论是“原始”还是用户 - 在数据模型中用Equals标记的所有字段。

* 位置属性：此字段属性将自动为所有用户定义的字段设置，从0开始，对每个后续新字段递增1。但是用户可以自由地改变字段之间的位置。在代码生成时，这个属性被翻译成“@ org.kie.api.definition.type.Position”注解，它可以被Drools引擎解释。此外，已建立的属性顺序决定了生成的Java类中构造函数参数的顺序。

例如，生成的采购订单数据对象的Java类代码与其定义相对应，如下图所示purchase\_example.jpg在本章底部的图中可见。请注意，数据对象的两个字段，即'标题'和'行'分别用Equals标记，并分别指定了位置2和1）。

*图265.采购订单配置*

**package** org.jbpm.examples.purchases;

*/\*\**

*\* This class was automatically generated by the data modeler tool.*

*\*/*

@org.kie.api.definition.type.Label("Purchase Order")

@org.kie.api.definition.type.TypeSafe(**true**)

@org.kie.api.definition.type.Role(org.kie.api.definition.type.Role.Type.EVENT)

@org.kie.api.definition.type.Expires("2d")

@org.kie.api.remote.Remotable

**public** **class** **PurchaseOrder** **implements** **java**.**io**.**Serializable**

{

**static** **final** **long** serialVersionUID = 1L;

@org.kie.api.definition.type.Label("Total")

@org.kie.api.definition.type.Position(3)

**private** java.lang.Double total;

@org.kie.api.definition.type.Label("Description")

@org.kie.api.definition.type.Position(0)

**private** java.lang.String description;

@org.kie.api.definition.type.Label("Lines")

@org.kie.api.definition.type.Position(2)

@org.kie.api.definition.type.Key

**private** java.util.List<org.jbpm.examples.purchases.PurchaseOrderLine> lines;

@org.kie.api.definition.type.Label("Header")

@org.kie.api.definition.type.Position(1)

@org.kie.api.definition.type.Key

**private** org.jbpm.examples.purchases.PurchaseOrderHeader header;

@org.kie.api.definition.type.Position(4)

**private** java.lang.Boolean requiresCFOApproval;

**public** **PurchaseOrder**()

{

}

**public** java.lang.Double **getTotal**()

{

**return** **this**.total;

}

**public** **void** **setTotal**(java.lang.Double total)

{

**this**.total = total;

}

**public** java.lang.String **getDescription**()

{

**return** **this**.description;

}

**public** **void** **setDescription**(java.lang.String description)

{

**this**.description = description;

}

**public** java.util.List<org.jbpm.examples.purchases.PurchaseOrderLine> getLines()

{

**return** **this**.lines;

}

**public** **void** **setLines**(java.util.List<org.jbpm.examples.purchases.PurchaseOrderLine> lines)

{

**this**.lines = lines;

}

**public** org.jbpm.examples.purchases.PurchaseOrderHeader **getHeader**()

{

**return** **this**.header;

}

**public** **void** **setHeader**(org.jbpm.examples.purchases.PurchaseOrderHeader header)

{

**this**.header = header;

}

**public** java.lang.Boolean **getRequiresCFOApproval**()

{

**return** **this**.requiresCFOApproval;

}

**public** **void** **setRequiresCFOApproval**(java.lang.Boolean requiresCFOApproval)

{

**this**.requiresCFOApproval = requiresCFOApproval;

}

**public** **PurchaseOrder**(java.lang.Double total, java.lang.String description,

java.util.List<org.jbpm.examples.purchases.PurchaseOrderLine> lines,

org.jbpm.examples.purchases.PurchaseOrderHeader header,

java.lang.Boolean requiresCFOApproval)

{

**this**.total = total;

**this**.description = description;

**this**.lines = lines;

**this**.header = header;

**this**.requiresCFOApproval = requiresCFOApproval;

}

**public** **PurchaseOrder**(java.lang.String description,

org.jbpm.examples.purchases.PurchaseOrderHeader header,

java.util.List<org.jbpm.examples.purchases.PurchaseOrderLine> lines,

java.lang.Double total, java.lang.Boolean requiresCFOApproval)

{

**this**.description = description;

**this**.header = header;

**this**.lines = lines;

**this**.total = total;

**this**.requiresCFOApproval = requiresCFOApproval;

}

**public** **PurchaseOrder**(

java.util.List<org.jbpm.examples.purchases.PurchaseOrderLine> lines,

org.jbpm.examples.purchases.PurchaseOrderHeader header)

{

**this**.lines = lines;

**this**.header = header;

}

@Override

**public** **boolean** **equals**(Object o)

{

**if** (**this** == o)

**return** **true**;

**if** (o == **null** || getClass() != o.getClass())

**return** **false**;

org.jbpm.examples.purchases.PurchaseOrder that = (org.jbpm.examples.purchases.PurchaseOrder) o;

**if** (lines != **null** ? !lines.equals(that.lines) : that.lines != **null**)

**return** **false**;

**if** (header != **null** ? !header.equals(that.header) : that.header != **null**)

**return** **false**;

**return** **true**;

}

@Override

**public** **int** **hashCode**()

{

**int** result = 17;

result = 31 \* result + (lines != **null** ? lines.hashCode() : 0);

result = 31 \* result + (header != **null** ? header.hashCode() : 0);

**return** result;

}

}

##### [20.6.7.6。使用外部模型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sect_datamodeler_externalmodels)

使用外部模型意味着可以在当前项目上下文中为已定义的POJO使用一个集合。为了使这些POJO可用，应该添加给定JAR的依赖项。一旦添加了依赖关系，可以从当前项目数据模型中引用外部POJO。

有两种方法可以将依赖项添加到外部JAR文件中：

* 依赖于当前本地M2存储库中已安装的JAR文件（通常与用户主页关联）。
* 依赖于当前KIE Workbench / Drools Workbench“Guvnor M2存储库”中安装的JAR文件。（在应用程序内部）

###### [依赖于本地M2存储库中的JAR文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dependency_to_a_jar_file_in_local_m2_repository)

要将依赖项添加到本地M2存储库中的JAR文件，请执行以下步骤。

[打开当前项目的项目编辑器并选择Dependencies视图。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_open_the_project_editor_for_current_project_and_select_the_dependencies_view)

*图266.项目编辑器。*

[点击“添加”按钮添加一个新的依赖线。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_click_on_the_add_button_to_add_a_new_dependency_line)

*图267.新的依赖关系线。*

[完成已安装在本地M2存储库中的JAR文件的GAV。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_complete_the_gav_for_the_jar_file_already_installed_in_local_m2_repository)

*图268.依赖线版本。*

[保存该项目以更新其依赖项。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_save_the_project_to_update_its_dependencies)

保存项目时，外部文件中定义的POJO将可用。

*图269.保存项目。*

###### [在当前“Guvnor M2存储库”中依赖于JAR文件。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_dependency_to_a_jar_file_in_current_guvnor_m2_repository)

要在当前的“Guvnor M2存储库”中向JAR文件添加依赖项，请按照以下步骤操作。

[打开Maven Artifact Repository编辑器。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_open_the_maven_artifact_repository_editor)

*图270. Guvnor M2储存库编辑器。*

[浏览您的本地文件系统并使用浏览按钮选择要上传的JAR文件。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_browse_your_local_file_system_and_select_the_jar_file_to_be_uploaded_using_the_browse_button)

*图271.文件浏览器。*

[使用上传按钮上传文件。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_upload_the_file_using_the_upload_button)

*图272.文件上传成功。*

[Guvnor M2存储库文件。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_guvnor_m2_repository_files)

文件加载完成后，它将显示在存储库文件列表中。

*图273.文件列表。*

[为上传的文件提供一个GAV（可选）。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_provide_a_gav_for_the_uploaded_file_optional)

如果上传的文件不是有效的Maven JAR（没有pom.xml文件），系统将提示用户为要安装的文件提供GAV。

*图274.无效的POM。*

*图275.手动输入GAV。*

[从存储库添加依赖项。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_add_dependency_from_repository)

打开项目编辑器（见下图）并点击“从存储库添加”按钮打开JAR选择器以查看当前“Guvnor M2存储库”中的所有已安装JAR文件。选择所需的文件时，应保存项目以使新的依赖关系可用。

*图276.从“Maven Artifact Repository”中选择JAR。*

###### [使用外部对象](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_the_external_objects)

当设置了对外部JAR的依赖关系时，可以通过以下方式在当前项目数据模型的上下文中使用外部POJO：

* 外部POJO可以通过当前模型数据对象进行扩展。
* 外部POJO可以用作当前模型数据对象的字段类型。

以下屏幕截图显示了外部对象如何以字符串“-ext-”作为前缀以便快速识别。

*图277.识别外部对象。*

##### [20.6.7.7。往返和并发](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_roundtrip_and_concurrency)

当前版本实现了Data modeller和Java源代码之间的往返和代码保存。无论Java代码是在哪里生成的（例如Eclipse，Data modeller），数据建模器都将只创建/删除/更新必要的代码元素以维护模型更新，即字段，getter / setters，构造函数，equals方法和hashCode方法。此外，无论Data Modeler所管理的Type或Field注释在Data modeller更新Java源时都将被保留。

除了保存代码，就像在其他工作台编辑器中一样，并发修改场景仍然是可能的。常见场景是两个不同的用户为同一个项目更新模型时，例如使用数据建模器或执行修改项目源的“git push command”。

从应用程序上下文的角度来看，我们可以基本确定两种不同的主要方案：

###### [应用程序未进行任何更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_no_changes_have_been_undertaken_through_the_application)

在这种情况下，应用程序用户基本上只是在数据模型中导航，而没有对其进行任何更改。同时，另一个用户从外部修改数据模型。

在这种情况下，不会向应用程序用户发出即时警告。但是，只要用户尝试进行任何类型的更改（例如添加或删除数据对象或属性）或更改任何现有的更改，就会显示以下弹出窗口：

*图278.外部更改警告*

用户可以选择：

* 重新打开数据模型，从而加载任何外部更改，然后执行他即将进行的修改，或者
* 忽略任何外部变化，并继续修改模型。在这种情况下，当试图保留这些更改时，会显示另一个弹出式警告：

*图279.强制保存/重新打开*

“强制保存”选项将有效覆盖任何外部更改，而“重新打开”将丢弃任何本地更改并重新加载模型。

|  |  |
| --- | --- |
|  | “强制保存”会覆盖所有外部更改！ |

###### [已通过申请进行了更改](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_changes_have_been_undertaken_through_the_application)

应用程序用户对数据模型进行了更改。同时，另一个用户从应用程序上下文外部同时修改数据模型。

在这种替代方案中，在外部用户向资产存储库提交其更改（或者例如将模型与数据建模器保存在不同的会话中）之后，立即向应用程序用户发出警告：

*图280.外部更改警告*

与前面的场景一样，用户可以选择：

* 重新打开数据模型，从而失去通过应用程序进行的任何修改，或者
* 忽略任何外部变化，并继续在模型上工作。

现在可以发生下列其中一种可能性：\*\*用户尝试通过单击数据建模器顶级菜单中的“保存”按钮来保存对模型所做的更改。这导致以下警告消息：

+

*图281.强制保存/重新打开*

“强制保存”选项将有效覆盖任何外部更改，而“重新打开”将丢弃任何本地更改并重新加载模型。

#### [20.6.8。数据集](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasets)

甲**数据组**基本上是一组填充了一些排，时间戳，文本和数字构成的数据的矩阵列组成。数据集可以存储在不同的系统中：数据库，Excel文件，内存或许多其他不同系统中。另一方面，**数据集定义**告诉工作台模块如何访问，读取和解析这些数据。

请注意，清楚数据集与其定义之间的区别非常重要，因为工作台并不负责存储任何数据，只是提供了一种标准方式来定义对这些数据集的访问权限，无论数据存储在何处。

举例来说，存储在远程数据库中的数据。例如，有效的数据集可以是整个数据库表或SQL查询的结果。在这两种情况下，数据库都会返回一堆列和行。现在，想象一下，我们想要访问这些数据，以便在新的工作台页面中提供一些图表。首先要创建并注册一个数据集定义，以表明以下内容：

* 数据集存储在哪里，
* 如何被访问，读取和解析
* 哪些列包含哪些类型。

本章介绍用于注册和处理数据集定义的可用工作台工具以及如何在其他工作台模块（如页面编辑器）中使用此定义。

|  |  |
| --- | --- |
|  | 为了简单起见，我们将使用术语数据集来引用实际的数据集定义，因为数据集和数据集定义在数据集创作上下文中可以被认为是同义词。 |

##### [20.6.8.1。数据集创作页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetauthoringperspective)

与创建数据集相关的所有内容都可以在Data Set Authoring页面下找到，该页面可从以下顶级菜单条目中访问：扩展>数据集，如以下屏幕截图所示。

*图282.数据集创作页面*

中央面板显示欢迎屏幕，而左侧面板包含数据集浏览器，其中列出了所有可用的数据集

|  |  |
| --- | --- |
|  | 此页面仅供管理员用户使用，因为定义数据集可被视为低级任务。 |

##### [20.6.8.2。数据集浏览器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetexplorer)

该数据集浏览器列出存在于系统中的数据集。每次用户点击数据集时，都会显示一个简短的摘要以及以下信息：

*图283.数据集浏览器*

* （1）用于创建新数据集的按钮
* （2）当前可用数据集的列表
* （3）表示数据集的提供者类型（Bean，SQL，CSV等）的图标
* （4）当前缓存和刷新策略状态的详细信息
* （5）后端当前大小（单位为行）和客户端当前大小的详细信息（单位为字节）
* （6）用于编辑数据集的按钮。点击后，数据集编辑器屏幕在中央面板上打开

接下来的部分将介绍如何创建，编辑和微调数据集定义。

##### [20.6.8.3。数据集创建](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetcreation)

点击New Data Set按钮打开一个新的屏幕，用户可以通过这个屏幕分三个步骤创建一个新的数据集定义：

* 提供者类型选择

指定远程存储系统的类型（BEAN，SQL，CSV，ElasticSearch）

* 提供者配置

指定能够从远程系统查找数据的属性。配置因所选数据提供者类型而异。

* 数据集栏和过滤器

实时数据预览，列类型和初始过滤器配置。

###### [第1步：提供商类型选择](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetcreationtypeselection)

允许用户指定正在创建的数据集的数据提供者的类型。

此屏幕列出了所有当前可用的数据提供程序类型以及带说明的帮助程序弹出窗口。每个数据提供者都用一个描述性图像来表示：

*图284.提供者类型选择*

目前支持四种类型：

* Bean（Java类） - 直接从Java生成数据集
* SQL - 用于从任何符合ANSI-SQL的数据库获取数据
* CSV - 上传远程或本地CSV文件的内容
* 弹性搜索 - 查询并获取存储在Elastic Search节点上的文档作为数据集

选择类型后，单击下一步按钮继续下一个工作流程步骤。

###### [第2步：配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetcreationconfiguration)

*图285. CSV配置*

在上一步中选择的提供程序类型将确定系统要求的配置设置。

*图286.每个数据集类型的配置屏幕*

|  |  |
| --- | --- |
|  | UUID属性是由系统生成的只读字段。它仅用于API调用或特定操作。 |

###### [第3步：数据集栏和预览](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetcreationpreview)

点击测试按钮后（参见上一步），系统将执行数据集查找测试呼叫，以检查远程系统是否已启动并且数据是否可用。如果一切正常，用户将看到以下屏幕：

*图287.数据集预览*

此屏幕显示实时数据预览以及用户希望成为结果数据集一部分的列。用户还可以浏览数据并对数据集结构应用一些更改。一旦完成，我们可以点击保存按钮为了注册新的数据集定义。

我们还可以随时更改配置设置，只需返回配置选项卡即可。我们可以根据需要重复配置>测试>预览循环，直到我们认为它已准备好保存。

**列**

在“ 列”选项卡区域中，用户可以选择哪些列是结果数据集定义的一部分。

*图288.数据集列*

* （1）添加或删除列。只选择那些你想成为结果数据集的一部分的列
* （2）使用下拉图像选择器更改列类型

数据集可能只包含以下4种类型的列：

* 标签 - 对于支持组操作的文本值（类似于SQL“group by”操作符），这意味着您可以执行数据查找调用并获得每个不同值的一行。
* 文本 - 对于不支持组操作的文本值。通常用于建模诸如摘要，描述等的大型文本列。
* 数字 - 用于数字值。它支持数据查询调用的聚合函数：sum，min，max，average，count，disctinct。
* 日期 - 用于日期或时间戳记值。它确实支持不同时间间隔的基于时间的群组操作：分钟，小时，日，月，年......

无论您想从哪个远程系统检索数据，生成的数据集总是会返回上述四种类型之一的一组列。默认情况下，存在远程系统列类型和数据集类型之间的映射。用户可以修改某些列的类型，具体取决于数据提供者和远程系统的列类型。系统支持以下对列类型的更改：

* 标签<>文本 - 当我们想要启用/禁用目标列的分类（分组）时很有用。例如，想象一个名为“document”的数据库表，其中包含一个名为“abstract”的大型文本列。由于我们不希望系统将此列作为“标签”对待，因此我们可能会将其列类型更改为“文本”。这样做，我们正在优化系统处理数据集的方式
* 数字<>标签 - 当我们想将数字列作为标签对待时很有用。这可以用于例如指示给定的数字列不是可用于聚合函数的数字值。尽管其值存储为我们想要将该列作为“标签”处理的数字。这样的专栏的一个例子是：一个项目的代码，一个评估ID，...

|  |  |
| --- | --- |
|  | BEAN数据集不支持更改列类型，因为它取决于开发人员决定哪些列是具体类型。 |

**过滤**

数据集定义可以定义一个过滤器。过滤器的目标是省去用户认为不必要的行。过滤器功能适用于任何数据提供程序类型，它允许用户对任何可用的数据集列应用过滤操作。

*图289.数据集过滤器*

在添加或删除过滤器条件和操作时，中央区域的预览表会更新为反映当前过滤器状态的实时数据。

有两种筛选数据集的策略，同样重要的是要注意在两者之间进行选择具有重要意义。设想一个仪表盘，其中有一些图表来自费用报告数据集，这些数据集是在SQL表的基础上构建的。想象一下，我们也只想从“伦敦”办公室获取费用报告。您可以定义一个包含过滤器“office = London”的数据集，然后使用这些数据集提供多个图表。这是推荐的方法。另一个选择是定义一个没有初始过滤器的数据集，然后让各个图表指定他们自己的过滤器。由用户决定最佳方法。

根据情况，最好在数据集级别定义过滤器以便在其他模块中重用。该决定也可能会对性能产生影响，因为过滤后的缓存数据集的性能要比许多单独的非缓存数据集查找请求好得多。（有关缓存数据集的更多信息，请参阅下一节）。

|  |  |
| --- | --- |
|  | 注意，对于SQL数据集，用户既可以使用引入的过滤器功能，也可以将自定义过滤条件添加到SQL语句中。虽然，第一种方法更适用于非技术用户，因为他们可能没有所需的SQL语言技能。 |

##### [20.6.8.4。数据集编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetdefeditor)

要编辑现有数据集定义，请进入数据集浏览器，展开所需的数据集定义并单击编辑按钮。这将导致一个新的编辑器面板被打开并放置在屏幕的中心，如下图所示：

*图290.数据集定义编辑器*

*图291.编辑器选择器*

* 保存 - 验证当前更改并存储数据集定义。
* 删除 - 从存储中永久删除数据集定义。任何引用数据集的客户端模块都可能受到影响。
* 验证 - 检查所有必需的参数是否存在并且是否正确，以及验证数据集是否可以被检索而没有问题。
* 复制 - 创建一个全新的定义作为当前副本的副本。

|  |  |
| --- | --- |
|  | 数据集定义作为JSON文件存储在底层的GIT存储库中。执行的任何操作都会在存储库日志中注册，以便稍后审核更改日志。 |

##### [20.6.8.5。高级设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetadvancedsettings)

在高级设置选项卡区域，用户可以指定缓存和刷新设置。这些对于充分利用系统功能非常重要，从而提高性能并提高应用响应水平。

*图292.高级设置*

* （1）启用或禁用客户端缓存并指定最大大小（字节）。
* （2）启用或禁用后端缓存并指定最大缓存大小（行数）。
* （3）启用或禁用数据集和刷新周期的自动刷新。
* （4）启用或禁用旧数据设置的刷新。

让我们深入了解这些设置的含义的更多细节。

##### [20.6.8.6。高速缓存](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetcachesettings)

该系统提供了用于保存数据集和使用内存策略执行数据操作的开箱即用的缓存机制。这些功能的使用带来了很多好处，如减少网络流量，远程系统负载，处理时间等。另一方面，用户可以正确调整缓存设置以避免出现性能问题。

支持两种缓存级别：

* 客户端级别
* 后端级别

下图显示了在任何数据集操作中如何涉及缓存：

*图293.数据集缓存*

任何数据查找调用都会生成一个结果数据集，因此使用缓存技术将确定数据查找调用的执行位置以及结果数据集的位置。

**客户端缓存**

如果为ON，则查询操作中涉及的数据集将被推入Web浏览器，以便从此数据集提供的所有组件**不需要对后端执行任何请求，**因为数据集操作在客户端解决：

* 数据集存储在Web浏览器的内存中
* 客户端组件来自存储在浏览器中的数据集
* 数据集操作（分组，聚合，过滤器和排序）在Web浏览器中通过Javascript数据集操作引擎进行处理。

如果您事先知道您的数据集将保持较小，您可以启用客户端缓存。它将减少后端请求的数量，包括对存储系统的请求。另一方面，如果您认为您的数据集相当大，请禁用客户端缓存，以免遇到浏览器问题，如性能下降或间歇性挂起。

**后端缓存**

它的目标是为后端数据集提供一个缓存机制。

通过将数据集保存在内存中并使用内存引擎执行组，过滤和排序操作，此功能可**减少对远程存储系统的请求数量**。

对于不经常更改的数据集非常有用，它们的大小可以被认为是可以接受的，以便在内存中进行保存和处理。它对远程存储的低延迟连接问题也有帮助。另一方面，如果您的数据集将经常更新，最好禁用后端缓存，并在每次查找请求时对远程存储执行请求，因此存储系统负责解析数据集查找请求。

|  |  |
| --- | --- |
|  | BEAN和CSV数据提供程序默认依赖后端高速缓存，因为在这两种情况下，数据集必须始终加载到内存中，以便使用内存引擎解决任何数据查找操作。这就是为什么后端设置在高级设置选项卡中不可见的原因。 |

##### [20.6.8.7。刷新](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasetrefreshsettings)

刷新功能允许在满足某些条件时使任何缓存数据无效。

*图294.刷新设置*

* （1）启用或禁用刷新功能。
* （2）指定刷新间隔。
* （3）在数据过期时启用或禁用数据集无效。

数据集刷新策略与数据集缓存密切相关，详见上一节。这个失效机制决定了缓存的生命周期。

根据数据的性质，存在三种主要用例：

* **源数据变化可预测** - 想象一下每晚更新的数据库。在这种情况下，建议的配置是使用“刷新间隔= 1天”并禁用“刷新陈旧数据”。这样，系统每天都会使缓存的数据集无效。当我们事先知道数据将会改变时，这是正确的配置。
* **源数据更改不可预测** - 另一方面，如果我们不知道数据库是否每天都更新，则建议的配置是使用“刷新时间间隔= 1天”并启用“刷新陈旧数据”。如果是这样，在使任何数据无效之前，系统将检查修改。在对数据进行修改时，系统将使当前陈旧的数据集失效，以便在下一次数据集查找调用时使用新数据填充高速缓存。
* **实时场景** - 实时场景缓存没有意义，因为数据将不断更新。在这种情况下，发送到客户端的数据必须不断更新，所以不是启用刷新设置（记住此设置影响缓存，并且缓存未启用），取决于使用数据集的客户端以决定何时刷新。当客户端是仪表板时，只需要修改“显示器编辑器”配置屏幕中的刷新设置，并设置适当的刷新周期，例如“刷新间隔= 1秒”。

#### [20.6.9。数据源管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasources)

数据源管理系统提供了定义用于访问外部数据库的数据源的功能。此数据源稍后可以由其他工作台组件（如数据集）使用。

##### [20.6.9.1。数据库驱动](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.databasedrivers)

为了能够与目标数据库通信，数据源将需要数据库驱动程序来访问它。这就是系统额外提供为数据源操作定义数据库驱动程序的能力的原因。数据库驱动程序基本上是符合JDBC的驱动程序。我们将在下一个主题中看到它们。

##### [20.6.9.2。数据源创作页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasourceauthoringperspective)

与数据源和驱动程序创作相关的所有内容都可以在数据源创作页面下找到，该页面可从以下顶级菜单条目访问：扩展>数据源，如以下屏幕截图所示。

*图295.数据源创作页面*

|  |  |
| --- | --- |
|  | 此页面仅供管理员用户使用，因为定义数据源可被视为低级别任务。 |

##### [20.6.9.3。数据源浏览器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasourceexplorer)

该数据源资源管理器中列出了当前系统中定义的数据源和驱动程序，它提供了一个管理他们所需要的操作同时进行。

*图296.数据源浏览器*

* （1）用于创建新数据源的操作链接
* （2）当前可用的数据源列表
* （3）创建新驱动程序的操作链接
* （4）当前可用的驱动程序列表

##### [20.6.9.4。新数据源向导](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasourcecreation)

单击新建数据源操作链接将打开新建数据源向导：

*图297.新建数据源向导*

以下所需的参数定义了一个数据源：

* 名称：数据源定义的唯一名称。
* 连接URL：与所选驱动程序类型兼容的JDBC数据库连接URL。这是PostgreSQL数据库连接URL的一个例子：jdbc：postgresql：// localhost：5432 / appformer。
* 用户：目标数据库中的用户名。
* 密码：相应的用户密码。
* 驱动程序：选择要用于连接到目标数据库的JDBC驱动程序。请注意，连接url格式可能因驱动程序而异，而不同的数据库供应商通常会提供不同的驱动程序。
* 测试连接：点击后，系统将显示一个类似于下图的对话框，显示连接测试状态。

*图298.测试连接状态*

|  |  |
| --- | --- |
|  | 虽然不是必需的，但建议在完成数据源创建之前使用测试连接按钮检查数据源参数的正确性。 |

##### [20.6.9.5。数据源编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasourceeditor)

所述数据源编辑器通过点击在一个数据源项打开数据源资源管理。

以下屏幕截图显示了为上述示例的数据源打开的数据源编辑器。

*图299.数据源编辑器*

* 主面板：主面板基本上允许您修改数据源配置参数。
* 测试连接：测试连接。

|  |  |
| --- | --- |
|  | 建议在保存修改的数据源之前测试连接。 |

##### [20.6.9.6。数据源内容浏览器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.datasourcecontent)

通过单击“ 浏览内容”按钮打开数据源内容浏览器，并启用通过数据源指向的数据库结构进行导航。导航分为三个级别：架构级别，当前架构级别和当前表级别。

* 模式级别：列出当前数据源可访问的所有数据库模式。列出哪些模式取决于授予连接配置中使用的用户的数据库访问权限。对于以下项目也是如此。
* 当前模式级别：显示所选模式的所有数据库表。
* 当前表格级别：显示所选表格的表格内容。

以下屏幕截图显示了实现以下导航步骤的用户在每个级别显示的信息。选择“公共”模式→选择“国家”表。

模式选择：

点击打开按钮打开所选模式的当前模式级别。

*图300.数据库模式*

表格选择：

点击打开按钮打开所选表格的当前表级别。

*图301.架构表*

表信息：

所选表格的行显示在此级别。

*图302.表格行*

##### [20.6.9.7。外部数据源](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.externaldatasources)

外部数据源通常不在当前工作台中定义，而是存在于当前容器中，对于像Wildfly 10或JBoss EAP 7服务器这样的容器，它们仍然可以以只读模式列出。在这种情况下，仅启用数据源内容浏览器。

*图303.外部数据源导航*

##### [20.6.9.8。新驱动程序向导](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.drivercreation)

点击新的驱动程序操作链接打开新的驱动程序向导：

*图304.新建驱动程序向导*

以下所需参数定义了一个驱动程序：

* 名称：驱动程序定义的唯一名称。
* 驱动程序类名称：实现JDBC驱动程序合同的类的java完全限定名称。
* 组ID：包含JDBC驱动程序实现的工件的Maven组标识。
* 工件标识：包含JDBC驱动程序实现的工件的Maven工件标识。
* 版本：包含JDBC驱动程序实现的工件的Maven版本。

|  |  |
| --- | --- |
|  | 某些商业数据库驱动程序（如Oracle）在maven中央存储库中不可用。您可以首先通过Artifact Repository页面上传这些文件，然后继续进行驱动程序配置，以获取maven中央存储库中可用的驱动程序。 |

##### [20.6.9.9。驱动编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.drivereditor)

该驱动程序编辑器通过点击在驾驶项目中打开数据源资源管理器。

以下屏幕截图显示为上述示例的驱动程序打开的驱动程序编辑器。

*图305.驱动程序编辑器*

* 主面板：主面板基本上允许您修改驱动程序配置参数。查看新驱动程序向导。

##### [20.6.9.10。按默认驱动程序](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.bydefaultdrivers)

系统附带一组默认配置的驱动程序，用于最常用的开源数据库。并且它们与Wildfly 10和JBoss EAP 7服务器支持的最新数据库版本保持一致。

*图306.通过默认驱动程序*

|  |  |
| --- | --- |
|  | 默认情况下，可以通过将datasource.management.disableDefaultDrivers配置属性设置为true来禁用驱动程序初始化。可以通过在datasource-management.properties文件中配置适当的值或通过将系统属性-Ddatasource.management.disableDefaultDrivers = true传递给JVM来设置它。有关更多信息，请参阅高级设置 |

##### [20.6.9.11。高级设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.advancedsettings)

数据源管理系统高级设置可以在给定Workbench分发文件的WEB-INF / classes目录下的datasource-management.properties文件中找到。

数据源管理系统能够处理数据源和驱动程序的两种不同的内部实现。基于Wildfly / EAP本机数据源和驱动程序的实现，以及独立于容器的实现。Wildfly / EAP Workbench发行版默认配置为使用原生的Wildlfy / EAP容器实现，Tomcat8发行版配置为使用容器独立实现。后一种实现也可以用于Wildfly / EAP容器。

有效的组合是：

WildflyDataSourceProvider + WildflyDriverProvider   
或  
DBCPDataSourceProvider + DBCPDriverProvider

datasource.management.wildfly.xxxxx属性仅适用于WildflyXXXProviders。

##### [20.6.9.12。Workbench Wildlfy / EAP分发的高级设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.advancedsettings.wildlfy)

| **属性名称** | **默认值** | **描述** |
| --- | --- | --- |
| datasource.management.DataSourceProvider | WildflyDataSourceProvider | 请参阅高级设置。 |
| datasource.management.DriverProvider | WildflyDriverProvider | 请参阅高级设置。 |
| datasource.management.disableDefaultDrivers | 假 | 设置为true以禁用默认数据库驱动程序初始化。 |
| datasource.management.wildfly.host | 本地主机 | 用于Wildlfy服务器管理接口绑定的名称或IP地址。 |
| datasource.management.wildfly.port | 9990 | 用于Wildlfy服务器管理接口绑定的端口。 |
| datasource.management.wildfly.admin |  | 管理用户连接到运行当前工作台的Wildfly服务器。通常，不需要设置此值，但在Wildlfy管理接口绑定到与localhost不同的地址时可能需要此值。 |
| datasource.management.wildfly.password |  | 管理用户密码以连接到运行当前工作台的Wildfly服务器。通常，不需要设置此值，但在Wildlfy管理接口绑定到与localhost不同的地址时可能需要此值。 |
| datasource.management.wildfly.realm | ManagementRealm | 用于管理用户认证的领域。 |
| datasource.management.wildfly.profile |  | 用于启动Wildfly域的配置文件名称，例如默认，完整，全角等。只有当Workbench在群集模式下运行并且托管Wildfly服务器使用域进行配置时，才能设置此值。如果Wildlfy服务器作为独立服务器运行，请勿设置。 |
| datasource.management.wildfly.serverGroup |  | 当前Wildfly服务器实例所属的服务器组，例如primary-server-group等。只有当Workbench在集群模式下运行并且托管Wildfly服务器使用域进行配置时，该值才必须设置。如果Wildlfy服务器作为独立服务器运行，请勿设置。 |
| datasource.management.DefChangeHandler |  | 此值必须仅在Workbench以群集模式运行时进行设置。如果托管Wildfly服务器是使用域进行配置的，则必须使用以下值DomainModeChangeHandler，并且在托管Wildlfy服务器作为独立服务器运行时必须使用以下值StandaloneModeChangeHandler。使用DBCPXXXProviders的集群安装必须配置为使用StandaloneModeChangeHandler。 |

|  |  |
| --- | --- |
|  | 上面的属性也可以通过使用Java标准机制将系统属性传递给JVM来设置。例如-Ddatasource.management.wildfly.port = 1234。使用此机制配置的值将覆盖datasource-management.properties文件中配置的值。 |

##### [20.6.9.13。Tomcat发行版的高级设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.advancedsettings.tomcat)

| **属性名称** | **默认值** | **描述** |
| --- | --- | --- |
| datasource.management.DataSourceProvider | DBCPDataSourceProvider | 这是Tomcat 8分发版的唯一选项，请参阅高级设置。 |
| datasource.management.DriverProvider | DBCPDriverProvider | 这是Tomcat 8分发版的唯一选项，请参阅高级设置。 |
| datasource.management.disableDefaultDrivers | 假 | 设置为true以禁用默认数据库驱动程序初始化。 |
| datasource.management.DefChangeHandler |  | 此值必须仅在Workbench以群集模式运行时进行设置。Tomcat发行版仅支持StandaloneModeChangeHandler值。 |

|  |  |
| --- | --- |
|  | 上面的属性也可以通过使用Java标准机制将系统属性传递给JVM来设置。例如-Ddatasource.management.wildfly.port = 1234。使用此机制配置的值将覆盖datasource-management.properties文件中配置的值。 |

### [20.7。安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.userandgroupmgmt)

本节介绍管理员用户如何使用直观友好的用户界面管理应用程序的用户，组和权限，以便配置谁可以访问可用的不同资源和功能。

#### [20.7.1。基本概念](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_basic_concepts)

为了理解所提供的安全管理功能，需要首先介绍一些核心概念。

##### [20.7.1.1。角色vs组](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_roles_vs_groups)

用户可以分配多个角色和/或组。为用户分配至少一个角色始终是强制性的，否则他/她将无法登录。角色在应用程序服务器级别定义，它们是webapp的web.xml描述符的一部分。另一方面，组是一个更灵活的概念，因为它们可以在运行时定义。

##### [20.7.1.2。权限](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_permissions)

基本上，用户可以在应用程序中执行许可。通常，与特定资源相关的操作。例如：

* 查看页面
* 保存一个项目
* 查看存储库
* 删除仪表板

权限可以被授予或拒绝，并且可以是全局的或特定于资源的。例如：

* 全球：“创建新页面”
* 具体：“查看主页”

正如你所看到的，许可是一resource + action对。在具体的页面中，我们有：读取，更新，删除和创建可用的操作。这意味着有四种可能的权限可以授予页面。

权限不一定需要绑定到资源。有时，保护对特定功能的访问也是必要的，例如“ 生成销售报告 ”。这意味着，权限不仅可用于保护对资源的访问，还可用于应用程序中的自定义功能。

##### [20.7.1.3。授权政策](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_authorization_policy)

分配给每个角色和/或组的权限集称为授权（或安全性）策略。每个应用程序都包含一个安全策略，每次系统检查权限时都会使用该安全策略。

授权策略文件存储在应用程序WAR结构下的名为WEB-INF / classes / security-policy.properties的文件中。

|  |  |
| --- | --- |
|  | 如果未定义策略，那么将禁用授权管理功能，并且应用程序的行为就像所有资源和功能默认授予一样。 |

以下是安全策略文件的示例：

**# Role "admin"**

role.admin.permission.perspective.read=true

role.admin.permission.perspective.read.Dashboard=false

**# Role "user"**

role.user.permission.perspective.read=false

role.user.permission.perspective.read.Home=true

role.user.permission.perspective.read.Dashboard=true

每个条目定义一个分配给角色/组的权限。在应用程序启动时，策略文件被加载并存储到内存中。

##### [20.7.1.4。安全提供商](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_provider)

安全环境通常通过使用领域来提供。领域用于限制对不同应用程序资源的访问。因此，领域包含有关用户，组，角色，权限和任何其他相关信息的信息。

在大多数典型情况下，应用程序的安全性被委托给容器的安全机制，该机制同时消耗给定的领域。重要的是要考虑存在多个领域实现，例如Wildfly提供基于application-users.properties / application-roles.properties文件的领域，Tomcat提供基于tomcat-users.xml文件的领域等。没有单一的安全领域依赖，它可以在每个安装不同。

由于需要支持的潜在不同安全环境，安全模块提供了一些定义良好的API以及一些默认的内置安全提供程序。一个**安全供应商**是给一个给定的领域的具体用户和组管理服务实施的正式名称。

可用的用户和组管理功能取决于所配置的安全提供程序。如果内置提供者不适合应用程序的安全领域，那么构建和注册您自己的提供者很容易。

#### [20.7.2。安装和设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installation_and_setup)

在撰写本文时，该应用程序提供了两个预安装的安全提供程序：

* **Wildfly 10 / EAP 7发行版** - 两个发行**版**均使用为配置默认领域文件application-users.properties和application-roles.properties而配置的[Wildfly安全提供](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management/uberfire-security-management-wildfly)程序
* **Tomcat分发** - 它使用配置为使用默认领域文件tomcat-users.xml的[Tomcat安全提供程序](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management/uberfire-security-management-tomcat)

请阅读每个[提供商的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)，以便为目标部署环境应用具体设置。

另一方面，当使用自定义安全提供程序或使用其中一个可用时，请考虑以下安装选项：

* 在现有WAR分发上启用安全管理功能
* 在现有或新建项目中进行设置和安装

注意：如果未安装安全提供程序，则将没有可用的用户界面来管理安全领域。一旦安装了安全提供程序并进行了安装，安全管理UI中将自动启用用户和组管理功能（请参阅下面的[使用情况](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_usage)部分）。

##### [20.7.2.1。启用用户和组管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_enabling_user_group_management)

鉴于现有WAR分发，请按照以下步骤安装并启用用户和组管理功能：

* 确保WEB-INF / lib中存在以下库：
  + WEB-INF / lib目录/ uberfire安全管理的API - ?.罐子
  + WEB-INF / lib目录/ uberfire安全管理，后台 - ?.罐子
* 将安全提供程序库复制到WEB-INF / lib：
  + 例如：WEB-INF / lib / uberfire-security-management-wildfly - ?. jar
  + 如果提供者需要额外的库，请复制它们（阅读每个[提供者的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)以获取更多信息）。
* 替换WEB-INF / classes / security-management.properties文件的全部内容，如果不存在，则创建它。该文件中的设置取决于所使用的具体实现。请阅读每个[提供商的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)以获取更多信息。
* 如果在Wildfly或EAP上部署，请检查WEB-INF / jboss-deployment-structure.xml是否需要更新（请阅读每个[提供商的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)以获取更多信息）。

##### [20.7.2.2。禁用用户和组管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_disabling_user_group_management)

用户和组管理功能可以被禁用，因此不会有服务或用户界面可用：

* 从应用程序中卸载安全提供程序

如果未安装具体的安全提供程序，则会禁用用户和组管理功能，并且不会向用户显示任何服务或用户界面。例如，在Weblogic和Websphere安装中就是这种情况， 因为在撰写本文时没有可用的安全提供程序实现。

* 删除或评论安全管理配置文件

删除或注释位于WEB-INF / classes / security-management.properties的配置文件中的所有行是禁用用户和组管理功能的另一种方法。

##### [20.7.2.3。升级现有安装](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_upgrading_an_existing_installation)

在7之前的版本中，授予对组织单位，存储库或项目等资源的访问权限的唯一方法是指出哪些角色能够访问给定实例。这些角色作为实例持久状态的一部分存储在GIT中。CLI是用于添加/删除角色的工具：

* **remove-role-repo**：从存储库中删除角色
* **add-role-org-unit**：将角色添加到组织单位
* **remove-role-org-unit**：从组织单位中删除角色
* **add-role-project**：将角色添加到项目中
* **remove-role-project**：从项目中删除角色

从版本7开始，授权策略基于权限。这意味着不再需要保留每个资源实例的角色列表。所需要的是使用安全管理UI将适当的权限条目定义到活动授权策略中（请参阅下面的[使用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_usage)部分）。

以上命令不再需要，因此它们已被删除。基本上，这些命令所做的是设置哪些角色能够读取特定项目。

为了保证与7之前版本的向后兼容性，自动迁移工具捆绑在应用程序中，该应用程序将分配给任何组织单位，存储库或项目的角色列表转换为安全策略的读取权限条目。

此应用程序在安全策略部署期间首次启动时执行。所以现有客户不必担心，因为他们将保持其安全设置。

#### [20.7.3。用法](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_usage_2)

安全管理页面位于顶部菜单栏的主页部分。

*图307.链接到安全管理页面*

下一个屏幕截图显示了这个新页面的外观：

*图308.安全管理主页*

此页面支持：

* 列出所有可用的角色，组和用户
* 创建和删除用户和组
* 编辑用户，分配角色或组，并更改用户属性
* 编辑角色和组安全设置，其中包括：
  + 登录后用户将被导向的主页
  + 授予或拒绝授予可用的不同工作台资源和功能的权限

上述所有功能一起提供了一个完整的用户和组管理子系统以及用于保护对特定资源或功能的访问的权限配置UI。

接下来的部分将深入分析所有这些功能。

|  |  |
| --- | --- |
|  | 用户和组管理相关功能可以完全禁用。请参阅上一节 [禁用用户和组管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_disabling_user_group_management)。如果是这种情况，那么组和用户选项卡将对用户保持隐藏状态。 |

##### [20.7.3.1。用户管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_user_management)

通过选择左侧栏中的“ 用户”选项卡，应用程序将在应用程序的安全领域中显示所有默认存在的用户：

* **搜索用户**

除列出所有用户外，还允许搜索：

+在搜索框中指定搜索模式时，列出的用户将被缩减为仅与搜索模式匹配的用户。

+

+搜索模式取决于应用程序使用的具体安全提供程序。请阅读每个[提供商的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)以获取更多信息。

* **创建新用户**

通过点击“ 新用户+ ”锚点，屏幕右侧会显示一个表单。

这是一个类似界面的向导，应用程序要求输入新的用户名，密码以及要分配的角色/组。

* **编辑用户**

点击左侧栏中的用户后，用户编辑器在屏幕右侧打开。

例如，使用Wildfly安全提供程序时，admin用户的详细信息屏幕如下所示：

同样的屏幕，但使用Keycloak安全提供程序时看起来如下所示：

请注意，使用Keycloak提供程序时，会显示新的用户属性部分，但在使用Wildfly提供程序时不会显示。这是由于以下事实：可用的信息和操作始终取决于每个提供商的功能，如以下“ [安全提供商功能”](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_provider_capabilities)部分所述。

接下来是在用户的详细信息屏幕中处理的信息类型：

* 用户名
* 用户的属性
* 分配的组
* 分配的角色
* 授予或拒绝权限

为了**更新或删除现有用户**，请点击用户编辑器屏幕中用户名旁边的编辑按钮：

编辑器处于编辑模式后，可以执行不同的操作（只要安全提供程序支持它们）：

例如，修改分配给用户的角色和组或者更改用户密码。

* **权限摘要**

“ 权限”选项卡显示分配给此特定用户的所有权限的摘要。这是一个非常有用的视图，因为它允许管理员用户根据角色和组的安全设置来验证目标用户是否拥有适当的权限级别。

有关如何为角色和组分配权限的更多详细信息位于下面的[安全设置编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_settings_editor)部分。

* **更新用户的属性**

可以使用属性表中可用的操作添加或删除用户属性：

* **更新分配的组**

从“ 组”选项卡中，单击“ 添加到组”按钮时会显示组选择弹出窗口：

该弹出屏幕允许用户搜索并选择或取消选择分配给用户的组。

* **更新分配的角色**

从“ 角色”选项卡中，单击添加到角色按钮时会显示一个角色选择弹出窗口：

该弹出屏幕允许用户搜索并选择或取消选择分配给用户的角色。

* **更改用户的密码**

点击更改密码按钮时会显示更改密码弹出窗口：

* **删除用户**

通过点击删除按钮可以删除当前正在编辑的用户。

###### [安全提供者功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_provider_capabilities)

每个安全领域可以为不同的操作提供支持。例如，考虑使用基于属性文件的Wildfly领域。application-users.properties的内容如下所示：

admin=207b6e0cc556d7084b5e2db7d822555c

salaboy=d4af256e7007fea2e581d539e05edd1b

maciej=3c8609f5e0c908a8c361ca633ed23844

kris=0bfd0f47d4817f2557c91cbab38bb92d

katy=fd37b5d0b82ce027bfad677a54fbccee

john=afda4373c6021f3f5841cd6c0a027244

jack=984ba30e11dda7b9ed86ba7b73d01481

director=6b7f87a92b62bedd0a5a94c98bd83e21

user=c5568adea472163dfc00c19c6348a665

guest=b5d048a237bfd2874b6928e1f37ee15e

kiewb=78541b7b451d8012223f29ba5141bcc2

kieserver=16c6511893651c9b4b57e0c027a96075

请注意，它基于键值对，其中键是用户名，值是用户密码的哈希值。所以一个用户只是由一个密钥和它的用户名来代表，它没有名字，也没有地址或其他元信息。

另一方面，考虑使用Keycloak服务器提供的领域。用户信息由更多元数据组成，例如姓氏，地址等，如下图所示：

因此，用户和组管理API中的不同服务和客户端组件都基于功能。**功能**用于公开或限制不同服务和客户端组件提供的可用功能。功能的例子是：

* 创建一个用户
* 更新用户
* 删除一个用户
* 更新用户的属性
* 创建一个组
* 更新一个组
* 将组分配给用户
* 为用户分配角色

每个安全提供者都必须指定一组支持的功能。从前面的例子可以看出，Wildfly安全提供程序不支持属性管理功能 - 用户只能由用户名组成。另一方面，Keycloak提供商确实支持这种功能。

