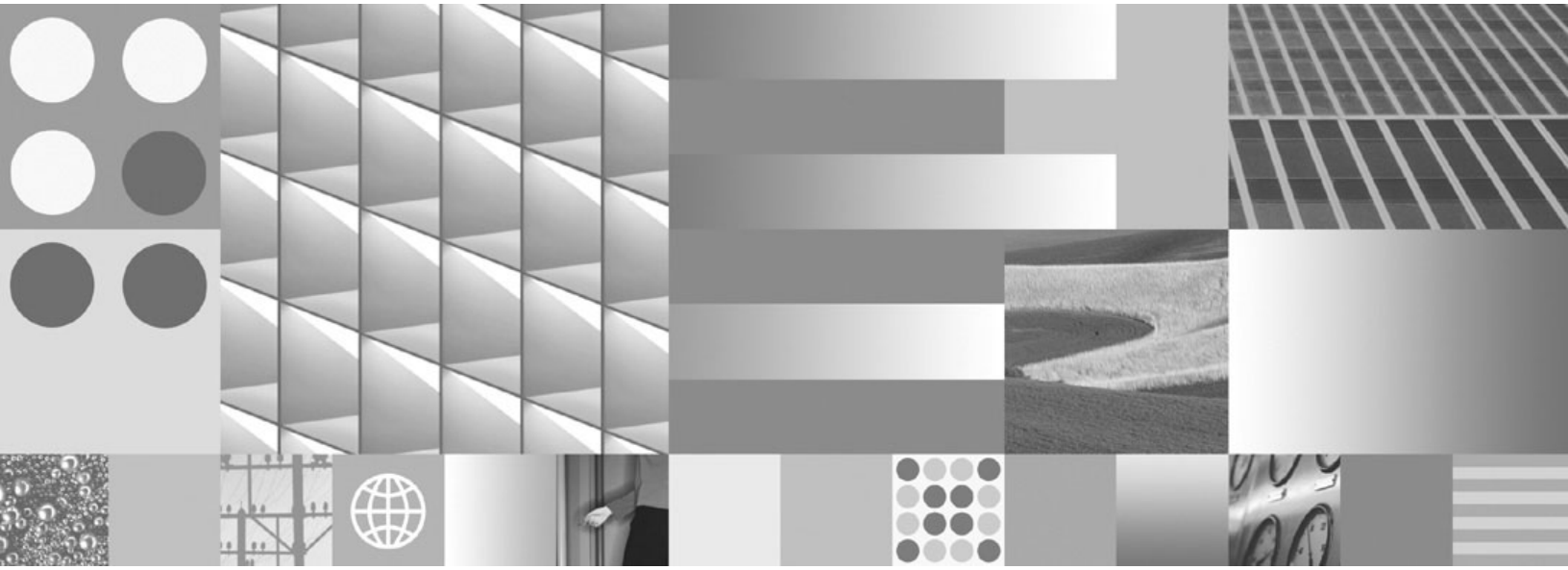
活动关联技术

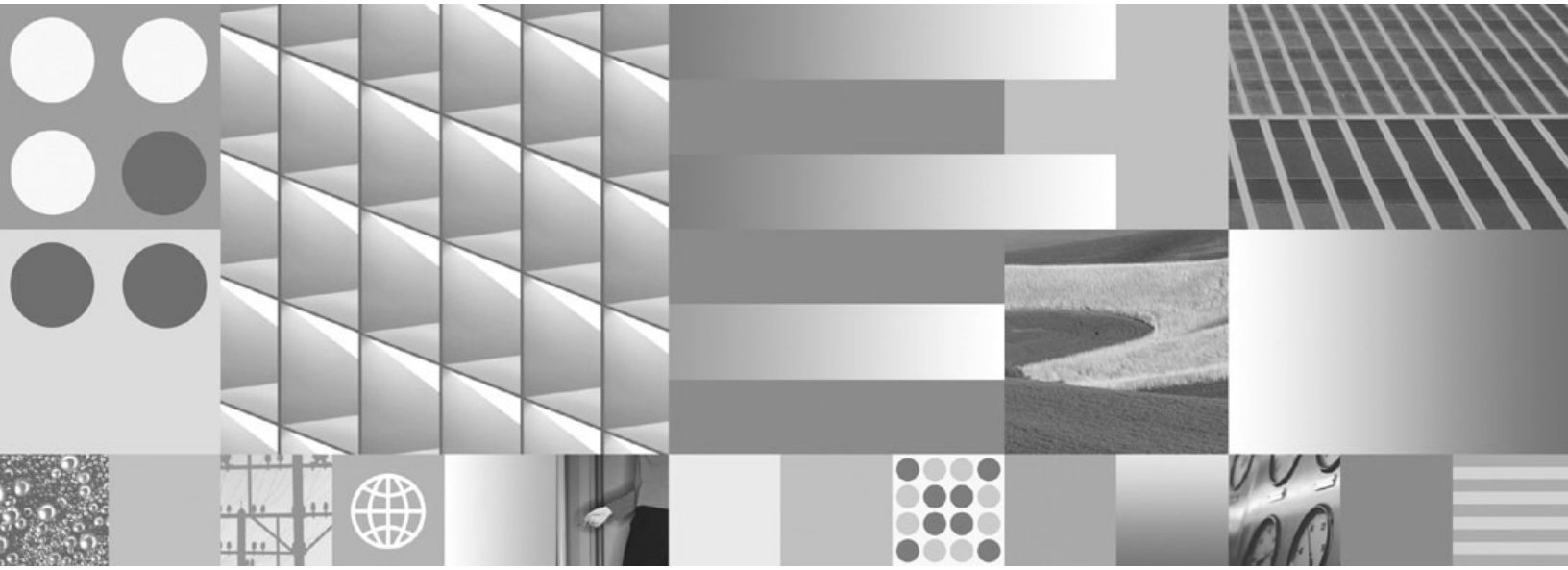
版本 6.0



规则编写者指南与规则编写者参考大全

活动关联技术

版本 6.0



规则编写者指南与规则编写者参考大全

注

在使用本资料及其支持的产品之前，请先阅读第 113 页的『声明』中的信息。

第一版（**2006** 年 **6** 月）

关于本资料

本资料概述了 IBM®活动关联技术及其在复杂事件处理中的作用。活动关联技术规则语言是基于 XML 的语言，用于编写关联事件的规则。本资料是为规则编写者设计的，编写者必须理解如何为业务机构编写用于关联事件的规则。

目录

第 1 部分 规则编写者指南

第 1 章 简介

本简介将简要描述复杂事件处理（也称为 CEP）并概述活动关联技术及其在复杂事件处理中的作用。

现今的商业环境

如今，商业和政府机构依赖于通过计算机网络（尤其是因特网）的电子信息处理。借助附加技术（例如，网格计算），这些机构无论何时何地均可运行关键任务应用程序。业务流程、活动、基础结构以及我们的整个社会都依赖于这些机构的信息技术（IT）层。

这些机构需要实时了解业务所发生的任何情况。例如，它们需要了解关键任务应用程序是否可用、是否运行正常，以及如何检测并避免业务流程、活动或基础结构中存在潜在的危机。如果危机发生，则它们立即需要了解所发生的问题、如何解决问题，以及原因所在。

由于信息量之大且存在于单个、无关联的片断中导致难于提炼出来，因此无法识别或了解涉及业务流程、活动和基础结构的大多数事件的重要性。然而，如果将这些事件聚集并进行关联以使它们的关系容易被了解，它们就可生成大量的信息。

复杂事件处理的用途在于实时获取关于事件的有价值的信息。

复杂事件处理

*事件*只是有关已发生情况的通知。

*复杂事件处理*是通过对事件驱动的系统中的低级别事件进行分析、关联和总结，来得到高级别事件。这些高级别事件称为复杂事件，适用于以易于理解的术语来通知相关人员存在业务机会或问题，或者触发自动流程。借助对潜在机会或问题的预警机制，以及对能够改变业务流程、活动或基础结构状况的根本原因的更好理解，这些机构从而可运作得更为有效。

*事件相关*是实时定义并检测事件流中的模式并实施操作来响应相关事件的过程。它用于根据检测到的症状来识别问题。可通过原因、时间、成员资格或这些元素的组合来关联事件。事件相关为复杂事件处理的一个不可缺少部分。

活动关联技术

活动关联技术使用规则来实时检测事件流中的模式。该技术基于以下认识：很多情况下，响应操作不应由单个低级别事件触发，而是由在不同时间和不同上下文中发生的事件的复杂组合来触发。活动关联技术使用事件之间的关系来提示业务机会和问题。例如，基于通过事件实时关联所得的业务提示，某家公司可执行以下类型的操作:

- 在节日销售期间，向部分或所有客户提供装运折扣。
- 在接下来的 30 天内，根据承运商、订单金额和订购量来计算装运成本。
- 向在 2005 年 7 月 1 日至 2005 年 12 月 31 日期间购买价值超过 500 美元产品的客户发送 25 美元的礼券。

- 如果任何订单处理在 36 小时内未完成，则将通知管理员。
- 如果在 30 秒内检测到超过 4 次登录相同计算机的尝试，则将通知管理员。

活动关联技术由以下主要项目组成:

活动关联技术规则语言（Active Correlation Technology rule language）
基于 XML 的语言，用于编写关联事件的规则。随后可将这些规则部署至活动关联技术运行时环境。

活动关联技术引擎（Active Correlation Technology engine）
活动关联技术组件，用于根据活动关联技术编译器的输出来处理事件。

活动关联技术规则构建器（Active Correlation Technology rule builder）
GUI，用于使用活动关联技术规则语言来编写关联规则。

活动关联技术运行时环境是活动关联技术引擎所嵌入至的应用程序。

第 2 章 规则语言概述

本概述描述活动关联技术规则语言的主要概念。

规则模式为事件相关情况（例如，阈值条件或重复事件检测）的表示法。活动关联技术规则语言中包含 7 种规则模式，这些模式已经过验证可代表 IBM 客户需要处理的大多数事件相关情况。其中 6 种规则模式定义全状态规则，1 种规则模式定义无状态规则。

全状态规则用于关联特定时间段内发生的多个事件，并生成对这些事件的响应。无状态规则仅处理符合特定条件的单个事件，并生成对该事件的响应。

全状态规则（stateful rule）

保留状态信息（有关规则实例特性的信息）的规则，用于在某个时间段内对一组事件执行操作。以下任何规则模式定义的规则均为全状态规则：收集、计算、重复、序列、阈值或定时器。

无状态规则（stateless rule）

不保留状态信息的规则，因此在某个时间仅可对一个事件执行操作。由过滤模式定义的规则为无状态规则。

相关参考

第 41 页的第 4 章，『规则集组织摘要』
本参考列出规则集、规则块和每种规则类型的所有语言元素。它可作为编写规则集的快速参考。

第 53 页的第 5 章，『语言元素参考』
本参考将描述活动关联技术规则语言的 XML 模式中的语言元素的详细信息。 这些语言元素以字母顺序列出，且对每个元素可用的属性在该元素的主题中有所描述。

第 109 页的第 6 章，『词汇表』
本词汇表包含活动关联技术中重要概念的术语和定义。

规则剖析

规则的最基本部分为事件选择、组密钥、全状态规则的时间窗口、规则响应、激活时间间隔以及生命周期操作。规则还包括表达式和变量。表达式是包含定制逻辑的代码，可将该逻辑添加至规则。

事件选择

事件选择条件将确定由规则接受以进行处理的事件。 <eventSelector> 元素定义规则的事件选择条件。除由定时器模式定义的规则外，事件选择适用于所有其他规则。由于定时器规则不处理事件，因此不包含事件选择条件。

组密钥

通常情况下，每个活动的规则都具有一个正在活动关联技术引擎中运行的规则实例或副本。然而，同一规则有时为不同的事件组所需，而不同的事件组一般与不同的资源组相关联。组密钥是用于指定规则针对每组共享公共特性的事件创建不同规则实例（或其副本）的方法。

组密钥将作为事件选择的附加形式。如果使用组密钥定义了某个规则，且该规则接收到带有组密钥定义的特性的事件，则该事件将被发送至正在处理共享该特性的事件的规则实例。例如，您可定义用于收集类型为Audit Failure的所有安全性事件的规则，并将组密钥定义为某个事件的主机名属性。现在，该规则可多次使用，针对主机名属性的每个唯一值而运行该规则的不同副本。还可监视接收Audit Failure事件的所有系统，以便确定每个主机名在 2 分钟的时间段内发生的此类事件是否超过 10 个。

<groupingKey> 元素定义规则的组密钥，并对由收集、计算、重复、序列和阈值模式定义的规则有效。

全状态规则的时间窗口

由于全状态规则与特定时间段内发生的多个事件相关联，因此全状态规则的基本部分为 <timeWindow> 元素定义的时间窗口。时间窗口指定时间段，在该时间段内，全状态规则正在进行处理以与其模式发生匹配。

规则响应

规则响应操作定义在规则完成其处理后要采取的操作。下列每个语言元素都定义不同类型的规则响应操作:

- <onDetection> 中的 <action>
- <onNextEvent> 中的 <action>
- <onTimeOut> 中的 <action>
- <onTimeWindowComplete> 中的 <action>

对规则可用的规则响应操作的类型取决于规则模式。

激活时间间隔

激活时间间隔定义规则何时处于活动状态、何时处于不活动状态。<activationInterval> 元素定义规则的激活时间间隔。

可在离散时间点上激活或停用规则，或者由特定事件对其进行激活或停用。

如果将规则指定为在离散时间点上*以及*由特定事件进行激活或停用，则该规则将在该时间点或在接收到事件时（取决于哪个发生在先）被激活或停用。然而在该情况下，在该规则的生命周期内可能由多个事件对其进行激活或停用。例如，规则可能由某个事件激活，然后被停用，接着在某个定义的时间点上被激活，然后再次被停用，随后再由另一个事件激活。

<activationByGroupingKey> 元素是包含于 <activationInterval> 元素中的一个元素。<activationByGroupingKey> 元素包含用于指定事件的元素，这些事件可激活和停用由 <groupingKey> 元素定义的规则实例。

生命周期操作

生命周期操作定义在规则生命周期的以下 4 个主要阶段要采取的操作: 装入、激活、停用和卸装。

<lifeCycleActions> 元素包含用于定义下列操作的下列元素:

- <onLoad> 中的 <action>
- <onActivation> 中的 <action>
- <onDeactivation> 中的 <action>
- <onUnload> 中的 <action>

规则的生命周期

规则的生命周期中的每个阶段均可具有多个原因和结果。通过在生命周期操作（由 <lifeCycleActions> 元素定义）中编写和包含表达式，规则编写者可定义在每个阶段要执行的操作。

规则的生命周期中的阶段

以下为规则的生命周期中的 4 个主要阶段:

装入 将规则装入运行中的活动关联技术引擎，这将触发 <onLoad> 元素中的操作。

激活 激活规则，这将触发 <onActivation> 元素中的操作。

停用 停用规则，这将触发 <onDeactivation> 元素中的操作。

卸装 从运行中的活动关联技术引擎卸装规则，这将触发 <onUnload> 元素中的操作。

激活和停用阶段可在规则的生命周期中发生多次，但装入和卸装阶段仅可发生一次。

通常情况下，无需定义生命周期操作。以下为可能要定义特定生命周期操作的情形的示例:

- 当装入某个规则时，可能要创建到需要在该规则中访问的外部系统（例如，数据库管理器）的连接。当卸装这同一个规则时，可能要断开该连接并运行任何必要的清除进程。
- 当激活某个规则时，可能要验证某些资源是否对该规则可用。
- 当停用某个阈值规则但未符合阈值，且时间段也尚未结束时，可能要将带有该信息的消息转发给某人。

由于规则的激活和停用可在生命周期内发生多次，因此为这些阶段编写的所有操作均可多次运行。

每个生命周期阶段的原因和结果

表 1 列出了每个生命周期阶段的原因和结果。

表 1. 每个生命周期阶段的原因和结果

生命周期阶段	原因	结果
装入	以下任一情况: • 添加或替换规则或规则块，这将导致装入新的规则。 • 在活动关联技术引擎中替换规则集，这将导致装入新规则集中的规则。	<onLoad> 元素中的操作将运行。
激活	激活规则。可使用以下任一方式来激活规则: • 根据 <activationInterval> 元素中的定义 • 使用 activate() 方法，该方法是通过 act_lib 变量提供的 • 通过应用程序调用，来调用活动关联技术引擎中的 activate() 方法	如果规则处于不活动状态，则 <onActivation> 元素中的操作将运行。
停用	停用规则。可使用以下任一方式来停用规则: • 根据 <activationInterval> 元素中的定义（例外情况: <activationByGroupingKey> 元素中的 <deactivateOnEvent> 元素不会导致规则停用） • 使用 deactivate() 方法，该方法是通过 act_lib 变量提供的 • 通过应用程序调用，来调用活动关联技术引擎中的 deactivate() 方法	如果规则处于活动状态，则 <onDeactivation> 元素中的操作将运行。
卸装	以下任一情况: • 活动关联技术引擎关闭，这将导致卸装规则。 • 除去或替换规则或规则块，这将导致卸装旧的规则。 • 从活动关联技术引擎中除去规则集或替换其中的规则集，这将导致卸装旧规则集中的规则。	如果规则处于活动状态，则将运行 <onDeactivation> 元素中的操作，随后为 <onUnload> 元素中的操作。否则，仅运行 <onUnload> 元素中的操作。

规则的结构

活动关联技术规则语言将规则组织为规则块，这些规则块是规则集的组成部分。

规则集

规则集包含规则，组织成规则块，由活动关联技术引擎来执行。它是规则执行单元。每个活动关联技术引擎每次仅对一个规则集执行操作。

规则集中包含的规则由发送至活动关联技术引擎的事件触发。根据每个规则的事件选择条件将事件顺序传递至相应的规则，且每次执行一个规则。同一事件可应用至（并因此触发）多个规则。虽然这些规则未必是关联的，但它们可能是关联的。

规则集中的规则块和规则的顺序将决定事件如何通过规则集。

可按规则集级别来定义变量和导入，以用于该规则集作用域内的表达式（即：包含定制逻辑的代码）中。导入是用于访问外部代码的特定于编程语言的方式。规则编写者可定义某个导入以便导入外部模块（例如 Java™ 类）以在规则内的表达式中使用。

规则块

规则块是规则集中根据功能将规则分组到域中的组织单元。域为一组规则根据自身的功能所应用至的类别。例如，域可代表具体的地区、IT 管理规程（例如，安全性检测或网络事件相关）或业务机构（例如，具体的公司或分公司）。

规则块可包含规则和其他规则块。由于可对规则块进行嵌套，因此可构建规则的层次结构。例如，规则集可包含网络事件相关的规则块，且网络事件相关的规则块可能包含两个其他规则块：一个用于第 2 层关联，另一个用于 IP 关联。

因此，规则集将对多种域提供事件相关能力，而规则块能够将这些可能需要访问一组相似事件的不同域进行组织。

可按规则块级别来定义变量和导入，以便用于该规则块作用域内的表达式中。规则块的作用域包括该规则块中包含的所有规则和其他规则块。

规则

规则是用于识别事件之间的关系并运行相应规则响应的关联单元。一个规则是下列 7 种规则模式之一的一种实施（按功能将规则组织成规则块，规则块是规则集的一部分）：

- 收集模式
- 计算模式
- 重复模式
- 过滤模式
- 序列模式
- 阈值模式
- 定时器模式

每个规则可根据其模式提供独特的事件相关能力，并可通过事件转发来构成规则链。通过该规则链，可将不同模式的事件相关能力进行组合或嵌套。

可按规则级别来定义变量，以便用于该规则作用域内的表达式中。

总结

总而言之，规则集为执行单元，规则块为组织单元，而规则为关联单元。规则集包含一个或多个规则块，每个规则块可包含其他规则块。每个规则块包含某个特定域的规则。可对规则块进行嵌套以构建规则的层次结构。规则集中的规则块和规则的顺序将决定事件如何通过规则集。

可按规则集或规则块级别来定义变量和导入，以便用于规则内的表达式中。变量或导入的作用域为各自的规则集或规则块。还可按规则级别来定义变量，但这将使它们的作用域仅限于该规则。

规则模式

规则模式为事件相关情况（例如，阈值条件或重复事件检测）的表示法。活动关联技术规则语言定义以下规则模式: 收集、计算、重复、过滤、序列、阈值和定时器。

当发生某个规则定义的情境时，将与该规则的模式匹配。当该模式匹配时，该规则将执 行相应的规则响应操作来结束其处理。在规则处于活动状态时，该规则模式可发生多次匹配。

由过滤模式定义的规则为规则语言中唯一的无状态规则。所有其他规则均为全状态规则。

收集模式

收集规则由收集模式定义。该规则将在一定时间间隔内收集一组选定的事件。该规则为全状态规则。

概述

收集模式用于在某个时间段内收集相似的事件。时间段由必需的时间窗口指示，该窗口使用规则语言中的 <timeWindow> 元素定义。

规则响应的运行条件

使用收集模式，规则响应将在时间窗口完成后（由 <onTimeWindowComplete> 元素定义）运行。

此规则模式的示例用法

收集模式的示例用法为执行以下操作的规则:

它在一定时间段内收集符合某个事件选择器条件的事件。当该时间段结束后，它将把所收集的事件汇总为单个事件，该事件中包含所收集事件的总数以及关于汇总事件的特征信息。

相关参考

第 42 页的『收集规则摘要』
本摘要列出收集规则的所有语言元素。

计算模式

计算规则由计算模式定义。该规则在一定时间间隔内接收到每个事件时，将计算（通过表达式）应用于所收集的事件。该规则为全状态规则。

概述

计算模式对在某个时间段内接收到每个事件时运行计算功能（使用语言中的 <computeFunction> 元素定义）。时间段由必需的时间窗口指示，该窗口由 <timeWindow> 元素定义。

规则响应的运行条件

使用计算模式，规则响应将在时间窗口完成后（由 <onTimeWindowComplete> 元素定义）运行。计算的值在 <onTimeWindowComplete> 操作期间中可用。

此规则模式的示例用法

假定某个应用程序正在处理客户订单事件。计算模式的示例用法为执行以下操作的规则:

每次接收到事件后，都将订单的总值添加至指定时间段内已出现的所有订单的总值，并且更新后的所有订单的总值将发布在用户界面上。

相关参考

第 44 页的『计算规则摘要』
本摘要列出计算规则的所有语言元素。

重复模式

重复规则由重复模式定义。该规则计算在指定的时间间隔内接受的第二个时间和后续事件，但跳过对这些事件的规则集处理。该规则为全状态规则。

概述

重复模式通常用于在某个时间段内隔离相似（重复）的事件。重复事件在某些方面与先前事件相似，但不一定是该事件的相同副本。只要事件符合规则的事件选择条件，则认为这些事件重复。时间段由必需的时间窗口指示，该窗口使用规则语言中的 <timeWindow> 元素定义。

规则响应的运行条件

使用重复模式，规则响应将在以下时间运行:

- 检测到第一个事件时（由 <onDetection> 元素定义）。
- 处理每个重复事件时（由 <onNextEvent> 元素定义）。
- 时间窗口完成时（由 <onTimeWindowComplete> 元素定义）。

即使未接收到重复事件，第一个事件仍将触发 <onDetection> 操作。设置该行为的原因在于您可能要转发第一个事件并跳过重复事件的规则集处理。在这种情况下，您可以添加规则响应操作，该操作在触发规则的 <onDetection> 操作时转发第一个事件。

重复事件（第二个以及后续事件）的缺省处理是计算重复事件数量，但跳过重复事件的规则集处理。如果要对重复事件执行其他操作，则可明确定义 <onNextEvent> 操作。例如在某些情况下，重复事件代表可能已记录到数据库或其他存储库中的事件。因此，可能要对 <onNextEvent> 操作进行编码以除去来自这些其他位置的重复事件。

<onTimeWindowComplete> 操作可用于创建包括那些已处理的重复事件在内的所有重复事件的摘要记录。

此规则模式的示例用法

假定持续出现来自相同资源类型（安全监视器）的『拒绝服务』消息。这表明可能存在安全性违规。重复模式的示例用法为执行以下操作的规则:

在出现来自安全监视器的『拒绝服务』消息后，将计算 30 秒时间内发生的所有重复事件的次数，但不发送至操作员控制台。同时，在 30 秒时间段结束时，规则将生成事件来指示在该时间段内出现的『拒绝服务』消息的次数。

相关参考

第 45 页的『重复规则摘要』
本摘要列出重复规则的所有语言元素。

过滤模式

过滤规则由过滤模式定义。该规则在接受事件时将执行某个操作。它只对单个事件执行操作，因此是无状态规则。

概述

过滤模式用于对符合事件选择条件的单个事件执行操作。与其他规则模式不同，过滤模式不保留相关的状态信息（例如，过去事件的历史记录）。

规则响应的运行条件

使用过滤模式，规则响应将在接收到任一符合事件选择条件的事件时（由 <onDetection> 元素定义）运行。

此规则模式的示例用法

过滤模式的示例用法为执行以下操作的规则:

如果 ServerStatus 事件表明 serverLoad 大于 95%，则该规则将运行呼叫管理员的操作。

相关参考

第 47 页的『过滤规则摘要』
本摘要列出过滤规则的所有语言元素。

序列模式

序列规则是由序列模式定义的。它将检测某个时间间隔内是否到达了一系列特定事件。序列可有序，也可随机。序列规则为全状态规则。

概述

序列模式将检查某个时间段内的事件序列，并检测该序列是否完整。不完整的序列是按指定序列包含一个或多个事件（但并非所有事件）的序列。

时间段由必需的时间窗口指示，该窗口使用规则语言中的 <timeWindow> 元素定义。序列中的每个事件由规则中单独的 <eventSelector> 元素来定义。可按以下任意顺序来检测事件序列:

- 按为规则编写 <eventSelector> 元素的顺序。在该情况下，当规则检测到由第 1 个 <eventSelector> 元素定义的事件时，将启动序列检测。现在，规则将等待由第 2 个 <eventSelector> 元素定义的事件。
- 按随机顺序。在该情况下，当规则检测到由 <eventSelector> 元素定义的任意一个事件时，都将启动序列检测。现在，规则将等待由 <eventSelector> 元素定义的另一个事件。

序列模式在下列主要方面不同于其他规则模式:

- 它具有多个 <eventSelector> 元素用于定义由规则接受的事件。最少需要两个 <eventSelector> 元素。

- 当事件符合由某个 <eventSelector> 元素定义的条件时，该 <eventSelector> 元素将被排除在该规则实例中的进一步事件处理之外。
- <eventSelector> 元素中的别名属性仅在序列规则中有效，且该属性唯一地为序列规则中的特定事件选择器选择的事件命名。在过滤谓词或操作内的表达式中，可使用 act_eventList 变量来通过某个事件的别名在序列规则中访问该事件。

规则响应的运行条件

使用序列模式，规则响应将在以下时间运行:

- 在时间窗口中检测到完整事件序列时，如 <onDetection> 元素所定义。
- 在时间窗口中一个或多个事件已到达而完整事件序列未到达时，如 <onTimeOut> 元素所定义。

  序列模式可用于检测给定的序列是否完整。例如，如果发生“系统关闭”事件而没有后续的“系统运行”事件，则规则编写者可编写 <onTimeOut> 操作以处理该类型的缺失事件。

此规则模式的示例用法

说明检测完整序列的场景:
假定在 IT 环境中，管理员希望了解 DB2® 堆大小的值是否正在影响 WebSphere® Application Server，如果是，则更正该问题。因此，如果在指定的时间段内按下列顺序发生以下事件，则管理员要增大 DB2 堆大小的值并重新启动数据库管理器:

1. WebSphere Application Server 资源分配异常。假定这是 WASResourceAllocationException 类型的事件。
2. DB2 错误消息，其内容为: 『堆不足以处理语句』。假定这是 DB2NotEnoughHeap 类型的事件。

对于此场景，在序列规则中定义了 2 个 <eventSelector> 元素，且事件必须按编写 <eventSelector> 元素代码的顺序（而非随机顺序）到达。第 1 个 <eventSelector> 元素将检查事件 WASResourceAllocationException，第 2 个 <eventSelector> 元素将检查事件 DB2NotEnoughHeap。假定系统在指定的时间窗口中存在下列事件:

1. WASResourceAllocationException
2. DB2BackupStarted
3. WASResourceAllocationException
4. WASResourceAllocationException
5. DB2NotEnoughHeap

规则行为如下:

1. 接受第 1 个事件 WASResourceAllocationException。由于已符合第 1 个 <eventSelector> 元素的条件，因此第 1 个 <eventSelector> 元素目前已被排除在该规则中的进一步事件处理之外。
2. 忽略第 2 个事件 DB2BackupStarted。
3. 忽略第 3 个事件 WASResourceAllocationException。
4. 忽略第 4 个事件 WASResourceAllocationException。

5. 接受第 5 个事件 DB2NotEnoughHeap，并完成该序列。<onDetection> 规则响应操作将运行。该操作被定义为增大 DB2 堆大小的值并重新启动数据库管理器。规则将返回到它的初始状态。

   该规则现在将第 1 个 <eventSelector> 元素包含到未来的事件处理中。

说明检测不完整序列的场景:
假定业务机构希望在接收到订单请求后的 1 小时内使所有的客户订单都可供货，并希望了解什么时候没有这样做到。

对于此场景，在序列规则中定义了 2 个 <eventSelector> 元素，且事件必须按编写 <eventSelector> 元素代码的顺序（而非随机顺序）到达。第 1 个 <eventSelector> 元素将检查事件 Netsales（带有 operationType=Order），第 2 个 <eventSelector> 元素将检查事件 Netsales（带有 operationType=Delivery）。假定系统在指定的时间窗口（1 小时）中存在下列事件:

1. Netsales 事件（带有 operationType=Order）
2. Netsales 事件（带有 operationType=Order）

规则行为如下:

1. 接受第 1 个事件。由于已符合第 1 个 <eventSelector> 元素的条件，因此第 1 个 <eventSelector> 元素目前已被排除在该规则中的进一步事件处理之外。
2. 忽略第 2 个事件。
3. 因为在指定的时间窗口内没有接收到 Netsales 事件（带有 operationType=Delivery），所以将运行 <onTimeOut> 规则响应操作。此操作被定义为通知业务主管: 未能在接收到订单请求后的 1 个小时内使客户订单可供货。规则将返回到它的初始状态。