不同的视图和用户界面组件依赖于每个提供程序支持的功能，因此如果提供程序不支持某项功能，UI将不提供该功能管理的视图。举个例子，考虑一个具体的提供者不支持删除用户 - 用户界面上的删除用户按钮将不可用。

请查看具体的服务提供商文档以检查每个支持的功能，默认的功能可以在[这里](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)找到。

##### [20.7.3.2。集团管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_group_management)

通过选择左边栏中的组选项卡，应用程序将显示应用程序安全领域中默认存在的所有组：

* **搜索组**

除列出所有组外，还允许搜索：

+在搜索框中指定搜索模式时，列出的组将被缩减为仅与搜索模式匹配的组。

+

+搜索模式取决于应用程序使用的具体安全提供程序。请阅读每个[提供商的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)以获取更多信息。

* **创建新的组**

通过点击“ 新组+ ”锚点，中央面板上将显示一个新屏幕，以执行新的组创建。

输入名称并点击保存后，下一步是将用户分配给它：

+

+点击“ 添加选定的用户 ”按钮完成组创建。

* **修改一个组**

点击左侧栏中的组后，所选组实例的安全设置编辑器将在屏幕右侧打开。[安全设置编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_settings_editor)部分的更多详细信息 。

* **删除组**

要删除一个现有的组，只需点击删除按钮。

##### [20.7.3.3。角色管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_role_management)

通过选择左边栏中的角色选项卡，应用程序将显示所有应用程序角色：

与用户和组不同，角色无法创建或删除，因为它们来自应用程序的web.xml描述符。点击左侧栏中的角色后，角色编辑器将在屏幕右侧打开，这与组使用的安全设置编辑器完全相同。[安全设置编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_settings_editor)部分的更多详细信息。

这意味着可以定义角色和基于组的权限。角色和组之间的主要区别是：

* 角色是应用程序定义的资源。它们在应用程序的web.xml描述符中定义为<security-role>条目。
* 组是动态的，可以在运行时定义。安装的安全提供程序确定存储组实例的位置。

它们可以一起使用，没有任何问题。建议使用组，因为它们比角色更灵活。

* **搜索角色**

除列出所有角色外，还允许搜索：

+在搜索框中指定搜索模式时，列出的角色将被缩减为仅与搜索模式匹配的角色。

+

+搜索模式取决于应用程序使用的具体安全提供程序。请阅读每个[提供商的文档](https://github.com/kiegroup/appformer/tree/master/uberfire-extensions/uberfire-security/uberfire-security-management)以获取更多信息。

#### [20.7.4。安全设置编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_settings_editor)

该编辑器用于为角色和组设置多个安全设置。

+

##### [20.7.4.1。主页](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_home_page)

这是登录后用户定向的页面。这样可以为不同的用户设置不同的主页，因为用户可以分配到不同的角色或组。

##### [20.7.4.2。优先](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_priority)

它用于确定哪些设置（主页，权限，...）优先于那些分配有多个该角色或组的用户。

如果没有这个设置，就不可能确定哪个角色/组应该优先。例如，管理角色比非管理角色具有更高的优先级。对于同时具有管理角色和非管理角色的用户，只要管理角色的优先级高于其他角色，管理权限将始终获胜。

##### [20.7.4.3。权限](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_permissions_2)

目前，该工作台支持以下权限类别。

* 工作台：一般工作台权限，不受任何特定资源类型的限制。
* 页面：如果访问页面被拒绝，则不会在任何应用程序菜单中显示。更新，删除和创建权限更改页面管理插件编辑器的行为。
* 组织单位：设置谁可以在“管理”页面的“组织单位”部分创建，更新或删除组织单位。还可以在项目创建页面的Project Explorer中设置哪些组织单位可见。
* 存储库：设置谁可以在“管理”页面的“存储库”部分创建，更新或删除存储库。还可以在Project Authoring页面的Project Explorer中设置哪些存储库可见。
* 项目：在“项目创作”页面中，设置谁可以从项目编辑器屏幕创建，更新，删除或构建项目以及哪些项目在项目浏览器中可见。

对于页面，组织单位，存储库和项目，可以定义全局权限并在之后添加单个实例异常。例如， 可以向所有页面授予读取访问权限，并且仅针对单个页面拒绝访问。这被称为授予全部否认一些策略。

相反，否认所有授予一些策略也是支持的：

|  |  |
| --- | --- |
|  | 在上面的示例中，更新和删除权限被禁用，因为如果用户甚至无法读取页面，则定义此类权限没有意义。 |

#### [20.7.5。安全策略存储](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_security_policy_storage)

安全策略存储在工作台的VFS下。最具体的，在一个GIT仓库中被称为“安全”。ACL表存储在“authz”目录下名为“security-policy.properties”的文件中。接下来是这个文件包含的条目的一个例子：

role.admin.home=HomePage

role.admin.priority=0

role.admin.permission.perspective.read=true

role.admin.permission.perspective.create=true

role.admin.permission.perspective.**delete**=true

role.admin.permission.perspective.update=true

每次从安全设置UI修改ACL时，更改都会存储到GIT仓库中。

最初，当应用程序第一次部署时，GIT中不存储安全策略。但是，应用程序可能需要为每个应用程序角色设置具有不同访问配置文件的默认策略。

为了支持默认策略，系统允许声明安全策略作为webapp内容的一部分。这可以通过在webapp的资源类路径下放置一个security-policy.properties文件来完成（WAR文件中的WEB-INF / classes目录是有效的）。在应用程序启动时，将执行以下步骤：

* 检查活动策略是否已存储在GIT中
* 如果不是，则检查是否已经在webapp的类路径下定义了一个策略
* 如果找到，则此政策存储在GIT下

以上是在工作台中使用的自动部署机制来设置其默认安全策略。

部署过程的一个细微变化是能够将“security-policy.properties”文件拆分成小块，以便例如可以为每个角色定义一个文件。拆分文件必须以“security-module-”前缀开头，例如：“security-module-admin.properties”。部署机制将读取并部署classpath上的“security-policy.properties”和所有可选的“security-module-?. properties”。

请注意，尽管使用拆分方法，但由于安全子系统将其用作标记文件以定位其他策略文件，因此必须始终存在“security-policy.properties”。这种分割机制可以更好地组织整个安全策略。

### [20.8。将嵌入式工作台嵌入您的应用程序中](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.embedding)

正如我们所知道的，Workbench提供了一组编辑器以不同格式创作资源。根据资产的格式，使用专门的编辑器。

Workbench提供的另一个功能是可以通过**独立**模式将其嵌入到自己的（Web）应用程序中。因此，如果您想在自己的应用程序中编辑规则，流程，决策表等，而无需切换到Workbench，则可以。

为了将Workbench嵌入到应用程序中，您需要的是Workbench应用程序已部署并在Web /应用程序服务器中运行，并且在您自己的Web应用程序中，具有适当HTTP查询参数的iframe，如下表所述。

| *表42.独立模式下的HTTP查询参数* | | | |
| --- | --- | --- | --- |
| **参数名称** | **说明** | **允许多个值** | **例** |
| 独立 | 只要存在该参数，工作台将切换到独立模式。 | 没有 | （没有） |
| 路径 | 要编辑的资产的路径。请注意，资产应该已经存在。 | 没有 | 混帐：//master@uf-playground/todo.md |
| 透视 | 引用现有的透视图名称。 | 没有 | LibraryPerspective |
| 头 | 定义应显示的标题的名称（对于上下文菜单标题很有用）。 | 是 | AppNavBar |

| *表43.示例* | | | |
| --- | --- | --- | --- |
| **网址** | **说明** | [**HTTP：//本地主机：8080 /纪伊-WB /纪伊-wb.jsp独立＆透视= LibraryPerspective**](http://localhost:8080/kie-wb/kie-wb.jsp?standalone&perspective=LibraryPerspective) | **打开库透视图。您可以选择要编辑的项目并获取所选项目的所有创作功能。** |
| [HTTP：//本地主机：8080 /纪伊-WB /纪伊-wb.jsp独立＆路径=默认：//master@myteam/Mortgages/src/main/java/mortgages/mortgages/Applicant.java](http://localhost:8080/kie-wb/kie-wb.jsp?standalone&path=default://master@myteam/Mortgages/src/main/java/mortgages/mortgages/Applicant.java) | 只打开给定资产的编辑器。 | [HTTP：//本地主机：8080 /纪伊-WB /纪伊-wb.jsp独立＆透视= AuthoringPerspective＆路径=默认：//master@myteam/Mortgages/src/main/java/mortgages/mortgages/Applicant.java](http://localhost:8080/kie-wb/kie-wb.jsp?standalone&perspective=AuthoringPerspective&path=default://master@myteam/Mortgages/src/main/java/mortgages/mortgages/Applicant.java) | 使用Project Explorer打开给定的文件。让用户可以浏览指定文件所在项目的内容，甚至打开项目中的其他文件。 |

### [20.9。执行服务器管理UI](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserverui)

Execution Server Management UI允许用户创建和修改服务器模板和容器，它还允许用户管理远程服务器。该屏幕通过“部署”→“规则部署”菜单提供。

*图309.执行服务器管理*

|  |  |
| --- | --- |
|  | 管理用户界面仅适用于KIE托管服务器。 |

#### [20.9.1。服务器模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserveruitemplate)

服务器模板用于定义可用于多个服务器的通用配置，因此名称为：Template。

服务器模板可以直接从管理用户界面创建，也可以在服务器连接到控制器时自动创建，并且该远程服务器没有模板定义。服务器模板可能具有一个或多个功能，如果您需要修改必须创建新模板的功能，则无法修改此功能。以下是当前功能列表：

* 规则（Drools）
* 过程（jBPM）
* 规划（Optaplanner）

|  |  |
| --- | --- |
|  | 对于规划者功能，也必须启用规则的功能。 |

为了创建新的服务器模板，您必须单击新建服务器模板按钮并按照向导。也可以在Wizard中创建一个容器，但现在我们只限于模板。

*图310.新建服务器模板向导*

创建完成后，您将看到左侧列出的新模板，并突出显示新的服务器模板。在右侧，您将看到列出与所选服务器模板相关的容器和远程服务器的第二级导航。

*图311.服务器模板*

在导航之上还可以删除当前的服务器模板或创建它的副本。

*图312.服务器模板操作*

#### [20.9.2。容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserveruicontainer)

容器是服务器模板的KIE容器配置。单击添加容器按钮为当前服务器模板创建一个新的容器。

搜索区域可以帮助用户找到他们正在寻找的特定KJAR。

*图313.新建容器向导*

对于启用了进程功能的服务器模板，向导具有第二个可选步骤，用户可以在其中配置一些与进程相关的行为。

*图314.进程配置*

**Kie Base Name**确定将使用已部署工件的哪个Kie Base。

**Kie会话名称**决定选定的Kie Base的Kie会话将被使用。

|  |  |
| --- | --- |
|  | 请注意，只有在部署的项目包含一些业务过程时，此选项卡上的配置才会生效。如果服务器模板启用了进程的扩展，则这是不够的。 |

一旦创建，新的容器将显示在远程服务器列表上方的容器列表中。刚创建容器时默认为Stopped，这是允许用户删除它的唯一状态。

*图315.容器*

容器有以下可用于管理和/或配置的选项卡：

* 状态
* 版本配置
* 过程配置

“状态”选项卡列出运行活动Container的所有远程服务器。每个远程服务器都呈现为一张卡片，显示给用户状态和端点。

|  |  |
| --- | --- |
|  | 只有启动的Container才会部署到远程服务器。 |

*图316.状态容器*

对于没有处理能力的容器，“版本配置”选项卡允许用户更改容器的当前版本。用户可以使用“升级”按钮手动升级到特定版本或启用/禁用扫描仪。也可以执行只扫描一次新版本的立即扫描操作。

要用最新的更改重新部署SNAPSHOT kjars，必须先删除该版本的所有现有容器。然后执行'build and deploy'将创建一个包含最新SNAPSHOT kjar的容器。但是，这对发布版本来说是不可能的。如果一个kjar的GAV是anthing但是SNAPSHOT，则需要将GAV更新到较新的发行版本并将其部署到其自己的容器中。如前所述，新版本也可用于升级现有容器，前提是容器不具有进程功能。

*图317.版本配置*

进程配置与具有进程能力的模板服务器的新容器向导期间显示的形式相同。如果模板服务器不具备此功能，则操作按钮将被禁用。

*图318.过程配置*

#### [20.9.3。远程服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.execserveruiremoteserver)

远程服务器是一个托管的KIE服务器实例，运行时配置了一个控制器。

|  |  |
| --- | --- |
|  | 默认情况下，Workbench附带嵌入式控制器。 |

远程服务器列表显示在容器列表的下方。一旦选择屏幕就会显示远程服务器的详细信息和一张卡片列表，每张卡片代表一个正在运行的Container。

*图319.远程服务器*

## [21.编写规则资产](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#drools.AuthoringAssets)

### [21.1。创建一个包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.newpackage)

配置软件包通常只需要完成一次，并且由具有规则/模型经验的人员完成。一般来说，很少有人需要配置软件包，一旦安装完成，可以根据需要反复复制。软件包配置绝对是一项需要适当专业知识的技术任务。

所有资产都存放在Drools Workbench的“软件包”中 - 软件包就像一个文件夹（它也可以作为“命名空间”）。规则资产居住的主文件夹。规则尤其需要知道事实模型是什么，命名空间是什么等等。

因此，虽然规则（和一般资产）可以出现在任何数量的类别中，但它们只能生活在一个包中。如果您将Drools Workbench看作是一个文件系统，那么每个软件包都是一个文件夹，并且资产位于该文件夹中 - 作为一个令人高兴的文件列表。

要创建一个空包，请从“创建新资产”菜单中选择“包”。

*图320.新包装*

#### [21.1.1。空包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_empty_package)

只需指定一个名称即可创建一个空包。

*图321.新的空包*

包创建完成后，它将显示在Project Explorer中。

*图322.显示新包的项目资源管理器*

#### [21.1.2。复制，重命名和删除软件包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_copy_rename_and_delete_packages)

正如在[Project Explorer](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.projectexplorercopyrenamedeleteactions)部分已经提到的，用户可以直接从Project Explorer复制，重命名或删除一个包。

### [21.2。业务规则与指导编辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditor)

使用UI创作引导规则，根据对象模型的知识来控制和提示用户输入。

这也可以用DSL语句来增强。

#### [21.2.1。引导规则编辑器的各个部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorparts)

引导规则编辑器由三个主要部分组成。

下图显示了编辑器的操作。以下说明适用于图表中的字母框：

*图323.导向的BRL编辑器*

答：规则的不同部分：

* 规则的“WHEN”部分或条件。
* 规则的“THEN”动作部分。
* 可能影响规则操作的可选属性。

#### [21.2.2。规则的“WHEN”（左侧）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorlhs)

B：这显示了一种模式，它声明规则正在寻找“贷款申请”事实（这些字段在下面列出，在这种情况下没有）。“申请人”的另一种模式列在“贷款申请”下面。列出了“creditRating”和“applicationDate”字段。点击事实名称（“LoanApplication”）将弹出一个选项列表以添加到事实声明中：

* 添加更多字段（例如他们的“位置”）。
* 为事实分配一个变量名称（如果需要，可以稍后使用）
* 添加“多字段”约束 - 即跨越字段的约束（例如，年龄> 42或风险> 2）。

C：“减号”图标（“[ - ]”）表示您可以删除某些内容。在这种情况下，它将删除整个“LoanApplication”事实声明。根据图标的位置，可以删除规则声明的不同组件，例如Fact Pattern，Field Constraint，其他条件元素（“exists”，“not exists”，“from”等）或Action。

D：“加号”图标（“+”）允许您为规则的条件或动作部分或更多属性添加更多模式。在所有情况下，都提供了一个弹出选项框。对于规则的“WHEN”部分，您可以从条件元素列表中进行选择以添加：

* 对事实的约束：它会给你一个事实清单。
* “以下不存在”：事实加上约束不得存在。
* “存在以下内容”：至少应存在一场比赛（但只需要一场比赛 - 每场比赛都不会触发）。
* “以下任何一个都是真实的”：任何模式都可以匹配（然后将模式添加到这些更高级模式中）。
* “发件人”：这会将新的“从条件元素”插入到规则中。
* “来自收集”：这将为规则插入一个新的收集条件元素。
* “从收集”：这将插入一个新的收集条件元素的规则。
* “从入口点”：这使您可以定义模式的入口点。
* “Free Form DRL”：这将让你插入一个免费的DRL块。

如果你只是提出一个事实（如上图所示），那么所有的模式组合在一起，所以它们都是真实的（“和”）。

E：这显示了“creditRating”字段的约束条件。从左至右看，你会发现：

* 字段名称：“creditRating”。点击它可以给它分配一个变量名，或者访问它的嵌套属性。
* 约束操作列表（“等于”被选中）：该列表的内容根据字段的数据类型而变化。
* 值字段：它可能是以下值之一：
  + 字面值：根据字段的数据类型显示不同的组件：
    - 字符串→文本框
    - 任何数值→文本框将条目限制为对数字子类型有效的值（例如，一个字节可以保存从-128到127的值）。BigDecimal和BigInteger数据类型也受支持。请确保已在包配置中导入适当的类。如果上传的POJO模型公开了BigDecimal或BigInteger字段的访问器或增变器，则会自动添加该导入。BigDecimal值自动后缀“B”，指示底层引擎该文字值应该被解释为BigDecimal。BigIntegers后缀为“I”。用户不需要输入后缀。
    - 日期→日历
    - 枚举→列表框
    - 布尔→复选框
  + “公式”：这是一个计算的表达式（仅适用于高级用户）
  + 表达式 - 这可让您使用表达式生成器来构建完整的表达式。（目前只支持基本表达式）

F：这显示了“applicationDate”字段的约束。从左至右看，你会发现：

* 字段名称：“applicationDate”。
* 约束操作列表：“之后”被选中。
* 一个“时钟”图标。由于“applicationDate”是Date数据类型，所以可用操作符的列表包括与复杂事件处理（CEP）相关的操作符。当使用CEP操作员时，会显示此附加图标以允许您输入其他CEP操作员参数。点击“时钟”将循环可用的CEP操作员参数组合。

|  |  |
| --- | --- |
|  | 当事实被声明为事件时，复杂事件处理操作符也可用。有关如何将注释添加到事实模型的详细信息，请参阅本用户指南的“事实模型”一章。活动可以访问所有CEP操作员; 日期字段类型被限制为“之后”，“之前”和“重合”。 |
|  | 注释为Events的事实也可以定义CEP滑动窗口。 |

##### [21.2.2.1。添加模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditoraddingpatterns)

当点击WHEN部分的+按钮时，会出现一个新的弹出窗口，让您将新的模式添加到规则。弹出窗口看起来类似于下面的图片。在这个弹出窗口中，您可以通过选择一个列表项来选择要添加的模式的类型。在列表中，除了已经提到的条件元素，例如“存在”，“不存在”，“来自”，“收集”，“累积”，“来自入口 - 点“和”自由格式DRL“。一旦你选择了这些元素之一，你可以通过点击“确定”按钮添加一个新的模式。新模式将添加在规则左侧的底部。如果您想选择不同的位置，可以使用放置在弹出框顶部的组合框。

您也可以通过单击模式操作工具栏上的[+]按钮来打开该弹出窗口。如果是这种情况，出现的弹出窗口不会限制位置组合框，因为新模式将被添加到您单击的模式之后。

*图324.添加模式*

##### [21.2.2.2。添加约束](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditoraddingconstraints)

下面的对话框是当你想添加约束到事实时你会得到的。上半部分是简单的选项：您可以直接添加字段约束（将显示适用事实的字段列表），也可以使用AND或OR操作数添加“多字段约束”。在窗口的下半部分有高级选项：你可以添加一个公式（这个公式可以解析为True或False - 这就像上面的例子：“... salary>（2500 \* 4.1）”）。你也可以分配一个变量名称的事实（这意味着你可以在规则的动作部分访问该变量，设置一个值等）。

*图325.添加约束*

#### [21.2.3。规则的“THEN”（右侧）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorrhs)

H：这表示规则的“动作”，规则的右手边包含动作列表。在这种情况下，我们正在更新“贷款申请”事实的“解释”字段。还有其他几种可以使用的操作：

* 插入一个全新的事实，并可选择在事实上设置一个字段。

值字段可以是以下值之一：

* + 字面值：根据字段的数据类型显示不同的组件：
    - 字符串→文本框
    - 任何数值→文本框将条目限制为对数字子类型有效的值（例如，一个字节可以保存从-128到127的值）。BigDecimal和BigInteger数据类型也受支持。请确保已在包配置中导入适当的类。如果上传的POJO模型公开了BigDecimal或BigInteger字段的访问器或增变器，则会自动添加该导入。BigDecimal值自动后缀“B”，指示底层引擎该文字值应该被解释为BigDecimal。BigIntegers后缀为“I”。用户不需要输入后缀。
    - 日期→日历
    - 枚举→列表框
    - 布尔→复选框
  + 一个变量绑定到左侧的事实或字段。被设置的字段的数据类型必须与变量的数据类型匹配。
  + “公式”：这是一个计算的表达式（仅适用于高级用户）
* 从逻辑上插入一个全新的事实（请参阅专家文档中的“真相维护”），并可选择在事实上设置一个字段。
  + 字面值：根据字段的数据类型显示不同的组件：

值字段可以是以下值之一：

* + - 字面值：根据字段的数据类型显示不同的组件：
      * 字符串→文本框
      * 任何数值→文本框将条目限制为对数字子类型有效的值（例如，一个字节可以保存从-128到127的值）。BigDecimal和BigInteger数据类型也受支持。请确保已在包配置中导入适当的类。如果上传的POJO模型公开了BigDecimal或BigInteger字段的访问器或增变器，则会自动添加该导入。BigDecimal值自动后缀“B”，指示底层引擎该文字值应该被解释为BigDecimal。BigIntegers后缀为“I”。用户不需要输入后缀。
      * 日期→日历
      * 枚举→列表框
      * 布尔→复选框
    - 一个变量绑定到左侧的事实或字段。被设置的字段的数据类型必须与变量的数据类型匹配。
    - “公式”：这是一个计算的表达式（仅适用于高级用户）
  + 一个变量绑定到左侧的事实或字段。被设置的字段的数据类型必须与变量的数据类型匹配。
  + “公式”：这是一个计算的表达式（仅适用于高级用户）
* 修改现有事实的一个字段（它告诉引擎事实已经改变）。

值字段可以是以下值之一：

* + 字面值：根据字段的数据类型显示不同的组件：
    - 字符串→文本框
    - 任何数值→文本框将条目限制为对数字子类型有效的值（例如，一个字节可以保存从-128到127的值）。BigDecimal和BigInteger数据类型也受支持。请确保已在包配置中导入适当的类。如果上传的POJO模型公开了BigDecimal或BigInteger字段的访问器或增变器，则会自动添加该导入。BigDecimal值自动后缀“B”，指示底层引擎该文字值应该被解释为BigDecimal。BigIntegers后缀为“I”。用户不需要输入后缀。
    - 日期→日历
    - 枚举→列表框
    - 布尔→复选框
  + 一个变量绑定到左侧的事实或字段。被设置的字段的数据类型必须与变量的数据类型匹配。
  + “公式”：这是一个计算的表达式（仅适用于高级用户）
* 在事实上设置一个字段（在这种情况下，引擎不知道更改 - 通常是因为您正在设置结果）。

值字段可以是以下值之一：

* + 字面值：根据字段的数据类型显示不同的组件：
    - 字符串→文本框
    - 任何数值→文本框将条目限制为对数字子类型有效的值（例如，一个字节可以保存从-128到127的值）。BigDecimal和BigInteger数据类型也受支持。请确保已在包配置中导入适当的类。如果上传的POJO模型公开了BigDecimal或BigInteger字段的访问器或增变器，则会自动添加该导入。BigDecimal值自动后缀“B”，指示底层引擎该文字值应该被解释为BigDecimal。BigIntegers后缀为“I”。用户不需要输入后缀。
    - 日期→日历
    - 枚举→列表框
    - 布尔→复选框
  + 一个变量绑定到左侧的事实或字段。被设置的字段的数据类型必须与变量的数据类型匹配。
  + “公式”：这是一个计算的表达式（仅适用于高级用户）
* 从引擎的工作内存中删除一个事实。
* 向现有全球列表添加事实。
* 调用一个变量的方法。
* 写一大块自由形式代码。

#### [21.2.4。可选属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditoroptionalattributes)

规则的属性部分提供了定义元数据和属性的方法（如“显着性”，“无回路”等）。

点击“+”图标添加新的元数据或属性定义。每个定义将出现在本节中。

点击每个元数据或属性旁边的“ - ”图标将其删除。

##### [21.2.4.1。凸显](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditoroptionalattributessalience)

每个规则都有一个突出值，该值是一个默认为零的整数值。显着性值代表具有较高优先级的较高显着性规则的优先级。显着性值可以是正值或负值。

#### [21.2.5。模式/操作工具栏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditortoolbar)

G：在每个模式或动作旁边，你会发现一个包含3个按钮的工具栏。

第一个“+”图标可让您在任意位置插入新的模式/动作。其他“+”图标允许您在所选的下方插入新的模式/动作。

剩余的箭头图标可以让您将当前的模式/动作向上或向下移动。

#### [21.2.6。用户驱动的下拉列表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorenumerations)

*图326.显示为下拉列表的数据枚举*

请注意，可以将字段值限制为预配置列表中的项目。该列表或者由Java枚举定义，或者被配置为包的一部分（使用数据枚举为下拉列表提供值）。这些值可以是固定列表，或者（例如）从数据库加载。这对代码和其他设置了值的字段很有用。也可以在屏幕上显示的内容与规则中使用的值（或代码）不同。有关数据枚举的配置方式，请参阅数据枚举部分。

可以在同一事实（例如取决于所选“国家地区”的“城市”列表）上定义依赖于一个或多个其他字段的值的一个字段的值的列表。有关更多信息，请参阅“枚举”部分。

#### [21.2.7。增加DSL语句](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditordsl)

如果包规则是DSL配置的一部分，那么当您添加条件或操作时，它将提供一个“DSL句子”列表，您可以从中进行选择 - 当您选择一个时，它会将一行添加到规则 - DSL指定值来自用户，然后会显示编辑框（文本）（因此它最终看起来有点像表单）。这是可选的，还有另一个DSL编辑器。请注意，这个编辑器中的DSL功能比全部的DSL功能略少一些（基本上你可以在[只有]和[然后] DSL的部分 - 这与drools 3的效果没有区别）。

下图显示了在指导编辑器中正在使用的DSL句子：

*图327.在指导编辑器中的DSL*

#### [21.2.8。一个更复杂的例子：](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruleeditorcomplexexample)

*图328.更复杂的BRL示例*

在上面的示例中，您可以看到如何使用条件元素，文字值和公式的混合。该规则有4个“顶级”模式和1个动作。“顶级”模式是：

1. 人的事实模式。此模式包含两个字段约束：一个用于birthDate字段，另一个用于公式。请注意，birthDate限制的值是从日历中选择的。另外需要注意的是，您可以进行计算并在公式限制中使用嵌套字段（即car.brand）。最后，我们为Person Fact Type设置一个变量名称（$ p）。然后你可以在其他模式中使用这个变量。

|  |  |
| --- | --- |
|  | 从这个模式生成的DRL将是：  $p : Person( birthDate < "19-Dec-1982" , eval( car.brand == "Ford" && salary > (2500 \* 4.1) )) |

1. A从模式。此条件将为每个街道名称为“Elm St.”的地址创建一个匹配项 从Person的地址列表中删除。from的左侧是一个常规的事实模式，右侧是一个表达式生成器，可以让我们检查变量的字段。

|  |  |
| --- | --- |
|  | 从这个模式生成的DRL将是： Address( street == "Elm St." ) from $p.addresses |

1. 一个“不存在”的条件元素。当其内容不匹配时，该条件将匹配。在这种情况下，其内容是一个常规的事实模式（个人）。在这个事实模式中，您可以看到变量（$ p）如何在公式值内使用。

|  |  |
| --- | --- |
|  | 从这个模式生成的DRL将是： not Person( salary == ( $p.salary \* 2 ) ) |

1. 一个“来自收集”的条件元素。这可能是您可以使用的最复杂模式之一。它包含一个左模式（它必须是一个事实模式，在这种情况下是一个数字模式，该数字命名为$ totalAddresses），一个源模式（可以是一个事实模式，来自，收集或积累条件元素。这种情况是一个地址模式限制，在zip字段中有一个字段限制）和一个公式部分，您可以使用任何内置或自定义累加函数（在本例中使用count（）函数）。基本上，这个条件元素将从Person的地址列表中计算邮政编码为43240的地址。

|  |  |
| --- | --- |
|  | 从这个模式生成的DRL将会是：`$ totalAddresses：Number（）来自$ p.addresses，count（$ a））的accumulate（$ a：Address（zipCode ==“43240”））`` |

### [21.3。资产/规则模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditor)

当您需要定义单个规则时，指导性规则编辑器非常棒，但是如果您需要定义遵循相同结构但在字段限制或操作部分中具有不同值的多个规则，则“规则模板”是非常宝贵的资产。规则模板允许用户定义一个规则结构，其中包含要从数据表中插值的值的占位符。文字值，公式和表达式也可以继续使用。

规则模板通常可以用作Drools Workbench中决策表的替代方案。

#### [21.3.1。创建一个规则模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditorcreating)

要为规则创建模板，只需从“New Item”菜单中选择“Guided Rule Template”。

#### [21.3.2。定义模板](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditordefining)

一旦创建了规则模板，就会显示编辑器。编辑器采用标准指导编辑器的形式，在“规则创作”部分详细解释。按照规则构建，您可以在您的字段限制和操作部分中插入“模板键”作为占位符。文字值，公式和表达式可以像标准指导编辑器一样继续使用。

*图329.模板键弹出*

下面的屏幕截图显示了一个简单的规则，该规则是针对申请人的最大年龄，最低年龄和信用等级使用“模板密钥”定义的。模板键已被分别定义为“$ max\_age”，“$ min\_age”和“$ cr”。

*图330.指导编辑器中的规则模板*

#### [21.3.3。定义模板数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditordefiningdata)

完成规则模板的定义后，您需要输入将用于插入“模板关键字”占位符的数据。Drools Workbench提供了在指导编辑器屏幕内的灵活网格中输入数据的功能。数据输入部分位于编辑器中的“数据”选项卡上。

规则模板数据网格非常灵活; 为底层字段的数据类型提供不同的弹出式编辑器。列可以调整大小和排序; 并且单元格可以合并和分组以便于快速输入数据。

一行数据为单个规则插入“Template Key”占位符; 因此一行成为一条规则。

|  |  |
| --- | --- |
|  | 如果某行的单元格留空，则不会生成适用行的规则。 |

*图331.模板数据网格*

##### [21.3.3.1。单元合并](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditorcellmerging)

网格左上方的图标切换单元格合并的开启和关闭。合并单元格时，将具有相同值的同一列中的单元格合并到一个单元格中。这简化了更改共享相同原始值的多个单元格的值。合并单元格时，它们还会在单元格的左上角获得一个图标，以允许跨合并单元格的行进行分组。

*图332.单元合并*

##### [21.3.3.2。细胞分组](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditorcellgrouping)

已合并的单元格可以进一步折叠为单行。单击合并单元格左上角的[+ \ - ]图标可将相应的行折叠为单个条目。跨越具有相同值的折叠行的其他列中的单元格未更改。跨越具有不同值的折叠行的其他列中的单元格将突出显示，并显示第一个值。

*图333.细胞分组*

当分组单元格的值被更改时，所有已折叠的单元格也会更新其值。

#### [21.3.4。生成的DRL](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedruletemplateeditorgenerateddrl)

虽然没有必要，但规则制定者可以查看将为“规则模板”和相关数据生成的DRL。此功能及其操作与其他资产没有区别。从编辑器屏幕底部选择“源”选项卡。将显示所有规则的DRL。

*图334.生成的DRL*

### [21.4。指导决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditor)

指导决策表编辑器支持创作决策表。规则属性，元数据，条件和动作可以以表格格式定义，从而有助于快速录入大量相关规则。基于Web的决策表规则像所有其他规则资产一样被编译到DRL中。

可以在列定义中使用项目中可用的数据对象，可以使用数据对象编辑器手动创建数据对象，也可以使用添加到项目中的依赖项。

#### [21.4.1。决策表的类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditortypesofdecisiontable)

大体上有两种不同类型的决策表，在Workbench中都支持这两种决策表： -

* 扩展条目
* 有限的条目

##### [21.4.1.1。扩展条目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorextendedentrydecisiontable)

扩展条目决策表是指列定义或存根指定模式，字段和运算符但不包含值的决策表。这些值或状态本身就是决策表的主体。通过限制列表中的值来限制可能值的范围是正常的，但不是必需的。Workbench支持使用Java枚举，Workbench枚举或决策表“可选值列表”来限制值的输入。

*图335. Extended Entry Decision表*

##### [21.4.1.2。有限的条目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorlimitedentrydecisiontable)

有限条目决策表是列模式，字段和运算符之外的列定义指定的值。在表格主体中保留的决策表是布尔值，其中正值（勾选的复选框）具有表示列应该应用或者匹配的效果。负值（清除的复选框）表示该列不适用。

*图336.限制进入决策表*

##### [21.4.1.3。导航](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditornavigation)

编辑器可以通过按下鼠标左键并拖动来“平移”。也可以通过左键单击表格标题并拖动来重新定位表格。双击表格标题可锁定编辑器中的表格，防止重新定位; 然而，通过平移编辑器，可以将编辑器可见边界溢出的内容滚动到视图中。

细胞可以以多种方式选择： -

* 首先可以双击单个单元格，并显示与底层数据类型相对应的内嵌单元编辑器。可以通过点击第一个单元格，然后在按下Shift键的同时单击所需范围的范围来选择同一列中的单元格组。
* 其次，键盘光标键可用于在桌子周围导航。按回车键将弹出相应的编辑器。通过按住Shift键可以选择范围，同时用光标键扩展范围。

通过将鼠标悬停在表格主体中相应的分隔线上可以调整列的大小。鼠标指针将改变，然后列宽被拖曳到更窄或更宽。

可以通过将鼠标悬停在表格标题中的相应标题上来移动列。鼠标指针将会改变，表明可以进行操作，然后可以将列拖动到新的允许位置。个别字段上的约束可以在其父模式中重新定位。通过拖动模式标题可以重新定位单个模式的列块。

###### [菜单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_menus)

[菜单栏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_menu_bar_2)

原理菜单栏包含用于执行以下操作的项目：

* 保存

保存编辑器中所做的更改。

* 删除

完全删除表格。

* 改名

重命名表格。

* 复制

复制文件。

* 验证

验证从表中生成的规则。

* 编辑

剪切，复制，粘贴，删除表格中选定的单元格。

* 视图

调整视图的“缩放”级别并在合并状态之间切换。审核日志也可以查看。

* 插

插入行或列。

* 雷达

雷达显示编辑器的“迷你视图”。在编辑器中可见的表格和区域仅由矩形表示。用户可以拖动可见区域矩形来重新定位编辑器的可见范围。

*图337.雷达*

* 版本历史

显示表格的不同版本。

[上下文菜单](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_context_menu)

在桌子上方按下鼠标右键会显示一个上下文菜单。上下文菜单根据鼠标指针悬停在单元格上还是第一个行号列上而不同。然而，每个可用选项都是一致的，为用户提供了剪切，复制，插入或删除项目的功能。

*图338.上下文菜单*

##### [21.4.1.4。单元合并](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorcellmerging)

“查看”→“合并”菜单项打开和关闭单元格合并。合并单元格时，将具有相同值的同一列中的单元格合并到一个单元格中。这简化了更改共享相同原始值的多个单元格的值。合并单元格时，它们还会在单元格的左上角获得一个图标，以允许跨合并单元格的行进行分组。

*图339.单元合并*

##### [21.4.1.5。细胞分组](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorcellgrouping)

已合并的单元格可以进一步折叠为单行。单击合并单元格右上角的三角形图标会将相应的行折叠为单个条目。跨越具有相同值的折叠行的其他列中的单元格未更改。跨越具有不同值的折叠行的其他列中的单元格将突出显示，并显示第一个值。

*图340.小区分组*

当分组单元格的值被更改时，所有已折叠的单元格也会更新其值。

#### [21.4.2。定义一个基于Web的决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditordefining)

##### [21.4.2.1。定义表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_the_table)

可以使用向导通过创建列或手动指导用户来定义表。该向导仅支持简单的列定义; 如果用户需要更复杂的列定义，则需要在初始创建后手动添加。

当创建新的空白决策表时，用户需要为事实，它们的约束和相应的操作定义列。表总是有两列自动创建并且不能被删除：行/规则编号的列和可以提供规则描述的列。

##### [21.4.2.2。使用向导定义表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizard)

向导可用于帮助定义决策表列。

首次选择创建新规则时可以选择向导。该向导提供了多个页面来定义表格： -

* 概要
* 添加事实模式
* 添加约束
* 添加操作以更新事实
* 添加操作以插入事实
* 要展开的列

###### [选择向导](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorselectingwizard)

“新建向导”对话框显示“使用向导”复选框。

*图341.选择向导*

###### [摘要页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizardsummarypage)

摘要页面显示了有关决策表的一些基本细节，并允许更改资产名称。

*图342.摘要页面*

###### [添加事实模式页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizardfactpatternspage)

此页面允许定义事实类型，以形成规则的“时间”列。在模型中可用的事实类型将显示在左侧的列表框中。选择一个事实类型并使用“>>”按钮将其添加到右侧列表框中所选事实的列表中。

移除是一个类似的过程：不再需要的事实可以在右侧列表框中选择，用“<<”按钮移除它。所有事实类型都需要绑定到一个变量。

*图343.添加事实模式页面*

###### [添加约束页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizardaddconstraintpage)

此页面允许用户在决定表中选择用于定义的事实类型的字段限制。

在上一个向导页面上选择的事实类型列在右侧的列表框中。通过单击选择一个事实类型将导致在中间列表框中显示可用字段的列表以及创建不需要特定字段的谓词的选项。

点击字段，然后点击“>>”按钮，可以将字段添加到模式的约束中。通过单击右侧列表框中的条件然后单击“<<”按钮，可以从模式定义中删除字段。

所有字段都需要有一个列标题和运算符。

不完整的字段将突出显示并显示警告消息。用户将无法完成定义，直到所有警告均已解决。

*图344.添加约束页面*

###### [添加操作以更新事实页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizardupdatefactspage)

已经定义的事实类型可以在规则的后果或动作中进行更新。此页面允许定义这些操作。

添加到决策表定义的事实类型列在左侧列表框中。通过点击选择事实类型将导致在中间列表框中显示可用字段的列表。

需要通过规则更新的字段可以通过选择可用字段并按下“>>”按钮来添加。可以通过点击选择的字段，然后点击“<<”按钮来删除字段。

所有操作都需要列标题。

任何不完整的动作将被突出显示并显示一条警告消息。用户将无法完成定义，直到所有警告均已解决。

*图345.添加操作以更新事实页面*

###### [添加操作以插入事实页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizardinsertfactspage)

也可以定义操作以将新事实插入规则引擎。

左侧列表框中列出了模型中可用的事实类型列表。通过点击它们并按下最左边的列表框和标题为“选择的图案”之间的“>>”按钮，选择用户希望包括在决策表定义中的用户。

去除是一个类似的过程，通过按“<<”按钮可以选择和去除选定的图案。

选择模式的选择会向用户显示可用字段的列表。需要通过动作设置值的字段可以通过选择它们并按下“可用字段”和“选择字段”列表框之间的“>>”按钮来添加。

去除过程与已经描述过的过程类似。

新事实需要绑定到一个变量，并有一个列标题指定。

不完整的事实和\或字段将突出显示并显示警告消息。用户将无法完成定义，直到所有警告均已解决。

*图346.添加操作以插入事实页面*

###### [列展开页面](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorwizardexpandcolumnspage)

该页面控制如何创建基于先前页面上定义的条件的决策表。

使用可选值列表定义的条件列可用于在决策表中创建行。在许多条件列已被定义为许可值列表的情况下，结果表将包含每一组值的一行; 即决策表将以扩展形式。

默认情况下，所有使用值列表定义的条件列都将包含在扩展中，但如果需要，用户可以选择列的子集。这可以通过取消选中“完全展开”复选框并将列添加到右侧列表框来完成。

如果不需要扩展，请取消选中“完全展开”复选框，并确保将零列添加到右侧列表框中。

*图347.列展开页面*

##### [21.4.2.3。手动添加列](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditordefiningmanuallycolumns)

如果选择在不借助向导的情况下创建表格，则首先将空表格呈现给用户。要添加一列，点击“添加列”按钮启动“新建列向导”。

*图348.新建列向导*

###### [简单的列类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorsimplecolumns)

默认情况下，“新建列向导”仅显示以下简单类型： -

* 添加一个简单的条件
* 删除现有的事实
* 设置一个字段的值

[简单的条件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorsimplecondition)

条件表示对规则的左侧定义的事实模式或“何时”部分的约束。要定义条件列，用户必须首先选择或定义绑定到模型类的Fact Pattern。

用户可以选择否定该模式。

一旦完成，用户可以定义字段约束。

如果使用相同的事实模式绑定定义两个或更多列，则字段约束将成为同一模式上的复合字段约束。

如果用户为单个模型类定义多个绑定，则每个绑定将成为规则左侧的单独模型类。

当用户编辑或创建一个新列时，他们将得到一个约束类型的选择： -

* 文字：单元格中的值将与使用操作符的字段进行比较。
* 公式：将评估单元格中的表达式，然后与字段进行比较。
* 谓词：不需要字段，表达式将被评估为真或假。

[“否则”的操作](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorotherwiseoperation)

使用等于（==）或不等式（！=）运算符的字面值定义的条件列可以利用“otherwise”的特殊决策表单元格值。这个特殊值允许定义一个规则，该规则与在表中定义的所有其他规则中未明确定义的所有值相匹配。举个例子来说明这一点：

**when**

Cheese( name **not** **in** ("Cheddar", "Edam", "Brie") )

...

**then**

...

**end**

**when**

Cheese( name **in** ("Cheddar", "Edam", "Brie") )

...

**then**

...

**end**

[删除现有的事实](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditordeletefact)

删除绑定事实的操作。

[设置一个字段的值](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.decisiontablesetfield)

一个Action，用于设置先前绑定事实或事实的新实例上的字段的值。用户可以选择通知规则引擎修改后的值，这可能导致其他规则被重新激活。

用户可以选择让新的事实“逻辑插入”，这意味着如果导致被调用的行为的条件停止为真，它将被自动删除。

请参阅Drools Expert文档以获取有关真相维护和逻辑插入的详细信息。

###### [高级列类型](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditoradvancedcolumns)

点击“包含高级选项”，为更高级的用例添加以下额外的“高级”列类型： -

* 添加条件BRL片段
* 添加一个新的属性列
* 添加一个新的元数据列
* 添加一个Action BRL片段
* 执行工作项目
* 使用工作项参数设置字段的值

[条件BRL碎片](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorconditionbrl)

允许在规则的左侧使用BRL片段的构造。使用引导规则编辑器创作BRL片段，因此该编辑器中可用的所有功能都可用于定义决策表列; 如“来自”，“收集”和“积累”等。

当使用定义为“模板键”的嵌入式引导规则编辑器字段值时，将在决策表中形成列。

在BRL片段中绑定的事实和事实字段可以被更简单的列类型引用，反之亦然。

在下面的例子中，已经定义了两个模板键，因此决策表中出现了两列。

|  |  |
| --- | --- |
|  | 用户也可以使用“Free Form DRL”来定义条件列。如果用户需要一个占位用于可变在DRL用户可以使用的Drools的模板语法@{variable-name}例如Smurf( name = "@{name}" )。 |

*图349.用BRL定义一个条件*

*图350.最终的决策表*

[属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorattributecolumns)

可以添加代表任何DRL规则属性的零个或多个属性列（例如，突出显示，定时器，启用等）。在指导决策表编辑器中提供额外的伪属性来“否定”规则。使用此属性可以取消完整的规则。例如，下面的简单规则也可以被否定，如图所示。

**when**

$c : Cheese( name == "Cheddar" )

**then**

...

**end**

**when**

**not** Cheese( name == "Cheddar" )

**then**

...

**end**

[元数据](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditormetadatacolumns)

可以定义零个或多个元数据列，每个列表示DRL规则中的正常元数据注释。

[操作BRL片段](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditoractionbrl)

允许在规则的右侧使用BRL片段的构造。使用引导规则编辑器创作BRL片段，因此该编辑器中可用的所有功能都可用于定义决策表列。当使用定义为“模板键”的嵌入式引导规则编辑器字段值时，将在决策表中形成列。BRL片段中绑定的事实可以通过更简单的列类型引用，反之亦然。

|  |  |
| --- | --- |
|  | 用户也可以使用“Free Form DRL”来定义Action列。如果用户在DRL中需要一个变量的占位符，他们可以使用@{variable-name}例如Drools Templates语法$s.setName( "@{name}" )。 |

在下面的例子中，已经定义了两个模板键，因此决策表中出现了两列。

*图351.用BRL定义一个动作*

*图352.结果决策表*

[执行工作项目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorexecuteworkitem)

调用jBPM Work Item Handler的Action设置其输入参数以绑定事实\事实字段值。

[使用工作项参数设置字段的值](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorsetworkitemparameter)

一个Action将现有事实或新事实的字段的值设置为jBPM工作项处理程序的结果参数的值。

|  |  |
| --- | --- |
|  | 通过撤销权限“编辑指导决策表”列，可以禁用用户的列定义创建和编辑。这些用户仍然可以修改决策表数据，但他们将无法更改决策表的结构，以避免意外中断其功能的情况。  请参阅[安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.userandgroupmgmt)以检查如何在常规工作台权限部分禁用它。 |

##### [21.4.2.4。编辑栏目](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditoreditcolumn)

要编辑列，请单击“编辑列”按钮以显示现有的列定义。点击“编辑”将启动一个窗口来编辑列定义。点击“删除”删除表中的列。

*图353.编辑列*

|  |  |
| --- | --- |
|  | 通过将鼠标指针悬停在列上也可以找到列的定义。将出现一个弹出窗口，显示定义的提示。它被认为是一个提示，因为所有条件都有一个内插的虚拟值。  *图354.列定义提示* |

#### [21.4.3。命中政策](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorhitpolicy)

命中策略决策表可以使用的类似于DMN规范。命中策略声明规则行之间的优先级以及表中有多少规则可以触发。所有命中策略都可以通过为每一行设置正确的激活组和显着值来进行复制，事实上，这就是DRL所做的。

命中策略只能在决策表第一次创建时设置。

*图355.选择Hit Policy*

一旦在编辑器中查看表格。正在使用的命中策略可以在表格的左上角看到。

*图356.命中策略*

* U =独特命中有了独特的命中策略，每一行必须是唯一的，这意味着不能有重叠。如果验证功能在开发时发出警告，则永远不会出现两行可触发的情况。
* F =第一次击中第一次击中只有一行，首先是自上而下满意的一击。
* RC =已解决的命中类似于第一次命中，但您可以例如为第5行提供第10行的优先级。这意味着您可以保持所需行的顺序以实现可视化，但指定优先级异常。
* R =规则顺序多行可以触发，并且Verification不会报告行之间的冲突，因为它们预计会发生。
* ＃=无这是正常的命中模式。旧的决策表默认使用这个，但由于7.0使用了PHREAK，现在行顺序很重要。旧表格不需要迁移工具。多行可以触发。验证会对冲突的行发出警告。

#### [21.4.4。审计日志](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorauditlog)

审计日志已添加到Web指导的决策表编辑器中，以跟踪添加，删除和修改。

默认情况下，审计日志未配置为记录任何事件，但是，用户可以轻松选择他们感兴趣的事件。

只要资产被签入，审计日志就会持续存在。

*图357.一个空的审计日志*

一旦事件捕获被启用，所有可能的操作都被记录下来。用户可以执行以下操作： -

* 在每个事件旁记录一个注释。
* 从日志中删除一个事件。事件详细信息保留在底层存储库中。

*图358.审计事件的示例*

#### [21.4.5。实时验证和验证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverification)

决策表在每个单元更改后进行验证。如果发现任何问题，结果将显示在表格右侧的列中。验证和验证包含以下问题：

##### [21.4.5.1。冗余](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverificationredundancy)

当两行在给定相同的事实集合时执行相同的操作时，两行之间存在冗余。

如果冗余规则在现有事实上设置了一个值，则冗余可能不会成为问题，只需将该值设置两次即可。当两条规则增加计数器或在工作记忆中添加更多事实时会出现问题，因为这会导致不必要的副作用。在这两种情况下，其他行都不需要。

##### [21.4.5.2。涵摄](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverificationsubsumption)

当给定相同的事实集合时，一行执行与另一行相同的行为时存在存在。行不是多余的，因为另一行更加复杂，并且它可以用不会触发subsumptant行的一组事实触发。

包含的问题类似于冗余的情况。

##### [21.4.5.3。冲突](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverificationconflicts)

冲突可能存在于单行或两行之间。

单行冲突会阻止执行行操作。

当两个规则的条件满足一组相同的事实时，两行之间存在冲突，但这些操作将事实字段设置为不同的值。这些条件可能是多余的或者仅仅是潜在的。冲突是一个问题，因为不可能知道最后做了什么动作。条件可能会对已批准和已拒绝的贷款进行贷款。每次运行规则和每个规则引擎软件版本时，最终结果可能会有所不同。

##### [21.4.5.4。缺少列](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverificationmissingrhslhs)

在某些情况下，通常是偶然的，用户可以删除所有条件或操作列。当条件被删除时，所有的动作都会被执行，当动作列被遗漏时，这些行将不起作用。

##### [21.4.5.5。范围检查](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontableeditorverificationmissinrange)

范围检查可帮助您制定更完整的决策表。他们检查表中是否包含对字段可能具有的所有可能值的约束。一旦表格完成，任何给定的事实将会有一个行动。

### [21.5。引导决策表图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontablegrapheditor)

虽然有可能编写单引导决策表，但也可以编写相关表格的图表，其中一个动作可以为另一个的条件提供可能的匹配。在这种情况下，表格被认为是相关的。

#### [21.5.1。编辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editor)

编辑器本身保留了引导式决策表编辑器的所有功能，仅增加了在单个视图中可视化多个表及其关系的功能。选择一个表导致列定义面板，源和数据对象选项卡被更新以反映选定的表。验证和验证报告也会更新以反映所选表格。“概览”选项卡总体上保持图形的状态。

*图359.图形编辑器和雷达*

#### [21.5.2。创建一个新图](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_a_new_graph)

当首次创建新的“指导决策表格图表”时，将扫描目标软件包以查找现有的单一指导决策表格。找到的那些被检查关系，并且如果存在的话，自动添加到新图形。

#### [21.5.3。向图表中添加新表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adding_new_tables_to_the_graph)

通过点击“Documents ...”菜单条目，可以将图表添加到图表中。这揭示了一个工具面板，其中列出了已经是图表一部分的表格。同一个面板使用户能够添加新表格或打开现有表格。

*图360.文档弹出*

##### [21.5.3.1。添加新表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adding_new_tables)

点击“New ...”启动New Guided Decision Table Wizard。在向导完成后，新表将被添加到图中，并自动添加任何关系。

##### [21.5.3.2。添加现有表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adding_existing_tables)

点击“打开...”会显示一个文件选择器，其中可以选择一个或多个现有的指导决策表并将其添加到图形中。如果没有找到附加的指导性决策表，则会向用户显示一条消息。

*图361.文档弹出 - 打开现有的*

#### [21.5.4。删除表格](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_removing_tables)

通过打开“Documents ...”面板并单击表格旁边的“X”图标，可以从图表中删除表格。从图表中删除的表格不会从项目中删除，并且可以继续由Gudied Decision Table Editor进行编辑。

### [21.6。指导决策树](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guideddecisiontreeeditor)

Workbench支持创建简单的决策树。

|  |  |
| --- | --- |
|  | 编辑器目前不支持嵌套的数据对象。因此建议仅使用平面数据对象模型的指导决策树。 |

#### [21.6.1。最初的编辑布局](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_initial_editor_layout)

当创建新的指导决策树时，编辑器最初是空白的。

左侧是可用数据对象，其字段和操作的调色板。

右侧是可以拖放数据对象，其字段或操作以构建树的区域。

编辑器将显示正在拖动的节点和可以连接的适用子节点之间的连接器。当拖动完成时，新节点将被附加到适用的孩子。将根节点拖到空树时不会显示连接器。完成拖动将位于编辑器中心的根节点。

编写树时有各种限制： -

1. 树必须在根上有一个数据对象。
2. 一棵树只能有一个根。
3. 数据对象可以具有其他数据对象，字段约束或操作作为子对象。

字段约束必须位于与父节点相同的数据对象的字段中。

1. 字段约束可以具有其他字段约束或Actions作为子字段。

字段约束必须位于与父节点相同的数据对象的字段中。

1. 行动只能将其他行为视为孩子。

*图362.指导决策树 - 空编辑器*

展开调色板显示数据对象及其字段的树节点。

*图363.指导决策树 - 扩展调色板*

#### [21.6.2。第一步](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_first_steps)

将数据对象拖动到树创作区域。

*图364.指导决策树 - 数据对象根节点*

点击一个节点选择它。

操作节点的图标显示在节点被选中时。

这些图标是：

1. 删除

删除一个节点也会删除所有的孩子。

1. 编辑

折叠的节点无法编辑，因为它们包含许多子节点。

1. 坍方

*图365.指导决策树 - 选定节点*

#### [21.6.3。编辑数据对象节点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editing_data_object_nodes)

选择数据对象节点并单击编辑图标将显示一个弹出窗口来管理节点。

弹出窗口显示数据对象类型并允许将其绑定到变量。绑定数据对象可以通过操作进行修改或收回。

*图366.指导决策树 - 数据对象编辑器*

#### [21.6.4。编辑字段约束节点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editing_field_constraint_nodes)

选择一个字段约束节点并单击编辑图标将显示一个弹出窗口来管理该节点。

弹出窗口显示数据对象的类型和字段，并允许该字段绑定到变量。可以选择适用于数据模型字段的数据类型的运算符，并输入相应的值。

*图367.导向决策树 - 字段约束编辑器*

#### [21.6.5。编辑动作节点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_editing_action_nodes)

选择插入操作节点并单击编辑图标将显示一个弹出窗口来管理节点。

弹出窗口允许选择要插入的数据对象以及它的插入是否为“逻辑”。有关更多信息，请参阅有关真相维护的Drools文档。新数据对象的字段可以设置值。

*图368.指导决策树 - 操作“插入”编辑器*

选择插入缩进节点并单击编辑图标将显示一个弹出窗口来管理节点。

该弹出框允许选择任何数据对象从选定节点到根节点的路径中进行回缩。

*图369.指导决策树 - 操作“撤回”编辑器*

选择“插入更新”节点并单击编辑图标将显示一个弹出窗口来管理节点。

弹出窗口允许修改从所选节点到根节点的路径中绑定的任何数据对象。修改后的数据对象的字段可以设置值。

*图370.指导决策树 - 操作“更新”编辑器*

#### [21.6.6。管理树](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_the_tree)

即使简单的树木也会变大，变得难以维护。

因此可以折叠树的一部分，在用户界面中留出更多空间来维护树的不同部分。

如果一个节点有子节点，当它被选中时，它将有一个图标来折叠子节点。点击此图标会折叠儿童。

折叠节点可以通过选择并单击exapnd图标同样进行扩展。折叠节点无法编辑，因为它包含许多子节点。删除折叠的节点也会删除所有的孩子。

*图371.指导决策树 - 折叠节点*

*图372.指导决策树 - 折叠节点*

### [21.7。电子表格决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.xlsdecisiontable)

多个规则可以存储在电子表格中。电子表格中的每一行都是一条规则，每一列都是条件，操作或选项。本文档的Drools Expert部分更详细地讨论电子表格决策表。

要使用电子表格，请上传XLS或XLSX文件。要创建一个新的决策表：启动新的“决策表（电子表格）”向导，您将可以选择上传一个。

*图373.电子表格决策表*

#### [21.7.1。转换为指导决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_convert_to_guided_decision_table)

通过单击XLS决策表编辑器中的“转换”菜单项，可将XLS决策表转换为导向决策表。根据XLS文件的内容创建一个或多个文件。转换结果会报告给用户。

*图374.电子表格转换结果*

|  |  |
| --- | --- |
|  | DateXLS文件中的字段格式必须符合为Drools配置的格式并由系统属性设置drools.dateformat。默认设置是dd-MMM-yyyy |

### [21.8。记分卡](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditor)

记分卡是用于计算总分的公式的图形表示。记分卡可以用来预测某种结果的可能性或可​​能性。Drools现在支持添加计分卡。加法计分卡通过将分配给各个规则条件的所有部分分数相加来计算总分数。

此外，Drools记分卡将允许设置原因代码，这有助于确定有助于总分的特定规则（桶）。Drools记分卡将基于PMML 4.1标准。

New Item菜单现在允许创建记分卡资产。

*图375.记分卡资产导向编辑器*

上图显示了具有一个特征的记分卡。每个记分卡由两部分组成（a）设置参数（b）特性部分

#### [21.8.1。（a）设置参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditorsetupparameters)

设置部分由定义此计分卡整体行为的参数组成。

1. 事实：此下拉列表显示了此资产可见的事实列表。
2. 合成分数字段：显示所选事实的字段列表。只显示'double'类型的字段。如果此下拉列表为空，请检查您的事实模型。最终计算得分将存储在此字段中。
3. 初始分数：数字文本字段捕获初始分数。生成的规则将使用此分数初始化“成绩分数字段”，然后在总结部分分数时将其添加到总分中。
4. 使用原因码：布尔指标以及最终得分计算原因码。在此字段中选择是/否将启用/禁用“结果原因代码字段”，“原因代码算法”和“基线分数”字段。
5. 结果原因代码字段：显示所选事实的字段列表。仅显示类型为'java.util.List'的字段。此集合将保存此记分卡选择的原因代码。
6. 原因代码算法：可以是“none”，“pointsAbove”或“pointsBelow”，描述相对于每个特征的基线得分或者在顶级记分卡中设置的原因代码的排名方式。
7. 基线分数：确定原因代码排名时，用作所有特征的基准比较分数的单个值。或者，可以为每个个体特征设置独特的基线分数，如下所示。只有当UseReasonCodes为“true”且未给出每个特性的baselineScore时，此值才是必需的。

|  |  |
| --- | --- |
|  | 如果UseReasonCodes为“true”，则必须在记分卡级别或每个特性定义BaselineScore，并且必须为每个特性或每个输入属性提供ReasonCode。如果UseReasonCodes为“false”，则不需要baselineScore和reasonCode。 |

#### [21.8.2。（b）特点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditorcharacteristics)

点击“新特性”按钮后，新的空白特征编辑器将添加到记分卡中。为每个记分卡特征（数字或分类）定义点分配策略。每个记分卡特征被分配一个单独的部分分数，用于计算整体分数。总分只是所有部分分数的总和。部分分数被假定为“double”类型的连续值。

##### [21.8.2.1。创造特征](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditorcreatingcharacteristics)

每个记分卡必须至少有一个特征

*图376.新特性*

1. 名称：此特征的描述性名称。仅供参考。
2. Remove Charteristic：在显示确认对话框后，将从记分卡中删除此特征。
3. 添加属性：将添加一个属性（bin）的行条目。
4. 事实：选择要计算部分分数的分数。
5. 特征：显示所选事实的字段列表。只显示“String”，“int”，“double”，“boolean”类型的字段。
6. 基线分数：设置用于比较确定原因代码排名时的实际部分分数的特征基准分数。当useReasonCodes属性为“true”且baselineScore未在元素记分卡中定义时，此值是必需的。每当为特征定义baselineScore时，它优先于在记分卡元素中定义的baselineScore值。
7. 原因代码：包含特征的原因代码，通常与不利的决定相关联。

##### [21.8.2.2。创建属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.guidedscorecardeditorcreatingattributes)

点击'新属性'按钮，一个新的空属性编辑器。在记分卡模型中，定义特定特征属性的所有元素都必须引用单个字段。

*图377.新属性*

1. 操作员：输入属性和部分分数之间的映射发生的条件。操作员下拉菜单将根据所选字段的数据类型显示不同的值。
   1. DataType字符串：“=”，“in”。
   2. DataType整数：“=”，“>”，“<”，“> =”，“⇐”，“> .. <”，“> = .. <”，“> = ..⇐”，“>。 .⇐”。
   3. 数据类型布尔值：“true”，“false”。

有关更多详细信息，请参阅下一小节（值）。

1. 值：根据操作员选择的指定值可以是单个值或用逗号（“，”）分隔的一组值。对于运算符类型布尔值，值字段被禁用。

|  |  |  |  |
| --- | --- | --- | --- |
| *表44.运算符/值* | | | |
| 数据类型 | 操作者 | 值 | 备注 |
| 串 | = | 单一价值 | 将寻找完全匹配 |
| 串 | 在 | 逗号分隔值（a，b，c，...） | 如果字段值包含在逗号分隔的值列表中，则运算符“in”表示评估为TRUE |
| 布尔 | 是真的 | N / A | 值字段不可编辑（只读） |
| 布尔 | 是错误的 | N / A | 值字段不可编辑（只读） |
| 数字 | = | 单一价值 | 等于操作员 |
| 数字 | > | 单一价值 | 比操作员还好 |
| 数字 | < | 单一价值 | 不及运营商 |
| 数字 | > = | 单一价值 | 大于或等于 |
| 数字 | <= | 单一价值 | 小于或等于 |
| 数字 | > .. < | 逗号分隔值（a，b） | （大于值'a'）和（小于值'b'） |
| 数字 | > = .. < | 逗号分隔值（a，b） | （大于或等于值'a'）和（小于值'b'） |
| 数字 | > = .. <= | 逗号分隔值（a，b） | （大于或等于值'a'）和（小于或等于值'b'） |
| 数字 | > .. <= | 逗号分隔值（a，b） | （大于值'a'）和（小于或等于值'b'） |

1. 部分分数：定义授予属性的分数分数。
2. 原因代码：定义属性的原因代码。如果在此级别使用reasonCode属性，则优先于与特性元素关联的ReasonCode。
3. 操作：删除此属性。提示用户进行确认。

|  |  |
| --- | --- |
|  | 如果使用原因代码为“真”，则必须在记分卡级别或每个特性定义基线分数，并且必须为每个特性或其每个输入属性提供原因代码。如果使用原因代码为“false”，则不需要BaselineScore和ReasonCode。 |

### [21.9。测试场景](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditor)

测试场景用于验证规则和知识库是否按预期工作。当知识库发展时，测试情景会防止回归。

*图378.示例测试场景*

给定部分列出了行为所需的事实。期望部分列出了行为完成的预期更改和操作。给定的事实在执行前通过测试场景。在规则执行期间，记录知识库中的更改。在执行结束记录的行动后，将知识库和知识库输出中的现有事实与预期进行比较。

*图379.执行后的示例测试场景*

#### [21.9.1。知识会话选择器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.ksessionselector)

*图380.知识会话选择器*

知识会话选择器用于选择规则运行的会话。知识会话需要是有状态的，并且在使用模拟时间时使用伪时钟。知识会话在项目编辑器的知识库和会话部分中定义。

#### [21.9.2。给定部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorgivensection)

*图381.给出弹出窗口*

* 插入一个新的事实 - 添加一个新的事实，在执行前插入到知识库中。
* 修改现有的事实 - 允许在知识库执行之间编辑事实。
* 删除现有的事实 - 允许在执行之间移除事实。
* 激活规则流程组 - 允许通过预先激活组来测试规则流程组中的规则。

#### [21.9.3。期望部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorexpectedsection)

*图382.预期弹出*

* 规则 - 验证是否触发了某个规则。
* 事实值 - 验证在给定部分中创建的事实的事实值。
* 任何匹配的事实 - 验证知识库中至少有一个事实与指定的字段值。

#### [21.9.4。全局部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorglobalsection)

*图383.全局弹出*

* 全局 - 验证全局字段值。

#### [21.9.5。新的输入部分](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.testscenarioeditorinpuotsection)

*图384.新建输入弹出窗口*

* 对现有事实调用方法 - 从规则执行开始的事实调用方法。

### [21.10。DSL编辑器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.dsleditor)

DSL编辑器允许创作DSL句子。读者应该花时间在Drools Expert文档中探索DSL功能; 因为Drools Workbench的DSL编辑器中的语法是相同的。正常的语法被扩展为提供“提示”来控制如何在用户界面内呈现和验证DSL变量。

支持以下“提示”： -

* {<varName>：<正则表达式>}

当DSL Sentence在指导编辑器中使用时，这将呈现一个文本字段来代替DSL变量。文本字段的内容将根据正则表达式进行验证。

* {<的varName>：ENUM：<factType.fieldName>}

当在导向编辑器中使用DSL Sentence时，这将呈现一个枚举来代替DSL变量。<factType.fieldName>将枚举绑定到模型Fact和Field枚举定义。这可以是“Drools工作台枚举”（即在工作台中定义）或Java枚举（即在模型POJO JAR文件中定义）。

* {<的varName>：日期：<DATEFORMAT>}

当在导向编辑器中使用DSL Sentence时，这将使日期选择器代替DSL变量。

* {<varName>：BOOLEAN：<[checked | 未选中]>}

这将使用下拉选择器代替DSL变量，从而在指导编辑器中使用DSL Sentence时提供布尔选项。

* {<的varName>：CF：<factType.fieldName>}

这将呈现一个按钮，允许您使用自定义窗体设置此变量的值。为了使用此功能，包含factType.fieldName的自定义表单配置的工作集必须处于活动状态。如果没有这样的工作集，则使用简单的文本框（就像常规变量一样）。

有关更多信息，请阅读有关工作集和自定义窗体配置的更多信息。

*图385. DSL规则*

### [21.11。数据枚举（下拉列表配置）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.enumerationeditor)

数据枚举是一种可选的资产类型，技术人员可以配置该类型以提供指导编辑器的下拉列表。这些资源像其他资源一样进行存储和编辑，并应用于它们所属的包。

enum配置的内容是一个Fact.field到下拉列表中要使用的值列表的映射。字符串可以是下拉列表中显示的值，也可以是代码值（规则中使用的结果）和显示值的映射。

*图386.数据枚举*

如果您希望使用规则中使用的值与UI中显示的值之间的映射，则需要使用等号分隔代码值和显示值。例如：

'Person.gender' : ['M=Male','F=Female']

在这个例子F中将用于规则但Female在UI中显示。

下拉列表还可以依赖于其他字段值。

让我们想象一个简单的事实模型，我们有一个名为Vehicle的类，它有两个字段：engineType和fuelType。我们想要选择engineType“汽油”或“柴油”。现在，显然燃料的选择类型必须取决于发动机类型（因此对于Petrol，我们有ULP和PULP，对于Diesel，我们有BIO和NORMAL）。我们可以在枚举中表示这种依赖关系：

这显示了如何使选择依赖于其他字段值。请注意，一旦您选择了engineType，燃料类型的选择列表将被确定。

#### [21.11.1。高级枚举概念](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.advancedenumconcepts)

还有一些其他的高级功能可以用数据枚举来完成。

##### [21.11.1.1。外部数据源](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.advanceenumconcepts.external)

您可以使用辅助类从外部数据源检索列表，而不是定义静态列表。

该助手类必须位于项目的类路径中; 通过添加一个JAR包含该类作为项目依赖项。辅助类必须是可实例化的，并且有一个返回a的非静态方法java.util.List。清单的人口是具体实施。

例如：

'Person.age' : (**new** org.yourco.DataHelper()).getListOfAges()

在其他一些情况下，您可能希望完全从外部数据源（如关系数据库）加载枚举数据。要做到这一点，你可以实现一个返回a的类Map<String, List<String>>。地图的关键在于，Fact.field 值是java.util.List<String>要使用的值之一。

**public** **class** **SampleDataSource2** {

**public** Map<String>, List<String> **loadData**() {

Map data = **new** HashMap();

List d = **new** ArrayList();

d.add("value1");

d.add("value2");

data.put("Fact.field", d);

**return** data;

}

}

枚举定义简单如下（不涉及事实或字段名称）：

=(**new** SampleDataSource2()).loadData()

该=运营商通知工作台从辅助类加载所有枚举数据。

|  |  |
| --- | --- |
|  | 枚举定义被请求在编辑器中使用时，将对助手方法进行静态评估。 |

##### [21.11.1.2。动态查找](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.advanceenumconcepts.dynamic)

以上所有情况都是通过在编辑器需要枚举时静态评估枚举定义来检索值列表。

通过将引用中的辅助类的调用封装在引号中，也可以从辅助类中动态加载依赖的枚举定义。例如：

'Country.region[countryCode]' : '(new org.yourco.DataHelper()).getListOfRegions("@{countryCode}")'

### [21.12。DRL文件](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.drleditor)

DRL文件存储为文本 - 可以在Drools Workbench中进行管理。DRL文件可以包含一个或多个全局变量，规则，函数，枚举，类型声明或DRL语言支持的任何内容。通常情况下，您将使用IDE编辑原始DRL文件，因为它具有所有先进的工具和内容帮助和调试。然而，有时候规则可能需要在Drools Workbench的包中处理相当技术性的东西。在任何典型的规则包中，您通常需要一些“技术规则” - 当然，您可以混合和匹配所有规则类型。

*图387. DRL技术规则*

## [22.工作台集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchintegration)

### [22.1。休息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.workbenchremoteapi)

对知识库的REST API调用允许您管理知识库内容并处理知识库存储库中的静态数据。这些调用是异步的，也就是说，他们在执行调用后继续执行它们的工作。每次调用都会返回作业ID，以允许（在执行REST API调用之后）请求作业状态并验证作业是否成功完成。这些调用的参数以JSON实体的形式提供。

在使用Java代码与REST API进行交互时，可以在[(org.uberfire:)uberfire-rest-client](https://origin-repository.jboss.org/nexus/#nexus-search;quick~uberfire-rest-client)JAR中找到POST操作中使用的各类或各种操作返回的类。下面提到的所有类都可以org.guvnor.rest.client在该JAR 的包中找到。

#### [22.1.1。求职电话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_job_calls)

每个知识库REST调用在发送之后都会返回其作业ID。这是非常必要的，因为调用是异步的，并且您需要能够引用作业来检查其作业的生命周期状态。在其生命周期中，作业可以具有以下状态：

* ACCEPTED：工作被接受并正在处理中
* BAD\_REQUEST：该请求未被接受，因为它包含不正确的内容
* RESOURCE\_NOT\_EXIST：请求的资源（路径）不存在
* DUPLICATE\_RESOURCE：资源已经存在
* SERVER\_ERROR：发生服务器上的错误
* SUCCESS：工作成功完成
* FAIL：工作失败
* DENIED：工作被拒绝了
* GONE：找不到工作ID

在下列情况下，作业可能会发生变化： **作业已被明确删除**作业完成并已从状态缓存中删除（作业在缓存达到其最大容量后从状态缓存中删除）\*\*作业从未存在

job提供以下电话：

**[得到]**/jobs/{jobID}

返回作业状态

**[删除]**/jobs/{jobID}

删除作业：如果作业尚未处理，这将从作业队列中删除作业。但是，这不会取消或停止正在进行的工作

#### [22.1.2。项目调用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_project_calls)

通过对知识库的项目调用，您可以管理Workspace项目。

project提供以下电话：

**[得到]**/projects

获取有关知识商店中所有项目的信息。

**[得到]**/spaces/{spaceName}/projects/{projectName}

获取有关该项目的信息。

**[POST]**/spaces/{spaceName}/projects/{projectName}

在给定的空间创建一个新项目。

**[删除]**/spaces/{spaceName}/projects/{projectName}

从知识库中删除该项目。

**[得到]**/spaces/{spaceName}/projects

获取有关该空间中所有项目的信息

**[POST]**/spaces/{spaceName}/git/clone

从提供的git地址克隆一个项目到该空间。

**[删除]**/spaces/{spaceName}/projects/{projectName}

从空间中删除项目。

#### [22.1.3。空间电话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_space_calls)

通过对知识库的空间调用，您可以管理空间。

spaces提供以下电话：

**[得到]**/spaces

获取所有空间。

**[得到]**/spaces/{spaceName}

获得单个空间。

**[POST]**/spaces

创建一个描述的空间 Space

**[删除]**/spaces/{spaceName}

删除空间。

#### [22.1.4。Maven的电话](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_calls)

Maven调用Knowledge Store中的项目可以编译，测试，安装和部署项目。

maven提供以下电话：

**[POST]**/spaces/{spaceName}/projects/{projectName}/maven/compile

编译项目（相当于 mvn compile

**[POST]**/spaces/{spaceName}/projects/{projectName}/maven/test

测试项目（相当于 mvn test

**[POST]**/spaces/{spaceName}/projects/{projectName}/maven/install

安装该项目（相当于 mvn install

**[POST]**/spaces/{spaceName}/projects/{projectName}/maven/deploy

部署该项目（相当于 mvn deploy）

#### [22.1.5。REST总结](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rest_summary)

下表中的网址模板与以下网址相关：

* http://{server}:{port}/kie-wb/rest

#### [22.1.6。嵌入式Kie服务器控制器调用](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_embedded_kie_server_controller_calls)

当使用嵌入式Kie Server Controller模式运行Workbench时，还可以使用与管理Kie Server模板，Kie Server服务器实例和Container的所有方面有关的一系列端点。有关更多详细信息，请参阅[Controller REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_controller_rest_api)。一个[Java客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_controller_client_api)也可以与这些端点进行交互。

### [22.2。Keycloak SSO集成](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.keycloakssointegration)

单点登录（SSO）和相关令牌交换机制正在成为Web上不同环境中身份验证和授权的最常见方案，尤其是在迁移到云中时。

本节讨论Keycloak与jBPM或Drools应用程序的集成，以便使用Keycloak上提供的所有功能。Keycloak是用于浏览器应用程序和RESTful Web服务的集成SSO和IDM。在[Keycloak的主页中](http://keycloak.jboss.org/)进一步了解它。

与Keycloak集成的结果具有许多优点，例如：

* 为不同的客户提供集成的SSO和IDM环境，包括jBPM和Drools工作台
* 社交登录 - 使用您的Facebook，Google，LinkedIn等帐户
* 用户会话管理
* 还有更多...

下一部分将介绍Keycloak的以下集成点：

* **通过Keycloak服务器进行工作台认证**

它基本上包括通过Keycloak SSO保护Web客户端和远程服务客户端。因此Web界面或远程服务消费者（无论是用户还是服务）都将通过KC进行身份验证。

* **通过Keycloak服务器执行服务器认证**

包括保护执行服务器提供的远程服务（因为它不提供Web界面）。任何远程服务使用者（无论是用户还是服务）都将通过KC进行身份验证。

* **消费远程服务**

本节介绍第三方客户端如何使用Workbench和Execution Server提供的远程服务端点（如REST API或远程文件系统服务）。

* **Keycloakd和Workbench的安全管理区域**

#### [22.2.1。脚本](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_scenario)

考虑下面的图表作为本文档示例的环境：

Keycloak是一个独立的进程，提供远程认证，授权和管理服务，这些服务可能会被网络上的一个或多个jBPM应用程序占用。

考虑构建这种环境的这些主要步骤：

* 安装并设置Keycloak服务器
* 为此示例创建并设置Realm - 配置Realm的客户端，用户和角色
* 安装并设置SSO客户端适配器和jBPM应用程序

注意：本文档的结果环境和不同配置基于jBPM（KIE）工作台，但同样适用于KIE Drools Workbench。

#### [22.2.2。安装并设置Keycloak服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_install_and_setup_a_keycloak_server)

Keycloak提供了大量文档和几篇关于在不同环境下安装的文章。本节介绍如何为示例构建集成环境的最小设置。如果您需要更多信息，请参阅[Keycloak文档](http://www.keycloak.org/documentation.html)。

以下是最小Keycloak安装和设置步骤：

* 从[下载](http://www.keycloak.org/downloads.html)部分下载最新版本的Keycloak 。这个例子基于Keycloak 1.9.0.Final
* 将下载的Keycloak分发解压缩到一个文件夹中，让我们将其作为参考

$KC\_HOME

* 运行KC服务器 - 此示例基于在同一主机上运行Keycloak和jBPM。为了避免端口冲突，您可以使用Keycloak服务器的端口偏移：

$KC\_HOME/bin/standalone.sh -Djboss.**socket**.binding.port-offset=100

* 创建Keycloak的管理员管理员用户，方法是导航到[http：// localhost：8180 / auth /](http://localhost:8180/auth/)

Keycloak管理控制台将在[http：// localhost：8180 / auth / admin /中提供](http://localhost:8180/auth/admin/)。

#### [22.2.3。创建并设置演示领域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_create_and_setup_the_demo_realm)

安全领域用于限制不同应用程序资源的访问。

一旦Keycloak服务器正在运行，下一步就是创建一个领域。这个领域将为jBPM应用程序提供不同的用户，角色，会话等。

Keycloak为领域创建和管理提供了几个例子，从[官方的例子](https://github.com/keycloak/keycloak/tree/master/examples)到有更多例子的不同文章。

按照以下步骤创建本文后面使用的演示领域：

* 转到Keycloak [管理控制台](http://localhost:8180/auth/admin)，然后单击添加领域按钮。给它一个名字演示。
* 转到“客户”部分（从主管理控制台菜单中）为演示领域创建一个新客户端：
  + 客户端ID： kie
  + 客户端协议：openid连接
  + 访问类型：机密
  + 根URL：[*http：// localhost：8080*](http://localhost:8080/)
  + 基本网址： / kie-wb-xyzFinal
  + 重定向URI：/ kie-wb-xyzFinal / \*

由此产生的kie客户端设置屏幕：

|  |  |
| --- | --- |
|  | 正如你在上面的设置中看到的那样，它被认为是应用程序上下文路径的值kie-wb-xyzFinal。如果您的jBPM应用程序将部署在不同的上下文路径，主机或端口上，请在此处使用您的具体设置。 |

能够从jBPM工作台使用演示领域的最后一步是创建应用程序的用户和角色：

* 转到Roles部分并创建角色admin，kiemgmt和rest-all
* 转到用户部分并创建管理员用户。在凭证选项卡中使用值密码设置密码，取消设置临时开关。
* 在用户部分导航到角色映射选项卡，并将admin，kiemgmt和其余所有角色分配给管理员用户

此时Keycloak服务器正在主机上运行，​​使用最低配置集进行设置。我们转到jBPM工作台设置。

#### [22.2.4。安装并设置jBPM Workbench](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_install_and_setup_jbpm_workbench)

在本教程中，让我们使用Wildfly作为jBPM工作台的应用程序服务器，因为jBPM安装程序默认是这样做的。

假设运行jBPM安装程序后，将$ JBPM\_HOME作为已部署应用程序的Wildfly服务器的根路径。

##### [22.2.4.1。安装KC适配器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_install_the_kc_adapter)

为了使用jBPM应用程序中的[Keycloak](http://www.keycloak.org/docs/latest/securing_apps/topics/oidc/java/jboss-adapter.html)身份验证和授权模块，必须在我们的$ JBPM\_HOME服务器上安装[用于Wildfly](http://www.keycloak.org/docs/latest/securing_apps/topics/oidc/java/jboss-adapter.html)的[Keycloak适配器](http://www.keycloak.org/docs/latest/securing_apps/topics/oidc/java/jboss-adapter.html)。Keycloak为不同容器提供多种适配器，如果您使用其他容器或需要使用其他适配器，请查看Keycloak文档中的[安全应用程序部分](http://www.keycloak.org/docs/latest/securing_apps/index.html)。以下是安装和设置Wildfly 8.2.x适配器的步骤：

* 从[这里](https://repository.jboss.org/nexus/service/local/repositories/central/content/org/keycloak/keycloak-wf8-adapter-dist/1.9.0.Final/keycloak-wf8-adapter-dist-1.9.0.Final.zip)下载适配器
* 在您的shell上执行以下命令：
* cd $JBPM\_HOME/unzip keycloak-wf8-adapter-dist.zip // **Install** the KC **client** adapter
* cd $JBPM\_HOME/**bin**
* ./**standalone**.sh -**c** **standalone**-**full**.**xml** // Setup the KC **client** adapter.
* // \*\* Once **server** **is** up, **open** a **new** command line terminal **and** run:
* cd $JBPM\_HOME/**bin**

./jboss-cli.sh -**c** *--file=adapter-install.cli*

##### [22.2.4.2。配置KC适配器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_configure_the_kc_adapter)

将KC适配器安装到Wildfly后，下一步是配置适配器以指定不同的设置，例如身份验证服务器的位置，要使用的领域等。

Keycloak提供了两种配置适配器的方法：

* 每个WAR配置
* 通过Keycloak子系统

在这个例子中，让我们使用第二个选项，使用Keycloak子系统，这样我们的WAR就没有这种设置。如果你想使用每个WAR方法，请看看[这里](http://www.keycloak.org/docs/latest/securing_apps/topics/oidc/java/jboss-adapter.html)。

编辑配置文件$ JBPM\_HOME / standalone / configuration / standalone-full.xml并找到子系统配置部分。添加以下内容：

<subsystem xmlns="urn:jboss:domain:keycloak:1.1">

<secure-deployment name="kie-wb-x.y.z.Final.war">

<realm>demo</realm>

<realm-public-key>MIIBIjANBgkqhkiG9w0BAQEFAAOCA...</realm-public-key>

<auth-server-url>http://localhost:8180/auth</auth-server-url>

<ssl-required>external</ssl-required>

<resource>kie</resource>

<enable-basic-auth>true</enable-basic-auth>

<credential name="secret">925f9190-a7c1-4cfd-8a3c-004f9c73dae6</credential>

<principal-attribute>preferred\_username</principal-attribute>

</secure-deployment>

</subsystem>

如果您在步骤2中从此文档导入了示例json文件，则可以使用具体的部署名称使用与上述相同的配置。否则，请使用您的值作为这些配置：

* 安全部署的名称 - 使用具体应用程序的WAR文件名
* 领域 - 在我们的示例中，应用程序将使用的领域是上一步创建的演示领域。
* 领域公钥 - 在这里提供演示领域的公钥。这不是强制性的，如果没有指定，它将从服务器检索。否则，您可以在Keycloak管理控制台→Realm设置（适用于演示领域）→Keys中找到它
* 认证服务器URL - Keycloak认证服务器的URL
* 资源 - 在步骤2中创建的客户端的名称。在我们的示例中，使用值kie。
* 启用基本身份验证 - 对于此示例，我们也启用基本身份验证机制，以便客户端可以使用令牌（承载）和基本方法来执行请求。
* Credential - 使用kie客户端的密码值。您可以在Keycloak管理控制台中找到它→客户端→kie→凭证选项卡→复制秘密值。

对于这个例子，你必须注意使用你的具体值来确定安全部署名称，领域公钥和证书密码。

|  |  |
| --- | --- |
|  | 确保Widfly / EAP配置文件的配置文件（例如：standalone.xml）中不存在以下标记：  <single-sign-on/>  它在某些服务器版本中默认启用。如果存在，必须将其删除/禁用，以便允许Keycloak正确处理客户端。 |

##### [22.2.4.3。运行环境](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_run_the_environment)

此时Keycloak服务器已启动并在主机上运行，​​并且为jBPM应用程序服务器安装并配置了KC适配器。您可以使用以下命令运行应用

$JBPM\_HOME/bin/standalone.sh -c standalone-full.xml

服务器启动后，您可以导航到应用程序：

http:*//localhost:8080/kie-wb-x.y.z.Final*

使用您的Keycloak的管理员用户凭据登录：admin /密码。

#### [22.2.5。通过Keycloak确保工作台远程服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_workbench_remote_services_via_keycloak)

jBPM和Drools工作台都提供了不同的远程服务端点，可供使用[远程API的](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_drools.workbenchremoteapi)第三方客户端使用。

为了通过Keycloak验证这些服务，必须禁用BasicAuthSecurityFilter，从jBPM的WAR文件中对这些WEB-INF / web.xml文件（应用程序部署描述符）应用这些修改：

* 从部署描述符中删除以下过滤器：
* <filter>
* <filter-name>HTTP Basic Auth Filter</filter-name>
* <filter-class>org.uberfire.ext.security.server.BasicAuthSecurityFilter</filter-class>
* <init-param>
* <param-name>realmName</param-name>
* <param-value>KIE Workbench Realm</param-value>
* </init-param>
* </filter>
* <filter-mapping>
* <filter-name>HTTP Basic Auth Filter</filter-name>
* <url-pattern>/rest/\*</url-pattern>
* <url-pattern>/maven2/\*</url-pattern>
* <url-pattern>/ws/\*</url-pattern>

</filter-mapping>

* 将远程服务URL模式约束为：
* <security-constraint>
* <web-resource-collection>
* <web-resource-name>remote-services</web-resource-name>
* <url-pattern>/rest/\*</url-pattern>
* <url-pattern>/maven2/\*</url-pattern>
* <url-pattern>/ws/\*</url-pattern>
* </web-resource-collection>
* <auth-constraint>
* <role-name>rest-all</role-name>
* </auth-constraint>

</security-constraint>

|  |  |
| --- | --- |
|  | 消费远程服务的用户必须是角色全员的成员。如前面所述，admin用户在本例中已经是rest-all角色的成员。 |

#### [22.2.6。通过Keycloak确保工作台的文件系统服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_workbench_s_file_system_services_via_keycloak)

为了使用文件系统等远程服务（例如远程GIT），**必须**在$ JBPM\_HOME / standalone / configuration / standalone-full.xml文件中为应用程序的安全域 **使用**特定的**Keycloak登录模块**。默认情况下，工作台使用另一个安全域，因此$ JBPM\_HOME / standalone / configuration / standalone-full.xml上的结果配置 应如下所示：

<security-domain name="other" cache-type="default">

<authentication>

<login-module code="org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule" flag="required">

*<!-- Parameter value can be a file system absolute path or a classpath (e.g. "classpath:/some-path/kie-git.json")-->*

<module-option name="keycloak-config-file" value="$JBPM\_HOME/kie-git.json"/>

</login-module>

</authentication>

</security-domain>

注意：

* $ JBPM\_HOME / standalone / configuration / standalone-full.xml文件中另一个安全域上的登录模块必须由上面给出的替换。
* 用用于远程服务客户端的json配置文件替换$ JBPM\_HOME / kie-git.json的路径（在文件系统上）或类路径（例如classpath：/some-path/kie-git.json）。请继续阅读以创建此Keycloak客户端以及如何获取此json文件。

此时，使用JAAS进行身份验证过程的远程服务（例如文件系统的远程服务（例如GIT））由Keycloak使用上述json配置文件中指定的客户端加以保护。所以让我们在Keycloak上创建这个客户端并生成所需的JSON文件：

* 导航到[KC管理控制台，](http://localhost:8180/auth/admin)并使用kie-git作为名称为演示领域创建新客户端。
* 启用直接访问授权启用选项
* 禁用标准流量启用选项
* 为此客户使用保密访问类型。以下图为例：
* 在相同的kie-git客户端配置屏幕中转到安装选项卡，并使用Keycloak OIDC JSON类型进行导出。
* 最后，将生成的JSON文件复制到服务器文件系统的可访问目录中，或将其添加到应用程序的类路径中。使用此路径值作为org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule登录模块的上述配置的keycloak-config-file参数。
* 更多关于Keycloak JAAS登录模块的信息可以在[这里](https://keycloak.gitbooks.io/securing-client-applications-guide/content/v/2.2/topics/oidc/java/jaas.html)找到。

此时，所有通过Keycloak服务器验证的用户都可以克隆内部Git存储库：

*# Command example:*

**git** clone ssh://admin@localhost:8001/system

#### [22.2.7。执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_execution_server)

KIE执行服务器提供可供任何第三方客户端使用的[REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.ksrestapi)。本节介绍如何将KIE Execution Server与Keycloak SSO集成，以便将第三方客户端身份管理委派给SSO服务器。

考虑上面的环境运行，所以考虑有：

* Keycloak服务器运行并监听[http：// localhost：8180 / auth](http://localhost:8180/auth)
* 一个名为demo的领域，其中一个名为kie的客户端用于jBPM Workbench
* 运行在[http：// localhost：8080 / kie-wb-xyzFinal上的](http://localhost:8080/kie-wb-x.y.z.Final) jBPM Workbench

按照以下步骤将执行服务器添加到此环境中：

* 在Keycloak上为执行服务器创建客户端
* 安装安装程序和执行服务器（使用KC客户端适配器）

##### [22.2.7.1。在Keycloak上创建执行服务器的客户端](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_create_the_execution_server_s_client_on_keycloak)

根据每个执行服务器的部署情况，您必须在Keycloak 的演示领域创建一个新的客户端：

* 转到[KC管理控制台](http://localhost:8180/auth/admin) →客户端→新客户端
* 名称：kie-execution-server
* 根URL：[*http：// localhost：8280 /*](http://localhost:8280/)
* 客户端协议：openid连接
* 访问类型：机密（如果您需要，可以公开），但不建议用于生产环境）
* 有效的重定向URI：/ kie-server-xyzFinal / \*
* 基本网址：/ kie-server-xyzFinal

在这个例子中，在前面的步骤中创建的admin用户是用于客户端请求的用户。因此，请确保管理员用户是角色kie-server的成员，以便使用执行服务器的远程服务。如果角色不存在，请创建它。

注意：此示例认为执行服务器将配置为使用端口偏移量200运行，因此HTTP端口将在localhost：8280处可用。

##### [22.2.7.2。在执行服务器上安装并设置KC适配器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_install_and_setup_the_kc_adapter_on_the_execution_server)