   该规则现在将第 1 个 <eventSelector> 元素包含到未来的事件处理中。

   相关概念

   第 20 页的『访问与事件相关的信息』
   以下示例说明如何通过活动关联技术提供的变量来访问与事件相关的信息。

   相关参考

   第 48 页的『序列规则摘要』
   本摘要列出序列规则的所有语言元素。

阈值模式

阈值规则是由阈值模式定义的。该规则将在某个时间间隔内收集一组选定的事件，并在接收到每个事件后确定是否已符合阈值条件。该规则为全状态规则。

概述

在符合阈值之前，阈值模式将在某个时间段内一直收集事件。时间段由必需的时间窗口指示，该窗口使用规则语言中的 <timeWindow> 元素定义。

阈值模式将为阈值类型提供以下三个选项:

事件计数阈值

使用该类型的阈值，可定义某个时间段内必须符合事件选择条件的事件的数量。定义的阈值将与已接受的事件数量进行比较。当事件计数与时间窗口中定义的限制相等时，将符合该阈值。

该类型的阈值可用于极为简单的事件计数检查。例如，可用于回答以下问题：『是否在 1 分钟内发生了 5 次登录失败事件？』

该阈值由 <eventCountThreshold> 元素定义。<eventCountThreshold> 元素还指定时间窗口的以下两个可能的时间间隔方式之一：

固定时间间隔

固定时间间隔在接收到第一个符合事件选择条件的事件时开始，在发生以下某种情况时结束：

- 规则在指定的持续时间内符合其阈值。
- 指定的持续时间已过。

滑动时间间隔

滑动时间间隔在接收到第一个符合事件选择条件的事件时开始。然而，当规则尚未符合其阈值且指定的持续时间已过时，时间窗口将把开始时间调整（滑动）为新的“第一个”事件的接收时间，通常情况下为接受的下一个事件。在发生以下某种情况之前，滑动时间间隔将继续以该方式进行调整：

- 规则在指定的持续时间内符合其阈值。
- 在接收到启动时间窗口的事件后，指定的持续时间内未接收到后续事件。

启动时间窗口的事件（成为新的“第一个”事件）为具有符合以下条件的接收时间的事件：接收时间加上规则的时间间隔持续时间大于当前时间。以下为公式形式的条件：

事件接收时间 + 规则的时间间隔持续时间 > 当前时间

当不存在此类事件时，滑动时间间隔将无法再调整时间，时间间隔将结束。

计算的阈值

使用该类型的阈值，可编写代码（或者使用他人编写的代码），该代码将对每个已接受事件执行计算，并将返回计算的阈值（保存在先前定义的变量中）。该计算的阈值将随后与定义的阈值进行比较，以确定是否符合阈值。

因此，可将复杂计算应用于创建（或更新）计算的阈值（可能使用从先前事件保存的数据），且规则编写者可以独立于用于计算“计算的阈值”的逻辑来设置“定义的阈值”。

该类型的阈值可用于将某个值聚集并与定义的阈值进行比较。例如，可用于计算在某个时间段内对某个客户的销售总额，并将该总额与定义的阈值进行比较。

该阈值由 <computedThreshold> 元素定义。

布尔阈值

使用该类型的阈值，可编写代码（或者使用他人编写的代码），该代码对每个已接受事件返回值 `true` 或 `false`。如果值为 `true`，则已符合阈值。如果值为 `false`，则阈值规则将在该时间段结束或接受其他事件之前继续处理。

该类型的阈值可用于检查某个范围的值。例如，如果 CPU 使用率必须始终介于 30% 和 80% 之间，则该阈值可经常验证使用率是否保持在该范围内。

该阈值由 <booleanThreshold> 元素定义。

规则响应的运行条件

对于阈值模式，规则响应将在以下时间运行:

- 当已符合阈值时，如 <onDetection> 元素定义。
- 在时间窗口中接受了一个或多个事件但未符合阈值时，如 <onTimeOut> 元素定义。

此规则模式的示例用法

带有事件计数阈值的阈值模式的示例用法是某个执行以下操作的规则:

如果在 30 秒的滑动时间间隔内，从相同子网发生的 `Server unreachable` 事件超过 4 次，则该规则将运行操作以检查路由器的状态。

相关参考

第 50 页的『阈值规则摘要』
本摘要列出阈值规则的所有语言元素。

定时器模式

定时器规则由定时器模式定义。该规则按照规定的时间间隔启动操作。该规则为全状态规则。虽然定时器规则不处理事件，但事件可对其进行激活或停用。

概述

定时器模式类似于在定时器，它在某个时间段开始时启动并在该时间段结束时停止。时间段由必需的时间窗口指示，该窗口使用规则语言中的 <timeWindow> 元素定义。

除非指定为不重复，否则在停用定时器规则之前定时器模式将一直重复。因此，当定时器规则启动时，将在启动任何操作前等待指定的时间，并在被停用或活动关联技术引擎关闭之前一直重复该行为。

定时器规则的特别之处在于它不包含事件选择条件。定时器规则根据规则的激活时间间隔来启动处理，时间间隔由 <activationInterval> 元素定义。如果使用缺省的 <activationInterval> 元素且定时器模式设置为重复，则定时器规则在活动关联技术引擎将其装入时启动并在活动关联技术引擎关闭时停止。要使用事件激活定时器规则，必须在 <activateOnEvent> 元素（位于规则的 <activationInterval> 元素内）中指定该事件。

规则响应的运行条件

使用定时器模式，规则响应将在时间窗口完成后（由 <onTimeWindowComplete> 元素定义）运行。

此规则模式的示例用法

定时器模式对于实现清除规则很有用。定时器模式的示例用法为执行以下操作的规则:

该规则将每 30 分钟运行一次操作，该操作将清除打开时间已超过 48 小时的无害参考事件。

相关参考

第 51 页的『定时器规则摘要』
本摘要列出定时器规则的所有语言元素。

不同规则模式的一般方面和独特方面

该矩阵将提供不同规则模式的一般方面和独特方面的高级别概述。

表 2 将列出规则的主语言元素，并在元素有效的每个规则类型对应的列上显示 X。主语言元素为不同规则类型的直接子元素。该列表不包括包含于这些直接子元素中的元素，这些元素也会随规则类型的不同而有所不同。此外，某些元素属性的有效性会随规则类型的不同而有所不同。

表 2. 矩阵显示不同规则模式的一般方面和独特方面

元素	收集	计算	重复	过滤	序列	阈值	定时器
<comment>	X	X	X	X	X	X	X
<variable>	X	X	X	X	X	X	X
<activationInterval>	X	X	X	X	X	X	X
<lifeCycleActions>	X	X	X	X	X	X	X
<eventSelector>	X	X	X	X	X	X	
<groupingKey>	X	X	X		X	X	
<timeWindow>	X	X	X		X	X	X
<computeFunction>		X					
<booleanThreshold>						X	
<computedThreshold>						X	
<eventCountThreshold>						X	
<onDetection>			X	X	X	X	
<onNextEvent>			X				
<onTimeOut>					X	X	
<onTimeWindowComplete>	X	X	X				X

表达式

表达式是包含定制逻辑的代码，可将该逻辑添加至规则。表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素中有效。

规则编写者可根据上下文以及要得到的结果来编写用于不同用途的表达式。表达式通常用于变量的初始化、事件选择条件的定义以及规则响应和生命周期操作的指定。

包含表达式的语言元素

每个包含表达式的语言元素都具有 expressionLanguage 属性，该属性标识编写表达式所使用的编程语言。Java 编程语言是唯一受支持的表达式语言。

表达式可包含于以下规则语言元素中。

- 规则集、规则块或规则变量的 <varInitializer>
- <eventSelector> 上的 <filteringPredicate>
- <groupingKey> 上的 <computedValue>

- 计算规则上的 <computeFunction>
- 阈值规则上的 <booleanThreshold>
- 阈值规则上的 <computedThreshold>
- 规则的规则响应操作:
  - <onDetection> 中的 <action>。该操作仅对重复规则、过滤规则、序列规则和阈值规则有效。
  - <onNextEvent> 中的 <action>。该操作仅对重复规则有效。
  - <onTimeOut> 中的 <action>。该操作仅对序列规则和阈值规则有效。
  - <onTimeWindowComplete> 中的 <action>。该操作仅对收集规则、计算规则、重复规则和定时器规则有效。
- 规则的生命周期操作:
  - <onLoad> 中的 <action>
  - <onActivation> 中的 <action>
  - <onDeactivation> 中的 <action>
  - <onUnload> 中的 <action>

活动关联技术提供的用于帮助编写表达式的能力

为帮助规则编写者编写表达式，活动关联技术提供用于执行以下操作的能力:

- 导入外部模块（例如 Java 类）和对象以在表达式中使用。
- 初始化并访问规则集、规则块或规则变量。
- 通过 act_event 变量，访问规则当前正在处理的事件。
- 通过 act_eventCount 变量，访问规则已接受的事件的数量。
- 通过 act_eventList 变量，访问规则已接受的事件的列表。这包括访问事件的各属性的能力，以及在序列规则中按每个事件的别名对其进行访问的能力。
- 通过 act_lib 变量，访问包含了下列能力的方法: 获取并设置变量以及控制通过规则集的事件流。
- 通过 act_location 变量，访问表达式在规则层次结构中的位置。
- 通过 act_nodeName 变量，访问节点的标准名称。
- 通过 act_threshold 变量，访问阈值规则的已定义阈值。

导入和访问外部模块和对象

本示例说明如何使外部代码（例如 Java 类）和外部对象可由表达式访问。外部对象是应用程序创建用来与表达式进行通信的对象。

从表达式访问外部代码之前，必须使该代码可由表达式访问。

导入是特定于编程语言的方式，用于使外部代码可由表达式访问。<import> 元素中包含特殊类型的表达式，该表达式指定要导入供规则内的其他表达式使用的外部模块（例如 Java 类）。导入可在规则集或规则块级别定义。

以下 <import> 元素中包含使用 Java 编程语言编写的表达式，该表达式导入 StaticHelper 类和 Queue 类，这些类可在其他表达式中引用:

```
<import expressionLanguage="java">
 import com.ibm.act.sample.StaticHelper;
 import com.ibm.act.test.Queue;
</import>
```

虽然在导入语句中可以不使用完整的类名，但为了避免很长的编译时间，您应当指定完整的名称。例如，应将 Java 类指定为 com.ibm.act.sample.StaticHelper，而不是 com.ibm.act.sample.* 或 com.ibm.act.*。

访问静态方法

以下示例说明在导入 StaticHelper 类之后，规则响应操作中的表达式如何引用该类:

```
<onDetection>
 <action expressionLanguage="java">
  StaticHelper.pageAdministrator("Too many login attempts for " + act_event.getAttribute("userID"));
 </action>
</onDetection>
```

访问对象的实例方法

以下示例说明在导入 Queue 类之后，规则响应操作中的表达式如何引用该类。此示例获取名为 OutputQueueOne 并且类型为 Queue 的外部对象，并用于将事件放置到特定队列中。

```
<onDetection>
 <action expressionLanguage="java">
  Queue myQueue = (Queue)act_lib.getExternalContext("OutputQueueOne");
  myQueue.enqueue(act_event);
 </action>
</onDetection>
```

初始化和访问变量

本示例说明如何初始化和访问规则集、规则块或规则变量。

变量可在规则集、规则块或规则级别定义。变量必须使用初始化表达式进行初始化之后才能访问。以下表达式初始化 2 个变量，分别名为 hostsList 和 hostsString:

```
<variable name="hostsList" dataType="java.util.ArrayList">
  <varInitializer expressionLanguage="java">
    return new ArrayList();
  </varInitializer>
</variable>
<variable name="hostsString" dataType="java.lang.String">
  <varInitializer expressionLanguage="java">
    return new String();
  </varInitializer>
</variable>
```

所有的变量都通过表达式访问。以下示例说明如何通过规则响应操作中的表达式来访问变量 hostsList 和 hostsString，这些变量已在先前示例中初始化。在以下示例中，将修改 hostsList，并为 hostsString 指定新值。

```
<onNextEvent>
 <action expressionLanguage="java">
   String hostname = act_event.getStringAttribute("hostname");
    ArrayList hostsList = (ArrayList)act_lib.getVariable("hostsList");
    hostsList.add(hostname);
    String hostsString = act_lib.getStringVariable("hostsString");
    String newHostString = hostsString + ", " + hostname);
    act_lib.setStringVariable("hostsString", newHostsString);
 </action>
</onNextEvent>
```

访问与事件相关的信息

以下示例说明如何通过活动关联技术提供的变量来访问与事件相关的信息。

访问当前事件的示例:
以下代码显示如何使用 act_event 变量来获取事件的主机名属性:

```
act_event.getAttribute("hostname");
```

根据索引通过事件列表访问事件的示例:
以下代码显示如何使用 act_eventList 变量来获取事件列表中的第一个事件:

```
act_eventList.get(0);
```

根据别名通过事件列表访问事件的示例:
与其他规则类型不同，序列规则允许多个事件选择器，它实际上最少需要 2 个事件选择器。<eventSelector> 元素中的别名属性仅在序列规则中有效，且该属性唯一地为序列规则中的特定事件选择器选择的事件命名。在过滤谓词或操作内的表达式中，可使用 act_eventList 变量来通过某个事件的别名在序列规则中访问该事件。

以下代码显示了序列规则的 2 个事件选择器。别名为 TECevent 和 WASevent。

```
<eventSelector alias="TECevent">
 <eventType type="serverStatus"/>
 <filteringPredicate expressionLanguage="java">
  return act_event.getStringAttribute("source").equals("TEC");
 </filteringPredicate>
</eventSelector>
<eventSelector alias="WASevent">
 <eventType type="serverStatus"/>
 <filteringPredicate expressionLanguage="java">
  return act_event.getStringAttribute("source").equals("WAS");
 </filteringPredicate>
</eventSelector>
```

以下代码显示如何使用 act_eventList 变量来获取已由第一个事件选择器接受的事件（名为 TECevent）：

```
act_eventList.get("TECevent");
```

编写表达式的最佳实践

本信息包含有效编写表达式的一些最佳实践、技巧与提示。

- 为了便于理解，本信息中提供的多数表达式示例都在 XML 构造中直接包含了 Java 代码。但是，创建规则时，最好使用外部模块来包含 Java 代码并将这些外部模块作为表达式的一部分进行调用。

  还可以在规则构建器中使用或编辑现有片段或者创建新片段，来为调用外部模块提供代码。片段为可在表达式中使用的源代码的引用。在规则构建器中，可通过“片段”视图使用片段。

  通过此方法，可在您所选择的集成开发环境（IDE）中完成设计、开发、编辑、测试和调试 Java 代码的任务，并作为开发流程的常规部分对这些任务进行控制。

- 要避免表达式代码被解析为 XML，请将该代码包括在 CDATA 部分中，其中 <![CDATA[直接加在该代码之前，]]> 紧随该代码之后，如以下示例所示:

  ```
  <onTimeOut>
  <action expressionLanguage="java">
  <![CDATA[
    IEvent firstEvent = act_eventList.get(0);
  ```

```
  System.out.println("Expired Item: " + firstEvent.getAttribute("sourceComponentId.location"));
]]>
</action>
</onTimeOut>
```