此时，在KC服务器上准备好一个名为kie-execution-server的客户端，以便从执行服务器使用该客户端。

让我们安装，设置和部署执行服务器：

* 安装另一个Wildfly服务器以用于执行服务器和KC客户端适配器。您可以按照上面的工作台说明进行操作，也可以按照[安全应用指南进行操作](http://www.keycloak.org/docs/latest/securing_apps/index.html)
* 编辑Wildfly服务器的配置路径中的standalone-full.xml文件，并将KC子系统适配器配置为：
* <secure-deployment name="kie-server-x.y.z.Final.war">
* <realm>demo</realm>
* <realm-public-key>MIGfMA0GCSqGSIb...</realm-public-key>
* <auth-server-url>http://localhost:8180/auth</auth-server-url>
* <ssl-required>external</ssl-required>
* <resource>kie-execution-server</resource>
* <enable-basic-auth>true</enable-basic-auth>
* <credential name="secret">e92ec68d-6177-4239-be05-28ef2f3460ff</credential>
* <principal-attribute>preferred\_username</principal-attribute>

</secure-deployment>

考虑你的具体环境设置，如果不同于这个例子：

* 安全部署名称→使用正在部署的执行服务器war文件的名称
* 公钥→使用演示领域公钥或将其保留为空，如果是，则服务器将提供一个
* 资源→这一次，而不是在WB配置中使用的kie客户端，使用kie-execution-server客户端
* 启用基本身份验证→由您决定。您可以为第三方服务使用者启用基本身份验证
* 凭证→为kie-execution-server客户端使用密钥。您可以在KC管理控制台的Credentials选项卡中找到它

##### [22.2.7.3。部署并运行执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_deploy_and_run_the_execution_server)

只需使用任何可用的机制在Wildfly中部署执行服务器即可。使用以下命令运行执行服务器：

$EXEC\_SERVER\_HOME/bin/standalone.sh -c standalone-full.xml -Djboss.socket.binding.port-offset=200 -Dorg.kie.server.id=<ID> -Dorg.kie.server.user=<USER> -Dorg.kie.server.pwd=<PWD> -Dorg.kie.server.location=<LOCATION\_URL> -Dorg.kie.server.controller=<CONTROLLER\_URL> -Dorg.kie.server.controller.user=<CONTROLLER\_USER> -Dorg.kie.server.controller.pwd=<CONTOLLER\_PASSWORD>

例：

$EXEC\_SERVER\_HOME/bin/standalone.sh -c standalone-full.xml -Djboss.socket.binding.port-offset=200 -Dorg.kie.server.id=kieserver1 -Dorg.kie.server.user=admin -Dorg.kie.server.pwd=password -Dorg.kie.server.location=http://localhost:8280/kie-server-x.y.z.Final/services/rest/server -Dorg.kie.server.controller=http://localhost:8080/kie-wb-x.y.z.Final/rest/controller -Dorg.kie.server.controller.user=admin -Dorg.kie.server.controller.pwd=password

|  |  |
| --- | --- |
|  | 将使用执行服务器远程服务端点的用户必须分配角色kie-server。因此，在KC管理控制台中为将使用执行服务器远程服务的用户创建并分配此角色。 |

一旦启动，您可以检查服务器状态（考虑使用此请求的基本身份验证，请参阅下一个[消费远程服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#consumingRemoteServices)以获取更多信息）：

curl http://admin:password@localhost:8280/kie-server-x.y.z.Final/services/rest/server/

#### [22.2.8。消费远程服务](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#consumingRemoteServices)

为了使用由Workbench或执行服务器提供的不同远程服务，您的客户端必须在KC服务器上进行身份验证，并拥有一个有效令牌来执行请求。

请记住，为了使用远程服务，已通过身份验证的用户必须分配：

* 使用WB远程服务的全部角色
* 用于使用Execution Server远程服务的kie-server角色

请确保创建必要的角色并将其分配给使用Keycloak管理控制台上的远程服务的用户。

您有两种选择来使用不同的删除服务端点：

* 使用基本身份验证，如果应用程序的客户端支持它
* 使用基于承载（令牌）的认证

##### [22.2.8.1。使用基本认证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_basic_authentication)

如果KC客户端适配器配置已启用基本身份验证（如本指南中WB（步骤3.2）和Execution Server所述），则可以避免令牌授予/刷新调用，并按以下示例调用服务。

WB远程存储库端点示例：

**curl** http://admin:password@localhost:8080/kie-wb-x.y.z.Final/rest/repositories

检查执行服务器状态的示例：

curl http://admin:password@localhost:8280/kie-server-x.y.z.Final/services/rest/server/

##### [22.2.8.2。使用基于令牌的认证](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#usingTokenBasedAuthentication)

第一步是在Keycloak上创建一个允许第三方远程服务客户端获取令牌的新客户端。它可以做到：

* 转到KC管理控制台并使用此配置创建新客户端：
  + 客户端ID：kie-remote
  + 客户端协议：openid连接
  + 访问类型：公共
  + 有效的重定向URI：[*http：// localhost /*](http://localhost/)
* 由于我们要手动获取令牌并调用服务，因此稍微增加令牌的使用寿命。在生产访问令牌中应该有一个相对较低的超时时间，最好少于5分钟：
  + 转到KC管理控制台
  + 点击你的领域设置
  + 点击代币标签
  + 将Access Token Lifespan的值更改为15分钟。这应该给我们足够的时间来获取令牌并在服务到期之前调用服务。

一旦创建了远程客户端的公共客户端，您现在可以通过对KC服务器的令牌端点执行HTTP请求来获取令牌。以下是命令行的示例：

RESULT=`curl --data "grant\_type=password&client\_id=kie-remote&username=admin&passwordpassword=<the\_client\_secret>" http:*//localhost:8180/auth/realms/demo/protocol/openid-connect/token*`

TOKEN=`echo $RESULT | sed 's/.\*access\_token":"//g' | sed 's/".\*//g'`

此时，如果您回显$ TOKEN，它将输出从KC服务器获取的令牌字符串，该令牌字符串现在可用于授权对远程端点的进一步调用。例如，如果您想检查内部jBPM存储库：

**curl** -H "Authorization: bearer $TOKEN" http://localhost:8080/kie-wb-x.y.z.Final/rest/repositories

#### [22.2.9。Keycloak和工作台的安全管理区域](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_keycloak_and_the_workbench_s_security_administration_area)

jBPM工作台提供了一个提供用户，组和角色管理功能的管理区域（请参阅[安全管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.userandgroupmgmt)）。

默认情况下，应用程序的安全管理系统指向应用程序的服务器领域。例如，在使用Wildfly的打包发行版时，它指向Wildfly的ApplicationRealm（基于属性）。这意味着来自行政管理领域的实体并不是来自应用程序正在使用的Keycloak领域的实体。存在以下选项来更改此默认行为：

* 禁用用户系统管理
* 使用内置的Keycloak安全管理提供程序，而不是默认的

为了定制使用Keycloak安全管理提供商的现有jBPM应用程序（WAR文件），请按照以下步骤操作：

1. 将artifact keycloak-core-1.9.x.Final.jar添加到WEB-INF / lib中
2. 将artifact keycloak-common-1.9.x.Final.jar添加到WEB-INF / lib中
3. 从WEB-INF / lib中删除正在使用的任何安全管理提供程序的实际jar工件（例如：删除WEB-INF / lib / uberfire-security-management-wdilfly-xyZjar）
4. 通过以下方式替换WEB-INF / classes / security-management.properties的内容：

org.uberfire.ext.security.management.api.userManagementServices = KCAdapterUserManagementService

org.uberfire.ext.security.management.keycloak.authServer = <authz\_server\_url>

＃eg：org.uberfire.ext.security.management.keycloak.authServer = http ：//本地主机：8180 / AUTH

1. 更新/META-INF/jboss-deployment-structure.xml以包含/排除以下模块：

<deployment>

<dependencies>

...

<module name =“org.jboss.resteasy.resteasy-jackson-provider”services =“import”/>

...

</ dependencies>

<exclusions>

...

<module name = “org.jboss.resteasy.resteasy-jackson2-provider”/>

...

</ exclusions>

</ deployment>

|  |  |
| --- | --- |
|  | 上述步骤中需要的jar工件可以从[JBoss Nexus](https://repository.jboss.org/nexus/)下载，也可以从[源代码](https://github.com/kiegroup/appformer)构建。 |

应用上述更改后，安全管理区将使用用户会话中存在的访问令牌来授权和管理特定的Keycloak领域数据。

|  |  |
| --- | --- |
|  | 为了能够远程管理Keycloak领域，请确保用户具有分配的领域管理客户端角色 |

## [23.工作台高可用性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.workbenchhighavailability)

### [23.1。VFS集群](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.vfsclustering)

该[VFS库](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.vfsrepository)（通常是git仓库）存储所有的资产（如规则，决策表，流程定义，表单等）。如果该VFS驻留在每个本地服务器上，则必须在群集的所有服务器之间保持同步。

使用[Apache Zookeeper](http://zookeeper.apache.org/)和[Apache Helix](http://helix.incubator.apache.org/) 完成此操作。Zookeeper将所有零件粘合在一起。Helix是注册所有群集细节（节点，资源和群集本身）的群集管理组件。Uberfire（最重要的是构建Workbench）使用这两个组件来提供VFS集群。

要创建VFS集群：

1. 下载[Apache Zookeeper](http://zookeeper.apache.org/)和[Apache Helix](http://helix.incubator.apache.org/)。
2. 同时安装：
   1. 将Zookeeper解压缩到一个目录（$ZOOKEEPER\_HOME）中。
   2. 在$ZOOKEEPER\_HOME，复制zoo\_sample.conf到zoo.conf
   3. 编辑zoo.conf。根据需要调整设置。通常只有这两个属性是相关的：
   4. # the directory where the snapshot is stored.
   5. dataDir=/tmp/zookeeper
   6. # the port at which the clients will connect

clientPort=2181

* 1. 将Helix解压到一个目录（$HELIX\_HOME）中。

1. 在Zookeeper中配置集群：
   1. 转到bin目录：

$ cd $ZOOKEEPER\_HOME/bin

* 1. 启动Zookeeper服务器：

$ sudo ./zkServer.sh start

如果服务器无法启动，请验证dataDir（如所指定的zoo.conf）是否可访问。

* 1. 要查看Zookeeper的活动，请打开zookeeper.out：

$ cat $ZOOKEEPER\_HOME/bin/zookeeper.out

1. 在Helix中配置群集：
   1. 转到bin目录：

$ cd $HELIX\_HOME/bin

* 1. 创建群集：

$ ./helix-admin.sh --zkSvr localhost:2181 --addCluster kie-cluster

该zkSvr值必须与使用的Zookeeper服务器匹配。群集名称（kie-cluster）可以根据需要进行更改。

* 1. 将节点添加到群集：
  2. **# Node 1**
  3. $ ./helix-admin.sh --zkSvr localhost:2181 --addNode kie-cluster nodeOne:12345
  4. **# Node 2**
  5. $ ./helix-admin.sh --zkSvr localhost:2181 --addNode kie-cluster nodeTwo:12346

...

通常，群集中节点数a等于群集中应用程序服务器的数量。节点名称（nodeOne:12345，...）可以根据需要进行更改。

|  |  |
| --- | --- |
|  | nodeOne:12345是该节点的唯一标识符，稍后将在配置应用程序服务器时引用该标识符。它不是主机和端口号，而是用于唯一标识逻辑节点。 |

* 1. 将资源添加到群集：

$ ./helix-admin.sh --zkSvr localhost:2181 --addResource kie-cluster vfs-repo 1 LeaderStandby AUTO\_REBALANCE

资源名称（vfs-repo）可以根据需要进行更改。

* 1. 重新平衡群集以初始化群集：

$ ./helix-admin.sh --zkSvr localhost:2181 --rebalance kie-cluster vfs-repo 2

* 1. 启动Helix控制器来管理群集：

$ ./run-helix-controller.sh --zkSvr localhost:2181 --cluster kie-cluster 2>&1 > /tmp/controller.log &

1. 在应用程序服务器上正确配置安全域。例如在WildFly和JBoss EAP上：
   1. 编辑文件$JBOSS\_HOME/domain/configuration/domain.xml。

为了简单起见，假设我们使用默认域配置，该配置full使用定义两个服务器节点的配置文件作为其一部分main-server-group。

* 1. 找到配置文件full并通过复制默认情况下已定义的其他安全域来添加新的安全域：
  2. <security-domain name="kie-ide" cache-type="default">
  3. <authentication>
  4. <login-module code="Remoting" flag="optional">
  5. <module-option name="password-stacking" value="useFirstPass"/>
  6. </login-module>
  7. <login-module code="RealmDirect" flag="required">
  8. <module-option name="password-stacking" value="useFirstPass"/>
  9. </login-module>
  10. </authentication>

</security-domain>

|  |  |
| --- | --- |
|  | 安全域名是一个神奇的价值。 |

1. 在应用程序服务器上配置群集的[系统属性](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wb.systemproperties)。例如在WildFly和JBoss EAP上：
   1. 编辑文件$JBOSS\_HOME/domain/configuration/host.xml。
   2. 找到server属于该XML元素main-server-group并添加必要的系统属性。

例如，对于nodeOne：

<system-properties>

<property name="jboss.node.name" value="nodeOne" boot-time="false"/>

<property name="org.uberfire.nio.git.dir" value="/tmp/kie/nodeone" boot-time="false"/>

<property name="org.uberfire.metadata.index.dir" value="/tmp/kie/nodeone" boot-time="false"/>

<property name="org.uberfire.cluster.id" value="kie-cluster" boot-time="false"/>

<property name="org.uberfire.cluster.zk" value="localhost:2181" boot-time="false"/>

<property name="org.uberfire.cluster.local.id" value="nodeOne\_12345" boot-time="false"/>

<property name="org.uberfire.cluster.vfs.lock" value="vfs-repo" boot-time="false"/>

*<!-- If you're running both nodes on the same machine: -->*

<property name="org.uberfire.nio.git.daemon.port" value="9418" boot-time="false"/>

</system-properties>

对于nodeTwo：

<system-properties>

<property name="jboss.node.name" value="nodeTwo" boot-time="false"/>

<property name="org.uberfire.nio.git.dir" value="/tmp/kie/nodetwo" boot-time="false"/>

<property name="org.uberfire.metadata.index.dir" value="/tmp/kie/nodetwo" boot-time="false"/>

<property name="org.uberfire.cluster.id" value="kie-cluster" boot-time="false"/>

<property name="org.uberfire.cluster.zk" value="localhost:2181" boot-time="false"/>

<property name="org.uberfire.cluster.local.id" value="nodeTwo\_12346" boot-time="false"/>

<property name="org.uberfire.cluster.vfs.lock" value="vfs-repo" boot-time="false"/>

*<!-- If you're running both nodes on the same machine: -->*

<property name="org.uberfire.nio.git.daemon.port" value="9419" boot-time="false"/>

</system-properties>

确保群集，节点和资源名称与Helix中配置的名称匹配。

### [23.2。jBPM集群](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jbpm_clustering)

除上述信息外，jBPM集群还需要额外的配置。看到[这篇博文](http://mswiderski.blogspot.com.br/2013/06/clustering-in-jbpm-v6.html)来正确配置数据库等。

# KIE服务器

KIE服务器是规则的独立执行服务器。

## [24. KIE执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_ch.kie.server)

### [24.1。概观](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_overview_4)

的纪伊服务器是可用于实例化和执行规则和流程模块化的，独立的服务器组件。它通过REST，JMS和Java接口向客户端应用程序公开此功能。它还提供与Kie Workbench的无缝集成。

Kie Server的核心是一个可配置的Web应用程序，打包成WAR文件。分发可用于纯Web容器（如Tomcat）以及JEE 6和JEE 7容器。

Kie服务器上的大多数功能都是可配置的，并且基于扩展的概念。每个扩展可以独立启用/禁用，允许用户根据需要配置服务器。

Kie Server的当前版本附带两个默认扩展：

* BRM：使用Drools规则引擎为执行业务规则提供支持。
* BPM：为使用jBPM流程引擎执行业务流程提供支持。它支持：
  + 流程执行
  + 任务执行
  + 同步作业执行

这两个扩展默认都是启用的，但可以通过设置相应的属性来禁用（详情请参阅配置章节）。

该服务器设计的占地面积小，内存消耗最少，因此可以轻松部署到云环境中。此服务器的每个实例都可以打开并实例化多个Kie容器，这允许您并行执行多个服务。

#### [24.1.1。词汇表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_glossary)

* **Kie服务器**：执行服务器纯粹专注于为规则和进程提供运行时环境。这些功能由Kie服务器扩展提供。更多的功能可以通过进一步扩展添加（例如，如果功能缺失，客户可以添加自己的扩展，然后使用基础设施的KIE服务器）。Kie Server实例是在给定的应用程序服务器/ Web容器上执行的独立Kie服务器。Kie服务器实例化并提供对多个Kie容器的支持。
* **Kie服务器扩展**：为服务器增加功能的Kie服务器的“插件”。Kie Server附带两个默认的kie服务器扩展：BRM和BPM。
* **Kie Container**：一个kjar的内存实例，允许实例化和使用其资产（域模型，过程，规则等）。Kie服务器通过传输协议（如REST和JMS）上的标准API公开Kie Containers。
* **控制器**：服务器支持的REST端点，负责管理KIE服务器实例。这样的终点必须提供以下功能：
  + 响应连接请求
  + 在相应的Kie服务器ID上同步所有注册的容器
  + 响应断开请求
* **Kie服务器状态**：当前已知的Kie服务器实例的状态。这是一个本地存储（默认情况下在文件中），它保持以下信息：
  + 注册控制器列表
  + 已知容器的列表
  + kie服务器配置

服务器状态在收到事件后持续存在，如：Kie Container创建，Kie Container处置，控制器接受Kie Server实例注册等。

* **Kie服务器ID**：分配了配置的任意分配标识符。在启动时，每个Kie服务器实例都被分配一个ID，并且该ID与控制器上的配置相匹配。Kie服务器实例获取并使用该配置进行自身设置。

### [24.2。安装KIE服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installing_the_kie_server)

KIE服务器作为Web应用程序归档（WAR）文件分发。WAR文件有三种不同的包装：

* webc - 对于普通Web（Servlet）容器（如Tomcat）的WAR
* ee6 - 适用于JavaEE 6容器（如JBoss EAP 6.x）的WAR
* ee7 - 适用于JavaEE 7容器（如WildFly 8.x）的WAR

要安装KIE执行服务器并验证它正在运行，请完成以下步骤：

1. 将WAR文件部署到您的Web容器中。
2. 使用kie-server容器上的角色创建一个用户。
3. 测试您可以通过在浏览器窗口中导航到端点来访问执行引擎：[http://SERVER:PORT/CONTEXT/services/rest/server/](http://server:PORT/CONTEXT/services/rest/server/)。
4. 当提示输入用户名/密码时，输入您在步骤2中创建的用户名和密码。
5. 通过身份验证后，您将看到引擎状态形式的XML响应，与此类似：

*示例225.示例握手服务器响应*

<response type="SUCCESS" msg="KIE Server info">

<kie-server-info>

<version>7.7.0.Final</version>

</kie-server-info>

</response>

#### [24.2.1。自举开关](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_bootstrap_switches)

Kie服务器接受许多引导程序开关（系统属性）来配置服务器的行为。以下是所有支持的交换机的表格。

| *表45. Kie服务器引导程序开关* | | | |
| --- | --- | --- | --- |
| **属性** | **值** | **描述** | **需要** |
| **org.drools.server.ext.disabled** | 布尔值（默认值为“false”） | 如果为true，则禁用BRM支持（即规则支持）。 | 没有 |
| **org.jbpm.server.ext.disabled** | 布尔值（默认值为“false”） | 如果为true，则禁用BPM支持（即进程支持） | 没有 |
| **org.jbpm.ui.server.ext.disabled** | 布尔值（默认值为“false”） | 如果为true，则禁用BPM UI支持（即处理图像支持） | 没有 |
| **org.optaplanner.server.ext.disabled** | 布尔值（默认值为“false”） | 如果为true，则禁用BRP支持​​（即规划支持） | 没有 |
| **org.kie.executor.disabled** | 布尔值（默认值为“false”） | 如果为true，则禁用BPM作业执行程序支持 | 没有 |
| **org.kie.server.id** | 串 | 要分配给此服务器的任意ID。如果配置了遥控器，则这是服务器将连接到控制器以获取kie容器配置的ID。 | 否。如果未提供，则会自动生成一个ID。 |
| **org.kie.server.user** | 字符串（默认是“kieserver”） | 用于在控制模式下运行时用于连接来自控制器的kieserver的用户名 | 没有 |
| **org.kie.server.pwd** | 字符串（默认为“kieserver1！”） | 在托管模式下运行时用于从控制器连接到kieserver的密码 | 没有 |
| **org.kie.server.controller** | 逗号分隔的网址列表 | 控制器REST端点的URL列表。例如：<http://localhost:8080/kie-wb/rest/controller> | 是的，当使用控制器 |
| **org.kie.server.controller.user** | 字符串（默认是“kieserver”） | 用于连接到控制器REST API的用户名 | 是的，当使用控制器 |
| **org.kie.server.controller.pwd** | 字符串（默认为“kieserver1！”） | 用于连接到控制器REST API的密码 | 是的，当使用控制器 |
| **org.kie.server.location** | kie服务器实例的URL位置 | 控制器用于在此服务器上回叫的URL。例如：<http://localhost:8230/kie-server/services/rest/server> | 是的，当使用控制器 |
| **org.kie.server.domain** | 串 | JAAS LoginContext域，用于在使用JMS时对用户进行身份验证 | 没有 |
| **org.kie.server.bypass.auth.user** | 布尔值（默认值为“false”） | 允许绕过经过认证的用户进行与任务相关的操作，例如查询 | 没有 |
| **org.kie.server.repo** | 有效的文件系统路径（默认为“。”） | 在存储kie服务器状态文件的本地文件系统上的位置 | 没有 |
| **org.kie.server.persistence.ds** | 串 | 数据源JNDI名称 | 是，当启用BPM支持时 |
| **org.kie.server.persistence.tm** | 串 | 事务管理器平台为Hibernate属性设置 | 是，当启用BPM支持时 |
| **org.kie.server.persistence.dialect** | 串 | Hibernate方言被使用 | 是，当启用BPM支持时 |
| **org.jbpm.ht.callback** | 串 | 任务服务支持的回调之一（默认jaas） | 没有 |
| **org.jbpm.ht.custom.callback** | 串 | 在org.jbpm.ht.callback被设置为'custom'的情况下自定义实现UserGroupCallback | 没有 |
| **kie.maven.settings.custom** | 有效的文件系统路径 | maven配置的自定义settings.xml的位置 | 没有 |
| **org.kie.executor.interval** | 整数（默认值是3） | 执行者轮询之间的时间单位数 | 没有 |
| **org.kie.executor.pool.size** | 整数（默认值为1） | 异步工作池中的线程数 | 没有 |
| **org.kie.executor.retry.count** | 整数（默认值是3） | 处理错误的重试次数 | 没有 |
| **org.kie.executor.timeunit** | TimeUnit（默认为“SECONDS”） | 表示时间间隔的TimeUnit | 没有 |
| **org.kie.executor.disabled** | 布尔值（默认值为“false”） | 完全禁用执行程序 | 没有 |
| **kie.server.jms.queues.response** | 字符串（默认为“queue / KIE.SERVER.RESPONSE”） | JMS的响应队列的JNDI名称 | 没有 |
| **org.kie.server.controller.connect** | 长（默认值为10000） | kie服务器启动时反复尝试将kie服务器连接到控制器的等待时间（以毫秒为单位） | 没有 |
| **org.drools.server.filter.classes** | 布尔值（默认值为“false”） | 如果为true，则仅接受使用@ org.kie.api.remote.Remotable或@ javax.xml.bind.annotation.XmlRootElement注释的类作为额外的JAXB类 | 没有 |
|  |  |  |  |

|  |  |
| --- | --- |
|  | 如果您同时运行KIE服务器和KIE工作台，则必须使用**org.kie.server.persistence.ds**属性将KIE服务器配置为使用与KIE Workbench不同的数据源 。KIE Workbench使用jBPM执行服务，如果它们共享相同的数据源，则可能与KIE服务器发生冲突。 |

#### [24.2.2。不同容器的安装细节](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_installation_details_for_different_containers)

##### [24.2.2.1。Tomcat 7.x / 8.x](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_tomcat_7_x_8_x)

1. 下载并解压缩Tomcat发行版。我们称之为分布的根TOMCAT\_HOME。这个目录以Tomcat版本命名，例如apache-tomcat-7.0.55。
2. 下载kie-server- -webc.war并将其放入TOMCAT\_HOME/webapps。
3. 配置用户和角色。确保该文件TOMCAT\_HOME/conf/tomcat-users.xml包含以下用户名和角色定义。您当然可以选择不同的用户名和密码，只要确保用户具有角色即可kie-server：

*例子226. Tomcat的用户名和角色定义*

<role rolename="kie-server"/>

<user username="serveruser" password="my.s3cr3t.pass" roles="kie-server"/>

1. 通过运行启动服务器TOMCAT\_HOME/bin/startup.[sh|bat]。您可以检查Tomcat登录TOMCAT\_HOME/logs以查看应用程序是否已成功部署。请阅读上面的表格，了解可用于正确配置实例的引导程序开关。例如：
2. ./startup.sh -Dorg.kie.server.id=first-kie-server

-Dorg.kie.server.location=http:*//localhost:8080/kie-server/services/rest/server*

1. 验证服务器正在运行。转到[http://SERVER:PORT/CONTEXT/services/rest/server/](http://server:PORT/CONTEXT/services/rest/server/)并键入指定的用户名和密码。您应该看到简单的XML消息以及有关服务器的基本信息。

|  |  |
| --- | --- |
|  | 运行Tomcat或任何其他Web容器时无法利用JMS界面。WAR的Web容器版本仅包含REST接口。 |

##### [24.2.2.2。WildFly 8.x](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wildfly_8_x)

1. 下载并解压缩WildFly发行版。我们称之为分布的根WILDFLY\_HOME。该目录以WildFly版本命名，例如wildfly-8.2.0.Final。
2. 下载kie-server- -ee7.war并放入WILDFLY\_HOME/standalone/deployments。
3. 配置用户和角色。执行以下命令 WILDFLY\_HOME/bin/add-user.[sh|bat] -a -u 'kieserver' -p 'kieserver1!' -ro 'kie-server'。您当然可以选择不同的用户名和密码，只要确保用户具有角色即可kie-server。
4. 通过运行启动服务器WILDFLY\_HOME/bin/standalone.[sh|bat] -c standalone-full.xml <bootstrap\_switches>。您可以检出标准输出或WildFly登录WILDFLY\_HOME/standalone/logs以查看应用程序是否已成功部署。请阅读上面的表格，了解可用于正确配置实例的引导程序开关。例如：
5. ./standalone.sh --server-config=standalone-full.xml
6. -Djboss.socket.binding.port-offset=150
7. -Dorg.kie.server.id=first-kie-server

-Dorg.kie.server.location=http:*//localhost:8230/kie-server/services/rest/server*

1. 验证服务器正在运行。转到[http://SERVER:PORT/CONTEXT/services/rest/server/](http://server:PORT/CONTEXT/services/rest/server/)并键入指定的用户名和密码。您应该看到简单的XML消息以及有关服务器的基本信息。

### [24.3。Kie服务器设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_setup)

|  |  |
| --- | --- |
|  | 服务器设置和注册从版本6.2和之前发生了显着变化。以下内容仅适用于版本6.3和转发。 |

#### [24.3.1。托管Kie服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managed_kie_server)

托管实例需要控制器可用于正确启动Kie Server实例。

控制器是一个负责保持和管理Kie服务器配置的组件。每个控制器可以同时管理多个配置，并且环境中可以有多个控制器。托管的KIE服务器可以配置一系列控制器，但一次只能连接一个。

|  |  |
| --- | --- |
|  | 需要提到的是，即使可以有多个控制器，它们也应该保持同步，以确保无论KIE服务器实例与哪个联系人联系，它都会提供相同的配置集。 |

在启动时，如果Kie服务器配置了一系列控制器，它将会连续尝试连接它们，直到与其中一个连接成功建立连接。如果由于任何原因无法建立连接，即使本地存储器配置可用，服务器也不会启动。这是为了确保一致性而设计的。例如，如果Kie服务器关闭并且配置已更改，则此限制可确保它将以最新配置运行，或根本不运行。

|  |  |
| --- | --- |
|  | 为了在独立模式下运行Kie服务器，无需连接任何控制器，请参阅“非托管Kie服务器”。 |

配置设置，除其他外：

* kie容器被部署和启动
* 配置项目 - 目前这是一个进一步增强功能的占位符，可以远程配置KIE执行服务器组件 - 定时器，持久性等

Controller除了提供配置管理外，还负责Kie服务器的整体管理。它提供了一个REST API，分为两部分：

* 控制器本身暴露与KIE Execution Server实例交互
* 一个允许远程管理Kie服务器实例的管理API：
  + 添加/删除服务器
  + 向服务器添加/删除容器
  + 在服务器上启动/停止容器

控制器仅处理Kie服务器配置或定义，以不同的方式进行处理。它不处理KIE Execution Server实例的任何运行时组件。他们总是被认为是远离控制器。控制器负责持续配置以保持控制器本身的重新启动。它应该管理同步，以防万一控制器被配置为在控制器的所有实例上保持最新的所有定义。

默认情况下，控制器随附Kie Workbench，并提供全功能管理界面（包括REST API和UI）。它使用底层的git存储库作为持久存储，因此，当GIT存储库被群集时（使用Apache Zookeeper和Apache Helix），它也将覆盖控制器同步。

上图说明了由其管理的多个Kie服务器实例的单个控制器（工作台）设置。

下图说明了通过Zookeeper同步多个控制器实例的集群设置。

在上图中，我们可以看到Kie Server实例能够连接到任何控制器，但它们只能连接到一个。只要达到一个，每个实例都会尝试连接到控制器。一旦建立与其中一个控制器的连接，它将跳过其他控制器。

##### [24.3.1.1。使用托管服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_working_with_managed_servers)

在使用托管的KIE服务器实例时，用户可以采取两种方法：

* 配置第一：通过这种方法，用户将开始使用控制器（UI或REST API）并创建和配置Kie Server定义。这主要由服务器定义（ID和名称+可选版本，以提高可读性）的标识以及Kie Containers在服务器上运行的配置组成。
* 首先注册：采用这种方法，Kie Server实例首先启动并自动注册到控制器上。用户然后可以配置Kie容器。此选项只需跳过第一种方法中完成的注册步骤，并在自动注册时直接填入服务器ID，名称和版本。这两种方法之间没有其他区别。

#### [24.3.2。非托管KIE执行服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_unmanaged_kie_execution_server)

非托管的Kie服务器反过来只是一个独立的实例，因此必须使用Kie服务器本身的REST / JMS api单独配置。没有涉及控制器。配置会自动由服务器持久保存到文件中，并在重启时用作内部服务器状态。

配置在以下操作中更新：

* 部署Kie Container
* 取消部署Kie Container
* 启动Kie Container
* 停止Kie Container

|  |  |
| --- | --- |
|  | 如果Kie服务器重新启动，它将尝试恢复在关机之前持续存在的相同状态。这意味着正在运行的Kie Containers将启动，但之前停止/处理的Kie Containers不会。 |

在大多数使用情况下，Kie服务器应该以托管模式执行，因为它提供了一些优点，例如Web用户界面（如果使用工作台作为控制器）以及一些集群设施。

### [24.4。创建Kie容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_creating_a_kie_container)

一旦你的Execution Server被注册，你就可以开始添加**Kie Containers**了。

**Kie Containers**是自备环境，它们已被调配以保存打包和部署的规则实例的实例。

1. 首先单击要在其中部署Container的Execution Server旁边的**\ +**图标。这将带来新的容器屏幕。
2. 如果您知道已部署软件包的**组名**，**工件标识**和**版本**（GAV），则可以输入这些详细信息并单击确定按钮以选择该实例（并为容器提供一个名称）。
3. 如果您不知道这些值，则可以在KIE Workbench中搜索可部署的所有软件包。单击“ **搜索”**按钮，而不在搜索字段中输入任何值（可以通过输入您想要部署的包中存在的任何术语来缩小搜索范围）。

|  |  |
| --- | --- |
|  | 在这里插入屏幕 |

1. 上图显示有三个可部署的包可用作Execution Server上的容器。通过单击选择按钮选择您想要的那个。这将自动填充**GAV**，然后您可以单击**确定**按钮将此可部署作为新容器使用。
2. 在顶部输入此容器的名称，然后按确定按钮。

|  |  |  |
| --- | --- | --- |
|  | | 容器名称在每个执行服务器内必须是唯一的，并且不得包含任何空格。 |
|  | | 在**GAV**行的下方，您将看到一个不可编辑的行，它向您显示Container 的**URL**，您将可以使用它执行REST命令。 | |

### [24.5。管理容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_containers)

Execution Server中的容器可以在KIE Workbench.H内启动，停止和更新

#### [24.5.1。启动一个容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_starting_a_container)

一旦注册，容器处于“停止”模式。可以通过先选择它然后单击开始按钮来启动它。您也可以选择多个容器并同时启动它们。

一旦容器处于“运行”模式，其旁边会出现一个绿色箭头。如果启动容器时出现任何错误，红色图标将出现在Container和部署它们的执行服务器旁边。

您应该检查执行服务器和当前Business Central的日志，以查看重新部署容器（可能还有执行服务器）之前的错误。

#### [24.5.2。停止并删除容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_stopping_and_deleting_a_container)

与启动容器类似，选择要停止（或删除）的容器，然后单击停止按钮（在进入“正在运行”模式时替换该容器的开始按钮）或删除按钮。 ⁠

#### [24.5.3。更新容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_updating_a_container)

您可以更新部署KieContainers而无需重新启动执行服务器。这在业务规则发生变化并创建要配置的新版本软件包的情况下非常有用。

您可以配置和部署相同软件包的多个版本，每个版本都不同KieContainer。

要KieContainer动态更新部署，请单击容器旁边的图标。这将打开Container Info屏幕。此屏幕的示例如下所示：

|  |  |
| --- | --- |
|  | 在这里插入屏幕 |

“容器信息”屏幕是一个有用的工具，因为它不仅允许您查看终端KieContainer，还可以在更新可用时手动或自动刷新设置。更新可以是手动或自动的：

**手动更新：**要手动更新a KieContainer，请在“版本”框中输入新的版本号，然后单击“ **更新”**按钮。你当然可以更新组ID或者神器ID，如果这些也改变了。一旦更新，执行服务器将更新容器，并在**Resolved Release Id**部分的屏幕底部显示已解析的GAV属性。

**自动更新：**如果您希望部署的Container始终具有最新版本的部署，而无需手动编辑它，则需要将Version属性设置为值LATEST并启动一个Scanner。这将确保部署的规定始终包含最新版本。通过点击“立即扫描”按钮，只需点击一次即可启动扫描器，或者您可以在指定时间间隔（以毫秒为单位）进行扫描的情况下在后台启动扫描器。您也可以LATEST在首次创建此部署时将此值设置为。在这种情况下，**Resolved Release Id**将显示实际的最新版本号。

### [24.6。Kie服务器REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie.ksrestapi)

Execution Server通过REST API支持以下命令。

在使用这些命令之前请注意以下几点：

* 对于这些基本URL将保持如前面定义的端点（例如：[http://SERVER:PORT/CONTEXT/services/rest/server/](http://server:PORT/CONTEXT/services/rest/server/)）
* 如前所述，所有请求都需要基本的HTTP身份验证。

#### [24.6.1。[GET] /](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get)

返回执行服务器信息

*示例227.示例服务器响应*

<response type="SUCCESS" msg="KIE Server info">

<kie-server-info>

<version>6.2.0.redhat-1</version>

</kie-server-info>

</response>

#### [24.6.2。[POST] / config](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_config)

使用POST HTTP方法，您可以在Execution Server上执行各种命令。例如：create-container，list-containers，dispose-container和call-container。

* CreateContainerCommand
* GetServerInfoCommand
* ListContainersCommand
* CallContainerCommand
* DisposeContainerCommand
* GetContainerInfoCommand
* GetScannerInfoCommand
* UpdateScannerCommand
* UpdateReleaseIdCommand

命令本身可以在org.kie.server.api.commands包中找到。

#### [24.6.3。[GET] /容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers)

返回已在此Execution Server上创建的容器列表。

*示例228.示例服务器响应*

<response type="SUCCESS" msg="List of created containers">

<kie-containers>

<kie-container container-id="MyProjectContainer" status="STARTED">

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</release-id>

<resolved-release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</resolved-release-id>

</kie-container>

</kie-containers>

</response>

端点还支持基于ReleaseId和的过滤container status。例子：

* /containers?groupId=org.example - 仅返回具有指定groupId的容器
* /containers?groupId=org.example&artifactId=project1&version=1.0.0.Final - 只返回指定的容器 ReleaseId
* /containers?status=started,failed - 返回启动或失败的容器

#### [24.6.4。⁠[GET] / containers / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_id)

返回有关特定容器的状态和信息。例如，执行[http://SERVER:PORT/CONTEXT/services/rest/server/containers/MyProjectContainer](http://server:PORT/CONTEXT/services/rest/server/containers/MyProjectContainer)可能会返回以下示例容器信息。

*示例229.示例服务器响应*

⁠<response type="SUCCESS" msg="Info for container MyProjectContainer">

<kie-container container-id="MyProjectContainer" status="STARTED">

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</release-id>

<resolved-release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</resolved-release-id>

</kie-container>

</response>

#### [24.6.5。[PUT] / containers / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_containers_id)

允许您在执行服务器中创建一个新的容器。例如，要创建一个ID为**MyRESTContainer**的Container，完整的端点将为：[http://SERVER:PORT/CONTEXT/services/rest/server/containers/MyRESTContainer](http://server:PORT/CONTEXT/services/rest/server/containers/MyRESTContainer)。请求的一个例子是：⁠

*示例230.示例请求创建一个容器*

<kie-container container-id="MyRESTContainer">

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</release-id>

</kie-container>

如果成功，服务器的响应将是：

*示例231.创建容器时的示例服务器响应*

<response type="SUCCESS" msg="Container MyRESTContainer successfully deployed with module com.redhat:Project1:1.0">

<kie-container container-id="MyProjectContainer" status="STARTED">

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</release-id>

<resolved-release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</resolved-release-id>

</kie-container>

</response>

#### [24.6.6。[DELETE] / containers / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_containers_id)

⁠配置由id指定的容器。例如，[http://SERVER:PORT/CONTEXT/services/rest/server/containers/MyProjectContainer](http://server:PORT/CONTEXT/services/rest/server/containers/MyProjectContainer)使用DELETE HTTP方法执行将返回以下服务器响应：⁠

*示例232.处理容器的示例服务器响应*

<response type="SUCCESS" msg="Container MyProjectContainer successfully disposed."/>

#### [24.6.7。[POST] / containers / instances / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_instances_id)

针对指定的容器执行操作和命令。您可以在POST请求的主体中向此容器发送命令。例如，要使用id MyRESTContainer（[http://SERVER:PORT/CONTEXT/services/rest/server/containers/instances/MyRESTContainer](http://server:PORT/CONTEXT/services/rest/server/containers/instances/MyRESTContainer)）激发Container的所有规则，您可以发送fire-all-rules命令给它，如下所示（在POST请求的主体中）：

*示例233.示例服务器请求触发所有规则*

<fire-all-rules/>

以下是支持的命令列表：

* AgendaGroupSetFocusCommand
* ClearActivationGroupCommand
* ClearAgendaCommand
* ClearAgendaGroupCommand
* ClearRuleFlowGroupCommand
* 的DeleteCommand
* InsertObjectCommand
* ModifyCommand
* GetObjectCommand
* InsertElementsCommand
* FireAllRulesCommand
* QueryCommand
* SetGlobalCommand
* GetGlobalCommand
* GetObjectsCommand
* BatchExecutionCommand

这些命令可以在org.drools.core.command.runtime包中找到。

#### [24.6.8。[GET] / containers / {id} / release-id](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_id_release_id)

返回由id指定的Container的完整版本ID。

*示例234.示例服务器响应*

⁠<response type="SUCCESS" msg="ReleaseId for container MyProjectContainer">

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</release-id>

</response>

#### [24.6.9。[POST] / containers / {id} / release-id](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_id_release_id)

允许您更新容器部署的发布ID。将新的完整版本ID发送到服务器。

*示例235.示例服务器请求*

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.1</version>

</release-id>

服务器将回复成功或错误消息，类似于以下内容：⁠

*示例236.示例服务器响应*

<response type="SUCCESS" msg="Release id successfully updated.">

<release-id>

<artifact-id>Project1</artifact-id>

<group-id>com.redhat</group-id>

<version>1.0</version>

</release-id>

</response>

#### [10年6月24日。[GET] / containers / {id} / scanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_id_scanner)

返回有关此容器的自动更新的扫描仪的信息

*示例237.示例服务器响应*

<response type="SUCCESS" msg="Scanner info successfully retrieved">

<kie-scanner status="DISPOSED"/>

</response>

#### [11年6月24日。[POST] / containers / {id} / scanner](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_id_scanner)

允许您启动或停止控制轮询更新的Container部署的扫描器。要启动扫描仪，请发送类似于[http://SERVER:PORT/CONTEXT/services/rest/server/containers/{container-id}/scanner](http://server:PORT/CONTEXT/services/rest/server/containers/%7Bcontainer-id%7D/scanner)以下内容的请求：使用以下POST数据

*示例238.示例服务器启动扫描器的请求*

<kie-scanner status="STARTED" poll-interval="2000"/>

⁠poll-interval属性以毫秒为单位。服务器的响应类似于：⁠

*示例239.示例服务器响应*

<response type="SUCCESS" msg="Kie scanner successfully created.">

<kie-scanner status="STARTED"/>

</response>

要停止扫描仪，请将状态替换为DISPOSED并删除poll-interval属性。

### [24.7。OptaPlanner REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_optaplanner_rest_api)

Kie服务器支持以下Planner REST API。所有这些API也可以通过JMS和Java客户端API获得。另请注意：

* 这些URL的基本URL将保持为先前定义的端点（例如[http://SERVER:PORT/CONTEXT/services/rest/server/](http://server:PORT/CONTEXT/services/rest/server/)）。
* 如前所述，所有请求都需要基本的HTTP身份验证。
* 要获得特定的编组格式，请在HTTP请求中添加HTTP标头Content-Type和可选项X-KIE-ContentType。例如：
* Content-Type: application/xml

X-KIE-ContentType: xstream

|  |  |
| --- | --- |
|  | X-KIE-ContentType支持以下值：xstream，xml，json。 |

下面例子中的请求和响应假设乃纪伊集装箱使用计划员工作台的optacloud例子建成，通过调用PUT上/services/rest/server/containers/optacloud-kiecontainer-1与此内容：

<kie-container container-id="optacloud-kiecontainer-1">

<release-id>

<group-id>opta</group-id>

<artifact-id>optacloud</artifact-id>

<version>1.0.0</version>

</release-id>

</kie-container>

#### [24.7.1。[GET] /容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_2)

返回创建的容器的列表。

*示例240.示例服务器响应（XStream）*

<response type="SUCCESS" msg="List of created containers">

<result class="kie-containers">

<kie-container>

<container-id>optacloud-kiecontainer-1</container-id>

<release-id>

<group-id>optacloud</group-id>

<artifact-id>optacloud</artifact-id>

<version>1.0.0</version>

</release-id>

<resolved-release-id>

<group-id>optacloud</group-id>

<artifact-id>optacloud</artifact-id>

<version>1.0.0</version>

</resolved-release-id>

<status>STARTED</status>

<scanner>

<status>DISPOSED</status>

</scanner>

</kie-container>

</result>

</response>

#### [24.7.2。[PUT] / containers / {containerId} / solvers / {solverId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_containers_containerid_solvers_solverid)

用{solverId}容器中的给定创建一个新求解器{containerId}。请求的主体是编组的SolverInstance实体，必须指定解算器配置文件。

以下是请求和相应响应的示例。

*示例241.示例服务器请求（XStream）*

<solver-instance>

<solver-config-file>optacloud/optacloud/cloudSolverConfig.solver.xml</solver-config-file>

</solver-instance>

*示例242.示例服务器响应（XStream）*

<solver-instance>

<container-id>optacloud-kiecontainer-1</container-id>

<solver-id>solver1</solver-id>

<solver-config-file>optacloud/optacloud/cloudSolverConfig.solver.xml</solver-config-file>

<status>NOT\_SOLVING</status>

<score />

</solver-instance>

#### [24.7.3。[GET] / containers / {containerId} /求解器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers)

返回容器中创建的求解器列表。

*示例243.示例服务器响应（XStream）*

<org.kie.server.api.model.instance.SolverInstanceList>

<solvers>

<solver-instance>

<container-id>optacloud-kiecontainer-1</container-id>

<solver-id>solver2</solver-id>

<solver-config-file>optacloud/optacloud/cloudSolverConfig2.solver.xml</solver-config-file>

<status>NOT\_SOLVING</status>

<score />

</solver-instance>

<solver-instance>

<container-id>optacloud-kiecontainer-1</container-id>

<solver-id>solver1</solver-id>

<solver-config-file>optacloud/optacloud/cloudSolverConfig.solver.xml</solver-config-file>