XML 解析器将忽略 CDATA 部分中的所有内容。

- 如果在表达式中使用 act_lib.getExternalContext() 方法，请不要在规则集或规则块变量中存储该方法返回的对象。原因是应用程序可能更改对象实例的引用，但关联的规则集或规则块变量却不更新。
- 如果在 <action> 元素内的表达式中使用返回语句（`return;`），且分别为规则响应或生命周期操作编写了其他 <action> 元素，则代码执行将在返回语句的位置结束并在下一个 <action> 元素中的表达式中再次开始。
- 规则管理和其他活动关联技术引擎方法不能从规则响应或生命周期操作中进行调用。

相关信息

第 18 页的『导入和访问外部模块和对象』
本示例说明如何使外部代码（例如 Java 类）和外部对象可由表达式访问。外部对象是应用程序创建用来与表达式进行通信的对象。

第 37 页的『将片段包含在规则内的表达式中』
在规则内的表达式中，您可以从“Eclipse Workbench”中的“片段”视图包含片段。

变量

在规则语言中，某些变量用于存储涉及不同事件或规则的事件相关信息。随后可通过规则内的表达式来访问这些与事件相关的信息。某些类型的变量由规则编写者定义，而其他类型的变量则由活动关联技术提供。某些类型可在表达式中直接访问，而其他类型则只能通过由活动关联技术提供的方法来访问。

在 <variable> 元素中定义并通过方法访问的变量

您可在规则、规则块或规则集的 <variable> 元素中定义变量。随后可使用以下某种方法在表达式中访问该变量:

- getVariable() 方法或 get*javatype*Variable() 方法之一
- setVariable() 方法或 set*javatype*Variable() 方法之一

例如，如果在规则的 <variable> 元素中定义变量 *rule_writer_variable*，则可使用以下代码访问该变量:

```
int sample_variable = act_lib.getIntVariable("rule_writer_variable");
```

由活动关联技术提供并在表达式中直接访问的变量

以下变量由活动关联技术提供。可在表达式中直接插入这些变量。

- act_event
- act_eventList
- act_lib

例如，可使用以下代码来访问 act_event 变量以获取事件的主机名属性:

```
act_event.getAttribute("hostname");
```

由活动关联技术提供并通过方法访问的变量

以下变量由活动关联技术提供。可通过使用 getVariable() 方法或 get*javatype*Variable() 方法之一在表达式中访问这些变量。

- act_eventCount
- act_location
- act_nodeName
- act_threshold

例如，可使用以下代码来访问 act_eventCount 变量:

```
int eventcount_integer = act_lib.getIntVariable(IACTLibrary.EVENTCOUNT);
```

表 3 显示 IACTLibrary 接口为这些变量提供的常量。在代码中，为确保在编译期间而不是在运行期间发现任何拼写错误或字符错误，请始终使用表示这些变量的常量，而不是使用这些变量本身。例如，使用 `act_lib.getIntVariable(IACTLibrary.EVENTCOUNT);` 而不是 `act_lib.getIntVariable("act_eventCount");`。

表 3. 具有关联常量的变量

变量	关联常量
act_eventCount	EVENTCOUNT
act_location	LOCATION
act_nodeName	NODENAME
act_threshold	THRESHOLD

相关参考

第 104 页的『 variable 元素 』

<variable> 元素定义变量，并按可由表达式引用的格式来包含信息。变量可在规则集、规则块或规则级别定义。

活动关联技术变量的数据类型

由活动关联技术提供的变量具有不同的数据类型。

表 4 显示了这些变量的数据类型。

表 4. 活动关联技术变量的数据类型

变量	数据类型
act_ event	由 com.ibm.correlation.IEvent 接口定义的类型
act_ eventCount	int
act_ eventList	由 com.ibm.correlation.IEventList 接口定义的类型
act_ lib	由 com.ibm.correlation.IACTLibrary 接口定义的类型
act_ location	java.lang.String
act_ nodeName	java.lang.String
act_ threshold	因为该变量是阈值规则的 <computedThreshold> 或 <eventCountThreshold> 元素上阈值属性的值，所以数据类型必须与阈值属性值的数据类型相同。

变量有效的表达式上下文

由活动关联技术提供的变量仅在特定表达式上下文中有效。

表 5 列出这些变量有效的表达式上下文。对于在各自的表达式上下文中有效的每个变量，该表会在对应的列上显示一个 X。应用于这些变量的其他用法限制将在对应变量的主题中进行描述。

表 5. 变量有效的表达式上下文

表达式上下文	act_ event	act_ eventCount	act_ eventList	act_ lib	act_ location	act_ nodeName	act_ threshold
<onActivation> 中的 <action>	X			X	X	X	X
<onDeactivation> 中的 <action>	X	X	X	X	X	X	X
<onDetection> 中的 <action>	X	X	X	X	X	X	X
<onLoad> 中的 <action>				X	X	X	X
<onNextEvent> 中的 <action>	X	X	X	X	X	X	
<onTimeOut> 中的 <action>		X	X	X	X	X	X
<onTimeWindowComplete> 中的 <action>		X	X	X	X	X	
<onUnload> 中的 <action>				X	X	X	X
<booleanThreshold>	X	X	X	X	X	X	
<computedThreshold>	X	X	X	X	X	X	X
<computedValue>	X			X	X	X	
<computeFunction>	X	X	X	X	X	X	
<filteringPredicate>	X	X	X	X	X	X	X
规则的 <varInitializer>				X	X	X	X

act_event 变量

act_event 变量提供对方法的访问，这些方法应用于当前事件。

详细信息

由于定时器规则不处理事件，因此定时器规则中的 act_event 变量仅应用于激活或停用规则的事件。

仅当事件激活或停用规则时，才在 <onActivation> 和 <onDeactivation> 操作内应用 act_event 变量。否则，此变量为空。

编码示例

以下代码访问 act_event 变量来获取事件的主机名属性:

```
String host = act_event.getStringAttribute("hostname");
```

可访问的方法

act_event 变量提供访问的方法在 IEvent 接口中定义，如表 6 中所示。

表 6. 具有相应方法和 Javadoc 方法描述位置的 IEvent 接口

接口	方法	Javadoc 方法描述的位置
IEvent	• get • getAttribute • getBooleanAttribute • getByteAttribute • getShortAttribute • getIntAttribute • getLongAttribute • getFloatAttribute • getDoubleAttribute • getStringAttribute • set • getTimeStamp • setTimeStamp • getType • getOriginal	com.ibm.correlation.IEvent

act_eventCount 变量

act_eventCount 变量是一个整数，它等于规则已接受事件的数量。

详细信息

对于重复规则，act_eventCount 变量的值为已接受事件（包括原始事件和所有重复事件）的总数。对于所有其他规则类型，该值与事件列表的大小相同，事件列表可使用 act_eventList.size() 方法通过 act_eventList 变量获取。

由于定时器规则不处理事件，因此 act_eventCount 和 act_eventList 变量在定时器规则中无效。

如果使用组密钥定义规则，则 act_eventCount、act_eventList 和 act_threshold 变量在以下表达式上下文中无效:

- 生命周期操作
- <activateOnEvent> 中的 <filteringPredicate>，或 <activationInterval> 中的 <deactivateOnEvent>
- <computedValue>

这是由于在这种情况下，规则变量仅应用于规则实例，而在这些表达式运行时不存在规则实例。

编码示例

以下代码访问 act_lib 变量来获取规则已接受的事件的数量:

```
int eventCt = act_lib.getIntVariable(IACTLibrary.EVENTCOUNT);
```

act_eventList 变量

act_eventList 变量提供对方法的访问，这些方法应用于规则的已接受事件的列表。

详细信息

由于过滤规则为无状态规，而重复规则仅保留第一个已分析的事件，因此过滤规则和重复规则的列表中的事件始终不超过一个。

由于定时器规则不处理事件，因此 act_eventCount 和 act_eventList 变量在定时器规则中无效。

如果使用组密钥定义规则，则 act_eventCount、act_eventList 和 act_threshold 变量在以下表达式上下文中无效:

- 生命周期操作
- <activateOnEvent> 中的 <filteringPredicate>，或 <activationInterval> 中的 <deactivateOnEvent>
- <computedValue>

这是由于在这种情况下，规则变量仅应用于规则实例，而在这些表达式运行时不存在规则实例。

编码示例

以下代码访问 act_eventList 变量来获取事件列表中的第二个事件:

```
IEvent second_event = act_eventList.get(1);
```

可访问的方法

act_eventList 变量提供访问的方法在 IEventList 接口中定义，如表 7 中所示。

表 7. 具有相应方法和 Javadoc 方法描述位置的 IEventList 接口

接口	方法	Javadoc 方法描述的位置
IEventList	• get • size • isEmpty • listIterator	com.ibm.correlation.IEventList

act_lib 变量

act_lib 变量提供对活动关联技术中库方法的访问权。

详细信息

act_lib 变量可访问的方法根据规则语言元素（包含使用了该变量的表达式）的不同而有所不同。请参阅 第 26 页的表 8。

表 8. act_lib 变量根据其包含表达式的上下文可访问的方法

表达式上下文	IACTLibrary 方法	IExitableActionLibrary 方法	IActionLibrary 方法
<onActivation> 中的 <action>	X		
<onDeactivation> 中的 <action>	X		
<onDetection> 中的 <action>	X	X	X
<onLoad> 中的 <action>	X		
<onNextEvent> 中的 <action>	X	X	X
<onTimeOut> 中的 <action>	X		X
<onTimeWindowComplete> 中的 <action>	X		X
<onUnload> 中的 <action>	X		
<booleanThreshold>	X		
<computedThreshold>	X		
<computedValue>	X		
<computeFunction>	X		
<filteringPredicate>	X		
<varInitializer>	X		

编码示例

以下代码访问 act_lib 变量来调用退出规则集的方法:

```
act_lib.exitRuleSet();
```

可访问的方法

act_lib 变量提供访问的方法在 IACTLibrary、IExitableActionLibrary 和 IActionLibrary 接口中进行定义，如 第 27 页的表 9 中所示。

表 9. 接口、相应方法以及 Javadoc 方法描述位置

接口	方法	Javadoc 方法描述的位置
IACTLibrary	• trace • getVariable • getBooleanVariable • getShortVariable • getIntVariable • getLongVariable • getFloatVariable • getDoubleVariable • getStringVariable • setVariable • setBooleanVariable • setShortVariable • setIntVariable • setLongVariable • setFloatVariable • setDoubleVariable • setStringVariable • getExternalContext	com.ibm.correlation.IACTLibrary
IActionLibrary	• forward • forwardOnCompletion • activate • deactivate 在 IACTLibrary 接口中定义的方法在 IActionLibrary 接口中同样可用。	com.ibm.correlation.IActionLibrary
IExitableActionLibrary	• exitRuleSet • exitRuleBlock 在 IACTLibrary 和 IActionLibrary 接口中定义的方法在 IExitableActionLibrary 接口中同样可用。	com.ibm.correlation.IExitableActionLibrary

act_location 变量

act_location 变量为标识表达式位置（位于规则层次结构中）的字符串。

详细信息

该位置为标准名称，指示表达式在规则层次结构中的位置。其格式为 *identifier.identifier....*，其中的每个 *identifier* 为以下某个值:

- XML 元素的名称属性的值，该元素位于各自的层次结构中。

- 对于在规则块或规则中多次出现且无名称属性的元素：包含表达式的 XML 元素，后面跟随带方括号的索引号。该索引号指示表达式在其包含元素中的位置。指定索引号的计数器开始自 0 而并不是 1。因此，如果元素包含于第 3 个 <action> 元素中，则索引号显示为 action[2]。

这些标识按照降序排列，从最高级别规则块到包含表达式的最低级别元素。

编码示例

以下代码访问 act_lib 变量来获取表达式的位置：

```
String location = act_lib.getStringVariable(IACTLibrary.LOCATION);
```

变量返回的位置的示例

以下值为从 act_location 变量返回的位置的示例。

ruleBlockA.ruleA.eventSelector[3].filteringPredicate

该表达式包含于以下位置中：

- 名称属性值为 ruleBlockA 的规则块
- 名称属性值为 ruleA 的规则
- 第 4 个 <eventSelector> 元素
- <filteringPredicate> 元素

ruleBlockA.ruleA.onDetection.action[5]

该表达式包含于以下位置中：

- 名称属性值为 ruleBlockA 的规则块
- 名称属性值为 ruleA 的规则
- <onDetection> 元素
- 第 6 个 <action> 元素

ruleBlockA.ruleA.variableA.varInitializer

该表达式包含于以下位置中：

- 名称属性值为 ruleBlockA 的规则块
- 名称属性值为 ruleA 的规则
- 名称属性值为 variableA 的变量
- <varInitializer> 元素

act_nodeName 变量

act_nodeName 变量为标识节点标准名称的字符串。

详细信息

在活动关联技术中，节点是规则层次结构中的对象，可单独在规则集添加、除去或替换。具体而言，以下对象为节点：规则、规则块、规则块变量和规则集变量。由于无法在规则级别以下以个别而独立的方式对某个对象进行操作，因此规则变量不是节点。

如果某个规则的名称属性值为 rule1，它位于名称属性值为 ruleBlockA 的规则块中，则该规则的标准节点名为 ruleBlockA.rule1。由于规则在规则集中按层次结构组织，因此在标准节点名中使用句点（.）来表示下行到较低级别节点。

编码示例

以下代码访问 act_nodeName 变量来获取节点的标准名称:

```
String nodeName = act_lib.getStringVariable(IACTLibrary.NODENAME);
```

act_threshold 变量

act_threshold 变量是阈值规则的 <computedThreshold> 或 <eventCountThreshold> 元素上阈值属性的值，该值是已定义的阈值。

详细信息

act_threshold 变量仅在阈值规则中有效。

如果使用组密钥定义规则，则 act_eventCount、act_eventList 和 act_threshold 变量在以下表达式上下文中无效:

- 生命周期操作
- <activateOnEvent> 中的 <filteringPredicate>，或 <activationInterval> 中的 <deactivateOnEvent>
- <computedValue>

这是由于在这种情况下，规则变量仅应用于规则实例，而在这些表达式运行时不存在规则实例。

编码示例

以下代码访问 act_lib 变量来获取已定义的阈值:

```
int threshold = act_lib.getIntVariable(IACTLibrary.THRESHOLD);
```

通过规则集的事件流

事件流按照所编写的规则块和规则的顺序通过规则集。当活动关联技术引擎接收到事件时，该引擎将确定事件类型，并确定使用此事件类型进行规则激活、事件处理或规则停用的规则。

规则使用事件的方式

使用事件的每个规则首先确定该事件是否符合规则激活、事件处理或规则停用的所有指定条件。如果符合，则规则将执行以下操作:

对于规则激活

将运行规则的 <onActivation> 元素中的操作（如果已编码）。

对于事件处理

规则将处理事件。当规则模式匹配时，将运行规则响应操作（如果已编码）。在某些情况下，规则响应操作可执行以下操作:

- 该操作可导致事件跳过规则块或规则集其余部分的处理。
- 该操作可向另一个规则或规则块发送新事件或现有事件以进行处理。

对于规则停用

将运行规则的 <onDeactivation> 元素中的操作（如果已编码）。

可影响事件流的方法

活动关联技术提供以下可调用方法来影响通过规则集的事件流。这些方法可通过 act_lib 变量访问。

exitRuleSet

该方法指定当前事件不再由规则集中的任何其他规则处理。

exitRuleBlock

该方法指定当前事件不再由当前规则块中或该规则块包含的所有规则块中的任何其他规则处理。不过，当前规则块范围之外的其他规则处理将会处理该事件。

forward

该方法指定即使当前规则未完成其处理，仍将事件发送至其他规则和规则块。随后，其他每个规则和规则块都将对事件进行完整的处理，之后再将事件返回至调用 forward 方法的规则。

forwardOnCompletion

该方法指定在当前规则完成其处理后，将事件发送至其他规则和规则块。

第 3 章 规则编写概述

在编写规则以关联事件之前，必须了解和计划事件相关，并设计规则。您可以使用活动关联技术规则构建器来编写规则和编译规则集。

活动关联技术规则构建器用于编写规则的 GUI。它包含联机帮助。在规则构建器中，所生成的规则集文件是文件类型为 actl 的 XML 文件。

活动关联技术不提供变更管理或版本控制系统。

事件相关规划

事件相关规划包括了解何为事件相关以及如何将其应用于您的应用程序中。

请确保了解以下概念:

- 第 3 页的第 1 章，『简介』和第 5 页的第 2 章，『规则语言概述』中的信息
- 应用程序处理的事件

每个应用程序均可处理一组不同的事件，如以下示例所述:

保险业务示例

在保险业务中，可生成并关联用于跟踪通过索赔流程的工作流的事件，以确定业务流程是否及时完成。

销售示例

在不同类型的业务中，可定期总结和报告销售结果，并与目标比较以表明某个时间段内销售目标的完成状态。

IT 环境示例

在 IT 环境中，关键任务系统可每分钟生成一个事件以表明数据库服务器运行正常。可编写关联规则以监视这些脉动信号事件的接收，并在未接收到期望的脉动信号事件时执行某些规则响应操作。

此外，还应该了解应用程序处理的事件的格式。活动关联技术提供 Java 类和方法以访问正由活动关联技术引擎处理的事件中的数据。然而，如果要使用这些类和方法以访问或更改正被处理的事件，则对底层的事件对象的基本了解十分重要。

要规划事件相关，请执行以下步骤:

1. 确定应用程序中要进行关联的事件。
2. 确定用于关联事件的规则模式。

   规则模式代表特定的事件相关情境，并可用于关联在某种程度上导致了该情境的事件。请考虑应用程序处理的事件如何与活动关联技术规则语言定义的规则模式相关联。这可有助于确定需要使用的规则模式。

   请始终使用最适合事件相关情境的模式。例如，如果想要规则检测特定的事件序列，请不要编写代码将序列模式行为包含在过滤规则的规则响应操作中。而应该使用序列模式来创建序列规则。

3. 确定要使用的每个规则模式的构造。

以下信息总结了规则语言中的主要构造（尽管每个构造的详细信息对于规则模式是唯一的）。本信息的组织方式与通过规则构建器 GUI 所呈现的方式大致相同:

特性 规则特性的定义，包括规则名称、描述和模式。有关详细信息，请参阅以下主题:

- 第 67 页的『collectionRule 元素』
- 第 69 页的『computationRule 元素』
- 第 75 页的『duplicateRule 元素』
- 第 82 页的『filterRule 元素』
- 第 95 页的『sequenceRule 元素』
- 第 100 页的『thresholdRule 元素』
- 第 102 页的『timerRule 元素』

变量 规则变量的定义，包括每个变量的名称、类型、描述和初始化表达式。有关详细信息，请参阅第 104 页的『variable 元素』。

事件选择

条件的定义，该条件确定由规则接受以进行处理的事件。有关详细信息，请参阅第 79 页的『eventSelector 元素』。

组密钥 组密钥的定义，组密钥可用于指定规则针对每组共享公共特性的事件创建不同的规则实例（或其副本）。有关详细信息，请参阅第 83 页的『groupingKey 元素』。

模式指定

时间段的指定，在该时间段内，全状态规则正在进行处理，以将其模式与某些全状态规则模式的一些唯一方面的定义相匹配。有关详细信息，请参阅第 103 页的『timeWindow 元素』。

对于计算规则，这将包括计算的定义以应用于收集的事件。有关详细信息，请参阅第 72 页的『computeFunction 元素』。

对于阈值规则，这将包括阈值类型的定义以及特定于阈值类型的其他信息。有关详细信息，请参阅以下主题:

- 第 66 页的『booleanThreshold 元素』
- 第 70 页的『computedThreshold 元素』
- 第 77 页的『eventCountThreshold 元素』

规则响应

规则完成其处理后要采取的操作的定义。

有关详细信息，请参阅以下主题:

- 对于重复规则、过滤规则、序列规则或阈值规则: 第 89 页的『onDetection 元素』
- 对于重复规则: 第 90 页的『onNextEvent 元素』
- 对于序列规则或阈值规则: 第 90 页的『onTimeOut 元素』
- 对于收集规则、计算规则、重复规则或定时器规则: 第 91 页的『onTimeWindowComplete 元素』

激活时间间隔

关于规则何时处于活动状态、何时处于不活动状态的定义。有关详细信息，请参阅第 61 页的『activationInterval 元素』。

生命周期

规则生命周期的以下 4 个主要阶段要执行的操作（如果存在）的定义：装入、激活、停用和卸装。通常情况下，这些操作无需定义。有关详细信息，请参阅第 86 页的『lifeCycleActions 元素』。

4. 标识要在规则表达式内调用的 Java 方法和关联片段。规则编写者不必在规则表达式内编写大量 Java 代码，而只需使用 Java 方法来调用外部模块。这些外部模块可由嵌入活动关联技术的应用程序提供，也可由规则编写者根据需要进行创建。还应标识与每个 Java 方法关联的片段。有关更多信息，请参阅第 20 页的『编写表达式的最佳实践』。

继续进行『设计规则以关联事件』。

设计规则以关联事件

计划事件相关后，必须设计规则以关联事件。

首先，请完成第 31 页的『事件相关规划』。

要设计规则，请执行以下步骤:

1. 设计规则集内规则和规则块的组织。
2. 设计定义不同变量所使用的级别；例如，规则集、规则块或规则的级别。
3. 根据规则模式来设计每个规则的元素。

   此步骤将使用您在计划阶段所确定的每个规则模式的结构。
4. 设计规则之间的交互，特别是关于转发事件以及跳过重复事件的规则集处理。

   有关更多详细信息，请参阅第 29 页的『通过规则集的事件流』。
5. 设计、开发和测试您先前创建用来从表达式中调用的所有 Java 方法和关联片段。

继续『规则构建器入门』。

规则构建器入门

设计了用于关联事件的规则之后，您可以使用活动关联技术规则构建器来编写规则。

以下步骤是使用规则构建器编写规则的主要步骤。

1. 打开“Eclipse Workbench”。
2. 在“Eclipse Workbench”中设置透视图。
3. 为活动关联技术设置首选项或接受缺省首选项。
4. 创建用于存储规则集文件的项目或使用现有的项目。
5. 创建规则集文件，并将其存储于所选择的项目中。
6. 在规则集中至少创建一个规则块。此外，您还可以在规则集中创建其他规则块，以及在规则块中创建规则块。
7. 在规则块中创建规则。

8. 验证规则集。
9. 编译规则集。
10. 根据需要更新规则集。

在规则内的表达式中，您还可以从“Eclipse Workbench”中的“片段”视图包含片段。

在“Eclipse Workbench”中设置透视图

在您进行任何其他操作之前，应先在“Eclipse Workbench”中设置透视图。本主题描述如何打开“规则构建器”透视图。

首先，打开“Eclipse Workbench”（如果尚未打开）。

虽然您可以使用除“规则构建器”透视图以外的其他透视图，但是根据所选的透视图的不同，执行各种任务的步骤也会有些许差异。

要打开“规则构建器”透视图，请执行以下步骤:

1. 在“Workbench”中，单击**窗口 → 打开透视图 → 其他...**。
2. 单击**规则构建器**，然后单击**确定**。 然后将显示“规则构建器”透视图。

继续『设置首选项』。

设置首选项

在创建规则集文件之前，应先确保活动关联技术的首选项已在“Eclipse Workbench”中正确设置。本主题描述如何设置这些首选项。

首先，打开“Eclipse Workbench”（如果尚未打开）。

要设置活动关联技术的首选项，请执行以下步骤:

1. 在工作台中，单击**窗口 → 首选项...**。
2. 单击**活动关联技术**，并在活动关联技术页面中指定是否要将 『act』 添加为规则构建器中新建规则和规则块的前缀。 缺省情况下，『act』 *没有*添加为前缀。
3. 展开**活动关联技术**。 根据应用程序，可能会显示以下补充项。单击这些项以设置关联首选项。

项	关联首选项
通用基础事件定义提供程序	可指定为一个或多个符合“通用基础事件”规范的事件类型（包括给定事件类型可包含的属性的名称和类型）提供结构的 XML 文件。然后，创建规则时可使用这些事件类型和属性。
编译器	可指定以下主要项: • 是否应保存已编译的规则集 • 已编译规则集的文件类型 • 在编译时使用的代码的类路径 缺省情况下，已编译的规则集将保存并显示在“导航器”视图中；该规则集的文件类型为 `acts`，这表明编译器输出为经过序列化的规则集。

继续『创建用于存储规则集文件的项目』。

创建用于存储规则集文件的项目

在“Eclipse Workbench”中，当创建规则集文件时，必须指定用于存储该文件的项目。因此，在创建规则集文件之前必须创建项目，或者使用现有的项目。本主题描述在“Eclipse Workbench”中创建项目的方法。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

要在工作台中创建简单项目，请执行以下步骤:

1. 单击**文件 → 新建 → 项目...**。
2. 在“新建项目”向导中，单击**简单 → 项目**，然后单击**下一步**。
3. 在**项目名称**字段中，输入项目的唯一名称。同时，使用项目的缺省位置，或者选择其他位置。
4. 单击**完成**。 随后，新项目将显示在“规则构建器”透视图的“导航器”视图中。

继续『创建规则集』。

创建规则集

本主题描述如何创建规则集。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

要创建规则集文件，请执行以下步骤:

1. 单击**文件 → 新建 → 规则集文件**。
2. 在“新建规则集文件”向导中，单击与要在其中存储规则集文件的项目关联的文件夹的名称。该项目可能是在『创建用于存储规则集文件的项目』中创建的项目。 文件夹名称随后将显示在第一个字段中。
3. 在**文件名**字段中，输入规则集文件的名称。 文件类型必须为 actl。
4. 单击**下一步** 。
5. 为规则集输入唯一的名称，并输入描述（可选）。如果不希望使用 **XML 编码**字段的缺省值，您还可以指定规则集文件（XML 文件）的编码样式。
6. 单击**完成**。 随后将在“导航器”视图的项目文件夹中显示规则集文件。由于文件在保存时已进行验证，因此该文件旁边将显示『已验证』字样。该规则集文件还将显示在“大纲”视图中。

继续『创建规则块』。

创建规则块

本主题描述如何在规则集内或在其他的规则块内创建规则块。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

如果是初次创建规则集，则必须先在规则集中至少创建一个规则块，然后才能创建任何规则。创建第一个规则块之后，可以创建其他规则块，包括在规则块中创建规则块。

要创建规则块，请执行以下步骤:

1. 在“导航器”视图中，双击想要更新的规则集文件的名称。 该文件随后在“大纲”视图中打开。在“大纲”视图中单击规则集时，该规则集的当前信息将显示在编辑器区域中。
2. 在“大纲”视图中，右键单击该规则集。
3. 单击**新建子代 → 规则块**。 “大纲”视图中的规则集内随后将显示规则块，且该规则块的当前信息将在编辑器区域中显示。

现在您可以使用以下方法来创建其他规则块:

- 要创建与现有规则块同级的规则块，请右键单击现有的规则块，然后单击**新建同代 → 规则块**。
- 要在现有规则块内创建规则块，请右键单击现有规则块，然后单击**新建子代 → 规则块**。

同时，您还可以使用编辑器来更新规则集和每个规则块的信息。继续『创建规则』。

创建规则

本主题描述如何创建规则。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

必须在规则块中创建规则。要创建规则，请执行以下步骤:

1. 在“大纲”视图中，右键单击要更新的规则块。
2. 单击**新建子代**并输入要创建的规则。 “大纲”视图中的规则块内将随后显示规则，并且编辑器区域中将显示规则的当前信息。

您可以使用相同方式向规则块添加其他规则。同时，您还可以使用编辑器来更新每个规则的信息。继续『验证规则集』。

验证规则集

本主题描述如何在编译规则集之前对其进行验证。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

要验证规则集，请执行以下步骤:

1. 在“大纲”视图中，右键单击规则、规则块或规则集。
2. 单击**验证规则集**。 验证完成后，将会显示一个消息窗口，指示是否存在问题。如果验证成功完成，则将在“导航器”视图的规则集文件名右边显示『已验证』字样。

验证成功完成后，继续『编译规则集』。

编译规则集

本主题描述如何编译规则集。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

在“导航器”或“大纲”视图中，右键单击想要编译的规则集，然后单击**编译规则集**。“问题”视图中将显示所有的编译错误。

缺省情况下，如果不存在编译错误，则已编译的规则集将显示在“导航器”视图中；该规则集的缺省文件类型为 acts，这表明它是经过序列化的规则集。在“导航器”视图中，还将在规则集文件名的右边显示『已编译』字样。

更新规则集

本主题描述如何更新规则集。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

在“大纲”视图中，单击要更新的项（规则、规则块或规则集）。编辑器区域中随后将显示对应于该项的当前信息，您可以使用编辑器来更新该信息。

要创建与现有的规则块或规则同级的规则块或规则，请执行以下步骤:

1. 右键单击现有的规则块或规则。
2. 单击**新建同代**和要添加的项（规则块或规则类型）。

要在现有的规则块中创建规则块或规则，请执行以下步骤:

1. 右键单击现有的规则块。
2. 单击**新建子代**和要添加的项（规则块或规则类型）。

要访问“大纲”视图中的其他更新功能，请执行以下步骤:

1. 右键单击要更新的项（规则块或规则）。
2. 单击功能的菜单项；例如，**删除**、**复制** 或**粘贴**。

将片段包含在规则内的表达式中

在规则内的表达式中，您可以从“Eclipse Workbench”中的“片段”视图包含片段。

首先，打开“Eclipse Workbench”（如果尚未打开）。同样，打开“规则构建器”透视图。

“片段”视图显示在“规则构建器”透视图中。片段根据自身的功能被组织成类别。

要从“片段”视图包含片段，请执行以下步骤:

1. 单击片段类别来查看此类别中片段的名称。
2. 单击要包含在表达式中的片段。
3. 将片段拖入相应的**表达式**字段中。 代码放置在**表达式**字段中光标的位置上。如果代码要求规则编写者进行输入（例如特定消息文本或变量值），则在代码包含在表达式中之前将提示您提供该输入。

继续第 36 页的『验证规则集』。

第 2 部分 规则编写者参考大全

第 4 章 规则集组织摘要

本参考列出规则集、规则块和每种规则类型的所有语言元素。它可作为编写规则集的快速参考。

表 10 说明了每个语言元素后面的表示法的含义。*n* 代表次数为无限大。

表 10. 对定义语言元素出现次数的表示法的说明

表示法	含义
(0, 1)	0 表示语言元素为可选。1 表示如果已编写，则仅允许出现 1 次。
(0, *n*)	0 表示语言元素为可选。*n* 表示如果已编写，则允许出现多次。
(1, 1)	第一个 1 表示语言元素为必需。第二个 1 表示则仅允许出现 1 次。
(1, *n*)	1 表示语言元素为必需。*n* 表示允许出现多次。
(2, *n*)	2 表示语言元素必需出现 2 次。*n* 表示允许出现更多次数。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