<status>NOT\_SOLVING</status>

<score />

</solver-instance>

</solvers>

</org.kie.server.api.model.instance.SolverInstanceList>

#### [24.7.4。[GET] / containers / {containerId} / solvers / {solverId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers_solverid)

返回{solverId}容器中解算器的当前状态{containerId}。

*示例244.示例服务器响应（XStream）*

<solver-instance>

<container-id>optacloud-kiecontainer-1</container-id>

<solver-id>solver1</solver-id>

<solver-config-file>optacloud/optacloud/cloudSolverConfig.solver.xml</solver-config-file>

<status>NOT\_SOLVING</status>

<score />

</solver-instance>

#### [24.7.5。[POST] / containers / {containerId} / solvers / {solverId} / state / solve](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_containerid_solvers_solverid_state_solving)

如果尚未执行，则在容器{containerId}中启动解算器{solverId}。请求的主体是一个PlanningSolution被优化的编组。

以下是使用2台计算机和6个进程解决OptaCloud问题的示例。解算器异步运行。向bestsolution网址发送请求以获得最佳解决方案。

*示例245.示例服务器请求（XStream）*

<optacloud.optacloud.CloudSolution id="1">

<computerList id="2">

<optacloud.optacloud.Computer id="3">

<cpuPower>24</cpuPower>

<memory>96</memory>

<networkBandwidth>16</networkBandwidth>

<cost>4800</cost>

</optacloud.optacloud.Computer>

<optacloud.optacloud.Computer id="4">

<cpuPower>6</cpuPower>

<memory>4</memory>

<networkBandwidth>6</networkBandwidth>

<cost>660</cost>

</optacloud.optacloud.Computer>

</computerList>

<processList id="5">

<optacloud.optacloud.Process id="6">

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process id="7">

<requiredCpuPower>3</requiredCpuPower>

<requiredMemory>6</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process id="8">

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>3</requiredNetworkBandwidth>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process id="9">

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>2</requiredMemory>

<requiredNetworkBandwidth>11</requiredNetworkBandwidth>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process id="10">

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process id="11">

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>5</requiredNetworkBandwidth>

</optacloud.optacloud.Process>

</processList>

</optacloud.optacloud.CloudSolution>

*示例246.示例服务器请求（JAXB）*

**<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>**

<cloudSolution>

<computerList>

<cost>4800</cost>

<cpuPower>24</cpuPower>

<memory>96</memory>

<networkBandwidth>16</networkBandwidth>

</computerList>

<computerList>

<cost>660</cost>

<cpuPower>6</cpuPower>

<memory>4</memory>

<networkBandwidth>6</networkBandwidth>

</computerList>

<processList>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

</processList>

<processList>

<requiredCpuPower>3</requiredCpuPower>

<requiredMemory>6</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

</processList>

<processList>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>3</requiredNetworkBandwidth>

</processList>

<processList>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>2</requiredMemory>

<requiredNetworkBandwidth>11</requiredNetworkBandwidth>

</processList>

<processList>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

</processList>

<processList>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>5</requiredNetworkBandwidth>

</processList>

</cloudSolution>

*示例247.示例服务器请求（JSON）*

{

"optacloud.optacloud.CloudSolution": {

"computerList": [

{

"cpuPower": 24,

"memory": 96,

"networkBandwidth": 16,

"cost": 4800

},

{

"cpuPower": 6,

"memory": 4,

"networkBandwidth": 6,

"cost": 660

}

],

"processList": [

{

"requiredCpuPower": 1,

"requiredMemory": 1,

"requiredNetworkBandwidth": 1

},

{

"requiredCpuPower": 3,

"requiredMemory": 6,

"requiredNetworkBandwidth": 1

},

{

"requiredCpuPower": 1,

"requiredMemory": 1,

"requiredNetworkBandwidth": 3

},

{

"requiredCpuPower": 1,

"requiredMemory": 2,

"requiredNetworkBandwidth": 11

},

{

"requiredCpuPower": 1,

"requiredMemory": 1,

"requiredNetworkBandwidth": 1

},

{

"requiredCpuPower": 1,

"requiredMemory": 1,

"requiredNetworkBandwidth": 5

}

]

}

}

#### [24.7.6。[POST] / containers / {containerId} / solvers / {solverId} / state / terminating-early](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_containerid_solvers_solverid_state_terminating_early)

如果它正在运行，请求解算器提前终止。这并不会删除求解器，仍然可以检索最佳解决方案。

#### [24.7.7。[GET] / containers / {containerId} / solvers / {solverId} / bestsolution](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers_solverid_bestsolution)

返回请求发现时找到的最佳解决方案。如果解算器尚未终止（因此该status字段仍然存在SOLVING），它将返回到目前为止发现的最佳解决方案，但稍后的调用可能会返回更好的解决方案。

*示例248.示例服务器响应（XStream）*

<solver-instance>

<container-id>optacloud-kiecontainer-1</container-id>

<solver-id>solver1</solver-id>

<solver-config-file>optacloud/optacloud/cloudSolverConfig.solver.xml</solver-config-file>

<status>NOT\_SOLVING</status>

<score scoreClass="org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore">0hard/-5460soft</score>

<best-solution class="optacloud.optacloud.CloudSolution">

<computerList>

<optacloud.optacloud.Computer>

<cpuPower>24</cpuPower>

<memory>96</memory>

<networkBandwidth>16</networkBandwidth>

<cost>4800</cost>

</optacloud.optacloud.Computer>

<optacloud.optacloud.Computer>

<cpuPower>6</cpuPower>

<memory>4</memory>

<networkBandwidth>6</networkBandwidth>

<cost>660</cost>

</optacloud.optacloud.Computer>

</computerList>

<processList>

<optacloud.optacloud.Process>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

<computer reference="../../../computerList/optacloud.optacloud.Computer[2]"/>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process>

<requiredCpuPower>3</requiredCpuPower>

<requiredMemory>6</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

<computer reference="../../../computerList/optacloud.optacloud.Computer"/>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>3</requiredNetworkBandwidth>

<computer reference="../../../computerList/optacloud.optacloud.Computer"/>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>2</requiredMemory>

<requiredNetworkBandwidth>11</requiredNetworkBandwidth>

<computer reference="../../../computerList/optacloud.optacloud.Computer"/>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>1</requiredNetworkBandwidth>

<computer reference="../../../computerList/optacloud.optacloud.Computer"/>

</optacloud.optacloud.Process>

<optacloud.optacloud.Process>

<requiredCpuPower>1</requiredCpuPower>

<requiredMemory>1</requiredMemory>

<requiredNetworkBandwidth>5</requiredNetworkBandwidth>

<computer reference="../../../computerList/optacloud.optacloud.Computer[2]"/>

</optacloud.optacloud.Process>

</processList>

<score>0hard/-5460soft</score>

</best-solution>

</solver-instance>

#### [24.7.8。[POST] / containers / {containerId} / solvers / {solverId} / problemfactchanges](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_containers_containerid_solvers_solverid_problemfactchanges)

实时计划功能。提交一个或多个ProblemFactChanges来更新求解器当前优化的数据集。

#### [24.7.9。[GET] / containers / {containerId} / solvers / {solverId} / problemfactchanges / processed](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_containers_containerid_solvers_solverid_problemfactchanges_processed)

实时计划功能。如果求解器处理了所有已提交的ProblemFactChanges，则返回true。否则返回false。

#### [10年7月24日。[DELETE] / containers / {containerId} / solvers / {solverId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_containers_containerid_solvers_solverid)

⁠ {solverId}在容器中放置求解器{containerId}。如果尚未终止，则首先终止。

### [24.8。控制器REST API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_controller_rest_api)

当您安装了受管理的Kie服务器时，您需要通过控制器管理Kie服务器和容器。一般来说，它是通过工作台UI完成的，但您也可以使用Controller REST API。

* 控制器基本URL由kie-wb war部署提供，与org.kie.server.controller属性相同。（例如：<http://localhost:8080/kie-wb/rest/controller>）
* 如前所述，所有请求都需要基本的HTTP身份验证。

#### [24.8.1。[GET] /管理/服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_servers)

返回Kie服务器模板列表

*示例249.示例服务器响应*

**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

<server-template-list>

<server-template>

<server-id>demo</server-id>

<server-name>demo</server-name>

<container-specs>

<container-id>hr</container-id>

<container-name>hr</container-name>

<server-template-key>

<server-id>demo</server-id>

</server-template-key>

<release-id>

<artifact-id>HR</artifact-id>

<group-id>org.jbpm</group-id>

<version>1.0</version>

</release-id>

<configs>

<entry>

<key>RULE</key>

<value xsi:type="ruleConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<scanner-status>STOPPED</scanner-status>

</value>

</entry>

<entry>

<key>PROCESS</key>

<value xsi:type="processConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<strategy>Singleton</strategy>

<kie-base-name></kie-base-name>

<kie-session-name></kie-session-name>

<merge-mode>Merge Collections</merge-mode>

</value>

</entry>

</configs>

<status>STARTED</status>

</container-specs>

<configs/>

<server-instances>

<server-instance-id>demo@localhost:8230</server-instance-id>

<server-name>demo@localhost:8230</server-name>

<server-template-id>demo</server-template-id>

<server-url>http://localhost:8230/kie-server/services/rest/server</server-url>

</server-instances>

<capabilities>RULE</capabilities>

<capabilities>PROCESS</capabilities>

<capabilities>PLANNING</capabilities>

</server-template>

</server-template-list>

#### [24.8.2。[GET] / management / server / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_server_id)

返回Kie服务器模板

*示例250.示例服务器响应*

**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

<server-template-details>

<server-id>product-demo</server-id>

<server-name>product-demo</server-name>

<container-specs>

<container-id>hr</container-id>

<container-name>hr</container-name>

<server-template-key>

<server-id>demo</server-id>

</server-template-key>

<release-id>

<artifact-id>HR</artifact-id>

<group-id>org.jbpm</group-id>

<version>1.0</version>

</release-id>

<configs>

<entry>

<key>RULE</key>

<value xsi:type="ruleConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<scanner-status>STOPPED</scanner-status>

</value>

</entry>

<entry>

<key>PROCESS</key>

<value xsi:type="processConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<strategy>Singleton</strategy>

<kie-base-name></kie-base-name>

<kie-session-name></kie-session-name>

<merge-mode>Merge Collections</merge-mode>

</value>

</entry>

</configs>

<status>STARTED</status>

</container-specs>

<configs/>

<server-instances>

<server-instance-id>demo@localhost:8230</server-instance-id>

<server-name>demo@localhost:8230</server-name>

<server-template-id>demo</server-template-id>

<server-url>http://localhost:8230/kie-server/services/rest/server</server-url>

</server-instances>

<capabilities>RULE</capabilities>

<capabilities>PROCESS</capabilities>

<capabilities>PLANNING</capabilities>

</server-template-details>

#### [24.8.3。[PUT] / management / server / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_management_server_id)

用指定的ID创建一个新的Kie服务器模板

*示例251.示例请求创建一个新的Kie服务器模板*

**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

<server-template-details>

<server-id>test-demo</server-id>

<server-name>test-demo</server-name>

<configs/>

<capabilities>RULE</capabilities>

<capabilities>PROCESS</capabilities>

<capabilities>PLANNING</capabilities>

</server-template-details>

#### [24.8.4。[DELETE] / management / server / {id}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_management_server_id)

使用指定的ID删除Kie服务器模板

#### [24.8.5。[GET] / management / server / {id} / containers](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_server_id_containers)

返回给定服务器上的所有容器

*示例252.示例服务器响应*

**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

<container-spec-list>

<container-spec>

<container-id>hr</container-id>

<container-name>hr</container-name>

<server-template-key>

<server-id>demo</server-id>

</server-template-key>

<release-id>

<artifact-id>HR</artifact-id>

<group-id>org.jbpm</group-id>

<version>1.0</version>

</release-id>

<configs>

<entry>

<key>RULE</key>

<value xsi:type="ruleConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<scanner-status>STOPPED</scanner-status>

</value>

</entry>

<entry>

<key>PROCESS</key>

<value xsi:type="processConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<strategy>Singleton</strategy>

<kie-base-name></kie-base-name>

<kie-session-name></kie-session-name>

<merge-mode>Merge Collections</merge-mode>

</value>

</entry>

</configs>

<status>STARTED</status>

</container-spec>

</container-spec-list>

#### [24.8.6。[GET] / management / server / {id} / containers / {containerId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__get_management_server_id_containers_containerid)

返回Container信息，包括其发布标识和配置

*示例253.示例服务器响应*

**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

<container-spec-details>

<container-id>hr</container-id>

<container-name>hr</container-name>

<server-template-key>

<server-id>demo</server-id>

</server-template-key>

<release-id>

<artifact-id>HR</artifact-id>

<group-id>org.jbpm</group-id>

<version>1.0</version>

</release-id>

<configs>

<entry>

<key>PROCESS</key>

<value xsi:type="processConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<strategy>Singleton</strategy>

<kie-base-name></kie-base-name>

<kie-session-name></kie-session-name>

<merge-mode>Merge Collections</merge-mode>

</value>

</entry>

<entry>

<key>RULE</key>

<value xsi:type="ruleConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<scanner-status>STOPPED</scanner-status>

</value>

</entry>

</configs>

<status>STARTED</status>

</container-spec-details>

#### [24.8.7。[PUT] / management / server / {id} / containers / {containerId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__put_management_server_id_containers_containerid)

使用指定的containerId和给定的发行标识以及可选的配置创建一个新Container

*示例254.示例服务器请求*

**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

<container-spec-details>

<container-id>hr</container-id>

<container-name>hr</container-name>

<server-template-key>

<server-id>demo</server-id>

</server-template-key>

<release-id>

<artifact-id>HR</artifact-id>

<group-id>org.jbpm</group-id>

<version>1.0</version>

</release-id>

<configs>

<entry>

<key>PROCESS</key>

<value xsi:type="processConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<strategy>Singleton</strategy>

<kie-base-name></kie-base-name>

<kie-session-name></kie-session-name>

<merge-mode>Merge Collections</merge-mode>

</value>

</entry>

<entry>

<key>RULE</key>

<value xsi:type="ruleConfig" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<scanner-status>STOPPED</scanner-status>

</value>

</entry>

</configs>

<status>STARTED</status>

</container-spec-details

#### [24.8.8。[DELETE] / management / server / {id} / containers / {containerId}](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__delete_management_server_id_containers_containerid)

处理具有指定containerId的Container

#### [24.8.9。[POST] / management / server / {id} / containers / {containerId} / status / started](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_management_server_id_containers_containerid_status_started)

启动容器。没有要求的身体要求

#### [10年8月24日。[POST] / management / server / {id} / containers / {containerId} / status / stopped](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#__post_management_server_id_containers_containerid_status_stopped)

停止容器。没有要求的身体要求

### [24.9。Kie服务器Java客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_java_client_api)

Kie服务器有一个很好的Java API来包装REST或JMS请求以发送到服务器。在本节中，我们将探讨此API的一些可能性。

#### [24.9.1。Maven配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_configuration_2)

如果您是Maven用户，请确保您在项目的pom.xml中至少有以下依赖项

*Maven的依赖*

<dependency>

<groupId>org.kie.server</groupId>

<artifactId>kie-server-client</artifactId>

<version>${kie.api.version}</version>

</dependency>

*<!-- Logging -->*

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.1.2</version>

</dependency>

*<!-- Drools Commands -->*

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-compiler</artifactId>

<scope>runtime</scope>

<version>${kie.api.version}</version>

</dependency>

版本kie.api.version取决于您使用的Kie Server版本。例如，对于jBPM 6.3，您可以使用6.3.1-SNAPSHOT。

#### [24.9.2。客户端配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_client_configuration)

客户端需要一个配置对象，您可以在其中设置大多数服务器通信方面的信息，例如协议（REST和JMS）凭据和有效负载格式（目前支持的格式为XStream，JAXB和JSON）。首先要做的是创建你的配置，然后创建**KieServicesClient**对象，这是启动服务器通信的入口点。请参阅下面的源代码，我们使用REST客户端配置：

*客户端配置示例*

**import** org.kie.server.api.marshalling.MarshallingFormat;

**import** org.kie.server.client.KieServicesClient;

**import** org.kie.server.client.KieServicesConfiguration;

**import** org.kie.server.client.KieServicesFactory;

**public** **class** **DecisionServerTest** {

**private** **static** **final** String URL = "http://localhost:8080/kie-server/services/rest/server";

**private** **static** **final** String USER = "kieserver";

**private** **static** **final** String PASSWORD = "kieserver1!";

**private** **static** **final** MarshallingFormat FORMAT = MarshallingFormat.JSON;

**private** KieServicesConfiguration conf;

**private** KieServicesClient kieServicesClient;

**public** **void** **initialize**() {

conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

conf.setMarshallingFormat(FORMAT);

kieServicesClient = KieServicesFactory.newKieServicesClient(conf);

}

##### [24.9.2.1。JMS交互模式](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_jms_interaction_patterns)

在版本6.5中，KIE服务器客户端JMS集成得到增强，可以使用各种交互模式。目前可用的有：

* 请求回复（这是默认设置） - 使JMS集成同步 - 阻止客户端，直到它获得响应 - 不适合JMS事务用例
* 火和遗忘 - 仅使集成成为一种方式，适用于与kie服务器集成的通知 - 非常适合事务性JMS传送 - 仅当成功提交ckie服务器客户端时才向kie服务器传递消息
* 异步与回调 - 允许在发送消息到kie服务器并异步接收响应后阻止客户端 - 可以与事务性JMS传送

响应处理程序可以全局设置 - 创建KieServicesConfiguration时，或者可以在运行时在单个客户端实例上更改（如RuleServiceClient，ProcessServicesClient等）

虽然'火和忘记'和'请求回复'模式不需要任何额外的配置'与回调异步'。主要的是回调。KIE Server CLient提供了一个开箱即用的功能 - BlockingResponseCallback它提供了基于内部阻塞队列支持的基本支持。队列的大小是可以容忍的，因此允许接收多条消息，尽管此回调的意图是它一次只能接收一条消息 - 所以它就像一条消息（请求），然后每个客户端交互一条响应。

|  |  |
| --- | --- |
|  | 切换响应处理程序时，Kie Server Client不是线程安全的，这意味着处理程序的更改将影响使用相同客户端实例的所有线程。因此，如果处理程序发生动态更改，建议使用单独的客户端实例。一个好方法是维护使用专用响应处理程序的客户端集合，然后根据需要的处理程序使用这些客户端。 |

例：

客户端1将使用fire并忘记，而客户端2将使用请求回复。因此客户端1可用于启动进程，客户端2可用于查询用户任务。

用户可以通过实现org.kie.server.client.jms.ResponseCallback接口来提供自己的回调。

**组态**

Global JMS configuration

----

InitialContext context = ...;

Queue requestQueue = (Queue) context.lookup("jms/queue/KIE.SERVER.REQUEST"));

Queue responseQueue = (Queue) context.lookup("jms/queue/KIE.SERVER.RESPONSE");

ConnectionFactory connectionFactory = (ConnectionFactory) context.lookup("jms/RemoteConnectionFactory");

KieServicesConfiguration jmsConfiguration = KieServicesFactory.newJMSConfiguration( connectionFactory, requestQueue, responseQueue, "user", "password");

*// here you set response handler globally*

jmsConfiguration.setResponseHandler(**new** FireAndForgetResponseHandler());

----

或者，实际上可能更常见的是在使用各个客户端之前设置处理程序

Per client configuration

----

ProcessServiceClient processClient = client.getServicesClient(ProcessServicesClient.class);

*// change response handler for processClient others are not affected*

processClient.setResponseHandler(**new** FireAndForgetResponseHandler());

---

#### [24.9.3。服务器响应](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_server_response)

所有的服务响应都由对象表示，org.kie.server.api.model.ServiceResponse<T>其中T是有效负载的类型。它具有以下属性：

**字符串消息：**响应消息;

org.kie.server.api.model.ServiceResponse.ResponseType**键入：**响应类型枚举，它可以是SUCCESS或FAILURE;

**T结果：**响应的实际有效负载，请求的对象。

请注意，如果您使用REST或JMS，则返回相同的对象，换句话说，它对协议不可知。

#### [24.9.4。服务器功能](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_server_capabilities)

Decision Server最初只支持规则执行，从6.3版开始支持业务流程执行。要知道您的服务器支持什么，您可以通过使用客户端访问对象org.kie.server.api.model.KieServerInfo**\*** \* 列出服务器功能：

*列出服务器功能*

**public** **void** **listCapabilities**() {

KieServerInfo serverInfo = kieServicesClient.getServerInfo().getResult();

System.out.print("Server capabilities:");

**for**(String capability: serverInfo.getCapabilities()) {

System.out.print(" " + capability);

}

System.out.println();

}

如果服务器支持规则和进程，则在运行上面的代码时应打印以下内容：

服务器功能：BRM KieServer BPM

#### [24.9.5。Kie容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_containers)

如果您想发布kjar来接收请求，则必须将其发布到容器中。容器在客户端由对象表示org.kie.server.api.model.KieContainerResource，资源列表是org.kie.server.api.model.KieContainerResourceList。以下是如何打印容器列表的示例：

*列出Kie容器*

**public** **void** **listContainers**() {

KieContainerResourceList containersList = kieServicesClient.listContainers().getResult();

List<KieContainerResource> kieContainers = containersList.getContainers();

System.out.println("Available containers: ");

**for** (KieContainerResource container : kieContainers) {

System.out.println("\t" + container.getContainerId() + " (" + container.getReleaseId() + ")");

}

}

也可以根据特定的ReleaseId（及其各个部分）或状态列出容器：

*列出具有自定义过滤器的Kie容器*

**public** **void** **listContainersWithFilter**() {

*// the following filter will match only containers with ReleaseId "org.example:contatner:1.0.0.Final" and status FAILED*

KieContainerResourceFilter filter = **new** KieContainerResourceFilter.Builder()

.releaseId("org.example", "container", "1.0.0.Final")

.status(KieContainerStatus.FAILED)

.build();

KieContainerResourceList containersList = kieServicesClient.listContainers(filter).getResult();

List<KieContainerResource> kieContainers = containersList.getContainers();

System.out.println("Available containers: ");

**for** (KieContainerResource container : kieContainers) {

System.out.println("\t" + container.getContainerId() + " (" + container.getReleaseId() + ")");

}

}

#### [24.9.6。管理容器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_managing_containers_2)

您可以使用客户端来处置和创建容器。如果你处理一个容器，一个ServiceResponse将返回一个虚拟有效载荷（无有效载荷），如果你创建它，KieContainerResource对象本身将被返回到响应中。示例代码：

*处理和创建容器*

**public** **void** **disposeAndCreateContainer**() {

System.out.println("== Disposing and creating containers ==");

List<KieContainerResource> kieContainers = kieServicesClient.listContainers().getResult().getContainers();

**if** (kieContainers.size() == 0) {

System.out.println("No containers available...");

**return**;

}

KieContainerResource container = kieContainers.get(0);

String containerId = container.getContainerId();

ServiceResponse<Void> responseDispose = kieServicesClient.disposeContainer(containerId);

**if** (responseDispose.getType() == ResponseType.FAILURE) {

System.out.println("Error disposing " + containerId + ". Message: ");

System.out.println(responseDispose.getMsg());

**return**;

}

System.out.println("Success Disposing container " + containerId);

System.out.println("Trying to recreate the container...");

ServiceResponse<KieContainerResource> createResponse = kieServicesClient.createContainer(containerId, container);

**if**(createResponse.getType() == ResponseType.FAILURE) {

System.out.println("Error creating " + containerId + ". Message: ");

System.out.println(responseDispose.getMsg());

**return**;

}

System.out.println("Container recreated with success!");

}

#### [24.9.7。Decision Server的可用客户端](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_available_clients_for_the_decision_server)

KieServicesClient也是其他客户执行特定操作的切入点，例如发送BRMS命令和管理进程。目前来自KieServicesClient，您可以访问org.kie.server.client包中提供的以下服务：

* JobServicesClient：该客户端允许您安排，取消，重新请求和获取工作请求;
* ProcessServicesClient：允许您启动，发出放弃信号的过程; 完成和放弃其他能力中的工作项目;
* QueryServicesClient：强大的查询客户端允许您查询进程，进程节点和进程变量;
* RuleServicesClient：简单但功能强大的规则客户端可用于向服务器发送命令以执行与规则相关的操作（插入工作内存中的对象，消防规则，获取全局...）;
* UserTaskServicesClient：最后，用户任务客户端允许您通过某些字段（流程实例ID，用户等）执行用户任务（启动，声明，取消等）和查询任务的所有操作。

有关这些接口的更多信息，请查看github：[https](https://github.com/kiegroup/droolsjbpm-integration/tree/master/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client)：[//github.com/kiegroup/droolsjbpm-integration/tree/master/kie-server-parent/kie-server-remote/kie-server-client/src/main/java /组织/纪伊/服务器/客户端](https://github.com/kiegroup/droolsjbpm-integration/tree/master/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client)

您可以使用getServicesClientKieServicesClient类中的方法访问任何这些客户端。例如：RuleServicesClient rulesClient = kieServicesClient.getServicesClient(RuleServicesClient.class);

#### [24.9.8。发送命令到服务器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sending_commands_to_the_server)

要为服务器构建命令，您必须使用可以使用创建的类org.kie.api.command.KieCommands org.kie.api.KieServices.get().getCommands()。要发送的命令必须是**BatchExecutionCommand**或单个命令（如果发送单个命令，则服务器将其包装到BatchExecutionCommand中）：

*示例255.将命令发送到容器*

**public** **void** **executeCommands**() {

System.out.println("== Sending commands to the server ==");

RuleServicesClient rulesClient = kieServicesClient.getServicesClient(RuleServicesClient.class);

KieCommands commandsFactory = KieServices.Factory.get().getCommands();

Command<?> insert = commandsFactory.newInsert("Some String OBJ");

Command<?> fireAllRules = commandsFactory.newFireAllRules();

Command<?> batchCommand = commandsFactory.newBatchExecution(Arrays.asList(insert, fireAllRules));

ServiceResponse<String> executeResponse = rulesClient.executeCommands("hello", batchCommand);

**if**(executeResponse.getType() == ResponseType.SUCCESS) {

System.out.println("Commands executed with success! Response: ");

System.out.println(executeResponse.getResult());

}

**else** {

System.out.println("Error executing rules. Message: ");

System.out.println(executeResponse.getMsg());

}

}

这种情况下的结果是一个带有命令执行结果的String。在我们的情况下，它将打印以下内容：

== Sending commands to the server ==

Commands executed **with** success! Response:

{

"results" : [ ],

"facts" : [ ]

}

\ \*您必须添加***org.drools：drools-compiler*** dependency才能使这部分工作

在创建BatchExecutionCommand期间，可以指定一个可选的**查找**参数，该参数确定命令将运行的位置。查找参数可以是以下之一：

* Kie会话名称在部署描述符中定义
* Kie Container Id
* Kie Container Id后面跟着Process Instance Id（流程实例运行时策略必须在容器上使用）

最后两个选项需要在Kie容器上激活jBPM扩展。

在第三种情况下，查找参数的用法如下所示：

Command<?> batchCommand = commandsFactory.newBatchExecution(Arrays.asList(insert, fireAllRules), "demo-container#1");

#### [24.9.9。列出可用的业务流程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_listing_available_business_processes)

要列出流程定义，我们使用QueryClient。QueryClient的方法通常使用分页，这意味着除了您正在创建的查询之外，还必须提供当前页面和每页的结果数量。在下面的代码中，来自给定容器的流程定义查询从第0页开始并列出1000个结果，换句话说，就是1000个第一个结果。

*清单业务流程定义示例*

**public** **void** **listProcesses**() {

System.out.println("== Listing Business Processes ==");

QueryServicesClient queryClient = kieServicesClient.getServicesClient(QueryServicesClient.class);

List<ProcessDefinition> findProcessesByContainerId = queryClient.findProcessesByContainerId("rewards", 0, 1000);

**for** (ProcessDefinition def : findProcessesByContainerId) {

System.out.println(def.getName() + " - " + def.getId() + " v" + def.getVersion());

}

}

### [24.10。使用密钥存储保护密码](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_securing_password_using_key_store)

KIE服务器正在使用密码进行一些通信（例如REST API）基本身份验证。从安全角度来看，在光盘上以明文形式存储这些密码是不安全的。为此，开发了一种机制将密码存储在密钥存储区中，然后在应用程序中使用它。

#### [24.10.1。简单的用例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_simple_usecase)

用户想要保护他的密码以通过REST客户端进行通信。他创建了一个新的密钥库，他将输入密码，他将设置系统变量和密钥库的信息，KIE将自动加载密钥库并使用密码保护通信。

#### [24.10.2。实施和业务逻辑](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_implementation_and_business_logic)

如果定义了当前实现，则使用密钥存储。如果不是，使用配置参数的功能将回退到旧的行为。

#### [24.10.3。系统要求](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_system_requirements)

要使用密钥存储，我们需要先创建它。由于JKS不支持对称密钥，我们必须创建JCEKS密钥库。此外，只有Java 8及以上版本才能将密码存储在密钥存储区中。为了生成密钥存储，您可以使用标准工具KeyTool，它是JDK安装的一部分。

#### [24.10.4。初始化密钥存储](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_initialization_of_a_key_store)

对于密钥库初始化，我们推荐使用keytool。语法如下：

$ keytool -importpassword -keystore \_keystore\_url\_ -keypass \_alias\_key\_password\_ -alias \_password\_alias\_ -storepass \_keystore\_password\_ -storetype JCEKS

* **别名** - 要处理的条目的别名
* **keypass** - 密钥
* **密钥库** - 密钥库名称
* **storepass** - 密钥库密码
* **storetype** - 密钥库类型

运行此命令后，用户将被要求输入他想要存储的密码。

#### [24.10.5。加载密钥库的系统参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_system_parameters_for_loading_key_store)

* **kie.keystore.keyStoreURL** - 应该使用的密钥库的URL
* **kie.keystore.keyStorePwd** - 密钥库的密码
* **kie.keystore.key.server.alias** - 存储密码的REST服务密钥的别名
* **kie.keystore.key.server.pwd** - 存储密码的REST服务别名的密码
* **kie.keystore.key.ctrl.alias** - 存储密码的默认REST控制器密钥的别名
* **kie.keystore.key.ctrl.pwd** - 存储密码的默认REST控制器的别名密码

#### [24.10.6。例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_example_3)

1. 在应用程序服务器中创建用户和密码（它必须具有kie-server角色）

$ {EAP\_HOME} /add-user.sh -a -e -u kieserver -p“kiePassword1！” -g kie-server

1. 使用密钥工具来创建包含密码的密钥库

$ keytool -importpassword -keystore /home/kie/keystores/droolsServer.jceks -keypass keypwd -alias droolsKey -storepass serverpwd -storetype JCEKS

Enter the password to be stored:

Re-enter password:

$ keytool -importpassword -keystore /home/kie/keystores/droolsServer.jceks -keypass keypwd -alias restKey -storepass serverpwd -storetype JCEKS

Enter the password to be stored:

Re-enter password:

1. 在应用程序服务器上设置以下系统属性，以使kie服务器/控制器从密钥库读取密码

<system-properties>

<property name =“kie.keystore.keyStoreURL”value =“droolsServer.jceks”/>

<property name =“kie.keystore.keyStorePwd”value =“serverpwd”/>

<property name =“kie。 keystore.key.server.alias“value =”restKey“/>

<property name =”kie.keystore.key.server.pwd“value =”keypwd“/>

<property name =”kie.keystore.key.ctrl。别名“value =”droolsKey“/>

<property name =”kie.keystore.key.ctrl.pwd“value =”keypwd“/>

</ system-properties>

1. 启动服务器以验证配置

### [24.11。Kie服务器控制器客户端API](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_controller_client_api)

Kie服务器控制器提供了一个Java API来封装[REST](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#ControllerRestAPI-section)或Web Socket请求以发送到服务器。在本节中，我们将探讨此API的一些功能。

#### [24.11.1。Maven配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_maven_configuration_3)

如果您是Maven用户，则必须在项目的pom.xml中具有以下依赖项

*Maven的依赖*

<dependency>

<groupId>org.kie.server</groupId>

<artifactId>kie-server-controller-client</artifactId>

<version>${kie.api.version}</version>

</dependency>

*<!-- REST Client -->*

<dependency>

<groupId>org.jboss.resteasy</groupId>

<artifactId>resteasy-client</artifactId>

<version>3.0.19.Final</version>

</dependency>

*<!-- Web Socket Client -->*

<dependency>

<groupId>io.undertow</groupId>

<artifactId>undertow-websockets-jsr</artifactId>

<version>1.4.21.Final</version>

</dependency>

*<!-- Logging -->*

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.1.2</version>

</dependency>

版本kie.api.version取决于您使用的Kie Server Controller版本。例如，对于jBPM 7.6，您可以使用7.6.0.Final。

#### [24.11.2。客户端配置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_client_configuration_2)

首先要做的是创建**KieServerControllerClient**对象，这是启动服务器通信的入口点，使用提供的工厂方法可用于KieServerControllerClientFactory设置大多数服务器通信方面（如协议（REST或Web Socket）），证书，和有效载荷格式（目前，JAXB和JSON是REST协议唯一支持的格式）。请参阅下面的示例开始使用：

##### [24.11.2.1。Kie服务器控制器端点](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_kie_server_controller_endpoints)

* REST：/ rest / controller
* Web Socket：/ websocket / controller

*使用REST的客户端配置示例*

**import** org.kie.server.controller.api.model.spec.ServerTemplateList;

**import** org.kie.server.controller.client.KieServerControllerClient;

**import** org.kie.server.controller.client.KieServerControllerClientFactory;

**public** **class** **ListServerTemplatesExample** {

**private** **static** **final** String URL = "http://localhost:8080/kie-server-controller/rest/controller";

**private** **static** **final** String USER = "kieserver";

**private** **static** **final** String PASSWORD = "kieserver1!";

**public** **static** **void** **main**(String[] args) {

KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,

USER,

PASSWORD);

**final** ServerTemplateList serverTemplateList = client.listServerTemplates();

System.out.println(String.format("Found %s server template(s) at controller url: %s",

serverTemplateList.getServerTemplates().length,

URL));

}

}

*使用Web Socket的客户端配置示例*

**import** org.kie.server.controller.api.model.spec.ServerTemplateList;

**import** org.kie.server.controller.client.KieServerControllerClient;

**import** org.kie.server.controller.client.KieServerControllerClientFactory;

**public** **class** **ListServerTemplatesExample** {

**private** **static** **final** String URL = "ws://localhost:8080/kie-server-controller/websocket/controller";

**private** **static** **final** String USER = "kieserver";

**private** **static** **final** String PASSWORD = "kieserver1!";

**public** **static** **void** **main**(String[] args) {

KieServerControllerClient client = KieServerControllerClientFactory.newWebSocketClient(URL,

USER,

PASSWORD);

**final** ServerTemplateList serverTemplateList = client.listServerTemplates();

System.out.println(String.format("Found %s server template(s) at controller url: %s",

serverTemplateList.getServerTemplates().length,

URL));

}

}

##### [24.11.2.2。协议特定的设置](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_protocol_specific_settings)

###### [休息](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rest)

当使用REST协议时，Kie服务器控制器客户端允许您指定您自己的javax.ws.rs.core.Configuration。这对微调REST客户端API非常有用。作为一个例子，你可以根据你的需要调整连接时间。

*REST客户端超时配置示例*

**import** java.util.concurrent.TimeUnit;

**import** javax.ws.rs.core.Configuration;

**import** org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;

**import** org.kie.server.api.marshalling.MarshallingFormat;

**import** org.kie.server.controller.api.model.spec.ServerTemplateList;

**import** org.kie.server.controller.client.KieServerControllerClient;

**import** org.kie.server.controller.client.KieServerControllerClientFactory;

**public** **class** **RESTTimeoutExample** {

**private** **static** **final** String URL = "http://localhost:8080/kie-server-controller/rest/controller";

**private** **static** **final** String USER = "kieserver";

**private** **static** **final** String PASSWORD = "kieserver1!";

**public** **static** **void** **main**(String[] args) {

**final** Configuration configuration =

**new** ResteasyClientBuilder()

.establishConnectionTimeout(10,

TimeUnit.SECONDS)

.socketTimeout(60,

TimeUnit.SECONDS)

.getConfiguration();

KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,

USER,

PASSWORD,

MarshallingFormat.JSON,

configuration);

**final** ServerTemplateList serverTemplateList = client.listServerTemplates();

System.out.println(String.format("Found %s server template(s) at controller url: %s",

serverTemplateList.getServerTemplates().length,

URL));

}

}

###### [Web Socket](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_web_socket)

当通过Web Socket协议连接时，Kie Server Controller Client允许您根据客户端API连接到的特定Kie Server Controller中发生的更改来接收事件通知。例如，您可以收到关于连接到控制器的Kie Sever实例的通知。

*Kie服务器控制器事件示例*

**import** org.kie.server.controller.api.model.events.\*;

**import** org.kie.server.controller.api.model.spec.ServerTemplateList;

**import** org.kie.server.controller.client.KieServerControllerClient;

**import** org.kie.server.controller.client.KieServerControllerClientFactory;

**import** org.kie.server.controller.client.event.EventHandler;

**public** **class** **WebSocketEventsExample** {

**private** **static** **final** String URL = "ws://localhost:8080/kie-server-controller/websocket/controller";

**private** **static** **final** String USER = "kieserver";

**private** **static** **final** String PASSWORD = "kieserver1!";

**public** **static** **void** **main**(String[] args) {

KieServerControllerClient client = KieServerControllerClientFactory.newWebSocketClient(URL,

USER,

PASSWORD,

**new** TestEventHandler());

**final** ServerTemplateList serverTemplateList = client.listServerTemplates();

System.out.println(String.format("Found %s server template(s) at controller url: %s",

serverTemplateList.getServerTemplates().length,

URL));

**try** {

Thread.sleep(60 \* 1000);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**static** **class** **TestEventHandler** **implements** **EventHandler** {

@Override

**public** **void** **onServerInstanceConnected**(ServerInstanceConnected serverInstanceConnected) {

System.out.println("serverInstanceConnected = " + serverInstanceConnected);

}

@Override

**public** **void** **onServerInstanceDeleted**(ServerInstanceDeleted serverInstanceDeleted) {

System.out.println("serverInstanceDeleted = " + serverInstanceDeleted);

}

@Override

**public** **void** **onServerInstanceDisconnected**(ServerInstanceDisconnected serverInstanceDisconnected) {

System.out.println("serverInstanceDisconnected = " + serverInstanceDisconnected);

}

@Override

**public** **void** **onServerTemplateDeleted**(ServerTemplateDeleted serverTemplateDeleted) {

System.out.println("serverTemplateDeleted = " + serverTemplateDeleted);

}

@Override

**public** **void** **onServerTemplateUpdated**(ServerTemplateUpdated serverTemplateUpdated) {

System.out.println("serverTemplateUpdated = " + serverTemplateUpdated);

}

@Override

**public** **void** **onServerInstanceUpdated**(ServerInstanceUpdated serverInstanceUpdated) {

System.out.println("serverInstanceUpdated = " + serverInstanceUpdated);

}

@Override

**public** **void** **onContainerSpecUpdated**(ContainerSpecUpdated containerSpecUpdated) {

System.out.println("onContainerSpecUpdated = " + containerSpecUpdated);

}

}

}

#### [24.11.3。设置Kie服务器控制器](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_setting_up_a_kie_server_controller)

以下是该API的其他功能的演示。您可以按照本指南开始使用空Kie服务器控制器实例。此示例说明如何使用一些基本配置创建服务器模板以及设置单个容器。它还显示如何启动和停止特定的容器并删除新创建的服务器模板。

*安装Kie服务器控制器示例*

**import** java.util.Arrays;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.kie.server.api.model.KieContainerStatus;

**import** org.kie.server.api.model.KieScannerStatus;

**import** org.kie.server.api.model.ReleaseId;

**import** org.kie.server.controller.api.model.spec.\*;

**import** org.kie.server.controller.client.KieServerControllerClient;

**import** org.kie.server.controller.client.KieServerControllerClientFactory;

**public** **class** **WebSocketSetupControllerExample** {

**private** **static** **final** String URL = "ws://localhost:8080/kie-server-controller/websocket/controller";

**private** **static** **final** String USER = "kieserver";

**private** **static** **final** String PASSWORD = "kieserver1!";

**private** **static** KieServerControllerClient client;

**public** **static** **void** **main**(String[] args) {

KieServerControllerClient client = KieServerControllerClientFactory.newWebSocketClient(URL,

USER,

PASSWORD);

ServerTemplate serverTemplate = createServerTemplate();

ContainerSpec container = createContainer(serverTemplate);

client.startContainer(container);

client.stopContainer(container);

client.deleteServerTemplate(serverTemplate.getId());

}

**protected** **static** ServerTemplate **createServerTemplate**() {

ServerTemplate serverTemplate = **new** ServerTemplate();

serverTemplate.setId("example-client-id");

serverTemplate.setName("example-client-name");

serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name(),

Capability.RULE.name(),

Capability.PLANNING.name()));

client.saveServerTemplate(serverTemplate);

**return** serverTemplate;

}

**protected** **static** ContainerSpec **createContainer**(ServerTemplate serverTemplate){

Map<Capability, ContainerConfig> containerConfigMap = **new** HashMap();

ProcessConfig processConfig = **new** ProcessConfig("PER\_PROCESS\_INSTANCE", "kieBase", "kieSession", "MERGE\_COLLECTION");

containerConfigMap.put(Capability.PROCESS, processConfig);

RuleConfig ruleConfig = **new** RuleConfig(500l, KieScannerStatus.SCANNING);

containerConfigMap.put(Capability.RULE, ruleConfig);

ReleaseId releaseId = **new** ReleaseId("org.kie.server.testing", "stateless-session-kjar", "1.0.0-SNAPSHOT");

ContainerSpec containerSpec = **new** ContainerSpec("example-container-id", "example-client-name", serverTemplate, releaseId, KieContainerStatus.STOPPED, containerConfigMap);

client.saveContainerSpec(serverTemplate.getId(), containerSpec);

**return** containerSpec;

}

}

# 流口水的例子

帮助你学习Drools的例子

## [25.例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_examples_2)

### [25.1。获取示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_getting_the_examples)

确保安装了Drools Eclipse插件，该插件需要先安装图形编辑框架（GEF）依赖项。然后下载并提取drools-examples zip文件，其中包含已创建的Eclipse项目。将该项目导入新的Eclipse工作区。规则都具有执行规则的示例类。如果您想在另一个项目（或另一个IDE）中尝试这些示例，那么您当然需要手动设置依赖关系。很多，但不是所有的例子都记录在下面，尽情享受吧！

一些示例需要运行Java 1.6。

### [25.2。你好，世界](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_hello_world)

名称：Hello World

主类：org.drools.examples.helloworld.HelloWorldExample

模块：drools-examples

类型：Java应用程序

规则文件：HelloWorld.drl

目标：演示使用中的基本规则

“Hello World”示例显示了一个使用规则的简单应用程序，使用MVEL和Java方言编写。

这个例子演示了如何创建和使用一个KieSession。此外，显示审计日志记录和调试输出，其他示例中省略了它们，因为它们都非常相似。

以下代码片段显示了如何使用三行代码创建会话。

*例256. HelloWorld：创建KieSession*

KieServices ks = KieServices.Factory.get();

KieContainer kc = ks.getKieClasspathContainer();

KieSession ksession = kc.newKieSession("HelloWorldKS");

|  |  |
| --- | --- |
|  | 获得KieServices工厂。这是应用程序用于与引擎交互的主要界面。 |
|  | 从项目类路径创建一个KieContainer。这将查找**/META-INF/kmodule.xml** 文件来配置和实例化KieModule到KieContainer中。 |
|  | 根据指定的“HelloWorldKS”会话配置创建会话。 |

Drools有一个事件模型，可以揭示内部发生的大部分事件。两个默认的调试监听器提供的，DebugAgendaEventListener而DebugWorkingMemoryEventListener其打印出的调试事件信息System.err在控制台窗口中显示的流。将侦听器添加到Session很简单，如下面的代码片段所示。在KieRuntimeLogger提供执行审计，这些都可以在一个图形可被观众看到的结果。记录器实际上是建立在一个专门的实现Agenda和RuleRuntime听众。当引擎完成执行时，logger.close()必须被调用。

大多数示例使用Drools的审计日志记录功能来记录执行流程以供日后检查。

*例子257. HelloWorld：事件记录和审计*

*// The application can also setup listeners*

ksession.addEventListener( **new** DebugAgendaEventListener() );

ksession.addEventListener( **new** DebugRuleRuntimeEventListener() );

*// To setup a file based audit logger, uncomment the next line*

*// KieRuntimeLogger logger = ks.getLoggers().newFileLogger( ksession, "./helloworld" );*

*// To setup a ThreadedFileLogger, so that the audit view reflects events whilst debugging,*

*// uncomment the next line*

*// KieRuntimeLogger logger = ks.getLoggers().newThreadedFileLogger( ksession, "./helloworld", 1000 );*

本例中使用的单个类非常简单。它有两个领域：消息，这是一个String和可以是两个整数HELLO或状态之一的状态GOODBYE。

*例子258. HelloWorld例子：Message Class*

**public** **static** **class** **Message** {

**public** **static** **final** **int** HELLO = 0;

**public** **static** **final** **int** GOODBYE = 1;

**private** String message;

**private** **int** status;

...

}

Message用消息文本“Hello World”和状态创建单个对象HELLO，然后将其插入引擎中，并在该位置fireAllRules()执行。

*例259. HelloWorld：执行*

*// The application can insert facts into the session*

**final** Message message = **new** Message();

message.setMessage( "Hello World" );

message.setStatus( Message.HELLO );

ksession.insert( message );

*// and fire the rules*

ksession.fireAllRules();

以Java应用程序的形式执行该示例：

1. org.drools.examples.helloworld.HelloWorldExample在Eclipse IDE中打开该类
2. 右键单击该类并选择“以...运行...”，然后选择“Java应用程序”

如果我们在fireAllRules()方法上放置一个断点并选择ksession变量，我们可以看到议程中的“Hello World”规则已经被激活。

*图388. Hello World：fireAllRules日程视图*

应用程序打印输出到System.out调试监听器打印输出时去System.err。

*例260.控制台窗口中的HelloWorld：System.out*

**Hello** World

Goodbye cruel world

*例261.控制台窗口中的HelloWorld：System.err*

==>[ActivationCreated(0): rule=Hello World;

tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]

[ObjectInserted: handle=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];

object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]

[BeforeActivationFired: rule=Hello World;

tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]

==>[ActivationCreated(4): rule=Good Bye;

tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]

[ObjectUpdated: handle=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];

old\_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96;

new\_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]

[AfterActivationFired(0): rule=Hello World]

[BeforeActivationFired: rule=Good Bye;

tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]

[AfterActivationFired(4): rule=Good Bye]

实际的规则在文件src / main / resources / org / drools / examples / helloworld / HelloWorld.drl中 ：

*例262. HelloWorld：规则“Hello World”*

rule "Hello World"

dialect "mvel"

**when**

m : Message( status == Message.HELLO, message : message )

**then**

System.out.println( message );

modify ( m ) { message = "Goodbye cruel world",

status = Message.GOODBYE };

**end**

规则的LHS（之后when）部分指出，对于Message插入状态为“规则运行时”的每个对象，它都将被激活Message.HELLO。除此之外，还创建了两个变量绑定：变量message绑定到message属性，变量m绑定到匹配的Message对象本身。

该规则的RHS（之后then）或结果部分是使用MVEL表达式语言编写的，如规则属性所声明的dialect。打印绑定变量的内容后message向System.out，规则改变的值message和status所述的属性Message绑定到对象m。这是通过使用MVEL的modify语句完成的，该语句允许您在一个语句中应用一个赋值块，引擎会自动通知块末尾的变化。

可以在modify呼叫中为DRL设置一个断点，并在执行规则的结果期间再次检查议程视图。这次我们通过“Debug As”和“Drools应用程序”开始执行，而不是运行“Java应用程序”：

1. org.drools.examples.HelloWorld在Eclipse IDE中打开该类。
2. 右键单击该类并选择“调试为...”，然后选择“Drools应用程序”。

现在我们可以看到"Good Bye"使用Java语言的另一个规则被激活并放置在议程中。

*图389. Hello World：规则“Hello World”议程视图*

指定“java”方言的“Good Bye”规则与“Hello World”规则相似，只不过它与Message状态为的对象相匹配Message.GOODBYE。

*例263. HelloWorld：规则“再见”*

rule "Good Bye"

dialect "java"

**when**

**Message**( status == Message.GOODBYE, message : message )

then

System.**out**.**println**( message );

end

实例化的Java代码KieRuntimeLogger创建了一个可以加载到审计视图中的审计日志文件。Audit视图在许多示例中用于演示示例执行流程。在下面的视图屏幕截图中，我们可以看到该对象已插入，这将创建“Hello World”规则的激活; 然后执行激活，其更新Message导致“再见”规则激活的对象; 最后还会执行“Good Bye”规则。在审计视图中选择一个事件以绿色突出显示原点事件; 因此“激活创建”事件以绿色突出显示为“激活已执行”事件的来源。

*图390. Hello World：审核视图*

### [25.3。状态示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_state_example)

这个例子实现了两种不同的版本，以演示实现相同基本行为的不同方式：前向链接，即引擎根据工作内存中事实的变化评估，激活和激发规则的能力。

#### [25.3.1。了解状态示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_understanding_the_state_example)

名称：状态示例

主类：org.drools.examples.state.StateExampleUsingSalience

模块：drools示例

类型：Java应用程序

规则文件：StateExampleUsingSalience.drl

目标：演示规则激发优先级的基本规则使用和冲突解决。

每个State类都有其名称和当前状态的字段（请参阅该类org.drools.examples.state.State）。每个对象的两种可能状态是：

* NOTRUN
* FINISHED

*例子264.状态类*

**public** **class** **State** {

**public** **static** **final** **int** NOTRUN = 0;

**public** **static** **final** **int** FINISHED = 1;

**private** **final** PropertyChangeSupport changes =

**new** PropertyChangeSupport( **this** );

**private** String name;

**private** **int** state;

... setters and getters go here...

}

忽略这个PropertyChangeSupport，稍后会解释，我们看到State名为A，B，C和D 的四个对象的创建。最初它们的状态设置为NOTRUN，这是所使用的构造函数的默认值。每个实例依次被声明为Session，然后fireAllRules()被调用。

*例265.突出状态：执行*

**final** State a = **new** State( "A" );

**final** State b = **new** State( "B" );

**final** State c = **new** State( "C" );

**final** State d = **new** State( "D" );

ksession.insert( a );

ksession.insert( b );

ksession.insert( c );

ksession.insert( d );

ksession.fireAllRules();

ksession.dispose(); *// Stateful rule session must always be disposed when finished*

要执行该应用程序：

1. org.drools.examples.state.StateExampleUsingSalience在Eclipse IDE中打开该类。
2. 右键单击该类并选择“以...运行...”，然后选择“Java应用程序”

您将在Eclipse控制台窗口中看到以下输出：

*示例266.突出状态：控制台输出*

**A** finished

B finished

C finished

D finished

共有四条规则。首先，Bootstrap规则触发，将A设置为状态FINISHED，然后导致B将其状态更改为FINISHED。C和D都依赖于B，从而导致由显着性值解决的冲突。我们来看看执行的方式。

了解正在发生的事情的最好方法是使用审计日志功能以图形方式查看每个操作的结果。查看由此示例运行生成的审核日志：

1. 如果审计视图不可见，请点击“窗口”，然后选择“显示视图”，然后选择“其他...”和“流水帐”，最后选择“审计视图”。
2. 在“审核视图”中点击“打开日志”按钮并选择文件“<drools-examples-dir> /log/state.log”。

之后，“审核视图”将如下图所示：

*图391.突出状态示例审计视图*

从上到下阅读“审计视图”中的日志，我们可以看到工作内存中的每个操作和相应的更改。通过这种方式，我们观察到状态对象A在状态中的断言NOTRUN激活Bootstrap规则，而其他State对象的断言不会立即生效。

*例267.突显状态：规则“引导”*

rule Bootstrap

**when**

a : State(name == "A", state == State.NOTRUN )

**then**

System.out.println(a.getName() + " finished" );

a.setState( State.FINISHED );

**end**

规则引导程序的执行将A的状态更改为FINISHED，从而激活规则“A到B”。

*例268.突出状态：规则“A到B”*

rule "A to B"

**when**

State(name == "A", **state** == State.FINISHED )

b : State(name == "B", **state** == State.NOTRUN )

then

System.out.println(b.getName() + " finished" );

b.setState( State.FINISHED );

end

规则“A到B”的执行将B的状态更改为FINISHED，同时激活规则“B到C”和“B到D”，将其激活置于议程中。从现在开始，这两条规则可能会触发，因此它们被认为是“冲突的”。冲突解决策略允许引擎的议程决定要触发哪个规则。由于规则“B到C”具有**较高的显着性值**（10对比默认显着性值0），所以它首先触发，修改对象C到状态FINISHED。上面显示的“审计”视图反映了修改State对象的规则“A到B”，这导致两个激活冲突。议程观点也可以用来调查议程的状态，调试点放在规则本身和议程观点中。下面的屏幕截图显示了规则“A到B”中的断点和具有两个冲突规则的议程状态。

*图392.状态示例议程视图*

*例269.突出状态：规则“B到C”*

rule "B to C"

salience 10

**when**

State(name == "B", **state** == State.FINISHED )

c : State(name == "C", **state** == State.NOTRUN )

then

System.out.println(c.getName() + " finished" );

c.setState( State.FINISHED );

end

规则“B到D”最后触发，将对象D修改为状态FINISHED。

*例270.显着性状态：规则“B到D”*

rule "B to D"

**when**

State(name == "B", **state** == State.FINISHED )

d : State(name == "D", **state** == State.NOTRUN )

then

System.out.println(d.getName() + " finished" );

d.setState( State.FINISHED );

end

没有更多规则可以执行，因此引擎停止。

这个例子中另一个值得注意的概念是基于对象的动态事实的使用PropertyChangeListener。如文档中所述，为了让引擎看到并响应事实属性的更改，应用程序必须告诉引擎发生了更改。这可以通过使用modify语句在规则中显式完成，或者通过让引擎知道JavaBeans规范PropertyChangeSupport定义的事实实现来隐式完成。这个例子演示了如何用来避免规则中显式语句的需要。要利用此功能，请确保您的事实与班级的实施方式相同PropertyChangeSupportmodifyPropertyChangeSupportorg.drools.example.State 并且在规则文件中使用以下代码来配置引擎以侦听这些事实的属性更改：

*例271.声明一个动态事实*

declare type State

@propertyChangeSupport

end

当使用PropertyChangeListener对象时，每个设置者必须为通知实现一些额外的代码。这是state在课堂上的二传手org.drools.examples：

*示例272.使用PropertyChangeSupport设置示例*

**public** **void** **setState**(**final** **int** newState) {

**int** oldState = **this**.state;

**this**.state = newState;

**this**.changes.firePropertyChange( "state",

oldState,

newState );

}

这个例子中还有另一个类：StateExampleUsingAgendaGroup。如图所示，它执行从A到B到C到D，但是StateExampleUsingAgendaGroup使用议程组来控制规则冲突以及哪一个先触发。

议程小组是将议程分成小组并控制哪些小组可以执行的一种方式。默认情况下，所有规则都在议程组“MAIN”中。“议程组”属性可让您为规则指定不同的议程组。最初，工作记忆将重点放在议程小组“主要”上。小组的规则只有在小组获得焦点时才会触发。这可以通过使用方法setFocus()或规则属性来实现auto-focus。“自动对焦”意味着规则在匹配和激活时自动将焦点设置到其议程组。这是“自动对焦”，使“B到C”规则能够在“B到D”之前发射。

*例子273.议程组状态例子：规则“B到C”*

rule "B to C"

agenda-**group** "B to C"

auto-focus **true**

**when**

**State**(name == "B", state == State.FINISHED )

c : **State**(name == "C", state == State.NOTRUN )

then

System.**out**.**println**(c.getName() + " finished" );

c.setState( State.FINISHED );

kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "B to D" ).setFocus();

end

规则“B到C”呼叫setFocus()议程组“B到D”，允许其激活规则触发，从而允许触发规则“B到D”。

*例274.议程组状态例子：规则“B到D”*

rule "B to D"

agenda-**group** "B to D"

**when**

**State**(name == "B", state == State.FINISHED )

d : **State**(name == "D", state == State.NOTRUN )

then

System.**out**.**println**(d.getName() + " finished" );

d.setState( State.FINISHED );

end

### [25.4。斐波那契示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_fibonacci_example)

名称：Fibonacci

主类：org.drools.examples.fibonacci.FibonacciExample

模块：drools-examples

类型：Java应用程序

规则文件：Fibonacci.drl

目标：演示递归，

CE不是和跨产品匹配

由比萨的莱昂纳多发现的斐波纳契数（参见<http://en.wikipedia.org/wiki/Fibonacci_number>）（参见<http://en.wikipedia.org/wiki/Fibonacci>）是一个以0和1开头的序列。下一个斐波那契数是通过加上两个前面的斐波那契数得到的。斐波那契数列以0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946 ...斐波那契示例演示递归和冲突解决与显着值。

Fibonacci本例中使用单个事实类。它有两个字段，顺序和值。序列字段用于指示斐波纳契数列中对象的位置。值字段显示该序列位置的Fibonacci对象的值，使用-1表示仍需要计算的值。

*例275.斐波那契类*

**public** **static** **class** **Fibonacci** {

**private** **int** sequence;

**private** **long** value;

**public** **Fibonacci**( **final** **int** sequence ) {

**this**.sequence = sequence;

**this**.value = -1;

}

... setters and getters go here...

}

执行示例：

1. org.drools.examples.fibonacci.FibonacciExample在Eclipse IDE中打开该类。
2. 右键单击该类并选择“以...运行...”，然后选择“Java应用程序”

Eclipse在其控制台窗口中显示以下输出（使用“... snip ...”表示为了节省空间而删除的行）：

*示例276. Fibonacci示例：控制台输出*

recurse **for** 50

recurse **for** 49

recurse **for** 48

recurse **for** 47

...snip...

recurse **for** 5

recurse **for** 4

recurse **for** 3

recurse **for** 2

1 == 1

2 == 1

3 == 2

4 == 3

5 == 5

6 == 8

...snip...

47 == 2971215073

48 == 4807526976

49 == 7778742049

50 == 12586269025

为了解决Java的问题，我们只插入一个单一的斐波那契对象，序列字段为50，然后用递归规则插入其他49个Fibonacci对象。这个例子不使用PropertyChangeSupport。它使用MVEL方言，这意味着我们可以使用modify关键字，该关键字允许块设置器操作，该操作还会通知引擎更改。

*例277. Fibonacci例子：执行*

ksession.insert( **new** Fibonacci( 50 ) );

ksession.fireAllRules();

规则递归非常简单。它将每个声明的Fibonacci对象与值-1 相匹配，创建并声明一个新的Fibonacci对象，其序列小于当前匹配的对象。每次添加斐波纳契对象时，序列字段等于1的对象不存在，则规则重新匹配并再次触发。在not有条件的元素被用来阻止规则的匹配，一旦我们在内存中的所有50个斐波那契对象。该规则还有一个显着的价值，因为我们需要Fibonacci在执行Bootstrap规则之前拥有所有50个对象。

*例278. Fibonacci例子：Rule“Recurse”*

rule Recurse

salience 10

when

f : Fibonacci ( value == -1 )

not ( Fibonacci ( sequence == 1 ) )

then

**insert**( **new** Fibonacci( **f**.**sequence** - 1 ) );

System.out.println( "recurse for " + f.sequence );

**end**

审计视图显示Fibonacci序列字段为50 的对象的原始断言，从Java代码完成。从那里开始，Audit视图显示规则的连续递归，其中每个声明的Fibonacci对象导致Recurse规则被激活并再次触发。

*图393. Fibonacci示例：“递归”审计视图1*

当Fibonacci序列字段为2 的对象被声明时，“Bootstrap”规则与“Recurse”规则一起被匹配和激活。注意字段上的多重限制sequence，测试与1或2相等。

*例279. Fibonacci例子：Rule“Bootstrap”*

rule Bootstrap

**when**

f : Fibonacci( sequence == 1 || == 2, **value** == -1 ) *// multi-restriction*

then

**modify** ( f ){ **value** = 1 };

System.**out**.println( f.sequence + " == " + f.**value** );

end

在这一点上，议程看起来如下所示。但是，“Bootstrap”规则不会触发，因为“Recurse”规则具有更高的显着性。

*图394. Fibonacci示例：“递归”议程视图1*

当Fibonacci序列为1 的对象被声明时，引导规则再次匹配，导致对此规则进行两次激活。请注意，“Recurse”规则不匹配并激活，因为not条件元素在Fibonacci序列为1 的对象存在时立即停止规则的匹配。

*图395.斐波纳契示例：“递归”议程视图2*

一旦我们有两个Fibonacci值不等于-1的对象，“计算”规则就可以匹配。这是“Bootstrap”规则，它将序列1和2的对象设置为值1.此时，我们在工作内存中有50个斐波纳契对象。现在我们需要选择一个合适的三元组来依次计算它们的每个值。在没有字段约束的规则中使用三个斐波纳契模式来限制可能的交叉产品将导致50x49x48的可能组合，导致约125,000个可能的规则触发，其中大多数是不正确的。“计算”规则使用字段约束以正确的顺序正确约束斐波纳契模式; 这种技术被称为。第一个模式找到值为！= -1的任意斐波那契，并绑定模式和字段。第二个斐波那契也是这样做的，但它增加了一个额外的场约束来确保它的序列比Fibonacci所绑定的序列大1 f1。当这个规则第一次触发时，我们知道只有序列1和2的值为1，并且这两个约束确保了f1参考序列1和f2参考序列2.最终模式发现斐波那契的值等于-1，比序列大一个f2。在这一点上，我们有三个Fibonacci对象从可用的交叉产品中正确选择，我们可以计算Fibonacci绑定的第三个对象的值f3。

*例280.斐波纳契例子：规则“计算”*

rule Calculate

**when**

*// Bind f1 and s1*

f1 : Fibonacci( s1 : sequence, **value** != -1 )

*// Bind f2 and v2; refer to bound variable s1*

f2 : Fibonacci( sequence == (s1 + 1), v2 : **value** != -1 )

*// Bind f3 and s3; alternative reference of f2.sequence*

f3 : Fibonacci( s3 : sequence == (f2.sequence + 1 ), **value** == -1 )

then

*// Note the various referencing techniques.*

modify ( f3 ) { **value** = f1.**value** + v2 };

System.**out**.println( s3 + " == " + f3.**value** );

end

该modify语句更新了Fibonacci绑定到的对象的值f3。这意味着我们现在有另一个新的斐波纳契对象，其值不等于-1，这允许“计算”规则重新匹配并计算下一个斐波那契数。下面的Audit视图显示了如何触发最后一个“Bootstrap”修改Fibonacci对象，使“计算”规则匹配，然后修改另一个允许“计算”规则再次匹配的Fibonacci对象。这一直持续到为所有Fibonacci对象设置值。

*图396.斐波那契示例：“Bootstrap”审计视图*

### [25.5。银行教程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_banking_tutorial)

名称：BankingTutorial

主类：org.drools.examples.banking.BankingExamplesApp.java

模块：drools-examples

类型：Java应用程序

规则文件：org.drools.examples.banking。\*。drl

目标：演示模式匹配，基本排序和计算规则。

本教程演示了开发完整的个人银行应用程序以处理多个账户的信用和借记的过程。它使用一组为设计过程创建的设计模式。

该类RuleRunner是一个简单的工具，可以根据一组数据执行一个或多个DRL文件。它编译软件包并为每个执行创建知识库，使我们能够轻松执行每个场景并检查输出。实际上，对于一个只需构建一次知识库并进行缓存的生产系统来说，这不是一个好的解决方案，但对于本教程来说，它就足够了。

*例281.银行教程：RuleRunner*

**public** **class** **RuleRunner** {

**public** **RuleRunner**() {

}

**public** **void** **runRules**(String[] rules,

Object[] facts) **throws** Exception {

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

**for** ( **int** i = 0; i < rules.length; i++ ) {

String ruleFile = rules[i];

System.out.println( "Loading file: " + ruleFile );

kbuilder.add( ResourceFactory.newClassPathResource( ruleFile,

RuleRunner.class ),

ResourceType.DRL );

}

Collection<KnowledgePackage> pkgs = kbuilder.getKnowledgePackages();

kbase.addKnowledgePackages( pkgs );

StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();

**for** ( **int** i = 0; i < facts.length; i++ ) {

Object fact = facts[i];

System.out.println( "Inserting fact: " + fact );

ksession.insert( fact );

}

ksession.fireAllRules();

}

}

我们的第一个示例Java类加载并执行一个DRL文件Example.drl ，但不插入任何数据。

*例子282.银行教程：Java例子1*

**public** **class** **Example1** {

**public** **static** **void** **main**(String[] args) **throws** Exception {

**new** RuleRunner().runRules( **new** String[] { "Example1.drl" },

**new** Object[0] );

}

}

要执行的第一条简单规则具有一个eval始终为真的单一条件，以便此规则将在启动后匹配并触发一次。

*例283.银行教程：Example1.drl中的规则*

rule "Rule 01"

**when**

**eval**( 1==1 )

then

System.**out**.**println**( "Rule 01 Works" );

end

规则的输出如下，显示规则匹配并执行单个打印语句。

*例284.银行教程：Example1.java的输出*

Loading file: Example1.drl

Rule 01 Works

下一步是提出一些简单的事实并将其打印出来。

*例285.银行教程：Java Example2*

**public** **class** **Example2** {

**public** **static** **void** **main**(String[] args) **throws** Exception {

Number[] numbers = **new** Number[] {wrap(3), wrap(1), wrap(4), wrap(1), wrap(5)};

**new** RuleRunner().runRules( **new** String[] { "Example2.drl" },

numbers );

}

**private** **static** Integer **wrap**( **int** i ) {

**return** **new** Integer(i);

}

}

这不使用任何特定的事实，而是声明一组java.lang.Integer对象。这不被认为是“最佳实践”，因为数字不是一个有用的事实，但我们在此使用它来演示基本技术，然后再添加更多复杂性。

现在我们将创建一个简单的规则来打印出这些数字。

*例286.银行教程：Example2.drl中的规则*

rule "Rule 02"

**when**

Number( $intValue : intValue )

**then**

System.out.println( "Number found with value: " + $intValue );

**end**

这个规则再一次没有什么特别之处。它识别任何属于Number对象的事实并打印出值。注意抽象类的使用Number：我们插入了Integer对象，但我们现在查找任何类型的数字。模式匹配引擎能够匹配断言对象的接口和超类。

输出显示正在加载的DRL，插入的事实以及匹配和触发的规则。我们可以看到，每个插入的数字都是匹配的，并被打印出来。

*例287.银行教程：Example2.java的输出*

Loading file: Example2.drl

Inserting fact: 3

Inserting fact: 1

Inserting fact: 4

Inserting fact: 1

Inserting fact: 5

Number found with value: 5

Number found with value: 1

Number found with value: 4

Number found with value: 1

Number found with value: 3

当然，有很多更好的方法来对数字进行排序，而不是使用规则，但是因为当我们开始查看银行业务规则时，我们将需要按日期顺序应用一些现金流，我们将开发基于规则的简单排序技术。

*例子288.银行教程：Example3.java*

**public** **class** **Example3** {

**public** **static** **void** **main**(String[] args) **throws** Exception {

Number[] numbers = **new** Number[] {wrap(3), wrap(1), wrap(4), wrap(1), wrap(5)};

**new** RuleRunner().runRules( **new** String[] { "Example3.drl" },

numbers );

}

**private** **static** Integer **wrap**(**int** i) {

**return** **new** Integer(i);

}

}

我们再次插入我们的Integer对象，但是这次规则稍有不同：

*例289.银行教程：Example3.drl中的规则*

rule "Rule 03"

**when**

$number : Number( )

**not** Number( intValue < $number.intValue )

**then**

System.out.println("Number found with value: " + $number.intValue() );

retract( $number );

**end**

规则的第一行标识a Number并提取值。第二行确保不存在比第一个模式找到的更小的数字。我们可能期望只匹配一个数字 - 在集合中最小的一个。但是，打印后的数字缩回意味着最小的数字已被删除，显示下一个最小的数字，依此类推。

结果输出显示数字现在按数字排序。

*例290. Banking教程：Example3.java的输出*

Loading file: Example3.drl

Inserting fact: 3

Inserting fact: 1

Inserting fact: 4

Inserting fact: 1

Inserting fact: 5

Number found with value: 1

Number found with value: 1

Number found with value: 3

Number found with value: 4

Number found with value: 5

我们准备开始走向个人会计规则。第一步是创建一个Cashflow

*例子291.银行教程：类现金流*

**public** **class** **Cashflow** {

**private** Date date;

**private** **double** amount;

**public** **Cashflow**() {

}

**public** **Cashflow**(Date date, **double** amount) {

**this**.date = date;

**this**.amount = amount;

}

**public** Date **getDate**() {

**return** date;

}

**public** **void** **setDate**(Date date) {

**this**.date = date;

}

**public** **double** **getAmount**() {

**return** amount;

}

**public** **void** **setAmount**(**double** amount) {

**this**.amount = amount;

}

**public** String **toString**() {

**return** "Cashflow[date=" + date + ",amount=" + amount + "]";

}

}

类Cashflow有两个简单的属性，一个日期和一个金额。（请注意，使用double货币单位类型通常不是一个好主意，因为浮点数不能准确表示大多数数字。）还有一个用于设置值的重载构造函数和一个toString用于打印现金流的方法。Example4.java的Java代码 插入五个现金流对象，具有不同的日期和金额。

*例292.银行教程：Example4.java*

**public** **class** **Example4** {

**public** **static** **void** **main**(String[] args) **throws** Exception {

Object[] cashflows = {

**new** Cashflow(**new** SimpleDate("01/01/2007"), 300.00),

**new** Cashflow(**new** SimpleDate("05/01/2007"), 100.00),

**new** Cashflow(**new** SimpleDate("11/01/2007"), 500.00),

**new** Cashflow(**new** SimpleDate("07/01/2007"), 800.00),

**new** Cashflow(**new** SimpleDate("02/01/2007"), 400.00),

};

**new** RuleRunner().runRules( **new** String[] { "Example4.drl" },

cashflows );

}

}

便利类SimpleDate扩展java.util.Date，提供了一个构造函数，它以String作为输入并定义日期格式。代码如下所示

*例293.银行教程：SimpleDate类*

**public** **class** **SimpleDate** **extends** **Date** {

**private** **static** **final** SimpleDateFormat format = **new** SimpleDateFormat("dd/MM/yyyy");

**public** **SimpleDate**(String datestr) **throws** Exception {

setTime(format.parse(datestr).getTime());

}

}

现在，让我们看一下Example4.drl ，看看我们如何打印排序后的Cashflow对象：

*例子294.银行教程：Example4.drl中的规则*

rule "Rule 04"

**when**

$cashflow : Cashflow( $date : date, $amount : amount )

**not** Cashflow( date < $date)

**then**

System.out.println("Cashflow: "+$date+" :: "+$amount);

retract($cashflow);

**end**

在这里，我们确定Cashflow并提取日期和金额。在该规则的第二行中，我们确保没有比发现的现金流更早的日期。结果，我们打印出Cashflow符合规则的文件，然后将其缩回，为下一个最早的文件做好准备Cashflow。所以，我们产生的输出是：

*例295.银行教程：Example4.java的输出*

Loading file: Example4.drl

Inserting fact: Cashflow[date=Mon Jan 01 00:00:00 GMT 2007,amount=300.0]

Inserting fact: Cashflow[date=Fri Jan 05 00:00:00 GMT 2007,amount=100.0]

Inserting fact: Cashflow[date=Thu Jan 11 00:00:00 GMT 2007,amount=500.0]

Inserting fact: Cashflow[date=Sun Jan 07 00:00:00 GMT 2007,amount=800.0]

Inserting fact: Cashflow[date=Tue Jan 02 00:00:00 GMT 2007,amount=400.0]

Cashflow: Mon Jan 01 00:00:00 GMT 2007 :: 300.0

Cashflow: Tue Jan 02 00:00:00 GMT 2007 :: 400.0

Cashflow: Fri Jan 05 00:00:00 GMT 2007 :: 100.0

Cashflow: Sun Jan 07 00:00:00 GMT 2007 :: 800.0

Cashflow: Thu Jan 11 00:00:00 GMT 2007 :: 500.0

接下来，我们扩展我们的业务Cashflow，产生TypedCashflow可以是信用卡或借记卡业务的信用卡。（通常，我们只是将它添加到Cashflow

*例子296.银行教程：Class TypedCashflow*

**public** **class** **TypedCashflow** **extends** **Cashflow** {

**public** **static** **final** **int** CREDIT = 0;

**public** **static** **final** **int** DEBIT = 1;

**private** **int** type;

**public** **TypedCashflow**() {

}

**public** **TypedCashflow**(Date date, **int** type, **double** amount) {

**super**( date, amount );

**this**.type = type;

}

**public** **int** **getType**() {

**return** type;

}

**public** **void** **setType**(**int** type) {

**this**.type = type;

}

**public** String **toString**() {

**return** "TypedCashflow[date=" + getDate() +

",type=" + (type == CREDIT ? "Credit" : "Debit") +

",amount=" + getAmount() + "]";

}

}

有很多方法可以改进这个代码，但是为了这个例子，它会这样做。

现在我们来创建Example5，这是一个运行我们的代码的类。

*例子297.银行教程：Example5.java*

**public** **class** **Example5** {

**public** **static** **void** **main**(String[] args) **throws** Exception {

Object[] cashflows = {

**new** TypedCashflow(**new** SimpleDate("01/01/2007"),

TypedCashflow.CREDIT, 300.00),

**new** TypedCashflow(**new** SimpleDate("05/01/2007"),

TypedCashflow.CREDIT, 100.00),

**new** TypedCashflow(**new** SimpleDate("11/01/2007"),

TypedCashflow.CREDIT, 500.00),

**new** TypedCashflow(**new** SimpleDate("07/01/2007"),

TypedCashflow.DEBIT, 800.00),

**new** TypedCashflow(**new** SimpleDate("02/01/2007"),

TypedCashflow.DEBIT, 400.00),

};

**new** RuleRunner().runRules( **new** String[] { "Example5.drl" },

cashflows );

}

}

在这里，我们简单地创建一组Cashflow对象，这些对象既可以是信用卡也可以是借记卡。我们将它们和Example5.drl提供 给RuleEngine。

现在，让我们看看打印排序的规则 Cashflow

*例298.银行教程：Example5.drl中的规则*

rule "Rule 05"

**when**

$cashflow : TypedCashflow( $date : date,

$amount : amount,

type == TypedCashflow.CREDIT )

**not** TypedCashflow( date < $date,

type == TypedCashflow.CREDIT )

**then**

System.out.println("Credit: "+$date+" :: "+$amount);

retract($cashflow);

**end**

在这里，我们Cashflow用一种类型标识事实CREDIT并提取日期和金额。在规则的第二行中，我们确保不存在Cashflow具有比找到的更早的日期的相同类型。结果，我们打印满足模式的现金流，然后将其收回，为下一个最早的现金流类型制造出路CREDIT

所以，我们产生的输出是

*例299.银行教程：Example5.java的输出*

加载文件：Example5.drl

插入事实：TypedCashflow [date = Mon Jan 01 00:00:00 GMT 2007，type = Credit，amount = 300.0]

插入事实：TypedCashflow [date = Fri Jan 05 00:00:00 GMT 2007， Type = Credit，amount = 100.0]

插入事实：TypedCashflow [date = Thu Jan 11 00:00:00 GMT 2007，type = Credit，amount = 500.0]

插入事实：TypedCashflow [date = Sun Jan 07 00:00:00 GMT 2007年，类型=借方，金额= 800.0]

插入事实：TypedCashflow [日期=星期二1月02 00:00:00 GMT 2007，类型=借方，金额= 400.0]

信用：Mon Jan 01 00:00:00 GMT 2007 :: 300.0

Credit：Fri Jan 05 00:00:00 GMT 2007 :: 100.0

Credit：Thu Jan 11 00:00:00 GMT 2007 :: 500.0

继续我们的银行业务，我们现在要处理两个银行账户的贷方和借记，计算账户余额。为了做到这一点，我们创建了两个独立的Account对象，并在将它们Cashflows传递给规则引擎之前将它们注入到对象中。这样做的原因是为了方便访问正确的帐户而不必求助于辅助类。我们先来看看这个Account班。这是一个带有帐号和余额的简单Java对象：

*例子300.银行教程：班级账户*

**public** **class** **Account** {

**private** **long** accountNo;

**private** **double** balance = 0;

**public** **Account**() {

}

**public** **Account**(**long** accountNo) {

**this**.accountNo = accountNo;

}

**public** **long** **getAccountNo**() {

**return** accountNo;

}

**public** **void** **setAccountNo**(**long** accountNo) {

**this**.accountNo = accountNo;

}

**public** **double** **getBalance**() {

**return** balance;

}

**public** **void** **setBalance**(**double** balance) {

**this**.balance = balance;

}

**public** String **toString**() {

**return** "Account[" + "accountNo=" + accountNo + ",balance=" + balance + "]";

}

}

现在我们来扩展我们的TypedCashflow结果AllocatedCashflow，包括一个Account参考。

*例子301. Banking教程：Class AllocatedCashflow*

**public** **class** **AllocatedCashflow** **extends** **TypedCashflow** {

**private** Account account;

**public** **AllocatedCashflow**() {

}

**public** **AllocatedCashflow**(Account account, Date date, **int** type, **double** amount) {

**super**( date, type, amount );

**this**.account = account;

}

**public** Account **getAccount**() {

**return** account;

}

**public** **void** **setAccount**(Account account) {

**this**.account = account;

}

**public** String **toString**() {

**return** "AllocatedCashflow[" +

"account=" + account +

",date=" + getDate() +

",type=" + (getType() == CREDIT ? "Credit" : "Debit") +

",amount=" + getAmount() + "]";

}

}

Example5.java的Java代码 创建两个Account对象，并在构造函数调用中将其中一个传递给每个现金流。

*例子302.银行教程：Example5.java*

**public** **class** **Example6** {

**public** **static** **void** **main**(String[] args) **throws** Exception {

Account acc1 = **new** Account(1);

Account acc2 = **new** Account(2);

Object[] cashflows = {

**new** AllocatedCashflow(acc1,**new** SimpleDate("01/01/2007"),

TypedCashflow.CREDIT, 300.00),

**new** AllocatedCashflow(acc1,**new** SimpleDate("05/02/2007"),

TypedCashflow.CREDIT, 100.00),

**new** AllocatedCashflow(acc2,**new** SimpleDate("11/03/2007"),

TypedCashflow.CREDIT, 500.00),

**new** AllocatedCashflow(acc1,**new** SimpleDate("07/02/2007"),

TypedCashflow.DEBIT, 800.00),

**new** AllocatedCashflow(acc2,**new** SimpleDate("02/03/2007"),

TypedCashflow.DEBIT, 400.00),

**new** AllocatedCashflow(acc1,**new** SimpleDate("01/04/2007"),

TypedCashflow.CREDIT, 200.00),

**new** AllocatedCashflow(acc1,**new** SimpleDate("05/04/2007"),

TypedCashflow.CREDIT, 300.00),

**new** AllocatedCashflow(acc2,**new** SimpleDate("11/05/2007"),

TypedCashflow.CREDIT, 700.00),

**new** AllocatedCashflow(acc1,**new** SimpleDate("07/05/2007"),

TypedCashflow.DEBIT, 900.00),

**new** AllocatedCashflow(acc2,**new** SimpleDate("02/05/2007"),

TypedCashflow.DEBIT, 100.00)

};

**new** RuleRunner().runRules( **new** String[] { "Example6.drl" },

cashflows );

}

}

现在，让我们看一下Example6.drl中的规则， 以了解我们如何按日期顺序应用每个现金流并计算并打印余额。

*例子303. Banking教程：Example6.drl中的规则*

rule "Rule 06 - Credit"

**when**

$cashflow : AllocatedCashflow( $account : account,

$date : date,

$amount : amount,

type == TypedCashflow.CREDIT )

**not** AllocatedCashflow( account == $account, date < $date)

**then**

System.out.println("Credit: " + $date + " :: " + $amount);

$account.setBalance($account.getBalance()+$amount);

System.out.println("Account: " + $account.getAccountNo() +

" - new balance: " + $account.getBalance());

retract($cashflow);

**end**

rule "Rule 06 - Debit"

**when**

$cashflow : AllocatedCashflow( $account : account,

$date : date,

$amount : amount,

type == TypedCashflow.DEBIT )

**not** AllocatedCashflow( account == $account, date < $date)

**then**

System.out.println("Debit: " + $date + " :: " + $amount);

$account.setBalance($account.getBalance() - $amount);

System.out.println("Account: " + $account.getAccountNo() +

" - new balance: " + $account.getBalance());

retract($cashflow);

**end**

虽然我们有单独的信用和借记规则，但我们在检查早期现金流量时未指定类型。这是为了使所有现金流按日期顺序应用，而不管现金流的类型如何。在这些条件下，我们确定要使用的账户，并在后果中使用现金流量进行更新。

*例子304. Banking教程：Example6.java的输出*

Loading file: Example6.drl

Inserting fact: AllocatedCashflow[account=Account[accountNo=1,balance=0.0],date=Mon Jan 01 00:00:00 GMT 2007,type=Credit,amount=300.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=1,balance=0.0],date=Mon Feb 05 00:00:00 GMT 2007,type=Credit,amount=100.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=2,balance=0.0],date=Sun Mar 11 00:00:00 GMT 2007,type=Credit,amount=500.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=1,balance=0.0],date=Wed Feb 07 00:00:00 GMT 2007,type=Debit,amount=800.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=2,balance=0.0],date=Fri Mar 02 00:00:00 GMT 2007,type=Debit,amount=400.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=1,balance=0.0],date=Sun Apr 01 00:00:00 BST 2007,type=Credit,amount=200.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=1,balance=0.0],date=Thu Apr 05 00:00:00 BST 2007,type=Credit,amount=300.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=2,balance=0.0],date=Fri May 11 00:00:00 BST 2007,type=Credit,amount=700.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=1,balance=0.0],date=Mon May 07 00:00:00 BST 2007,type=Debit,amount=900.0]

Inserting fact: AllocatedCashflow[account=Account[accountNo=2,balance=0.0],date=Wed May 02 00:00:00 BST 2007,type=Debit,amount=100.0]

Debit: Fri Mar 02 00:00:00 GMT 2007 :: 400.0

Account: 2 - **new** balance: -400.0

Credit: Sun Mar 11 00:00:00 GMT 2007 :: 500.0

Account: 2 - **new** balance: 100.0

Debit: Wed May 02 00:00:00 BST 2007 :: 100.0

Account: 2 - **new** balance: 0.0

Credit: Fri May 11 00:00:00 BST 2007 :: 700.0

Account: 2 - **new** balance: 700.0

Credit: Mon Jan 01 00:00:00 GMT 2007 :: 300.0

Account: 1 - **new** balance: 300.0

Credit: Mon Feb 05 00:00:00 GMT 2007 :: 100.0

Account: 1 - **new** balance: 400.0

Debit: Wed Feb 07 00:00:00 GMT 2007 :: 800.0

Account: 1 - **new** balance: -400.0

Credit: Sun Apr 01 00:00:00 BST 2007 :: 200.0

Account: 1 - **new** balance: -200.0

Credit: Thu Apr 05 00:00:00 BST 2007 :: 300.0

Account: 1 - **new** balance: 100.0

Debit: Mon May 07 00:00:00 BST 2007 :: 900.0

Account: 1 - **new** balance: -800.0

### [25.6。定价规则决策表示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pricing_rule_decision_table_example)

定价规则决策表演示了在电子表格中使用Excel的XLS格式的决策表来计算保险单的零售成本。提供规则集的目的是计算申请特定政策的汽车司机的基本价格和折扣。驾驶员的年龄，历史和政策类型都会影响基本保费，另外一部分规则会以折扣百分比来处理。

Name: Example Policy Pricing

Main class: org.drools.examples.decisiontable.PricingRuleDTExample

Module: drools-examples

Type: Java application

Rules file: ExamplePolicyPricing.xls

Objective: demonstrate spreadsheet-based decision tables.