规则集摘要

本摘要列出规则集的语言元素。

规则集元素

<ruleSet> 包含以下元素:

- <comment> (0, 1)
- <import> (0, n)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <ruleBlock> (0, n)

相关参考

『规则块摘要』
本摘要列出规则块的语言元素。

规则块摘要

本摘要列出规则块的语言元素。

规则块元素

<ruleBlock> 包含以下元素。

如已编写，则必须按显示的顺序编写 <comment>、 <import> 和 <variable> 元素。剩余元素可按任意顺序进行编写。

- <comment> (0, 1)
- <import> (0, n)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <ruleBlock> (0, n)
- <collectionRule> (0, n)
- <computationRule> (0, n)
- <duplicateRule> (0, n)
- <filterRule> (0, n)
- <sequenceRule> (0, n)
- <thresholdRule> (0, n)
- <timerRule> (0, n)

相关参考

『收集规则摘要』
本摘要列出收集规则的所有语言元素。

第 44 页的『计算规则摘要』
本摘要列出计算规则的所有语言元素。

第 45 页的『重复规则摘要』
本摘要列出重复规则的所有语言元素。

第 47 页的『过滤规则摘要』
本摘要列出过滤规则的所有语言元素。

第 48 页的『序列规则摘要』
本摘要列出序列规则的所有语言元素。

第 50 页的『阈值规则摘要』
本摘要列出阈值规则的所有语言元素。

第 51 页的『定时器规则摘要』
本摘要列出定时器规则的所有语言元素。

收集规则摘要

本摘要列出收集规则的所有语言元素。

规则元素

<collectionRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)

- <activationTime> (0, 1)
  - <start> (0, 1)
    - 以下三个元素之一（1, 1）：
      - <dateTime>
      - <whenLoaded>
      - <inactiveWhenLoaded>
  - <stop> (0, 1)
    - 以下三个元素之一（1, 1）：
      - <dateTime>
      - <never>
      - <after>
- <activateOnEvent> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <deactivateOnEvent> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <activationByGroupingKey> (0, 1)
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
    - <stopAfter> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)

- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
  - 以下三个元素，按任意顺序 (1, n):
    - <attributeAlias>

        - <eventAttribute> (2, n)
      - <attributeName>
      - <computedValue>
- <timeWindow> (1, 1)
  - 以下两个元素之一 (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

计算规则摘要

本摘要列出计算规则的所有语言元素。

规则元素

<computationRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <activationByGroupingKey> (0, 1)
    - <activateOnEvent> (0, 1)
      - <eventType> (0, n)

        - <filteringPredicate> (0, 1)
        - <stopAfter> (0, 1)
      - <deactivateOnEvent> (0, 1)
        - <eventType> (0, n)
        - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
  - 以下三个元素，按任意顺序 (1, n):
    - <attributeAlias>
      - <eventAttribute> (2, n)
    - <attributeName>
    - <computedValue>
- <computeFunction> (1, 1)
- <timeWindow> (1, 1)
  - 以下两个元素之一 (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

重复规则摘要

本摘要列出重复规则的所有语言元素。

规则元素

<duplicateRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)

- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <activationByGroupingKey> (0, 1)
    - <activateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
      - <stopAfter> (0, 1)
    - <deactivateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
  - 以下三个元素，按任意顺序 (1, n):

- <attributeAlias>
  - <eventAttribute> (2, n)
- <attributeName>
- <computedValue>

- <timeWindow> (1, 1)
  - 以下两个元素之一 (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onDetection> (0, 1)
  - <action> (0, n)
- <onNextEvent> (0, 1)
  - <action> (0, n)
- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

过滤规则摘要

本摘要列出过滤规则的所有语言元素。

规则元素

<filterRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)

- <eventType> (0, n)
- <filteringPredicate> (0, 1)

- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <onDetection> (0, 1)
  - <action> (0, n)

序列规则摘要

本摘要列出序列规则的所有语言元素。

规则元素

<sequenceRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)

- <filteringPredicate> (0, 1)

– <deactivateOnEvent> (0, 1)
- <eventType> (0, n)
- <filteringPredicate> (0, 1)

– <activationByGroupingKey> (0, 1)
- <activateOnEvent> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
  - <stopAfter> (0, 1)
- <deactivateOnEvent> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)

• <lifeCycleActions> (0, 1)
– <onLoad> (0, 1)
- <action> (0, n)

– <onActivation> (0, 1)
- <action> (0, n)

– <onDeactivation> (0, 1)
- <action> (0, n)

– <onUnload> (0, 1)
- <action> (0, n)

• <eventSelector> (2, n)
– <eventType> (0, n)
– <filteringPredicate> (0, 1)

• <groupingKey> (0, 1)
– 以下三个元素，按任意顺序 (1, n):
- <attributeAlias>
  - <eventAttribute> (2, n)
- <attributeName>
- <computedValue>

• <timeWindow> (1, 1)
– 以下两个元素之一 (1, 1):
- <timeInterval>
- <runUntilDeactivated>

• <onDetection> (0, 1)
– <action> (0, n)

• <onTimeOut> (0, 1)
– <action> (0, n)

阈值规则摘要

本摘要列出阈值规则的所有语言元素。

规则元素

<thresholdRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <activationByGroupingKey> (0, 1)
    - <activateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
      - <stopAfter> (0, 1)
    - <deactivateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)

    - <action> (0, n)
  – <onDeactivation> (0, 1)
    - <action> (0, n)
  – <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  – <eventType> (0, n)
  – <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
  – 以下三个元素，按任意顺序 (1, n):
    - <attributeAlias>
      - <eventAttribute> (2, n)
    - <attributeName>
    - <computedValue>
- 以下三个元素之一（1, 1）:
  – <booleanThreshold>
  – <computedThreshold>
  – <eventCountThreshold>
- <timeWindow> (1, 1)
  – 以下两个元素之一 (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onDetection> (0, 1)
  – <action> (0, n)
- <onTimeOut> (0, 1)
  – <action> (0, n)

定时器规则摘要

本摘要列出定时器规则的所有语言元素。

规则元素

<timerRule> 包含以下元素:

- <comment> (0, 1)
- <variable> (0, n)
  – <comment> (0, 1)
  – <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  – <activationTime> (0, 1)
    - <start> (0, 1)
      - 以下三个元素之一（1, 1）:

        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - 以下三个元素之一（1, 1）:
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <timeWindow> (1, 1)
  - 以下两个元素之一 (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

第 5 章 语言元素参考

本参考将描述活动关联技术规则语言的 XML 模式中的语言元素的详细信息。这些语言元素以字母顺序列出，且对每个元素可用的属性在该元素的主题中有所描述。

在 XML 和其他标记语言（例如 SGML 和 HTML）中，元素为由以下部分组成的基本单元：开始标记、结束标记、关联的属性和它们的值，以及开始标记与结束标记之间包含的任何文本。属性是在元素上编写的“名称 – 值”对，用于定义该元素的某个特性。属性具有数据类型，该数据类型标识其值中提供的信息的类型（例如，数字、文本或布尔值信息）。

在 XML 中，名称空间是统一资源标识（URI），该 URI 提供唯一的名称，以将模式中的元素与类型定义关联。 URI 指示哪个 XML 模式包含元素定义。名称空间是通过前缀字符串后跟冒号指定的。活动关联技术规则语言模式是在三个不同的文件中定义的，并使用以下 3 个名称空间:

xsd: 该名称空间表明语言元素在标准 XML 模式中定义，在 http://www.w3.org 中有述。

br: 该名称空间表明语言元素在活动关联技术基本规则集模式中定义，位于 com/ibm/correlation/ruleparser/xml/RuleSetBase.xsd 子目录中的 ACTparser.jar 文件。例如，`br:ruleSet` 引用 RuleSetBase.xsd 文件中定义的 ruleSet 元素。

act: 该名称空间表明语言元素在活动关联技术语言模式中定义，位于 com/ibm/correlation/ruleparser/xml/ACTL.xsd 子目录中的 ACTparser.jar 文件。例如，`act:ruleSet` 引用 ACTL.xsd 文件中定义的 ruleSet 元素。

在规则语言模式中，语言元素定义为元素或复杂类型，例如:

```
<xsd:element name="symbol" minOccurs="1" maxOccurs="unbounded"></element>
<xsd:complexType name="symbol"></complexType>
```

在此模式中，minOccurs 和 maxOccurs 属性为语言元素发生次数分别定义最小值和最大值。表 11 描述 minOccurs 和 maxOccurs 属性不同的值的含义。

表 11. 模式中用于定义语言元素发生次数的属性

属性	属性值	含义
minOccurs	0	该语言元素是可选的。
minOccurs	1	该语言元素必须至少出现一次。1 为 minOccurs 属性的缺省值。
minOccurs	2	该语言元素必须至少出现 2 次。
maxOccurs	1	该语言元素不得出现多次。1 为 maxOccurs 属性的缺省值。
maxOccurs	无限	该语言元素可出现任意多次。

action 元素

<action> 元素包含定义规则响应操作或生命周期操作的表达式。

详细信息

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<action> 具有以下属性:

表 12. <action> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是
name	标识操作。该标识可用于故障诊断用途，特别是在为特定规则响应或生命周期操作定义的所有 <action> 元素中它是唯一的名称时非常有用。	xsd:NMTOKEN	否

包含于

<action> 包含于以下元素中:

- <onActivation>
- <onDeactivation>
- <onDetection>
- <onLoad>
- <onNextEvent>
- <onTimeOut>
- <onTimeWindowComplete>
- <onUnload>

包含

<action> 不包含元素。

相关概念

第 17 页的『表达式』
表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

activateOnEvent 元素

<activateOnEvent> 元素定义可激活规则或（对于使用 <groupingKey> 元素定义的规则）规则实例的事件。

以下为选择事件的 3 种可能方式:

- 使用带有 <filteringPredicate> 元素的一个或多个 <eventType> 元素

- 使用不带有 <filteringPredicate> 元素的一个或多个 <eventType> 元素
- 使用不带有任何 <eventType> 元素的 <filteringPredicate> 元素

如果规则处于不活动状态且未对 <eventType> 或 <filteringPredicate> 元素进行编码，则将选择发生的任意事件。

不编写任何 <eventType> 元素可对系统性能造成负面影响。

假定要选择Audit Failure 类型的所有事件。可使用过滤谓词来进一步优化选择条件，以便仅包含事件属性具有特定值的事件。例如，可编写 <eventType> 元素以选择Audit Failure类型的所有事件，并编写 <filteringPredicate> 元素以仅选择主机名属性具有值 MyCriticalSystem）的那些事件。

属性

<activateOnEvent> 不具有属性。

包含于

<activateOnEvent> 包含于以下元素中:

- <activationInterval>
- <activationByGroupingKey>

包含

<activateOnEvent> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 13. <activateOnEvent> 元素中包含的元素

元素	必需或可选?
<eventType>	可选。允许出现 0 次或多次。
<filteringPredicate>	可选。允许出现 0 或 1 次。
<stopAfter>	仅当 <activateOnEvent> 元素包含于 <activationByGroupingKey> 元素中时，该元素才有效。 可选。允许出现 0 或 1 次。

activationByGroupingKey 元素

<activationByGroupingKey> 元素包含用于指定事件的元素，这些事件可激活和停用由 <groupingKey> 元素定义的规则实例。由于 <groupingKey> 元素对于过滤规则和定时器规则无效，因此 <activationByGroupingKey> 元素不适用于这些规则。

详细信息

由 <activationByGroupingKey> 元素提供的功能将用在那些可用于定义组密钥的规则中。该功能使您能够基于组密钥来控制规则实例的激活和停用。当编写 <activationByGroupingKey> 元素时，可基于 <activationByGroupingKey> 中的 <activateOnEvent> 和 <deactivateOnEvent> 条件对每个规则实例单独进行激活和停用。

以下示例将说明 <activationByGroupingKey> 元素在计算规则中的用法。

- 以下计算规则将接受 StockSharesTraded 类型的事件。这些事件表明了许多不同的公司正在进行交易的股票的数量。
- 组密钥为事件的 stockSymbol 属性。stockSymbol 属性的值为特定公司的名称。
- 事件的 sharesTraded 属性的值是对应公司（公司由 stockSymbol 属性的值来表明）已交易的股票的数量。
- 对于某家特定的公司，将创建报告以表明该公司在 10 分钟内已交易的股票的数量。然而，在计算规则可创建该报告之前，它必须接收到 StartReporting 类型的事件，并将对应公司的名称作为 stockSymbol 属性的值。

```
<computationRule name="StockReporter">
 <variable dataType="java.lang.Integer" name="totalSharesTraded">
  <varInitializer expressionLanguage="java">
   return new Integer(0);
  </varInitializer>
 </variable>

 <activationInterval>
  <activationTime>
   <start>
    <inactiveWhenLoaded/>
   </start>
  </activationTime>
  <activationByGroupingKey>
   <activateOnEvent>
    <eventType type="StartReporting"/>
   </activateOnEvent>
  </activationByGroupingKey>
 </activationInterval>

 <eventSelector>
  <eventType type="StockSharesTraded"/>
 </eventSelector>

 <groupingKey>
  <attributeName>stockSymbol</attributeName>
 </groupingKey>

 <computeFunction assignTo="totalSharesTraded" expressionLanguage="java">
  return new Integer(act_lib.getIntVariable("totalSharesTraded")
            + act_event.getIntAttribute("sharesTraded"));
 </computeFunction>

 <timeWindow>
  <timeInterval unit="ISO-8601" duration="PT10M"/>
 </timeWindow>

 <onTimeWindowComplete>
  <action expressionLanguage="java">
   StockReport.createReport(act_eventList.get(0).getStringAttribute("stockSymbol"),
                act_lib.getIntVariable("totalSharesTraded"));
  </action>
 </onTimeWindowComplete>
</computationRule>
```

属性

<activationByGroupingKey> 无属性。

包含于

<activationByGroupingKey> 包含于以下元素中:

- <activationInterval>

包含

<activationByGroupingKey> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 14. <activationByGroupingKey> 元素中包含的元素

元素	必需或可选?
<activateOnEvent>	可选。允许出现 0 或 1 次。
<deactivateOnEvent>	可选。允许出现 0 或 1 次。

<activationInterval> 元素与 <activationByGroupingKey> 元素之间的关系

<activateOnEvent> 元素和 <deactivateOnEvent> 元素均包含于以下两个元素中:

- <activationInterval>
- <activationByGroupingKey>，该元素同时包含在 <activationInterval> 中

根据当前规则活动，以及根据 <activationInterval> 和 <activationByGroupingKey> 元素中的 <activateOnEvent> 和 <deactivateOnEvent> 定义之间的相互作用，规则行为会有所不同。以下示例将说明这些定义如何相互作用。