#### [25.6.1。执行示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_executing_the_example)

打开PricingRuleDTExample.java文件 并将其作为Java应用程序执行。它应该在控制台窗口中生成以下输出：

Cheapest possible

BASE PRICE IS: 120

DISCOUNT IS: 20

执行示例的代码遵循通常的模式。加载规则，插入事实并创建无状态会话。有什么不同是如何添加规则。

DecisionTableConfiguration dtableconfiguration =

KnowledgeBuilderFactory.newDecisionTableConfiguration();

dtableconfiguration.setInputType( DecisionTableInputType.XLS );

KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

Resource xlsRes = ResourceFactory.newClassPathResource( "ExamplePolicyPricing.xls",

getClass() );

kbuilder.add( xlsRes,

ResourceType.DTABLE,

dtableconfiguration );

请注意DecisionTableConfiguration对象的使用。它的输入类型设置为DecisionTableInputType.XLS。如果您使用BRMS，所有这些当然都是为您照顾的。

还有在这个例子中使用了两个事实类型，Driver和Policy。两者都使用它们的默认值。这Driver是30岁，没有先前的要求，目前有一个风险简介LOW。该Policy所申请的COMPREHENSIVE，而且还没有被批准。

#### [25.6.2。决策表](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_decision_table)

在这个决策表中，每一行都是一个规则，每一列都是一个条件或一个行为。

*图397.决策表配置*

参考上面的电子表格显示，我们有RuleSet声明，它提供了包名。还有其他可选项，例如Variables全局变量和Imports导入类。在这种情况下，规则的命名空间与我们使用的事实类相同，所以我们可以省略它。

再往下看，我们可以看到RuleTable声明。此后的名称（定价范围）用作所有生成的规则的前缀。在此之下，我们有“条件或行动”，表明该列的目的，即它是否构成条件的一部分或将生成的规则的结果。

你可以看到有一个驱动程序，他的数据跨越三个单元格，这意味着它下面的模板表达式适用于该事实。我们观察驾驶员的年龄范围（使用逗号分隔值$1并$2使用逗号分隔） locationRiskProfile，以及priorClaims在各个列中。在行动栏中，我们设定政策基准价格并记录消息。

*图398.基础价格计算*

在前面的电子表格部分中，有很多类别括号，由最左列中的注释表示。当我们知道司机的细节和他们的政策时，我们可以说（有点想法）他们应该匹配第18行，因为他们以前没有发生事故，并且已经有30年的历史了。这给了我们120的底价。

*图399.折扣计算*

以上部分包含我们可能授予我们的驱动程序的折扣条件。Age括号中的折扣结果，以前的索赔数量和政策类型。在我们的案例中，司机是30，没有事先声明，并正在申请一项COMPREHENSIVE政策，这意味着我们可以给予20％的折扣。请注意，这实际上是一个单独的表格，但在同一张工作表中，以便应用不同的模板。

重要的是要注意决策表生成规则。这意味着它们不仅仅是自上而下的逻辑，而更多的是捕获数据导致规则的手段。这是一个细微的差别，让一些人感到困惑。规则的评估不一定​​按给定顺序进行，因为规则引擎的所有常规机制仍然适用。

### [25.7。宠物商店示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pet_store_example)

Name: Pet Store

Main class: org.drools.examples.petstore.PetStoreExample

Module: drools-examples

Type: Java application

Rules file: PetStore.drl

Objective: Demonstrate **use** **of** Agenda **Groups**, **Global** **Variables** **and** integration **with** a GUI,

**including** callbacks **from** **within** the **rules**

Pet Store示例演示如何将规则与GUI集成在一起，在这种情况下是基于Swing的桌面应用程序。在规则文件中，它演示了如何使用议程组和自动对焦来控制允许在任何给定时间触发一组规则中的哪一个。它还说明了规则中Java和MVEL方言的混合，累积函数的使用以及从规则集中调用Java函数的方式。

所有Java代码都包含在一个文件PetStore.java中 ，定义了以下主要类（除了处理Swing事件的几个类）：

* Petstore包含main()我们将很快看到的方法。
* PetStoreUI负责创建和显示基于Swing的GUI。它包含几个较小的类，主要用于响应各种GUI事件，例如鼠标按钮点击。
* TableModel保存表格数据。把它看作是扩展Swing类的JavaBean AbstractTableModel。
* CheckoutCallback 允许GUI与规则交互。
* Ordershow 保留我们希望购买的物品。
* Purchase 存储订单和我们购买的产品的详细信息。
* Product 是一种JavaBean，它可以提供可供购买的产品及其价格。

大部分Java代码都是纯JavaBeans或基于Swing的。本节仅讨论几个与Swing相关的要点，但可以在Sun的Swing网站[http://java.sun.com/docs/books/tutorial/uiswing/上](http://java.sun.com/docs/books/tutorial/uiswing/)找到有关Swing组件的良好教程。

下面显示了Petstore.java 中与规则和事实相关的Java代码片段。

*例子305.在PetStore.main中创建PetStore KieContainer*

*// KieServices is the factory for all KIE services*

KieServices ks = KieServices.Factory.get();

*// From the kie services, a container is created from the classpath*

KieContainer kc = ks.getKieClasspathContainer();

*// Create the stock.*

Vector<Product> stock = **new** Vector<Product>();

stock.add( **new** Product( "Gold Fish", 5 ) );

stock.add( **new** Product( "Fish Tank", 25 ) );

stock.add( **new** Product( "Fish Food", 2 ) );

*// A callback is responsible for populating the*

*// Working Memory and for firing all rules.*

PetStoreUI ui = **new** PetStoreUI( stock,

**new** CheckoutCallback( kc ) );

ui.createAndShowGUI();

上面显示的代码KieContainer根据类路径创建并基于kmodule.xml 文件中的定义。与事实被断言并立即开除的其他示例不同，此示例将此步骤推迟到稍后。这样做的方式是通过第二个最后一行，其中PetStoreUI使用接受收集我们产品的Vector对象的构造函数创建对象stock，以及CheckoutCallback包含我们刚加载的规则库的类的实例。

启动规则的Java代码在该 CheckoutCallBack.checkout()方法中。当用户按下结账按钮时，这被触发（最终）。

*例306.激发规则 - 从CheckoutCallBack.checkout（）中提取*

**public** String **checkout**(JFrame frame, List<Product> items) {

Order order = **new** Order();

*// Iterate through list and add to cart*

**for** ( Product p: items ) {

order.addItem( **new** Purchase( order, p ) );

}

*// Add the JFrame to the ApplicationData to allow for user interaction*

*// From the container, a session is created based on*

*// its definition and configuration in the META-INF/kmodule.xml file*

KieSession ksession = kcontainer.newKieSession("PetStoreKS");

ksession.setGlobal( "frame", frame );

ksession.setGlobal( "textArea", **this**.output );

ksession.insert( **new** Product( "Gold Fish", 5 ) );

ksession.insert( **new** Product( "Fish Tank", 25 ) );

ksession.insert( **new** Product( "Fish Food", 2 ) );

ksession.insert( **new** Product( "Fish Food Sample", 0 ) );

ksession.insert( order );

ksession.fireAllRules();

*// Return the state of the cart*

**return** order.toString();

}

两个项目被传入此方法。一个是JFrameGUI底部围绕输出文本框架的Swing组件的句柄。第二个是订单项目清单; 这来自TableModel将GUI表格区域中的信息存储在GUI的右上部分。

for循环将来自GUI的订单项目列表转换为Order也包含在文件PetStore.java中的JavaBean 。请注意，可以直接在规则中引用Swing数据集，但使用简单的Java对象这样做更好地实现编码。这意味着如果我们想将示例转换为Web应用程序，我们就不会与Swing绑定。

请注意，本例中的所有状态都存储在Swing组件中，并且规则实际上是无状态的，这一点很重要。每次按下“签出”按钮时，此代码将Swing的内容复制TableModel到会话的工作存储器中。

在这个代码中，有9个电话给这个电话号码KieSession。其中的第一个创建了一个新KieSession的KieContainer。请记住，KieContainer当我们CheckoutCallBack在main()方法中创建类时我们通过了这个。接下来的两个调用传入两个对象，我们将其作为全局变量保存在规则中：Swing文本区域和用于写入消息的Swing框架。

更多插入信息将产品信息放入KieSession订单列表中。最后的电话是标准fireAllRules()。接下来，我们看看这个方法在规则文件中发生了什么。

*例307. Package，Imports，Globals和Dialect：从PetStore.drl中提取*

**package** org.drools.examples

**import** org.kie.api.runtime.KieRuntime

**import** org.drools.examples.petstore.PetStoreExample.Order

**import** org.drools.examples.petstore.PetStoreExample.Purchase

**import** org.drools.examples.petstore.PetStoreExample.Product

**import** java.util.ArrayList

**import** javax.swing.JOptionPane;

**import** javax.swing.JFrame

global JFrame frame

global javax.swing.JTextArea textArea

文件PetStore.drl的第一部分 包含标准包和导入语句，以使各种Java类可用于规则。新来的我们是两个全局frame和textArea。它们持有对调用该方法的Java代码先前传递的Swing组件JFrame和引用的引用。与规则中的变量不同，只要规则被触发，全局变量就会在会话的整个生命周期中保留其值。JTextAreasetGlobal()

PetStore.drl文件中的下一个摘录 包含两个函数，这些函数由我们很快会看到的规则引用。

*例308.规则中的Java函数：从PetStore.drl中提取*

function **void** **doCheckout**(JFrame frame, KieRuntime krt) {

Object[] options = {"Yes",

"No"};

**int** n = JOptionPane.showOptionDialog(frame,

"Would you like to checkout?",

"",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE,

**null**,

options,

options[0]);

**if** (n == 0) {

krt.getAgenda().getAgendaGroup( "checkout" ).setFocus();

}

}

function **boolean** **requireTank**(JFrame frame, KieRuntime krt, Order order, Product fishTank, **int** total) {

Object[] options = {"Yes",

"No"};

**int** n = JOptionPane.showOptionDialog(frame,

"Would you like to buy a tank for your " + total + " fish?",

"Purchase Suggestion",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE,

**null**,

options,

options[0]);

System.out.print( "SUGGESTION: Would you like to buy a tank for your "

+ total + " fish? - " );

**if** (n == 0) {

Purchase purchase = **new** Purchase( order, fishTank );

krt.insert( purchase );

order.addItem( purchase );

System.out.println( "Yes" );

} **else** {

System.out.println( "No" );

}

**return** **true**;

}

在规则文件中具有这些功能只是使Pet Store示例更加紧​​凑。在现实生活中，您可能在自己的文件中，在相同的规则包中，或者在标准Java类中作为静态方法来使用这些函数，并使用它们导入它们import function my.package.Foo.hello。

这两个功能的目的是：

* doCheckout()会显示一个询问用户是否希望结账的对话框。如果他们这样做，则重点放在checkout议程小组上，允许该小组中的规则（可能）触发。
* requireTank()会显示一个对话框询问用户是否希望购买坦克。如果是这样，一个新的鱼缸Product被添加到工作记忆中的订单列表中。

我们将看到稍后调用这些函数的规则。下一组示例来自Pet Store规则本身。第一个提取是发生第一个提取的部分，部分原因是它的auto-focus属性设置为true。

*例309.将项目放入工作内存：从PetStore.drl中提取*

// **Insert** **each** item **in** the shopping cart **into** the Working **Memory**

// **Insert** **each** item **in** the shopping cart **into** the Working **Memory**

rule "Explode Cart"

agenda-**group** "init"

**auto**-focus true

salience 10

dialect "java"

**when**

$**order** : **Order**( grossTotal == -1 )

$item : Purchase() **from** $**order**.items

**then**

**insert**( $item );

kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "**show** items" ).setFocus();

kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "**evaluate**" ).setFocus();

end

此规则与尚未grossTotal计算的所有订单匹配。它按照该顺序为每个采购项目循环。“爆炸购物车”规则的某些部分应该很熟悉：规则名称，显着性（建议规则被触发的顺序）以及设置的方言"java"。有三个新功能：

* 议程组`"init"定义议程组的名称。在这种情况下，组中只有一条规则。但是，Java代码和规则后果都不会将焦点集中到这个组，因此它依赖于下一个属性来触发它。
* auto-focus`true ensures that this rule, while being the only rule in the agenda group, gets a chance to fire when `fireAllRules() 从Java代码调用。
* kcontext…​.setFocus()依次将重点放在"show items"和"evaluate"议程小组上，允许他们的规则开火。在实践中，我们遍历订单上的所有项目，将它们插入到内存中，然后在每次插入后触发其他规则。

接下来的两个列表显示中的规则"show items"和evaluate议程组。我们按照它们被调用的顺序来看它们。

*示例310.在GUI中显示项目 - 从PetStore.drl中提取*

rule "Show Items"

agenda-group "show items"

dialect "mvel"

**when**

$order : Order( )

$p : Purchase( order == $order )

**then**

textArea.append( $p.product + "\n");

**end**

该"show items"议程组只有一个规则，被称为“显示项目”（注意的情况下的差异）。对于工作内存（或会话）中当前订单上的每笔购买，它将详细信息记录到GUI底部的文本区域。该textArea用来做这个变量就是我们在前面的全局变量之一。

该evaluate议程集团还从获得焦点"Explode Cart"前面列出的规则。这个日程基团具有两个规则，"Free Fish Food Sample"和"Suggest Tank"，如下所示。

*例311.评估议程组：从PetStore.drl中提取*

*// Free Fish Food sample when we buy a Gold Fish if we haven't already bought*

*// Fish Food and don't already have a Fish Food Sample*

rule "Free Fish Food Sample"

agenda-group "evaluate"

dialect "mvel"

when

$order : Order()

not ( $p : Product( name == "Fish Food") &amp;&amp; Purchase( product == $p ) )

not ( $p : Product( name == "Fish Food Sample") &amp;&amp; Purchase( product == $p ) )

exists ( $p : Product( name == "Gold Fish") &amp;&amp; Purchase( product == $p ) )

$fishFoodSample : Product( name == "Fish Food Sample" );

then

System.out.println( "Adding free Fish Food Sample to cart" );

purchase = **new** Purchase($order, $fishFoodSample);

insert( purchase );

$order.addItem( purchase );

end

*// Suggest a tank if we have bought more than 5 gold fish and don't already have one*

rule "Suggest Tank"

agenda-group "evaluate"

dialect "java"

when

$order : Order()

not ( $p : Product( name == "Fish Tank") &amp;&amp; Purchase( product == $p ) )

ArrayList( $total : size &gt; 5 ) from collect( Purchase( product.name == "Gold Fish" ) )

$fishTank : Product( name == "Fish Tank" )

then

requireTank(frame, kcontext.getKieRuntime(), $order, $fishTank, $total);

end

该规则"Free Fish Food Sample"将只会触发

* 我们还没有任何鱼食，而且
* 我们不已经有一个免费的鱼类食物样本，并
* 我们的订单中确实有金鱼。

如果规则确实触发了，它将创建一个新产品（鱼食品样本），并将其添加到工作记忆中的订单中。

该规则"Suggest Tank"将只会触发

* 我们的订单中没有鱼缸，而且
* 我们的订单中确实有5种以上的黄金鱼产品。

如果规则确实触发了，它会调用requireTank()我们之前查看的函数（向用户显示一个对话框，并在确认后向订单/工作内存中添加一个Tank）。当调用requireTank（）函数时，规则将传递全局框架变量，以便函数具有Swing GUI的句柄。

我们看到的下一个规则是"do checkout"。

*示例312.执行签出 - 从PetStore.drl中提取（6）*

rule "do checkout"

dialect "java"

**when**

**then**

doCheckout(frame, kcontext.getKieRuntime());

**end**

该规则"do checkout"有**没有议程组集并没有自动对焦属性**。因此，它被视为默认（MAIN）议程组的一部分。这个小组在默认情况下，当所有明确有焦点的议程组规则中的所有规则都已经运行时，焦点就集中在焦点上。

规则没有LHS，因此RHS将始终调用该doCheckout()函数。当调用该doCheckout()函数时，规则将传递全局frame变量给该函数一个Swing GUI的句柄。正如我们前面看到的，该doCheckout()功能向用户显示一个确认对话框。如果确认，该功能将重点放在结账议程组上，允许触发下一批规则。

*例子313.结账规则：从PetStore.drl中提取*

rule "Gross Total"

agenda-group "checkout"

dialect "mvel"

**when**

$order : Order( grossTotal == -1)

Number( total : doubleValue )

from accumulate( Purchase( $price : product.price ), sum( $price ) )

**then**

modify( $order ) { grossTotal = total };

textArea.append( "\ngross total=" + total + "\n" );

**end**

rule "Apply 5% Discount"

agenda-group "checkout"

dialect "mvel"

**when**

$order : Order( grossTotal &gt;= 10 &amp;&amp; &lt; 20 )

**then**

$order.discountedTotal = $order.grossTotal \* 0.95;

textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );

**end**

rule "Apply 10% Discount"

agenda-group "checkout"

dialect "mvel"

**when**

$order : Order( grossTotal &gt;= 20 )

**then**

$order.discountedTotal = $order.grossTotal \* 0.90;

textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );

**end**

结帐议程组中有三条规则：

* 如果我们还没有计算总Gross Total价格，则将产品价格累计为总价格，然后将此总额存入会话中，并通过Swing 再次JTextArea使用textArea全局变量进行显示。
* 如果我们的总额在10到20之间，则"Apply 5% Discount"计算折扣总额并将其添加到会话并显示在文本区域中。
* 如果我们的总额不低于20，则"Apply 10% Discount"计算折扣总额并将其添加到会话并显示在文本区域中。

现在我们已经完成了代码中发生的事情，让我们看看实际运行代码时会发生什么。PetStore.java文件 包含一个main()方法，以便它可以作为标准Java应用程序运行，无论是从命令行还是通过IDE。这假定你有你的类路径设置正确。（有关更多信息，请参阅示例部分的开头部分。）

我们看到的第一个屏幕是Pet Store Demo。它包含可用产品列表（左上角），所选产品的空白列表（右上角），结帐和重置按钮（中间）以及空白系统消息区域（底部）。

*图400.启动后的PetStore Demo*

为了达到这一点，发生了以下情况：

1. 该main()方法已经运行并加载了规则库，但尚未开启规则。到目前为止，这是与运行的规则相关的唯一代码。
2. 一个新的PetStoreUI对象已经创建并给予了规则库的一个句柄，供以后使用。
3. 各种Swing组件都在做它们的东西，并且显示上面的屏幕并等待用户输入。

点击列表中的各种产品可能会给您一个类似于下面的屏幕。

*图401.选择产品的PetStore演示*

请注意，这里没有任何规则代码被解雇。这只是Swing代码，侦听鼠标点击事件，并将一些选定的产品添加到TableModel对象以显示在右上角部分。（另外，请注意，这是模型视图控制器设计模式的经典用法）。

只有当我们按下“Checkout”按钮时，我们才会执行业务规则，这与我们前面通过代码的顺序大致相同。

1. 方法CheckOutCallBack.checkout()被调用（最终）由Swing类等待点击“签出”按钮。这将插入来自TableModel对象（GUI右上角）的数据，并将其插入会话的工作存储器中。然后它触发规则。
2. 该"Explode Cart"规则是第一个触发的，因为它已经auto-focus设置为true。它通过购物车中的所有产品循环，确保产品在工作记忆中，然后给予"Show Items"和Evaluation议程小组一个机会开火。这些团队的规则将购物车的内容添加到文本区域（在窗口的底部），决定是否给我们免费的鱼类食品，并询问我们是否要购买鱼缸。如下图所示。

*图402.我们想购买鱼缸吗？*

1. 该待办事项结帐规则是火，因为它（一）没有其他议程集团目前专注及（b）它是默认的（主）议程集团的一份子。它总是调用doCheckout（）函数，该函数显示“您想要结账吗？” 对话框。
2. 该doCheckout()功能将重点放在checkout议程组上，让该组中的规则成为可选项。
3. checkout议程组中的规则显示购物车的内容并应用适当的折扣。
4. 然后，Swing等待用户输入要么检出更多产品（并导致规则再次触发），要么关闭GUI - 请参见下图。

*图403.在所有规则开启后的Petstore Demo。*

我们可以添加更多的System.out调用来演示这种事件流。输出，正如它当前出现在控制台窗口中，在下面的列表中给出。

*示例314.运行PetStore GUI的Console（System.out）*

Adding free Fish Food Sample to cart

SUGGESTION: Would you like to buy a tank **for** your 6 fish? - Yes

### [25.8。诚实的政治家例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_honest_politician_example)

Name: Honest Politician

Main class: org.drools.examples.honestpolitician.HonestPoliticianExample

Module: drools-examples

Type: Java application

Rules file: HonestPoliticianExample.drl

Objective: Illustrate the concept of "truth maintenance" based on the logical insertion of facts

Honest Politician示例通过逻辑断言来演示真理维护。基本前提是一个对象只能在声明为真时才存在。规则的结果可以在逻辑上用该insertLogical()方法插入对象。这意味着只要逻辑插入的规则保持为真，该对象将只保留在工作存储器中。当规则不再为真时，对象会自动缩回。

在这个例子中有一个类Politician，名字和布尔值是诚实的。插入四位诚实国家的政治家设置为真。

*例315.班级政治家*

**public** **class** **Politician** {

**private** String name;

**private** **boolean** honest;

...

}

*例316.诚实的政治家：执行*

Politician blair = **new** Politician("blair", **true**);

Politician bush = **new** Politician("bush", **true**);

Politician chirac = **new** Politician("chirac", **true**);

Politician schroder = **new** Politician("schroder", **true**);

ksession.insert( blair );

ksession.insert( bush );

ksession.insert( chirac );

ksession.insert( schroder );

ksession.fireAllRules();

控制台窗口输出显示，尽管至少有一位诚实的政客，民主仍然存在。然而，由于每个政治家又被一个邪恶的公司腐化，所以所有的政客都变得不诚实，民主就死了。

*例317.诚实的政治家：控制台输出*

Hurrah!!! Democracy Lives

I'm an evil corporation and I have corrupted schroder

I'm an evil corporation **and** I have corrupted chirac

I'm an evil corporation and I have corrupted bush

I'm an evil corporation **and** I have corrupted blair

We are all Doomed!!! Democracy **is** Dead

只要在工作记忆中至少有一位诚实的政客，就会在Hope逻辑上断言一个新对象。这个对象只会存在，至少有一个诚实的政治家。一旦所有政客都不诚实，Hope对象将自动收回。为了确保它在任何其他规则之前触发，这个规则被赋予了10的显着性，因为在这个阶段，“希望是死的”规则实际上是真实的。

*例318.诚实的政治家：规则“我们有一个诚实的政治家”*

rule "We have an honest Politician"

salience 10

**when**

exists( Politician( honest == true ) )

**then**

insertLogical( **new** Hope() );

end

只要有一个Hope对象存在，“Hope Lives”规则就会匹配并触发。它突出了10个，因此它优先于“腐败诚实”。

*例319.诚实的政治家：规则“希望生活”*

rule "Hope Lives"

salience 10

**when**

**exists**( Hope() )

then

System.**out**.**println**("Hurrah!!! Democracy Lives");

end

现在有了希望，我们在开始的时候有四位诚实的政治家，我们对这个规则有四种激励，都是冲突的。他们会依次开火，腐蚀每个政客，使他们不再诚实。当所有四位政治家遭到腐败时，我们都没有财产的政治家honest == true。因此，“我们有一个诚实的政治家”的规则不再是真实的，它逻辑插入的对象（由于最后一次执行new Hope()）会自动收回。

*例320.诚实的政治家：规则“腐化诚实”*

rule "Corrupt the Honest"

**when**

politician : Politician( honest == **true** )

exists( Hope() )

**then**

System.out.println( "I'm an evil corporation and I have corrupted " + politician.getName() );

modify ( politician ) { honest = **false** };

**end**

随着Hope对象被自动缩回，通过真相维护系统，not应用于的条件元素Hope不再是真实的，以便下面的规则将匹配并触发。

*例321.诚实的政治家：规则“希望已死”*

rule "Hope is Dead"

**when**

**not**( Hope() )

**then**

System.out.println( "We are all Doomed!!! Democracy is Dead" );

**end**

让我们来看看这个应用程序的审计线索：

*图404.诚实的政治家示例审计视图*

当我们插入第一位政治家时，我们有两次激活。规则“我们有一个诚实的政治家”对于第一位插入的政治家只激活一次，因为它使用了一个exists条件元素，该元素匹配任何数字一次。规则“希望是死的”在这个阶段也被激活，因为我们还没有插入Hope对象。规则“我们有一个诚实的政治家”首先发射，因为它比插入Hope物体的“希望是死的”更显着。（该操作以绿色突出显示。）插入Hope对象激活“希望生活”并且去激活“希望是死的”; 它还为每位插入的诚实政客激活“腐败诚实”。规则“希望生活”执行，印刷“万岁!!!民主生活”。然后，对于每个政治家来说，统治“腐化诚实”火，印刷“我是一个邪恶的公司，我已经腐化了X”，其中X是政治家的名字，并将政治家的诚实价值修改为虚假。当最后一位诚实的政治家受到损坏时，Hope通过真相维护系统自动撤回，如蓝色突出显示的区域所示。绿色突出显示区域显示当前所选蓝色突出显示区域的原点。一旦Hope事实收回，“希望死了”激活并引发印刷“ 我们都是注定的！民主是死的“。

### [25.9。数独例子](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_example)

Name: Sudoku

Main class: org.drools.examples.sudoku.SudokuExample

Type: Java application

Rules file: sudoku.drl, validate.drl

Objective: Demonstrates the solving of logic problems, and complex pattern matching.

此示例演示了如何使用Drools在大量潜在解决方案空间中找到基于许多约束的解决方案。我们使用Sudoku流行的拼图。这个例子还显示了如何将Drools集成到图形界面中，以及如何使用回调函数与正在运行的Drools规则引擎进行交互，以根据运行时工作内存中的更改更新图形界面。

#### [25.9.1。数独概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_overview)

数独是一种基于逻辑的数字拼图游戏。目标是填充9x9网格，以便每个列，每行，以及九个3x3区域中的每一个都包含从1到9的数字，并且只有一次。

谜题制定者提供了一个部分完成的网格，谜题求解器的任务是完成具有这些约束的网格。

解决这个问题的一般策略是确保当你插入一个新号码时，它应该在其特定的3x3区域，行和列中是唯一的。

有关更详细的说明，请参阅[维基百科](http://en.wikipedia.org/wiki/Sudoku)。

#### [25.9.2。运行示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_running_the_example)

如上所述下载并安装drools-examples，然后执行java org.drools.examples.DroolsExamplesApp 并点击“SudokuExample”。

该窗口包含一个空的网格，但程序带有一些内部存储的网格，可以加载和解决。点击“文件”，然后点击“样本”并选择“简单”加载其中一个例子。请注意，在加载网格之前，所有按钮都被禁用。

*图405.初始屏幕*

加载“简单”示例根据拼图的初始状态填充网格。

*图406.加载“简单”*

点击“解决”按钮，基于Drools的引擎将填写剩余的值，并且按钮再次处于非活动状态。

*图407.“简单”解决*

或者，您可以单击“步骤”按钮查看规则集找到的下一位数字。控制台窗口将显示正在执行的规则的详细信息，以便以人类可读的形式解决步骤。下面介绍这些消息的一些示例。

由于[1,2]，在[0,1] 列消除中得到单个8 ：从[4,2]中删除9从

[2,8]中删除

[1,2] 行消除中的隐藏单个9 ：从[2 ，4]

由于[3,2]和[3,7]中的裸对

在[4,6]和[4,4]中排成一排，所以从[3,8 ]中删除6，

点击“Dump”按钮查看网格的状态，单元格显示已建立的值或剩余的可能性候选。

Col: 0 Col: 1 Col: 2 Col: 3 Col: 4 Col: 5 Col: 6 Col: 7 Col: 8

Row 0: 2 4 7 9 2 456 4567 9 23 56 9 --- 5 --- --- 1 --- 3 67 9 --- 8 --- 4 67

Row 1: 12 7 9 --- 8 --- 1 67 9 23 6 9 --- 4 --- 23 67 1 3 67 9 3 67 9 --- 5 ---

Row 2: 1 4 7 9 1 456 --- 3 --- 56 89 5 78 5678 --- 2 --- 4 67 9 1 4 67

Row 3: 1234 12345 1 45 12 5 8 --- 6 --- 2 5 78 5 78 45 7 --- 9 ---

Row 4: --- 6 --- --- 7 --- 5 --- 4 --- 2 5 8 --- 9 --- 5 8 --- 1 --- --- 3 ---

Row 5: --- 8 --- 12 45 1 45 9 12 5 --- 3 --- 2 5 7 567 4567 2 4 67

排6：1 3 7 1 3 6 --- 2 --- 3 56 8 5 8 3 56 8 --- 4 --- 3 567 9 1 678

排7：--- 5 --- 1 34 6 1 4 678 3 6 8 --- 9 --- 34 6 8 1 3 678 --- 2 --- 1 678

排8：34 --- 9 --- 4 6 8 --- 7 --- - - 1 --- 23456 8 3 56 8 3 56 6 8

现在，让我们加载故意无效的数独网格。点击“文件”，“样本”和“！DELIBERATELY BROKEN！”。请注意，此网格以某些问题开始，例如，值5在第一行中出现两次。

*图408.破碎的初始状态*

加载网格后，几条简单的规则会执行完整性检查。在这种情况下，以下消息打印在标准输出上：

单元格[0,8]：5在第0行

单元格中有一个副本[0,0]：5在第0行

单元格中有一个副本[6,0]：8 在第0列单元格中有一个副本

[4,0]：8在第0列中有重复

验证完成。

不过，单击“解决”按钮将解决规则应用于此无效网格。这不会完成; 一些细胞保持空白。

*图409.断开的“已解决”状态*

解决功能已通过使用实施标准解决技术的规则来实现。它们基于仍然是候选单元的值集合。例如，如果这样的集合包含单个值，那么这是单元格的值。稍微不太明显的是在九个单元组中的一个组中出现单值。检测这些情况的规则将某种特定单元格的解决方案值插入Setting类型的事实。这个事实导致从该单元所属的任何组中的所有其他单元中消除该值。最后，它被收回。

其他规则只会减少某些单元的允许值。规则“裸体对”，“行中的隐藏对”，“列中的隐藏对”和“正方形中的隐藏对”仅仅消除了可能性，但没有建立解决方案。“X翼成排”，“X翼成列”，“路口拆除排”和“路口拆除柱”完成更复杂的拆除工作。

#### [25.9.3。Java源代码和规则概述](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_java_source_and_rules_overview)

Java源代码可以在/ src / main / java / org / drools / examples / sudoku目录中找到，其中两个DRL文件定义了位于/ src / main / rules / org / drools / examples / sudoku中的规则目录。

该软件包org.drools.examples.sudoku.swing包含一组实现Sudoku拼图框架的类。请注意，该软件包对Drools库没有任何依赖关系。SudokuGridModel定义了一个接口，可以将Sudoku拼图存储为9x9的Cell对象网格。SudokuGridView是一个Swing组件，它可以可视化任何实现SudokuGridModel。SudokuGridEvent并SudokuGridListener用于传达模型和视图之间的状态更改：当单元格的值被解析或更改时触发事件。如果您熟悉其他Swing组件中的模型视图控制器模式，JTable那么此模式应该很熟悉。SudokuGridSamples为演示目的提供了一些部分填充的数独谜题。

包中org.drools.examples.sudoku.rules包含一个实用程序类和一个用于编译DRL文件的方法。

该包装org.drools.examples.sudoku包括一组执行的基本类Cell对象和它的各种聚合：所述CellFile亚型CellRow和CellCol以及CellSqr，所有这些都是亚型CellGroup。这是有趣的是，Cell和CellGroup是的子类SetOfNine，它提供了一个属性free与类型Set<Integer>。对于Cell它代表了个人候选集; 因为CellGroup该集合是其所有候选集合的单元的联合，或者简单地说，仍然需要分配的一组数字。

用81 Cell个27 CellGroup目的和通过所提供的联动Cell性 cellRow，cellCol以及cellSqr和所述CellGroup属性cells的列表，Cell对象，则它是可以编写检测允许值的分配，以细胞或值的消除的具体情况的规则来自一些候选集。

类的对象Setting用于触发伴随分配值的操作：将其从同胞单元和关联单元组的候选集中移除。此外，Setting在所有应检测新情况的规则中都会使用事实的存在; 这是为了避免对不一致的中间状态的反应。

Stepping当一个“步骤”没有定期终止时，一个类的对象被用在一个低优先级规则中来执行一个紧急停止。这表明该方案无法解决这个难题。

该类org.drools.examples.sudoku.SudokuExample实现了一个组合了所描述组件的Java应用程序。

#### [25.9.4。数独验证器规则（validate.drl）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_validator_rules_validate_drl)

验证规则检测单元组中的重复号码。它们结合在一个议程组中，使我们能够在加载拼图后明确地激活它们。

这三条规则“在单元格中复制......”非常相似。第一个模式定位一个具有分配值的单元格。第二种模式拉入该单元所属的三个单元组中的任何一个。最终模式会找到一个单元格（不是第一个单元格），其值与第一个单元格的值相同，并且分别位于同一行，列或方块中。

规则“终止小组”最后发射。它打印一条消息并呼叫暂停。

#### [25.9.5。数独解决规则（sudoku.drl）](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_sudoku_solving_rules_sudoku_drl)

该文件中有三种规则：一个组处理将一个数字分配给一个单元格，另一个组检测可行的分配，第三个组从候选集合中删除值。

规则“设置值”，“消除单元格中的值”和“收回设置”取决于Setting对象的存在。第一条规则处理对单元格的分配以及从单元格的三个组的“空闲”集合中删除值的操作。此外，它递减一个计数器，该计数器在零时将控制权返回给已调用的Java应用程序fireUntilHalt()。规则“从单元中消除一个值”的目的是减少与新分配的单元相关的所有单元的候选列表。最后，当全部消除完成后，规则“收回设置”收回触发Setting事实。

只有两条规则可以检测到可以向单元格分配数字的情况。规则“单一”会针对Cell包含单个号码的候选集合进行触发。当没有单个候选细胞时，但当存在含有候选细胞但该候选细胞不属于该细胞属于的三个组中的其中一个细胞的所有其他细胞时，规则“隐藏单个”会发生。两条规则都会创建并插入一个Setting事实。

来自最大规则组的规则单独或以两到三组的形式实施各种解决技巧，因为它们是在手动解决数独谜题时使用的。

规则“裸对”在一个组的两个单元中检测到大小为2的相同候选集合; 这两个值可以从该组的所有其他候选集中移除。

一个类似的想法激发了三条规则“隐藏在...中”; 在这里，规则在一组中的两个单元中查找两个数字的子集，在这个组的其他任何单元中都不出现任何值。那么这意味着所有其他的候选人都可以从隐藏的对中的两个单元中消除。

一对规则在行和列中处理“X-wings”。当两个不同行（或列）中的每一个中只有两个可能的单元格值并且这些候选者也位于相同的列（或行）中时，则列（或行）中的该值的所有其他候选者可以是淘汰。如果您遵循这些规则之一的模式序列，您将看到如何通过诸如“相同”或“唯一”等词语方便表达的条件导致具有适当约束或以“不”为前缀的模式。

规则对“交叉点删除...”基于一个方格内某个数字的限制出现，无论是单行还是单列。这意味着这个数字必须位于行或列的两个或三个单元格之一中; 因此它可以从该组的所有其他单元的候选集中移除。该模式建立限制发生，然后触发正方形之外和同一单元文件内的每个单元格。

这些规则对于很多人来说都是足够的，但对于所有的数独谜题来说，这些规则都是不够的 要解决非常困难的网格，规则集需要用更复杂的规则进行扩展。（最终，除了通过试验和错误之外，还有难以解决的难题。）

### [25.10。猜数字](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_number_guess)

Name: Number Guess

Main class: org.drools.examples.numberguess.NumberGuessExample

Module: droolsjbpm-integration-examples (Note: this is in a different download, the droolsjbpm-integration download.)

Type: Java application

Rules file: NumberGuess.drl

Objective: Demonstrate use of Rule Flow to organise Rules

“猜数字”示例显示了使用规则流，这是一种控制规则被触发顺序的方式。它使用广为人知的工作流程图来定义执行规则组的顺序。

*示例322.创建数字猜测规则库：NumberGuessExample.main（） - 第1部分*

**final** KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

kbuilder.add( ResourceFactory.newClassPathResource( "NumberGuess.drl",

ShoppingExample.class ),

ResourceType.DRL );

kbuilder.add( ResourceFactory.newClassPathResource( "NumberGuess.rf",

ShoppingExample.class ),

ResourceType.DRF );

**final** KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

kbase.addKnowledgePackages( kbuilder.getKnowledgePackages() );

包的创建和规则的加载（使用add()方法）与前面的例子相同。有一个额外的行添加规则流（NumberGuess.rf ），它提供了为同一知识库指定不同规则流的选项。否则，知识库的创建方式与以前相同。

*示例323.启动RuleFlow：NumberGuessExample.main（） - 第2部分*

**final** StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();

KnowledgeRuntimeLogger logger =

KnowledgeRuntimeLoggerFactory.newFileLogger(ksession, "log/numberguess");

ksession.insert( **new** GameRules( 100, 5 ) );

ksession.insert( **new** RandomNumber() );

ksession.insert( **new** Game() );

ksession.startProcess( "Number Guess" );

ksession.fireAllRules();

logger.close();

ksession.dispose();

一旦我们有了知识库，我们就可以用它来获得一个有状态的会话。在我们的会话中，我们插入了我们的事实，即标准Java对象。（为简单起见，在本示例中，这些类都包含在我们的NumberGuessExample.java 文件中，Class GameRules提供了允许的最大范围和猜测次数，Class RandomNumber自动生成一个0到100之间的数字，并使其可用于我们的规则，通过getValue()方法插入，Class Game跟踪我们之前做出的猜测，以及他们的数量。

请注意，在我们调用标准fireAllRules()方法之前，我们还通过startProcess()方法启动之前加载的过程。当我们谈论规则流文件和下面的图形规则流编辑器时，我们将学习何处获取我们传递的参数（“数字猜测”，即规则流的标识符）。

在我们完成对Java代码的讨论之前，我们注意到在一些真实应用程序中，我们将检查对象的最终状态。（在这里，我们可以检索猜测的数量，将它添加到高分表中。）在这个例子中，我们满足于确保通过调用dispose()方法来清除工作内存会话。

*图410. NumberGuess示例的RuleFlow*

如果你打开 Drools IDE中的NumberGuess.rf文件（假设你已经在Eclipse中正确安装了JBoss规则扩展），你应该看到上面的图表，类似于一个标准流程图。它的图标与JBoss jBPM工作流程产品中的图标相似（但不完全一样）。如果您希望编辑图表，IDE中图表左侧的可用组件菜单应该可用，它称为调色板。该图使用XStream以XML（一种几乎是人类可读的格式）保存。

如果尚未打开，请确保在IDE中显示属性视图。可以通过点击“窗口”，然后“显示视图”和“其他”来打开它，您可以在其中选择“属性”视图。如果您在选择规则流中的任何项目之前执行此操作（或单击规则流程中的空白区域），则应该显示以下一组属性。

*图411.数字猜测规则流的属性*

在我们通过示例规则流程进行时，请关注“属性视图”，因为它提供了有价值的信息。在这种情况下，当我们调用时，它为我们提供了我们在之前的代码片段中使用的规则流程的标识session.startProcess()。

在“猜数字”规则流程中，我们遇到了几种节点类型，其中许多节点类型由一个图标标识。

* 开始节点（绿色圆圈中的白色箭头）和结束节点（红色框）标记规则流程的开始和结束。
* 规则流组框（黄色，没有图标）表示我们将在稍后查看的规则（DRL）文件中定义的规则流组。例如，当流量到达规则流组“太高”时，只有那些标有“ruleflow-group”属性的规则才有`"Too High"可能触发。
* 动作节点（黄色，齿形图标）执行标准的Java方法调用。本示例中的大多数操作节点都会调用System.out.println()，指示程序对用户的进度。
* 拆分并加入节点（蓝色椭圆形，无图标），例如“猜猜正确吗？” 和“更多猜测加入”标志的地方，根据不同的条件，控制流量可以分裂，并分别重新加入
* 箭头指示各个节点之间的流量。

与规则相结合的各种节点使猜数游戏有效。例如，“猜测”规则流组只允许触发规则“获取用户猜测”，因为只有该规则具有匹配的属性ruleflow-group `"Guess"。

*例子324.规则只在规则流的特定点触发：NumberGuess.drl*

rule "Get user Guess"

ruleflow-group "Guess"

no-**loop**

**when**

$r : RandomNumber()

rules : GameRules( allowed : allowedGuesses )

game : Game( guessCount < allowed )

**not** ( Guess() )

**then**

System.out.println( "You have " + ( rules.allowedGuesses - game.guessCount )

+ " out of " + rules.allowedGuesses

+ " guesses left.\nPlease enter your guess from 0 to "

+ rules.maxRange );

br = **new** BufferedReader( **new** InputStreamReader( System.**in** ) );

i = br.readLine();

modify ( game ) { guessCount = game.guessCount + 1 }

insert( **new** Guess( i ) );

end

这个规则的其余部分是相当标准的。该when规则的LHS部分（之后）指出，对于RandomNumber插入到工作存储器中的每个对象，其中guessCount小于allowedGuesses来自GameRules对象且用户未猜到正确数量的对象将被激活。

RHS部分（或后面的部分then）向用户输出消息，然后等待用户输入System.in。获得此输入后（readLine()方法调用将阻塞，直到按下返回键）它会修改猜测计数并插入新猜测，使两者都可用于工作内存。

规则文件的其余部分非常标准：包将方言声明为MVEL，并导入各种Java类。总共有五个规则在这个文件中：

1. 获取用户猜测，我们上面检查的规则。
2. 记录最高猜测的规则。
3. 记录最低猜测的规则。
4. 如果不正确，检查猜测并将其从内存中收回的规则。
5. 通知用户所有猜测已用完的规则。

ruleflow-group正如上面所讨论的，标准规则和RuleFlow之间的一个整合点是通过规则的属性。一个规则（.drl）文件和规则流.rf文件之间的集成的第二点是，拆分节点（蓝色椭圆），可以在工作内存使用值（用规则更新）决定作用的，其流采取。要查看它是如何工作的，请点击“猜测正确节点”; 然后在属性视图中，通过单击一次点击“约束”属性行后出现的右侧按钮打开约束编辑器。您应该看到类似于下图的内容。

*图412.“猜测正确”节点的编辑约束*

点击“To node Too High”旁边的“编辑”按钮，你会看到一个如下所示的对话框。“文本编辑器”窗口中的值遵循LHS的标准规则格式，并可以引用工作内存中的对象。结果（RHS）是如果LHS表达式评估为真，则控制流遵循该节点（即，“至节点太高”）。

*图413.“猜测正确”节点的约束编辑器：值太高*

由于NumberGuess.java文件 包含一个main()方法，因此它可以作为标准的Java应用程序运行，无论是从命令行还是通过IDE。一个典型的游戏可能会导致下面的交互。粗体数字由用户输入。

*示例325.示例控制台输出的数字猜测示例击败了人！*

You have 5 out of 5 guesses left.

Please enter your guess from 0 to 100

50

Your guess was too high

You have 4 out of 5 guesses left.

Please enter your guess from 0 to 100

25

Your guess was too low

You have 3 out of 5 guesses left.

Please enter your guess from 0 to 100

37

Your guess was too low

You have 2 out of 5 guesses left.

Please enter your guess from 0 to 100

44

Your guess was too low

You have 1 out of 5 guesses left.

Please enter your guess from 0 to 100

47

Your guess was too low

You have no more guesses

The correct guess was 48

本示例中发生的情况总结如下：

1. 所述main()的方法NumberGuessExample.java加载规则库，创建一个状态会话和插入Game，GameRules和RandomNumber（含有目标数）对象到它。该方法还会设置我们要使用的流程流，并触发所有规则。控制权转移到规则流。
2. 文件号Guess.rf，规则流程从“开始”节点开始。
3. 控制权（通过“更多猜测”加入节点）传递给猜测节点。
4. 在猜测节点处，启用适当的规则流组（“获取用户猜测”）。在这种情况下，规则“猜测”（在NumberGuess.drl文件中）被触发。此规则向用户显示一条消息，获取响应并将其放入工作内存。流程转到下一个规则流程节点。
5. 在下一个节点“猜测正确”时，约束检查当前会话并决定采用哪条路径。

如果步骤4中的猜测过高或过低，则流程沿着具有动作节点的路径前进，其中具有正常Java代码的动作节点打印适当的消息，而规则流组引起最高猜测或最低猜测规则被触发。流从这些节点传递到步骤6。

如果步骤4中的猜测是正确的，我们沿着朝向规则流结束的路径前进。在我们到达那里之前，具有普通Java代码的动作节点会打印出一句“你猜对了”的语句。这里有一个连接节点（就在Rule Flow结束之前），这样我们的无更多猜测路径（步骤7）也可以终止Rule Flow。

1. 控制按照规则流经由联接节点传递，猜测错误的规则流组（触发规则以从工作内存收回猜测）到“更多猜测”决策节点上。
2. “更多猜测”决策节点（位于规则流的右侧）使用约束条件，再次查看规则已放入工作内存的值，以决定是否有更多猜测，如果有，则转到步骤3。如果不是，我们通过一个规则流程组进入规则流程的末尾，触发规则说明“你没有更多的猜测”。
3. 步骤3至步骤7的循环继续，直到数字被正确猜测，或者我们用尽猜测。

### [25.11。康威的人生游戏](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conway_s_game_of_life)

Name: Conway's Game Of Life

Main class: org.drools.examples.conway.ConwayAgendaGroupRun

org.drools.examples.conway.ConwayRuleFlowGroupRun

Module: droolsjbpm-integration-examples (Note: this is in a different download, the droolsjbpm-integration download.)

Type: Java application

Rules file: conway-ruleflow.drl conway-agendagroup.drl

Objective: Demonstrates 'accumulate', 'collect' and 'from'

康威的生命游戏，在[http://en.wikipedia.org/wiki/Conway's\_Game\_of\_Life](http://en.wikipedia.org/wiki/Conway%E2%80%99s_Game_of_Life)和[http://www.math.com/students/wonders/life/life.html中有](http://www.math.com/students/wonders/life/life.html)描述，是数学家John Conway在1970年代初期构思的一种着名的元胞自动机。虽然该系统被称为“康威的生命游戏”，但它确实不是一款游戏。康威的系统更像是一种生活形式的模拟。不要被吓倒。该系统非常简单而且非常有趣。数学和计算机科学的学生们对康威的系统已经超过30年了。这里介绍的应用程序是基于Swing的康威生命游戏实施。管理系统的规则是作为使用Drools的业务规则来实现的。本文将解释驱动模拟的规则并讨论实施的Drools部分。

我们将首先介绍如下所示的网格视图，该网格视图专为游戏的可视化而设计，展示生活模拟发生的“竞技场”。最初网格是空的，这意味着系统中没有活细胞。每个细胞活着或死亡，活细胞呈现绿色的球。预先选择的活细胞模式可以从“模式”下拉列表中选择。或者，可以双击单个单元格以在实时和死亡之间切换它们。了解每个单元格与其相邻单元格相关是非常重要的，这对于游戏规则来说非常重要。邻居不仅包括左侧，右侧，顶部和底部的单元格，还包括对角连接的单元格，以便每个单元格总共有8个邻居单元。例外是只有三个邻居的四个角落单元，

*图414.康威的生命之游：开始一场新的比赛*

那么管理这个游戏的基本规则是什么？它的目标是一代又一代地展示一个人口的发展。基于所有单元的同时评估，每一代都来自前一个。这是控制下一代看起来像什么的一套简单的规则：

* 如果一个活细胞中只有不到两个活的邻居，它就会死于孤独。
* 如果一个活细胞有超过3个活的邻居，它会死于过度拥挤。
* 如果一个死细胞恰好有3个活的邻居，它就会复活。

这就是它的全部。任何不满足任何这些标准的细胞都将保留下一代。考虑到这些简单的规则，回头再玩一下这个系统，并逐步完成几代人，并且注意到这些规则正在发挥作用。

下面的屏幕截图显示了一个具有多个活动单元格的示例代。不要担心匹配屏幕截图中呈现的确切模式。只需将一些单元格添加到网格中即可。一旦在网格中有活细胞组，或者选择预先设计的模式，请单击“下一代”按钮并注意发生了什么。一些活细胞被杀死（绿球消失），一些死细胞复活（绿球出现）。步入几代人，看看你是否注意到任何模式。如果你点击“开始”按钮，系统会自行演变，所以你不需要反复点击“下一代”按钮。稍微玩一下系统，然后回到这里以获取应用程序工作方式的更多细节。

*图415.康威的生命之游：一款正在运行的游戏*

现在让我们深入研究代码。由于这是一个高级示例，我们假设您现在知道了解Drools框架的方式，并且能够连接所呈现的突出显示，以便我们只关注高级别概述。该示例有两种执行方式，一种方式使用议程组来管理执行流程，另一种使用规则流程组来管理执行流程。这两个版本分别在ConwayAgendaGroupRun和中实现ConwayRuleFlowGroupRun。在这里，我们将讨论Rule Flow版本，因为它是大多数人使用的版本。

所有的Cell对象都被插入会话中，并且ruleflow-group“注册邻居”中的规则被规则流程允许执行。这组四条规则创建Neighbor了一些细胞与东北，北方，西北和西方邻居之间的关系。这种关系是双向的，它关注了其他四个方向。边界细胞不需要任何特殊处理 - 它们根本不会与没有任何特定细胞的邻近细胞配对。当所有激活都为这些规则触发时，所有单元都与它们的所有相邻单元有关。

*例326.康威的生命游戏：注册细胞邻近关系*

rule "register north east"

ruleflow-group "register neighbor"

when

$cell: Cell( $row : row, $col : col )

$northEast : Cell( row == ($row - 1), col == ( $col + 1 ) )

then

**insert**( **new** Neighbor( $cell, $northEast ) );

**insert**( **new** Neighbor( $northEast, $cell ) );

**end**

rule "register north"

ruleflow-**group** "register neighbor"

**when**

$cell: Cell( $**row** : **row**, $**col** : **col** )

$north : Cell( **row** == ($**row** - 1), **col** == $**col** )

**then**

**insert**( **new** Neighbor( $cell, $north ) );

**insert**( **new** Neighbor( $north, $cell ) );

**end**

rule "register north west"

ruleflow-**group** "register neighbor"

**when**

$cell: Cell( $**row** : **row**, $**col** : **col** )

$northWest : Cell( **row** == ($**row** - 1), **col** == ( $**col** - 1 ) )

**then**

**insert**( **new** Neighbor( $cell, $northWest ) );

**insert**( **new** Neighbor( $northWest, $cell ) );

**end**

rule "register west"

ruleflow-**group** "register neighbor"

**when**

$cell: Cell( $**row** : **row**, $**col** : **col** )

$west : Cell( **row** == $**row**, **col** == ( $**col** - 1 ) )

**then**

**insert**( **new** Neighbor( $cell, $west ) );

**insert**( **new** Neighbor( $west, $cell ) );

**end**

一旦插入了所有的单元格，一些Java代码将该模式应用到网格，将某些单元格设置为活动。然后，当用户点击“开始”或“下一代”时，它执行“生成”规则流程。此规则流程负责管理每个生成周期中的所有单元格更改。

*图416.康威的生命游戏：规则流程“生成”*

规则流程首先进入“评估”组，这意味着组中的任何活动规则都可以触发。这个小组的规则应用了在这个例子开始讨论的生命游戏规则，确定要杀死的细胞和被赋予生命的细胞。我们使用“阶段”属性通过特定的规则组来驱动Cell的推理; 通常该阶段与规则流程定义中的规则流程组相关联。请注意，Cell在这一点上它实际上并没有改变任何物体的状态; 这是因为它依次评估网格，并且必须完成全面评估，直到可以应用这些更改。为了实现这一点，它的小区集“相位”，其为任一Phase.KILL或Phase.BIRTH，Cell

*例327. Conway的生命游戏：评估具有状态变化的细胞*

rule "Kill The Lonely"

ruleflow-group "evaluate"

no-**loop**

**when**

// A live cell has fewer than 2 live neighbors

theCell: Cell( liveNeighbors < 2, cellState == CellState.LIVE,

phase == Phase.EVALUATE )

**then**

modify( theCell ){

setPhase( Phase.KILL );

}

end

rule "Kill The Overcrowded"

ruleflow-group "evaluate"

no-**loop**

**when**

// A live cell has more than 3 live neighbors

theCell: Cell( liveNeighbors > 3, cellState == CellState.LIVE,

phase == Phase.EVALUATE )

**then**

modify( theCell ){

setPhase( Phase.KILL );

}

end

rule "Give Birth"

ruleflow-group "evaluate"

no-**loop**

**when**

// A dead cell has 3 live neighbors

theCell: Cell( liveNeighbors == 3, cellState == CellState.DEAD,

phase == Phase.EVALUATE )

**then**

modify( theCell ){

theCell.setPhase( Phase.BIRTH );

}

end

一旦Cell对网格中的所有对象进行了评估，我们首先清除之前任何数据更改发生的任何计算激活。这是通过“重置计算”规则完成的，该规则清除“计算”组中的任何激活。然后，我们在规则流中输入一个分割，允许“杀”和“分娩”组中的任何激活触发。这些规则负责应用状态更改。

*例328.康威的生命游戏：应用状态变化*

rule "reset calculate"

ruleflow-group "reset calculate"

**when**

**then**

WorkingMemory wm = drools.getWorkingMemory();

wm.clearRuleFlowGroup( "calculate" );

end

rule "kill"

ruleflow-group "kill"

no-**loop**

**when**

theCell: Cell( phase == Phase.KILL )

**then**

modify( theCell ){

setCellState( CellState.DEAD ),

setPhase( Phase.DONE );

}

end

rule "birth"

ruleflow-group "birth"

no-**loop**

**when**

theCell: Cell( phase == Phase.BIRTH )

**then**

modify( theCell ){

setCellState( CellState.LIVE ),

setPhase( Phase.DONE );

}

end

在这个阶段，许多Cell对象已经被修改，状态改变为LIVE或DEAD。现在我们可以看到Neighbor定义单元关系的事实的力量。当一个细胞活着或死亡时，我们使用这个Neighbor关系遍历所有周围的细胞，增加或减少细胞liveNeighborEVALUATE阶段，以确保它在规则流程的评估阶段包含在推理中。请注意，我们不必自己做任何迭代; 只需将规则中的关系应用到规则引擎中，我们就可以用最少量的代码为我们完成所有艰苦的工作。一旦为所有单元确定并设置了实时计数，规则流程过程即告结束。如果用户最初点击了“开始”按钮，引擎将重新启动规则流程; 否则用户可能会要求另一代。

*例329.康威的生命游戏：评估具有状态变化的细胞*

rule "Calculate Live"

ruleflow-group "calculate"

lock-on-active

when

theCell: Cell( cellState == CellState.LIVE )

Neighbor( cell == theCell, $neighbor : neighbor )

**then**

modify( $neighbor ){

setLiveNeighbors( $neighbor.getLiveNeighbors() + 1 ),

setPhase( Phase.EVALUATE );

}

end

rule "Calculate Dead"

ruleflow-group "calculate"

lock-on-active

when

theCell: Cell( cellState == CellState.DEAD )

Neighbor( cell == theCell, $neighbor : neighbor )

**then**

modify( $neighbor ){

setLiveNeighbors( $neighbor.getLiveNeighbors() - 1 ),

setPhase( Phase.EVALUATE );

}

end

### [25.12。侵略者](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders)

Space Invaders游戏的简化版本。使用Z和K键向左和向右移动，并使用M键射击。这个例子建立在6个以上的项目上，每个项目的复杂性都会稍微增加一点。

Name: Example Invaders

Main class: org.drools.games.invaders.Invaders1Main

Main class: org.drools.games.invaders.Invaders2Main

Main class: org.drools.games.invaders.Invaders3Main

Main class: org.drools.games.invaders.Invaders4Main

Main class: org.drools.games.invaders.Invaders5Main

Main class: org.drools.games.invaders.Invaders6Main

*图417. Pong截图*

#### [25.12.1。Invaders1Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders1main)

Invaders1Main创建框架并附加KeyListener，将关键事件提供给引擎。它还设置了主要的游戏循环，可以在“Main.drl”中找到。整个例子中使用的典型惯例是每个文件有一个议程组，并且该文件中的所有规则都在同一个议程组中。

Run事实用于驱动Game循环的重复。最初只有一个组被评估，Keys。“keys.drl”文件由几个示例共享，并说明跨multipel项目的规则重用。

*例330.游戏循环*

rule "init" when

then

insert( new Run() );

setFocus( "Init" );

end

rule GameLoop when

r : Run()

then

setFocus( "Keys" );

end

rule Draw when

r : Run()

then

ui.show();

modify( r ) {} // force loop

end

#### [25.12.2。Invaders2Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders2main)

Invaders2Main将“绘制”阶段添加到游戏循环中并绘制SpaceShip

*例331.游戏循环*

rule GameLoop when

r : Run()

then

setFocus( "Draw" );

setFocus( "Keys" );

end

#### [25.12.3。Invaders3Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders3main)

入侵者3主要将移动控制添加到飞船，注意船舶移出屏幕边界。检测到KeyPressed，并在船的方向上设置dx的增量。该三角洲然后重复应用于船舶的x位置

*例332.移动船*

rule ShipDeltaMoveLeft agenda-group "Move" when

s : Ship()

KeyPressed( keyText == "Z" )

then

modify( s ) { dx = 0 - s.speed }

end

rule ShipDeltaStopLeft agenda-group "Move" when

s : Ship()

not KeyPressed( keyText == "Z" )

then

modify( s ) { dx = 0 }

end

rule ShipMove agenda-group "Move" when

s : Ship( dx != 0 )

Run()

then

modify( s ) { x = s.x + s.dx }

end

#### [25.12.4。Invaders4Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders4main)

Invaders4Main将限制控制添加到ShipMove规则中，所以它不会离开屏幕。注意使用“@watch（！x）”，这可以确保虽然规则将修改x属性，但它不会对x的更改作出反应，从而避免递归问题。

*示例333.移动具有边界的船*

rule ShipMove agenda-group "Move" when

s : Ship( dx != 0, x + dx > 0, x + dx + width < conf.windowWidth ) @watch( !x )

Run()

then

modify( s ) { x = s.x + s.dx }

end

#### [25.12.5。Invaders5Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders5main)

入侵者5主要更新“绘制”组以吸引5个入侵者。

#### [25.12.6。Invaders6Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders6main)

侵略者6主要增加更多的肉。按下“M”键可触发沿屏幕传播的导弹，同时检查导弹与入侵者之间的碰撞。

#### [25.12.7。Invaders4Main](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_invaders4main_2)

Invaders4Main将限制控制添加到ShipMove规则中，所以它不会离开屏幕。注意使用“@watch（！x）”，这可以确保虽然规则将修改x属性，但它不会对x的更改作出反应，从而避免递归问题。

*例334.消防导弹*

rule InsertBullet agenda-group "Bullet" when

KeyPressed( keyText == "M" )

s : Ship()

not Bullet()

then

b = new Bullet();

b.x = s.x + (s.width/2) - (b.width/2);

b.y = s.y - s.height - b.height;

b.width = conf.bulletWidth;

b.height = conf.bulletHeight;

b.dy = 0 - conf.bulletSpeed;

insert( b );

end

rule BulletMove agenda-group "Bullet" when

b : Bullet( y > 0 ) @watch( !y )

Run()

then

modify( b ) { y = b.y + b.dy }

end

rule Collision agenda-group "Bullet" when

b : Bullet( ) @watch( y )

i : Invader( x < b.x, x + width > b.x, y > b.y)

Run()

then

modify( i ) { alive = false }

end

### [25.13。冒险与流口水](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_adventures_with_drools)

基于Prolog中的冒险游戏，在Amzi网站[http://www.amzi.com/AdventureInProlog/上](http://www.amzi.com/AdventureInProlog/)，我们开始为Drools开发文本冒险游戏。它们是理想的，因为它们可以随着时间的推移简单起来并构建复杂性和大小，它们还展示了声明式关系编程的关键方面。

Name: Example Text Adventure

Main class: org.drools.games.adventure.TextAdventure

游戏允许你扮演英雄或怪物。如果你点击“新窗口”，你可以打开一个窗口作为英雄，另一个作为怪物打开，并同时播放它们。游戏允许任何角色在房间四处移动，拾起，放下或使用东西。通过使用出口房间的钥匙门可以被锁定和解锁，并且英雄可以使用怪物上的雨伞杀死怪物。

您可以在[http://downloads.jboss.org/drools/videos/text-adventures.swf上](http://downloads.jboss.org/drools/videos/text-adventures.swf)查看该示例的8分钟演示和介绍。请注意，视频现在比当前改进的示例要古老得多。

*图418.文本冒险屏幕截图*

#### [25.13.1。使用游戏。](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_using_the_game)

每个操作都遵循关联的Command类的构造函数参数。

*例335. MoveCommand*

@PropertyReactive

**public** **class** **MoveCommand** **extends** **Command** {

@Position(1)

**private** Character character;

@Position(2)

**private** Room room;

**public** **MoveCommand**(Character character, Room room) {

**this**.character = character;

**this**.room = room;

}

要发出移动操作，请选择“移动”按钮，然后选择退出房间。注意当您按下“移动”时，它将文本添加到底部的白色条。当退出房间被选中时，它也被添加到白色栏中。然后按发送，游戏引擎将执行该命令。它在内部使用反射来实例化命令并将其插入引擎。如果您选择不正确的参数，例如多次按退出，则反射将失败，您可以再次尝试。

*图419.移动操作*

事物列表显示任何你可以在房间里看到的东西，并不是所有的东西都可以被拾取。例如，你可以拿起钥匙和火炬，但不是怪物。当东西被拿起时，它从物品清单移动到清单清单。当某些东西被丢弃时，情况正好相反。

*图420.拾取动作*

关键是在办公室，上楼进入办公室。然后拿起钥匙。回到楼下，进入厨房。试着走进地下室，注意它被锁定。

选择“使用”动作，选择钥匙，然后选择地下室出口。这将打开门，你现在可以走过去。

杀死怪物从休息室拿起雨伞，然后选择“使用”，然后选择imbrella，最后选择怪物。

不要忘记打开一个“新窗口”来扮演怪物，尽管在英雄用钥匙打开它之前，您将无法离开地下室。怪物和英雄也可以将物品交给对方，在每个玩家库存之间移动物品。

#### [25.13.2。代码](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_code)

该模型是用Java类编写的。每个类使用@PropertyReactive和@Position。@PropertyReactive允许控制哪些字段模式作出反应，@Position将字段映射到允许模式的位置以及命名参数的参数位置。

*例子336.游戏世界数据例子*

@PropertyReactive

**public** **class** **Thing** {

@Position(0)

**private** **long** id;

@Position(1)

**private** String name;

**public** **Thing**(**long** id, String name) {

**this**.id = id;

**this**.name = name;

}

...

}

MVEL数据文件用于填充我们的世界，请参阅“data.mvel”。您可以编辑此文件以添加新房间，物品和角色，以及门锁。

*例子337.游戏世界数据例子*

rooms = [

"basement" : new Room("basement"),

"lounge" : new Room("lounge"),

"kitchen" : new Room("kitchen"),

"ground floor hallway" : new Room("ground floor hallway"),

"bedroom" : new Room("bedroom"),

"office" : new Room("office"),

"first floor hallway" : new Room("first floor hallway")

];

doors = [

"d1" : new Door( rooms["kitchen"], rooms["basement"] ),

"d2" : new Door( rooms["ground floor hallway"], rooms["lounge"]),

"d4" : new Door( rooms["ground floor hallway"], rooms["kitchen"]),

"d5" : new Door( rooms["ground floor hallway"], rooms[ "first floor hallway"] ),

"d8" : new Door( rooms["first floor hallway"], rooms[ "bedroom"] ),

"d9" : new Door( rooms["first floor hallway"], rooms[ "office"] )

];

locations = [

"monster" : new Location( characters["monster"], rooms["basement"] ),

"hero" : new Location( characters["hero"], rooms["ground floor hallway"] ),

"umbrella" : new Location( items["umbrella"], rooms["lounge"] ),

"key1" : new Location( items["key1"], rooms["office"] )

];

游戏创建命令，将其插入引擎。这些命令然后用于改变世界的状态，并且该状态反映回UI中。这些命令可以在“commands.drl”文件中找到。以下规则与MoveCommand相匹配，如果它有效，则会使移动发生。

*例子338.移动一个字符*

rule validMove agenda-group "commands" when

mc : MoveCommand( c : character, r : room )

l : Location( thing == c, ltarget : target ) @watch( !target )

?connect( d, r, ltarget; )

then

exit = new ExitEvent( c, (Room) l.target );

enter = new EnterEvent( c, r );

modify( l ) { target = r };

insert( exit );

insert( enter );

mc.session.channels["output"].send( "You have entered the " + l.target.name + "\n" );

end

在上面的规则中注意到“连接”模式，这实际上是一个查询。在MVEL数据文件门中只描述一种方式，我们可以使用查询来双向检查连接。查询可以在“queries.drl”文件中找到。

*例339.连接*

query connect( Door $d, Room $x, Room $y )

$d := Door($id, $name, $x, $y;)

or

$d :=Door($id, $name, $y, $x;)

end

UI的列表框由“UiView.drl”中的规则填充，这些规则反过来使用查询。当世界发生变化时，“物品”列表框将如何填充。

*示例340.更新UI*

rule updateThings salience 5 when

session : UserSession( $char : character )

things( $char, $things; )

then

session.channels["things"].send( $things );

end

query things(Character $char, List $things)

$char := Character()

Location( $char, $room; )

$things := List() from accumulate( Location($thing, $room; thing != $char),

collectList( $thing ) )

end

### [25.14。傍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_pong)

经典游戏Pong的转换。使用按键A，Z和K，M。每次弹跳后球应该变快。

Name: Example Pong

Main class: org.drools.games.pong.PongMain

*图421. Pong截图*

### [25.15。Wumpus世界](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_wumpus_world)

Name: Example Wumpus World

Main class: org.drools.games.wumpus.WumpusWorldMain

Wumpus World是“人工智能：现代方法”一书中的AI范例。当游戏第一次开始时，所有单元格变灰。当你走动时，他们变得可见。洞穴里有坑，黄金和黄金。当你在一个坑的旁边时，你会感到轻而易举，当你在旁边的时候，你会闻到一股恶臭，并在黄金旁边看到闪闪发光。传感器图标显示在移动按钮的上方。如果你走进一个坑或瓮，你就会死。有关Wumpus World更详细的概述，请参阅<http://www.cis.temple.edu/~giorgio/cis587/readings/wumpus.shtml>。显示游戏创建和运行情况的20分钟视频位于[http://www.youtube.com/watch?v=4CvjKqUOEzM。](http://www.youtube.com/watch?v=4CvjKqUOEzM)

*图422. Wumpus世界*

*图423.洞穴截图*

*图424.信号截图*

*图425.气味恶臭*

*图426.向上移动，Wumpus碰撞*

### [25.16。礼仪小姐和标杆管理](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_miss_manners_and_benchmarking)

Name: Miss Manners

Main class: org.drools.benchmark.manners.MannersBenchmark

Module: drools-examples

Type: Java application

Rules file: manners.drl

Objective: Advanced walkthrough on the Manners benchmark, covers Depth conflict resolution in depth.

#### [25.16.1。介绍](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_introduction_5)

曼纳斯小姐正在举办派对，作为一个好主人，她想要安排好座位。她最初的设计将每个人都安排在男女双人中间，但她担心人们有事要谈。什么是一个好的主持人？她决定记下每位客人的爱好，以便她可以安排客人不仅根据交替的性别进行配对，还要确保客人有一个有共同爱好的人，至少在一侧。

*图427.礼仪小姐的客人*

##### [25.16.1.1。标杆](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_benchmarking)

David Brant，Timothy Grose，Bernie Lofaso和Daniel P. Miranker在1991年的论文“数据库规模对规则系统性能的影响：五个案例研究”中建立了五个基准：

* **Manners**使用深度优先搜索方法来确定男女交替的座位安排，并确保邻居的共同爱好。
* **华尔兹**通过约束传播的线标记建立线图的三维解释。
* **WaltzDB**是华尔兹的一个更通用的版本，支持超过三行的连接并使用数据库。
* **ARP**是使用A \*搜索算法实现最低成本的机器人飞行器的路线规划器。
* **Weaver** VLSI路由器，用于使用黑板技术的通道和盒子。

礼仪已成为事实上的规则引擎基准。然而，它的行为现在是众所周知的，许多引擎都为此而优化，因此否定了它作为基准的用处，这也是华尔兹变得更加有利的原因。这五个基准也发布在德克萨斯大学<http://www.cs.utexas.edu/ftp/pub/ops5-benchmark-suite/>。

##### [25.16.1.2。礼仪小姐执行流程](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_miss_manners_execution_flow)

在分配了第一个座位布置之后，发生深度优先递归，其重复分配正确的座位布置，直到最后的座位被分配为止。礼仪使用一个Context实例来控制执行流程。活动图被分区以显示规则执行与当前Context状态的关系。

*图428.礼仪活动图*

##### [25.16.1.3。数据和结果](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_the_data_and_results)

在深入讨论规则之前，我们先来看看所声明的数据和由此产生的座位安排。数据是一组简单的五位客人，他们应该进行安排，以便两性替补和邻居有共同的爱好。

**数据**

数据在OPS5语法中给出，每个属性都有一个带有括号的名称和值对列表。每个人只有一个爱好。

（客人（名字n1）（性别m）（爱好h1））

（客人（名字n2）（性别f）（爱好h1））

（客人（名字n2）（性别f）（爱好h3））

（客人（名字n3） ）（性别m）（爱好h3））

（客人（姓名n4）（性别m）（爱好h1））

（客人（姓名n4）（性别f）（爱好h2））

（客人（姓名n4）（性别f） （爱好h3））

（客人（名字n5）（性别f）（爱好h2））

（客人（名字n5）（性别f）（爱好h1））

（last\_seat（座位5））

\* 结果\*

每次执行“分配座位”规则时都会打印结果列表的每一行。他们需要注意的一点是，每行都有一个比最后一个大的“pid”值。（这一点的意义将在讨论“指定座位”规则时予以解释。）“ls”，“rs”，“ln”和“rn”分别指左右座位和邻居的名字。实际的实现使用更长的属性名称（例如，leftGuestName但是，在这里我们将坚持原始实现的符号。

[座位id = 1，pid = 1，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

[座位id = 2，pid = 1，done = false，ls = 1，ln = n5，rs = 2，rn = n4]

[座位id = 3，pid = 2，done = false，ls = 2，ln = n4，rs = 3，rn = n3]

[座位id = 4，pid = 3，完成=假，ls =

4，ln = n2，rs = 5，rn = n1] = [ 假设= ls = 3，rn = n3，rs = 4，rn = n2 ]

#### [25.16.2。深入讨论](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_in_depth_discussion)

##### [25.16.2.1。作弊](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cheating)

礼仪的设计旨在进行跨产品加入和议程活动。许多人不理解这个例子，以获得更好的表现，使得他们的Manners基准点变得毫无意义。已知的秘籍或移植错误的方式是：

* 使用数组为客人的兴趣爱好，而不是断言每一个作为一个单一的事实大量减少交叉产品。
* 改变数据的顺序也可以减少匹配的数量，提高执行速度。
* 可以更改not条件元素，以便测试算法仅使用“第一最佳匹配”，即基本上将测试算法转换为反向链接。结果只能与其他反向链接规则引擎或方式端口相比较。
* 删除上下文以使规则引擎过早地匹配来宾和座位。正确的端口将防止事实匹配使用上下文启动。
* 可以防止规则引擎执行组合模式匹配。
* 如果在推理循环中没有事实回缩，则由于notCE，端口不正确。

##### [25.16.2.2。解决冲突](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_conflict_resolution_3)

Manners基准是为OPS5编写的，其中有两个冲突解决策略LEX和MEA。LEX是包括显着性，新近性和复杂性在内的多种策略链。策略的最近部分驱动深度优先（LIFO）点火顺序。CLIPS手册记录了新近度策略，如下所示：

*每个事实和实例都在内部标有“时间标记”，以表示其相对于系统中每个其他事实和实例的相对新近程度。与每个规则激活相关联的模式实体按降序排序以确定布局。在使用不太新近的模式实体进行激活之前，使用更近期的模式实体进行激活。要确定两次激活的放置顺序，请从最大时间标记开始逐个比较两次激活的排序时间标记。应该继续比较，直到一个激活的时间标记大于另一个激活的相应时间标记。具有更大时间标记的激活放在议程中的其他激活之前。*

*- CLIPS参考手册*

然而，Jess和CLIPS都使用深度策略，该策略更简单，更轻便，而Drools也采用了这种策略。CLIPS手册将深度策略记录为：

*新激活的规则放置在相同显着性的所有规则之上。例如，如果事实-a激活rule-1和rule-2并且fact-b激活了rule-3和rule-4，那么如果事实-a在fact-b，rule-3和rule-4之前被声明议程上面的规则1和规则2。但是，规则1相对于规则2和规则3相对于规则4的位置将是任意的。*

*- CLIPS参考手册*

如果没有在“make\_path”规则中使用突出显示，Depth策略的初始Drools实现将不适用于Manners。CLIPS支持团队曾这样说过：

*CLIPS的默认冲突解决策略Depth与OPS5使用的默认冲突解决策略不同。因此，如果您直接将OPS5程序翻译为CLIPS，但使用默认的深度冲突解决策略，则只有巧合才能获得正确的行为。在CLIPS中提供了LEX和MEA冲突解决策略，以允许您在CLIPS中快速转换并正确运行OPS5程序。*

*- CLIPS支持论坛*

对CLIPS代码的调查显示深度策略中存在未公开的功能。这个策略中使用了一个累积的时间标签; 它与新近战略中的事实比较并不是一个广泛的事实，它只是为每次激活添加所有时间标记的总和并进行比较。

##### [25.16.2.3。规则“assignFirstSeat”](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_assignfirstseat)

一旦上下文更改为START\_UP，将为所有声明的来宾创建激活。由于所有激活都是通过单个工作内存操作创建的，因此它们都具有相同的激活时间标记。最后断言的Guest对象将具有更高的实际时间标记，并且其激活会触发，因为它具有最高的累计事实时间标记。这条规则的执行顺序没有什么重要性，但对“分配座位”规则有很大影响。激活触发并声明第一个Seating排列和a Path，然后设置Context属性state以创建规则的激活findSeating。

rule assignFirstSeat

when

context : Context( state == Context.START\_UP )

guest : Guest()

count : Count()

then

String guestName = guest.getName();

Seating seating =

**new** Seating( count.getValue(), 1, true, 1, guestName, 1, guestName);

insert( seating );

Path path = **new** Path( count.getValue(), 1, guestName );

insert( path );

modify( count ) { setValue ( count.getValue() + 1 ) }

System.out.println( "assign first seat : " + seating + " : " + path );

modify( context ) {

setState( Context.ASSIGN\_SEATS )

}

end

##### [25.16.2.4。规则“findSeating”](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rule_findseating)

这条规则决定了每个Seating安排。该规则为针对所有客户的所有声明Seating安排创建了交叉产品解决方案，除了自身或任何已分配的选定解决方案。

rule findSeating

**when**

context : Context( **state** == Context.ASSIGN\_SEATS )

$s : Seating( pathDone == true )

$g1 : Guest( name == $s.rightGuestName )

$g2 : Guest( sex != $g1.sex, hobby == $g1.hobby )

count : Count()

**not** ( Path( id == $s.id, guestName == $g2.name) )

**not** ( Chosen( id == $s.id, guestName == $g2.name, hobby == $g1.hobby) )

then

**int** rightSeat = $s.getRightSeat();

**int** seatId = $s.getId();

**int** countValue = count.getValue();

Seating seating =

new Seating( countValue, seatId, false, rightSeat,

$s.getRightGuestName(), rightSeat + 1, $g2.getName() );

insert( seating );

Path path = new Path( countValue, rightSeat + 1, $g2.getName() );

insert( path );

Chosen chosen = new Chosen( seatId, $g2.getName(), $g1.getHobby() );

insert( chosen );

System.err.println( "find seating : " + seating + " : " + path +

" : " + chosen);

modify( count ) {setValue( countValue + 1 )}

modify( context ) {setState( Context.MAKE\_PATH )}

end

但是，从前面显示的印刷结果可以看出，选择Seating最高的pid交叉产品至关重要。如果我们对几乎所有的现有对象Seating和Guest对象都有激活，同一时间标记，这怎么可能？例如，findSeating生产激活的第三次迭代将如下所示。请记住，这是来自一个非常小的数据集，并且对于更大的数据集，将会有更多可能的激活Seating解决方案，每个解决方案有多个解决方案pid：

=> [ActivationCreated（35）：rule = findSeating

[fid：19：33]：[ Seating\_id = 3，pid = 2，done = true，ls = 2，ln = n4，rs = 3，rn = n3]

[ fid：4：4]：[Guest name = n3，sex = m，hobbies = h3]

[fid：3：3]：[Guest name = n2，sex = f，hobbies = h3]

=> [ActivationCreated（35） ：rule = findSeating

[fid：15：23]：[Seating id = 2，pid = 1，done = true，ls = 1，ln = n5，rs = 2，rn = n4]

[fid：5：5]： [Guest name = n4，sex = m，hobbies = h1]

[fid：2：2]：[Guest name = n2，sex = f，hobbies = h1]

=> [ActivationCreated（35）：rule = findSeating

[fid： 13:13]：[座位id = 1，pid = 0，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

[fid：9：9]：[guest name = n5，sex = f，hobbies = h1]

[fid：1：1]：[客人姓名= n1，性别= m，爱好= h1]

所有这些多余激活的创造可能看起来毫无意义，但必须记住，礼仪并非关于优秀的规则设计; 它有目的地被设计为一个糟糕的规则集来完全压力测试跨产品匹配过程和议程，这显然是。请注意，每个激活都具有相同的时间标记35，因为它们都是由Context对象中的更改激活的ASSIGN\_SEATS。通过OPS5和LEX，它可以正确激活Seating最后断言的激活。随着深度，累计的事实时间标签可确保激活最后一次Seating触发的火灾。

##### [25.16.2.5。规则“makePath”和“pathDone”](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rules_makepath_and_pathdone)

规则makePath必须始终在之前开火pathDone。一个Path对象被声明为每个Seating安排，直到最后声明 Seating。请注意，这些条件pathDone是其中的一部分，makePath因此我们如何确保makePath首先发生火灾？

rule makePath

**when**

Context( state == Context.MAKE\_PATH )

Seating( seatingId:id, seatingPid:pid, pathDone == false )

Path( id == seatingPid, pathGuestName:guestName, pathSeat:seat )

**not** Path( id == seatingId, guestName == pathGuestName )

**then**

insert( **new** Path( seatingId, pathSeat, pathGuestName ) );

end

rule pathDone

when

context : Context( state == Context.MAKE\_PATH )

seating : Seating( pathDone == false )

**then**

modify( seating ) {setPathDone( true )}

modify( context ) {setState( Context.CHECK\_DONE)}

end

*图429. Rete图*

这两条规则最终在冲突的议程上，并具有相同的激活时间标签。但是，对于“制作路径”，累积的实际时间标记更大，因此它优先。

##### [25.16.2.6。规则“继续”和“areWeDone”](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_rules_continue_and_arewedone)

规则areWeDone仅在最后一个席位被分配时激活，此时这两个规则将被激活。出于同样的原因，makePath总是胜过path Done，areWeDone将优先于规则continue。

rule areWeDone

**when**

context : Context( state == Context.CHECK\_DONE )

LastSeat( lastSeat: seat )

Seating( rightSeat == lastSeat )

**then**

modify( context ) {setState(Context.PRINT\_RESULTS )}

**end**

rule continue

when

context : Context( state == Context.CHECK\_DONE )

**then**

modify( context ) {setState( Context.ASSIGN\_SEATS )}

end

#### [25.16.3。输出摘要](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_output_summary)

分配第一个座位

=> [fid：13：13]：[座位id = 1，pid = 0，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

=> [fid：14： 14]：[Path id = 1，seat = 1，guest = n5]

==> [ActivationCreated（16）：rule = findSeating

[fid：13：13]：[Seating id = 1，pid = 0，done = true ，ls = 1，ln = n5，rs = 1，rn = n5]

[fid：9：9]：[客人姓名= n5，性别= f，hobbies = h1]

[fid：1：1]：[ = n1，sex = m，hobbies = h1]

==> [ActivationCreated（16）：rule = findSeating

[fid：13：13]：[Seating id = 1，pid = 0，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

[fid：9：9]：[客人姓名= n5，性别= f，hobbies = h1]

[fid：5：5]：[客人姓名= n4，性别= m ，hobbies = h1] \*

Assign Seating

=> [fid：15：17]：[Seating ID = 2，pid = 1，done = false，ls = 1，lg = n5，rs = 2，rn = n4]

=> [fid：16：18]：[路径id = 2，seat = 2，guest = n4]

=> [fid：17：19]：[chosen id = 1，name = n4，hobbies = h1]

=> [ActivationCreated（21）：rule = makePath

[fid：15：17]：[Seating ID = 2，pid = 1，done = false，ls = 1，ln = n5，rs = 2，rn = n4]

[fid： [14：14]：[Path id = 1，seat = 1，guest = n5] \*

==> [ActivationCreated（21）：rule = pathDone

[Seating id = 2，pid = 1，done = false，ls = ln = n5，rs = 2，rn = n4] \*

Make Path

=> [fid：18：22：[Path id = 2，seat = 1，guest = n5]]

Path Done

Continue Process

=> [ActivationCreated（25） ：rule = findSeating

[fid：15：23]：[Seating id = 2，pid = 1，done = true，ls = 1，ln = n5，rs = 2，rn = n4]

[fid：7：7]： [客人姓名= n4，性别= f，爱好= h3]

[fid：4：4]：[客人姓名= n3，性别= m，爱好= h3] \*

=> [ActivationCreated（25）：rule = findSeating

[fid：15：23]：[ Seating\_id = 2，pid = 1，done = true，ls = 1，ln = n5，rs = 2，rn = n4]

[ fid：5：5]：[guest name = n4，sex = m，hobbies = h1]

[fid：2：2]：[guest name = n2，sex = f，hobbies = h1]，[fid：12：20 ]：[Count value = 3]

=> [ActivationCreated（25）：rule = findSeating

[fid：13：13]：[Seating id = 1，pid = 0，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

[fid：9：9]：[guest name = n5，sex = f，hobbies = h1]

[fid：1：1]：[guest name = n1，sex = m，hobbies = h1 ]

分配座位

=> [fid：19：26]：[座位id = 3，pid = 2，done = false，ls = 2，lnn4，rs = 3，rn = n3]]

> [fid：20：27 ]：[Path id = 3，seat = 3，guest = n3]]

> [fid：21：28]：[Chosen id = 2，name = n3，hobbies = h3}]

=> [ActivationCreated（30）：规则= makePath

[fid：19：26]：[座位id = 3，pid = 2，done = false，ls = 2，ln = n4，rs = 3，rn = n3]

[fid：18：22] 2，seat = 1，guest = n5] \*

=> [ActivationCreated（30）：rule = makePath

[fid：19：26]：[Seating id = 3，pid = 2，done = false，ls = 2，ln = n = 3，n = 3，n = 3]

[fid：16：18]：[Path id = 2，seat = 2，guest = n4] \*

=> [ActivationCreated（30）：rule = done

[fid：19：26 ]：[

路径id = 3，pid = 2，done = false，ls = 2，ln = n4，rs = 3，rn = n3] \* Make Path

=> [fid：22：31]：[Path id = 3 ，

路径= 1，guest = n5] Make Path

=> [fid：23：32] [path id = 3，seat = 2，guest = n4]

Path Done

Continue Processing

=> [ActivationCreated（35）：rule = findSeating

[ fid：19：33]：[座位id = 3，pid = 2，done = true，ls = 2，ln = n4，rs = 3，rn = n3]

[fid：4：4]：[guest name = n3，sex = m，hobbies = h3]

[fid：3：3]：[guest name = n2，sex = f，hobbies = h3]，[fid：12： 29

= [ActivationCreated（35）：rule = findSeating

[fid：15：23]：[Seating id = 2，pid = 1，done = true，ls = 1，ln = n5，rs = 2， n4]

[fid：5：5]：[guest name = n4，sex = m，hobbies = h1]

[fid：2：2]：[guest name = n2，sex = f，hobbies = h1]

=> [ActivationCreated （35）：rule = findSeating

[fid：13：13]：[Seating id = 1，pid = 0，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

[fid：9： 9]：[Guest name = n5，sex = f，hobbies = h1]，[fid：1：1]：[guest name = n1，sex = m，hobbies = h1]

Assign Seating

=> [fid：24：36 ]：[座位id = 4，pid = 3，done = false，ls = 3，ln = n3，rs = 4，rn = n2]]

> [fid：25：37]：[路径id = 4，座位= 4，guest = n2]]

=> [fid：26：38]：[Chosen id = 3，name = n2，hobbies = h3]

==> [ActivationCreated（40）：rule = makePath

[fid：24：36]：[Seating id = pid = 3，done = false，ls = 3，ln = n3，rs = 4，rn = n2]

[fid：23：32]：[path id = 3，seat = 2，guest = n4] \*

==> [ActivationCreated（40）：rule = makePath

[fid：24：36]：[Seating id = 4，pid = 3，done = false，ls = 3，ln = n3，rs = 4，rn = n2]

[fid： 20:27]：[Path id = 3，seat = 3，guest = n3] \*

=> [ActivationCreated（40）：rule = makePath

[fid：24：36]：[Seating id = 4，pid = 3，done = [false]，ls = 3，ln = n3，rs = 4，rn = n2]

[fid：22：31]：[Path id = 3，seat = 1，guest = n5] \*

=> [ActivationCreated（40） rule = done

[fid：24：36]：[

Make id = 4，pid = 3，done = false，ls = 3，ln = n3，rs = 4，rn = n2] \* Make Path

=> fid：27： 41：[路径ID = 4，座位= 2，客人= n4]

Make Path

=> fid：28：42]：[Path id = 4，seat = 1，guest = n5]]

Make Path

=> fid：29：43]：[Path id = 4，seat = 3，guest = n3 ]

的路径完成

继续进行处理

=> [ActivationCreated（46）：规则= findSeating

[FID：15：23]：[座位ID = 2，PID = 1，进行=真，LS = 1，LN = N5，RS = 2 ，

[ rn = n4] [fid：5：5]：[guest name = n4，sex = m，hobbies = h1]，[fid：2：2]

[guest name = n2，sex = f，hobbies = h1]

= > [ActivationCreated（46）：rule = findSeating

[fid：24：44]：[ Seating\_id = 4，pid = 3，done = true，ls = 3，ln = n3，rs = 4，rn = n2]

[fid ：[Guest name = n2，sex = f，hobbies = h1]

[fid：1：1]：[Guest name = n1，sex = m，hobbies = h1] \*

=> [ActivationCreated（46） ：rule = findSeating

[fid：13：13]：[座位id = 1，pid = 0，done = true，ls = 1，ln = n5，rs = 1，rn = n5]

[fid：9：9]：[Guest name = n5，sex = f，hobbies = h1]

[fid：1：1]：[guest name = n1，sex = m，hobbies = h1]

Assign Seating

=> [fid：30：47] pid = 4，done = false，ls = 4，ln = n2，rs = 5，rn = n1]

=> [fid：31：48]：[path id = 5，seat = 5，guest = n1]

=> [fid：32：49]：[Chosen id = 4，name = n1，hobbies = h1]

### [25.17。逆向链接](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_backward_chaining)

反向链式规则体系是目标驱动的。这意味着系统从引擎试图满足的结论开始。如果它不能这样做，它会搜索子目标，即结果将完成当前目标的一部分。它继续这个过程，直到最初的结论得到满足或者没有更不满意的子目标。**Prolog**是一个后向链接引擎的例子。

*图430.反向链接图*

#### [25.17.1。后向链系统](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_backward_chaining_systems)

反向链接是最近添加到JBoss规则引擎的功能。这个过程通常被称为派生查询，与反应系统相比，它并不常见，因为JBoss规则主要是反应式前向链接。也就是说，它会响应您的数据更改。添加到引擎中的反向链接用于产品类推导。

#### [25.17.2。克隆传递闭包](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_cloning_transitive_closures)

*图431.推理图*

前面的图表演示了一个House传递项目的例子。通过实施以下规则可以创建类似的推理图表：

1.首先，创建一些Java规则来开发传递项目的推理。它插入每个位置。

2.接下来，创建**Location**类; 它有该项目和它所在的位置。

3.输入House例子的规则，如下所示：

ksession.insert( **new** Location("office", "house") );

ksession.insert( **new** Location("kitchen", "house") );

ksession.insert( **new** Location("knife", "kitchen") );

ksession.insert( **new** Location("cheese", "kitchen") );

ksession.insert( **new** Location("desk", "office") );

ksession.insert( **new** Location("chair", "office") );

ksession.insert( **new** Location("computer", "desk") );

ksession.insert( **new** Location("drawer", "desk") );

4.创建一个传递式设计，其中物品位于其指定位置，例如位于“办公室”中的“桌子”。

*图432.房屋的传递推理图*

#### [25.17.3。定义查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_defining_a_query)

1.创建一个查询来查看插入到规则引擎中的数据：

query isContainedIn( String x, String y )

Location( x, y; )

**or**

( Location( z, y; ) **and** isContainedIn( x, z; ) )

**end**

注意查询是如何递归的，并调用“isContainedIn”。

2.创建一个规则来打印出插入到系统中的每个字符串，以查看如何实施。该规则应该类似于以下格式：

rule "go" salience 10

**when**

$s : String( )

**then**

System.out.println( $s );

**end**

3.使用步骤2作为模型，创建调用步骤1查询“isContainedIn”的规则。

rule "go1"

**when**

**String**( **this** == "go1" )

**isContainedIn**("office", "house"; )

then

System.**out**.**println**( "office is in the house" );

end

当第一个字符串插入引擎时，“go1”规则将会触发。也就是说，它询问“办公室”项目是否在“房屋”位置。因此，当插入“go1”字符串时，步骤1查询由前面的规则引起。

4.创建“go1”，将其插入引擎，然后调用fireAllRules。

ksession.insert( "go1" );

ksession.fireAllRules();

---

go1

office **is** **in** the house

---线表示引擎输出与“go”规则和“go1”规则的触发分离。

#### [25.17.4。传递闭包示例](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_transitive_closure_example)

1.通过实施以下规则创建传递闭包：

rule "go2"

**when**

**String**( **this** == "go2" )

**isContainedIn**("drawer", "house"; )

then

System.**out**.**println**( "Drawer in the House" );

end

2.回想克隆传递闭包的话题，“房子”中没有“抽屉”的例子。“抽屉”位于“桌子”。

*图433.抽屉的传递推理图*

3.使用先前的查询来获得这个递归信息。

query isContainedIn( String x, String y )

Location( x, y; )

**or**

( Location( z, y; ) **and** isContainedIn( x, z; ) )

**end**

4.创建“go2”，将其插入引擎，然后调用fireAllRules。

ksession.insert( "go2" );

ksession.fireAllRules();

---

go2

Drawer **in** the House

当规则被解雇时，它正确地告诉你已经插入了“go2”，并且“抽屉”在“房子”中。

5.检查发动机如何确定这个结果

* 该查询必须递减几个级别才能确定这一点。
* 由于“drawer”不在“house”中，查询使用**（z，y;）**的值，而不是使用**Location（x，y;）**。
* 该**ž**目前未绑定的，这意味着它没有价值，并将返回的一切，是在争论。
* **y**现在必须“住”，所以**z**将返回“办公室”和“厨房”。
* 信息从“办公室”收集，并递归检查“抽屉”是否在“办公室”。为这些参数调用以下查询行：**isContainedIn（x，z;）**

“办公室”中没有“抽屉”的例子; 因此，它不匹配。随着z被解除绑定，它将返回“办公室”内的数据，并且它将收集**z ==台**。

isContainedIn(x==drawer, z==desk)

isContainedIn递归三次。在最后的递归中，一个实例触发“桌子”中的“抽屉”。

Location(x==drawer, y==desk)

这匹配在第一个位置并递归备份，所以我们知道“抽屉”位于“办公桌”，“办公桌”位于“办公室”，“办公室”位于“房间”中。因此，“抽屉”在“房子”中并且返回真实。

#### [25.17.5。反应性传递查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_reactive_transitive_queries)

1.通过实施以下规则创建一个无效传输查询：

rule "go3"

**when**

**String**( **this** == "go3" )

**isContainedIn**("key", "office"; )

then

System.**out**.**println**( "Key in the Office" );

end

即使答案不能得到满足，反应性传递查询也可以提出问题。稍后，如果满意，它会返回一个答案。

2.对这个反应信息使用相同的查询。

query isContainedIn( String x, String y )

Location( x, y; )

**or**

( Location( z, y; ) **and** isContainedIn( x, z; ) )

**end**

3.创建“go3”，将其插入引擎，然后调用fireAllRules。

ksession.insert( "go3" );

ksession.fireAllRules();

---

go3

匹配任何字符串的第一条规则返回“go3”，但没有其他内容被返回，因为没有答案; 然而，在系统中插入“go3”时，它会一直等到它满足为止。

4.在“抽屉”中插入一个新的“钥匙”位置：

ksession.insert( **new** Location("key", "drawer") );

ksession.fireAllRules();

---

Key **in** the Office

这个新位置满足了传递闭包，因为它正在监视整个图。另外，这个过程现在有四个递归级别，在这个级别中匹配和触发规则。

#### [25.17.6。带有未绑定参数的查询](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_queries_with_unbound_arguments)

1.通过实施以下规则创建具有未绑定参数的查询：

rule "go4"

**when**

**String**( **this** == "go4" )

**isContainedIn**(thing, "office"; )

then

System.**out**.**println**( "thing" + thing + "is in the Office" );

end

这条规则是要求“办公室”中的所有内容，它会告诉下面所有行。这个例子中的未绑定参数（out变量）会返回每个可能的值; 因此，它与反应式查询示例中使用的z值非常相似。

2.使用查询获取未绑定的参数。

query isContainedIn( String x, String y )

Location( x, y; )

**or**

( Location( z, y; ) **and** isContainedIn( x, z; ) )

**end**

3.创建“go4”，将其插入引擎，然后调用fireAllRules。

ksession.insert( "go4" );

ksession.fireAllRules();

---

go4

thing Key **is** **in** the Office

thing Computer **is** **in** the Office

thing Drawer **is** **in** the Office

thing Desk **is** **in** the Office

thing Chair **is** **in** the Office

当插入“go4”时，它将返回所有先前在“Office”下传递的信息。

#### [25.17.7。多个未绑定的参数](https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html#_multiple_unbound_arguments)

1.通过实施以下规则创建一个包含Mulitple Unbound Arguments的查询：

rule "go5"

**when**

**String**( **this** == "go5" )

**isContainedIn**(thing, location; )

then

System.**out**.**println**( "thing" + thing + "is in" + location );

end

这条规则是要求“办公室”中的所有内容，它会告诉下面所有行。这个例子中的未绑定参数（out变量）会返回每个可能的值; 因此，它与反应式查询示例中使用的z值非常相似。

这两个**东西**和**位置**都绑定了变量，没有绑定参数，一切都在呼吁。

2.对多个未绑定的参数使用查询。

query isContainedIn( String x, String y )

Location( x, y; )

**or**

( Location( z, y; ) **and** isContainedIn( x, z; ) )

**end**

3.创建“go5”，将其插入引擎，然后调用fireAllRules。

ksession.insert( "go5" );

ksession.fireAllRules();

---

go5

thing Knife **is** **in** House

thing Cheese **is** **in** House

thing Key **is** **in** House

thing Computer **is** **in** House

thing Drawer **is** **in** House

thing Desk **is** **in** House

thing Chair **is** **in** House

thing Key **is** **in** Office

thing Computer **is** **in** Office

thing Drawer **is** **in** Office

thing Key **is** **in** Desk

thing Office **is** **in** House

thing Computer **is** **in** Desk

thing Knife **is** **in** Kitchen

thing Cheese **is** **in** Kitchen

thing Kitchen **is** **in** House

thing Key **is** **in** Drawer

thing Drawer **is** **in** Desk

thing Desk **is** **in** Office

thing Chair **is** **in** Office

当“go5”被调用时，它会返回所有内容中的所有内容。

版本7.7.0.Final   
上次更新时间2018-03-26 11:11:06 CEST