在本示例中，将定义重复规则，以便在处于维护方式的系统上禁止事件，并在维护周期结束时提供关于被禁止事件数的摘要报告。

缺省情况下，定义了组密钥的规则允许处理所有的组密钥值。因此，当事件符合规则的事件选择条件时，所有的规则实例均处于活动状态，并已准备好根据组密钥的任一值来接受这些事件。该规则的激活时间间隔将与规则没有组密钥时相同，因为实际上，符合该规则的事件选择条件的所有事件都将得到处理。

在以下示例中，组密钥为 hostname，且 <activationInterval> 元素中的定义指定以下操作:

1. 当接收到 StartMaintenanceModeAllHosts 类型的事件时，将激活所有规则实例。
2. 当 2 个小时后或接收到 StopMaintenanceModeAllHosts 类型的事件时，将停用所有规则实例。

```
<duplicateRule name="Maintenance_Supression">
  <activationInterval>
    <activationTime>
      <start>
        <inactiveWhenLoaded/>
      </start>
      <stop>
        <after duration="PT2H" unit="ISO-8601"/>
      </stop>
    </activationTime>
    <activateOnEvent>
      <eventType type="StartMaintenanceModeAllHosts"/>
    </activateOnEvent>
```

```
    <deactivateOnEvent>
      <eventType type="StopMaintenanceModeAllHosts"/>
    </deactivateOnEvent>
  </activationInterval>
  <groupingKey missingAttributeHandling="ignoreEvent">
    <attributeName>hostname</attributeName>
  </groupingKey>
  <timeWindow>
    <runUntilDeactivated/>
  </timeWindow>
  <onDetection>
    <action expressionLanguage="java" name="DropEvent">
      <![CDATA[act_lib.exitRuleSet();]]>
    </action>
  </onDetection>
  <onTimeWindowComplete>
    <action expressionLanguage="java" name="CreateSummaryOfSupressedEvents">
      <![CDATA[Helper.createSummaryEvent("MaintenanceSummary", act_eventList, act_lib);]]>
    </action>
  </onTimeWindowComplete>
</duplicateRule>
```

在某些情况下，您可能希望控制要激活哪些规则实例以及何时激活。针对这些情况，应编写 <activationByGroupingKey> 元素。

以下示例将扩展先前的示例，并说明如何使用组密钥值来选择允许处理的规则实例。<activationByGroupingKey> 元素中的定义将指定以下操作:

1. 当接收到特定主机名的 StartMaintenanceMode 类型的事件时，将允许处理这些主机名的规则实例。
2. 当激活 2 个小时后或接收到相应主机名的 StopMaintenanceMode 类型的事件时，将停用这些规则实例。

```
<activationByGroupingKey>
  <activateOnEvent>
    <eventType type="StartMaintenanceMode"/>
    <stopAfter duration="PT2H" unit="ISO-8601"/>
  </activateOnEvent>
  <deactivateOnEvent>
    <eventType type="StopMaintenanceMode"/>
  </deactivateOnEvent>
  </activationByGroupingKey>
```

以下描述将总结在编写 <activationByGroupingKey> 元素时将发生的情况:

- 当在 <activationByGroupingKey> 元素中编写了 <activateOnEvent> 元素时，仅允许处理那些共享特定事件（符合 <activationByGroupingKey> <activateOnEvent> 条件）的组密钥值的事件。
- 当在 <activationByGroupingKey> 元素中编写了 <deactivateOnEvent> 元素时，不允许处理那些共享特定事件（符合 <activationByGroupingKey> <deactivateOnEvent> 条件）的组密钥值的事件。

不同激活和停用定义对规则状态的影响

第 59 页的表 15 和 第 60 页的表 16 将显示不同激活和停用定义如何影响规则的状态。在这些表中，将使用以下约定:

- *A* 为激活事件。
- 在表示法 『*A[x]* 中，』*x* 代表组密钥值。例如，*A[1]* 为带有组密钥值 1 的激活事件。
- *D* 为停用事件。

- 在表示法 『*D[x]* 中，』*x* 代表组密钥值。例如，*D[1]* 为带有组密钥值 1 的停用事件。

表 15. 规则状态根据激活定义的不同而变化

开始规则状态	可能影响规则状态的因素	结束规则状态
不活动	<activationInterval> <activationTime> <start> 中定义的时间	1. 激活规则。 2. <onActivation> 操作运行。 3. 允许所有组密钥值。
	activate() 方法	
	事件 *A*，在 <activationInterval> <activateOnEvent> 中定义	
	事件 *A[1]*，在 <activationByGroupingKey> <activateOnEvent> （*不带有* <stopAfter>）中定义	1. 激活规则。 2. <onActivation> 操作运行。 3. 仅允许组密钥值 1。当规则模式与该规则实例匹配时，将不再允许组密钥值 1。
	事件 *A[2]*，在 <activateOnEvent> （*带有* <stopAfter>）中定义	1. 激活规则。 2. <onActivation> 操作运行。 3. 仅允许组密钥值 2，且该值仅允许用于 <stopAfter> 元素指定的持续时间。在该持续时间内，该规则实例的规则模式可有多次匹配。
• 活动 • 允许所有组密钥值	<activationInterval> <activationTime> <start> 中定义的时间	规则状态中尚未发生变化。与开始规则状态相同。
	activate() 方法	
	事件 *A*，在 <activationInterval> <activateOnEvent> 中定义	
	事件 *A[1]*，在 <activationByGroupingKey> <activateOnEvent> （*不带有* <stopAfter>）中定义	
	事件 *A[2]*，在 <activateOnEvent> （*带有* <stopAfter>）中定义	

表 15. 规则状态根据激活定义的不同而变化 (续)

开始规则状态	可能影响规则状态的因素	结束规则状态
• 活动 • 仅允许根据 <activationByGroupingKey> <activateOnEvent> 定义触发了规则实例的组密钥值	<activationInterval> <activationTime> <start> 中定义的时间	规则状态中尚未发生变化。与开始规则状态相同。
	activate() 方法	
	事件 *A*，在 <activationInterval> <activateOnEvent> 中定义	允许所有组密钥值。
	事件 *A[1]*，在 <activationByGroupingKey> <activateOnEvent> （*不带有* <stopAfter>）中定义	• 除先前允许的组密钥值外，目前还允许组密钥值 1。 • 当规则模式与该规则实例匹配时，将不再允许组密钥值 1。
	事件 *A[2]*，在 <activateOnEvent> （*带有* <stopAfter>）中定义	• 除先前允许的组密钥值外，目前还允许组密钥值 2。 • 该值仅允许用于 <stopAfter> 元素指定的持续时间。 • 在该持续时间内，该规则实例的规则模式可有多次匹配。
• 活动 • 除了根据 <activationByGroupingKey> <deactivateOnEvent> 定义而不允许的组密钥值外，允许所有其他组密钥值	<activationInterval> <activationTime> <start> 中定义的时间	规则状态中尚未发生变化。与开始规则状态相同。
	activate() 方法	
	事件 *A*，在 <activationInterval> <activateOnEvent> 中定义	允许所有组密钥值。
	事件 *A[1]*，在 <activationByGroupingKey> <activateOnEvent> （*不带有* <stopAfter>）中定义	除先前允许的组密钥值外，目前还允许组密钥值 1。
	事件 *A[2]*，在 <activateOnEvent> （*带有* <stopAfter>）中定义	除先前允许的组密钥值外，目前还允许组密钥值 2。

表 16. 规则状态根据停用定义的不同而变化

开始规则状态	可能影响规则状态的因素	结束规则状态
不活动	<activationInterval> <activationTime> <stop> 中定义的时间	规则状态中尚未发生变化。与开始规则状态相同。
	deactivate() 方法	
	事件 *D*，在 <activationInterval> <deactivateOnEvent> 中定义	
	事件 *D[1]*，在 <activationByGroupingKey> <deactivateOnEvent> 中定义	
	事件 *A[2]* 的持续时间（在 <activationByGroupingKey> <activateOnEvent> <stopAfter> 中定义）结束	

表 16. 规则状态根据停用定义的不同而变化 (续)

开始规则状态	可能影响规则状态的因素	结束规则状态
• 活动 • 允许所有组密钥值	<activationInterval> <activationTime> <stop> 中定义的时间	1. 所有规则实例均已停用。 2. <onDeactivation> 操作运行。 3. 停用规则。
	deactivate() 方法	
	事件 *D*，在 <activationInterval> <deactivateOnEvent> 中定义	
	事件 *D[1]*，在 <activationByGroupingKey> <deactivateOnEvent> 中定义	• 不再允许组密钥值 1。 • 如果带有组密钥值 1 的规则实例处于活动状态，则将其停用。
	事件 *A[2]* 的持续时间（在 <activationByGroupingKey> <activateOnEvent> <stopAfter> 中定义）结束	已停用带有组密钥值 2 的规则实例。
• 活动 • 仅允许根据 <activationByGroupingKey> <activateOnEvent> 定义触发了规则实例的组密钥值	<activationInterval> <activationTime> <stop> 中定义的时间	1. 所有规则实例均已停用。 2. <onDeactivation> 操作运行。 3. 停用规则。
	deactivate() 方法	
	事件 *D*，在 <activationInterval> <deactivateOnEvent> 中定义	
	事件 *D[1]*，在 <activationByGroupingKey> <deactivateOnEvent> 中定义	• 不再允许组密钥值 1。 • 如果带有组密钥值 1 的规则实例处于活动状态，则将其停用。
	事件 *A[2]* 的持续时间（在 <activationByGroupingKey> <activateOnEvent> <stopAfter> 中定义）结束	• 不再允许组密钥值 2。 • 已停用带有组密钥值 2 的规则实例。
• 活动 • 除了根据 <activationByGroupingKey> <deactivateOnEvent> 定义而不允许的组密钥值外，允许所有其他组密钥值	<activationInterval> <activationTime> <stop> 中定义的时间	1. 所有规则实例均已停用。 2. <onDeactivation> 操作运行。 3. 停用规则。
	deactivate() 方法	
	事件 *D*，在 <activationInterval> <deactivateOnEvent> 中定义	
	事件 *D[1]*，在 <activationByGroupingKey> <deactivateOnEvent> 中定义	• 不再允许组密钥值 1。 • 如果带有组密钥值 1 的规则实例处于活动状态，则将其停用。
	事件 *A[2]* 的持续时间（在 <activationByGroupingKey> <activateOnEvent> <stopAfter> 中定义）结束	已停用带有组密钥值 2 的规则实例。

activationInterval 元素

<activationInterval> 元素包含一些元素，它们用于定义：规则何时处于活动，何时处于不活动状态。

详细信息

可在离散时间点上激活或停用规则，或者由特定事件对其进行激活或停用。

如果将规则指定为在离散时间点上以及由特定事件进行激活或停用，则该规则将在该时间点或在接收到事件时（取决于哪个发生在先）被激活或停用。然而在该情况下，在该规则的生命周期内可能由多个事件对其进行激活或停用。例如，规则可能由某个事件激活，然后被停用，接着在某个定义的时间点上被激活，然后再次被停用，随后再由另一个事件激活。

在商业环境中，可能要在接收到表明股票交易开市的事件时便激活规则。在 IT 环境中，可能要在 2005/10/29 06:00 启动维护窗口，并在以下某个时间（取决于哪个发生在先）结束:

- 2005/10/30 11:30
- 当接收到表明维护工作已完成的事件时

属性

<activationInterval> 无属性。

包含于

<activationInterval> 包含于以下元素中:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

包含

<activationInterval> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 17. <activationInterval> 元素中包含的元素

元素	必需或可选?
<activationTime>	可选。允许出现 0 或 1 次。
<activateOnEvent>	可选。允许出现 0 或 1 次。
<deactivateOnEvent>	可选。允许出现 0 或 1 次。
<activationByGroupingKey>	可选。允许出现 0 或 1 次。

包含的元素之间的关系

包含于 <activationTime> 元素中的 <start> 和 <stop> 元素是用于激活和停用规则的静态方法。通过这些元素，将在某个离散时间点上激活或停用某个规则。与此相反，<activateOnEvent> 和 <deactivateOnEvent> 元素是用于激活和停用规则的动态方法。通过这些元素，将在发生某个事件的情况下激活或停用某个规则。例如，如果某个规则尚未处于活动状态，则符合为 <activateOnEvent> 元素定义的条件的任何事件都将激活

该规则。如果某个规则尚未处于不活动状态，则符合为 <deactivateOnEvent> 元素定义的条件的任何事件都将停用该规则。因此，某些事件可更改关于何时激活或停用规则的静态定义。

表 18 描述了如何基于某些组合（可在其中编写以下元素）来激活或停用规则，以及何时激活或停用它:

- <start>
- <stop>
- <activateOnEvent>
- <deactivateOnEvent>

在 表 18 中，*X* 代表激活该规则的事件的名称，*Y* 代表停用该规则的事件的名称。

如果没有编写 <start> 元素，则缺省开始时间与 <whenLoaded> 元素定义的时间相同。

如果没有编写 <stop> 元素，则缺省停止时间与 <never> 元素定义的时间相同。

表 18. 规则活动（基于编写 <activationInterval> 中所含元素的不同组合）

<activationTime>				
<start>	**<stop>**	**<activateOnEvent>**	**<deactivateOnEvent>**	**规则活动**
<whenLoaded>	<never>			规则在装入后处于活动状态，并在活动关联技术引擎运行时保持活动状态。
<whenLoaded>	<never>		*Y*	规则在装入后处于活动状态。事件 *Y* 将停用规则。
<whenLoaded>	<never>	*X*	*Y*	规则在装入后处于活动状态。事件 *Y* 将停用规则，而事件 *X* 将重新激活规则。该停用和重新激活可多次进行。
<whenLoaded>	<after>			规则在装入后处于活动状态，并在指定的时间间隔之后停用。
<whenLoaded>	<dateTime>			规则在装入后处于活动状态，并在指定的日期和时间停用。
<inactiveWhenLoaded>	<never>	*X*		规则在装入后处于不活动状态。事件 *X* 将激活该规则，且该规则在活动关联技术引擎运行时保持活动状态。
<inactiveWhenLoaded>	<never>	*X*	*Y*	规则在装入后处于不活动状态。事件 *X* 将激活该规则，而事件 *Y* 将停用该规则。该激活和停用可多次进行。
<dateTime>	<dateTime>			规则将在指定的日期和时间被激活，并在指定的日期和时间被停用。
<dateTime>	<dateTime>	*X*	*Y*	规则将在指定的日期和时间被激活，并在指定的日期和时间被停用。事件 *X* 将激活该规则，而事件 *Y* 将停用该规则。事件 *X* 和 *Y* 可多次激活和停用规则。
<dateTime>	<never>			规则将在指定的日期和时间被激活，并在活动关联技术引擎运行时保持活动状态。
<dateTime>	<never>		*Y*	规则将在指定的日期和时间被激活。事件 *Y* 将停用规则。
<dateTime>	<never>	*X*	*Y*	规则将在指定的日期和时间被激活。事件 *Y* 将停用规则，而事件 *X* 将重新激活规则。该停用和重新激活可多次进行。
<dateTime>	<after>			规则将在指定的日期和时间被激活，并在指定的时间间隔之后被停用。
<dateTime>	<after>	*X*	*Y*	规则将在指定的日期和时间被激活，并在指定的时间间隔之后被停用。事件 *X* 将激活该规则，而事件 *Y* 将停用该规则。该激活和停用可多次进行。

activationTime 元素

<activationTime> 元素定义激活或停用某个规则的离散时间点。

属性

<activationTime> 不具有属性。

包含于

<activationTime> 包含于以下元素中:

- <activationInterval>

包含

<activationTime> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 19. <activationTime> 元素中包含的元素

元素	必需或可选?
<start>	可选。允许出现 0 或 1 次。
<stop>	可选。允许出现 0 或 1 次。

after 元素

<after> 元素将指定规则在开始处于活动状态后继续保持该状态的持续时间。在该持续时间过后，将停用该规则。

属性

<after> 具有以下属性:

表 20. <after> 元素的属性

名称	描述	数据类型	必需?
duration	指定持续时间的长度。该属性的数据类型取决于 unit 属性的值。	• 如果 unit 属性的值为 ISO-8601，则该数据类型为 xsd:duration。 • 如果 unit 属性的值为 milliseconds，则该数据类型为 xsd:positiveInteger。	是
unit	指定要使用的时间单位。该属性的有效值为: • ISO-8601 • milliseconds	xsd:string	是

对持续时间使用 ISO 8601 标准

将 ISO-8601 编码为 unit 属性的值，则表明根据 ISO 8601 标准对 duration 属性的值进行编码，以将持续时间指定为一个字符串。标准 XML 模式数据类型规范将使用 ISO 8601 来提供名为 duration 的数据类型。将在 http://www.w3.org/TR/xmlschema-2/#duration 中对该数据类型作详细描述。

标准 XML 模式中的 duration 数据类型的格式为以下字符串:

P*n*Y*n*M*n*DT*n*H*n*M*n*S

- P 为字符，字符串始终以该字符开头。
- *n*Y 代表年数。 1 年等同于 365 天。因此，编码为 1Y 等同于编码为 365D。
- *n*M 代表月数。 1 个月等同于 30 天。因此，编码为 1M 等同于编码为 30D。
- *n*D 代表天数。
- T 为分隔符，用于将日期单位（年、月、日）与时间单位（小时、分、秒）分隔开。时间单位始终跟随在 T 之后。
- *n*H 代表小时数。
- *n*M 代表分钟数。
- *n*S 代表秒数。

以下为该格式的示例:

- P5DT12H 为 5.5 天。
- PT59M59S 为 59 分 59 秒。
- P1M 为 1 个月。

包含于

<after> 包含于以下元素中:

- <stop>

包含

<after> 不包含元素。

attributeAlias 元素

<attributeAlias> 元素提供别名来与不同事件中具有相同含义但名称不同的事件属性相关联。例如，3 个不同的事件可能对某个事件属性使用以下 3 个不同的名称，该事件属性指示启动该事件的系统的名称: host、hostname 和 source。<attributeAlias> 元素包含描述各个事件属性的 <eventAttribute> 元素，这些事件属性必须关联为组密钥的一个事件属性。

详细信息

<attributeAlias> 元素及其 aliasName 属性仅在组密钥的上下文中有效。不能在任何表达式中引用该元素及其属性，包括 <computedValue> 元素中的表达式。

属性

<attributeAlias> 具有以下属性:

表 21. <attributeAlias> 元素的属性

名称	描述	数据类型	必需?
aliasName	定义事件属性的名称，该事件属性在 <eventAttribute> 元素中描述，需要关联为组密钥的一个事件属性。该名称在规则中必须是唯一的。	xsd:NMTOKEN	是

包含于

<attributeAlias> 包含于以下元素中:

- <groupingKey>

包含

<attributeAlias> 包含以下元素:

表 22. <attributeAlias> 元素中包含的元素

元素	必需或可选?
<eventAttribute>	该元素需要出现 2 次。允许出现更多次。

attributeName 元素

<attributeName> 元素包含作为组密钥组成部分的特定事件属性的名称。该名称必须与在 act_event 变量的 getAttribute 方法调用中使用的名称匹配。

属性

<attributeName> 不具有属性。

包含于

<attributeName> 包含于以下元素中:

- <groupingKey>

包含

<attributeName> 不包含元素。

booleanThreshold 元素

<booleanThreshold> 元素仅对于阈值规则有效。该元素中包含在接收到每个事件时调用的表达式。该表达式将基于当前事件以及该规则已接受的任何其他事件来计算或比较阈值。该表达式将返回布尔值 `true` 或 `false`，以指示是否已达到阈值。

详细信息

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<booleanThreshold> 具有以下属性:

表 23. <booleanThreshold> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是

包含于

<booleanThreshold> 包含于以下元素中:

- <thresholdRule>

包含

<booleanThreshold> 不包含元素。

相关概念

第 17 页的『表达式』

表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

collectionRule 元素

<collectionRule> 元素将根据收集模式来定义规则。

属性

<collectionRule> 具有以下属性:

表 24. <collectionRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	确定规则是接收所有事件还是仅接收从其他规则转发的事件。缺省值为 `false`，表明规则将接收所有事件，包括从其他规则转发的事件。	xsd:boolean	否

包含于

<collectionRule> 包含于以下元素中:

- <ruleBlock>

包含

<collectionRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 25. <collectionRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<eventSelector>	可选。允许出现 0 或 1 次。
<groupingKey>	可选。允许出现 0 或 1 次。
<timeWindow>	必需。仅允许出现 1 次。
<onTimeWindowComplete>	可选。允许出现 0 或 1 次。

相关概念

第 10 页的『收集模式』
收集规则由收集模式定义。该规则将在一定时间间隔内收集一组选定的事件。 该规则为全状态规则。

comment 元素

<comment> 元素可包含其所包含的规则集、规则块、规则或变量的功能和用途的描述。

属性

<comment> 不具有属性。

包含于

<comment> 包含于以下元素中:

- <ruleSet>
- <ruleBlock>
- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>

- <timerRule>
- <variable>

包含

<comment> 不包含元素。

computationRule 元素

<computationRule> 元素将根据计算模式来定义规则。

属性

<computationRule> 具有以下属性:

表 26. <computationRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	确定规则是接收所有事件还是仅接收从其他规则转发的事件。缺省值为 `false`，表明规则将接收所有事件，包括从其他规则转发的事件。	xsd:boolean	否

包含于

<computationRule> 包含于以下元素中:

- <ruleBlock>

包含

<computationRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 27. <computationRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<eventSelector>	可选。允许出现 0 或 1 次。
<groupingKey>	可选。允许出现 0 或 1 次。
<computeFunction>	必需。仅允许出现 1 次。
<timeWindow>	必需。仅允许出现 1 次。
<onTimeWindowComplete>	可选。允许出现 0 或 1 次。

相关概念

第 10 页的『计算模式』

计算规则由计算模式定义。该规则在一定时间间隔内接收到每个事件时，将计算（通过表达式）应用于所收集的事件。该规则为全状态规则。

computedThreshold 元素

<computedThreshold> 元素仅对阈值规则有效。该元素包含在接收到每个事件时调用的表达式，该表达式将根据当前事件以及与规则的事件选择条件匹配的任何其他事件来计算阈值。该表达式将返回该计算的阈值，并存储在为该规则定义的变量中。该规则将随后使用该计算的阈值与定义的阈值进行比较。

详细信息

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<computedThreshold> 具有以下属性:

表 28. <computedThreshold> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是
阈值	定义要符合的阈值。该定义的阈值必须为数值的字符串表示法，并可将其转换为对于规则变量有效的数据类型。	xsd:string	是
assignTo	标识变量名称，该变量用于保存从该表达式返回的该计算的阈值。必须已使用 <variable> 元素（在规则集、规则块或规则级别）为规则定义了该变量。必须将其定义为以下一种数字数据类型: • java.lang.Double • java.lang.Float • java.lang.Integer • java.lang.Long • java.lang.String 如果在规则集或规则块级别定义变量，则在规则模式匹配后不会重新初始化该变量。	xsd:NMTOKEN	是

表 28. <computedThreshold> 元素的属性 (续)

名称	描述	数据类型	必需?
thresholdComparison	定义用于将计算的阈值与定义的阈值进行比较的运算符。该运算符的有效值为: • lessThan • lessThanOrEqualTo • greaterThan • greaterThanOrEqualTo	xsd:string	是

包含于

<computedThreshold> 包含于以下元素中:

- <thresholdRule>

包含

<computedThreshold> 不包含元素。

相关概念

第 17 页的『表达式』
表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

computedValue 元素

<computedValue> 元素中包含一个表达式，该表达式在规则接收到以下事件时运行: 该事件要求根据该事件的一个或多个属性的值来创建字符串值。该字符串值随后可在组密钥中使用。

详细信息

有时，规则编写者可能要使用诸如组密钥中的以下项:

- 事件属性值的子串。例如，如果事件属性值包含嵌入的 IP 地址，则 <computedValue> 元素中的表达式可将该 IP 地址抽取为唯一的值，以便在组密钥中使用。
- 来自几个不同事件属性的值的子串。例如，<computedValue> 元素中的表达式可抽取一些子串并将它们进行组合来创建唯一的值，以便在组密钥中使用。

如果 <computedValue> 元素中的表达式返回空值，则规则将该空值视为缺少属性值。

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<computedValue> 具有以下属性:

表 29. <computedValue> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是

包含于

<computedValue> 包含于以下元素中:

- <groupingKey>

包含

<computedValue> 不包含元素。

相关概念

第 17 页的『表达式』
表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

computeFunction 元素

<computeFunction> 元素仅对于计算规则有效。该元素中包含一个表达式，该表达式在接收到每个事件时调用，并返回将存储于为规则定义的变量中的值。从该表达式返回的值必须与在 <computeFunction> 元素的 assignTo 属性中指定的变量的数据类型匹配。

详细信息

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<computeFunction> 具有以下属性:

表 30. <computeFunction> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是

表 30. <computeFunction> 元素的属性 (续)

名称	描述	数据类型	必需?
assignTo	标识变量的名称，该变量保存从该表达式返回的值。必须已使用 <variable> 元素（在规则集、规则块或规则级别）为规则定义了该变量。如果该变量在规则集或规则块级别定义，则在规则模式匹配后不会将它重新初始化。	xsd:NMTOKEN	是

包含于

<computeFunction> 包含于以下元素中:

- <computationRule>

包含

<computeFunction> 不包含元素。

相关概念

第 17 页的『表达式』

表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

dateTime 元素

<dateTime> 元素指定激活或停用规则的日期和时间。然而，仅当该规则在这一指定时间之前已装入到运行中的活动关联技术引擎时，才会激活或停用该规则。

详细信息

如果尚未在指定的激活时间之前将该规则装入至运行中的活动关联技术引擎，则不会激活该规则。如果尚未在指定的停用时间之前将该规则装入至运行中的活动关联技术引擎，会将该规则设置为 <start> 元素定义的状态，且不会由 <stop> 元素停用。

<dateTime> 元素的内容必须是字符串，该字符串遵循标准 XML 模式中 dateTime 数据类型的格式。例如，dateTime 由以下格式的有限长度的字符序列组成:

yyyy '-' *mm* '-' *dd* 'T' *hh* ':' *mm* ':' *ss* ('.' *s*+)? (*zzzzzz*)?

- *yyyy* 是代表年份的 4 位或更多的数字。如果多于 4 位，则不允许出现前导零，因而不允许出现 0000。
- 剩余的 '-' 是日期部分的各个部分之间的分隔符。
- 第一个 *mm* 是代表月份的 2 位数字，从 01 开始。
- *dd* 是代表日期的 2 位数字，从 01 开始。
- T 是分隔符，指示后跟时刻。
- *hh* 是代表小时（24 小时制）的 2 位数字，开始自 00 并结束于 23。
- : 是时刻部分的各个部分之间的分隔符。
- 第二个 *mm* 是代表分钟的 2 位数字，开始自 00 并结束于 59。

- *ss* 是代表整秒的 2 位数字，开始自 00 并结束于 59。
- '.' *s*+（如果存在）代表小数秒。
- *zzzzzz*（如果存在）代表时区。时区由有限长度的字符序列组成，格式为 (('+' | '-') *hh* ':' *mm*) | 'Z'，其中:
  - '+'（如果存在）代表非负的持续时间，因而不得出现 '-'。
  - '-'（如果存在）代表非正的持续时间，因而不得出现 '+'。
  - *hh* 是代表小时的 2 位数字，开始自 00 并结束于 14。
  - *mm* 是代表分钟的 2 位数字，开始自 00 并结束于 59。然而，如果小时的值为 14，则分钟的值必须为 00
  - Z 为 UTC（+00:00 或 -00:00）的速记符；鉴于此，不得出现其他时区元素。

以下为 <dateTime> 元素内容的两个示例:

- 2005-06-01T13:05:06.07 为当地时间 2005 年 6 月 1 日下午 1:05 超过 6.07 秒。
- 2005-06-01T13:05:06.07Z 为 UTC 时间 2005 年 6 月 1 日下午 1:05 超过 6.07 秒，该时间将是 EDT 时间 2005 年 6 月 1 日上午 9:05 超过 6.07 秒（或 2005-06-01T09:05:06.07-04:00）

属性

<dateTime> 无属性。

包含于

<dateTime> 包含于以下元素中:

- <start>
- <stop>

包含

<dateTime> 不包含元素。

deactivateOnEvent 元素

<deactivateOnEvent> 元素定义可停用规则或（对于使用 <groupingKey> 元素定义的规则）规则实例的事件。

以下为选择事件的 3 种可能方式:

- 使用带有 <filteringPredicate> 元素的一个或多个 <eventType> 元素
- 使用不带有 <filteringPredicate> 元素的一个或多个 <eventType> 元素
- 使用不带有任何 <eventType> 元素的 <filteringPredicate> 元素

如果规则处于活动状态且未对 <eventType> 或 <filteringPredicate> 元素进行编码，则将选择发生的任意事件。

不编写任何 <eventType> 元素可对系统性能造成负面影响。

假定要选择Audit Failure 类型的所有事件。可使用过滤谓词来进一步优化选择条件，以便仅包含事件属性具有特定值的事件。例如，可编写 <eventType> 元素以选择Audit Failure类型的所有事件，并编写 <filteringPredicate> 元素以仅选择主机名属性具有值 MyCriticalSystem）的那些事件。

属性

<deactivateOnEvent> 不具有属性。

包含于

<deactivateOnEvent> 包含于以下元素中:

- <activationInterval>
- <activationByGroupingKey>

包含

<deactivateOnEvent> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 31. <deactivateOnEvent> 元素中包含的元素

元素	必需或可选?
<eventType>	可选。允许出现 0 次或多次。
<filteringPredicate>	可选。允许出现 0 或 1 次。

duplicateRule 元素

<duplicateRule> 元素将根据重复模式来定义规则。

属性

<duplicateRule> 具有以下属性:

表 32. <duplicateRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	确定规则是接收所有事件还是仅接收从其他规则转发的事件。缺省值为 false，表明规则将接收所有事件，包括从其他规则转发的事件。	xsd:boolean	否

包含于

<duplicateRule> 包含于以下元素中:

- <ruleBlock>

包含

<duplicateRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 33. <duplicateRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<eventSelector>	可选。允许出现 0 或 1 次。
<groupingKey>	可选。允许出现 0 或 1 次。
<timeWindow>	必需。仅允许出现 1 次。
<onDetection>	可选。允许出现 0 或 1 次。
<onNextEvent>	可选。允许出现 0 或 1 次。
<onTimeWindowComplete>	可选。允许出现 0 或 1 次。

相关概念

第 11 页的『重复模式』
重复规则由重复模式定义。该规则计算在指定的时间间隔内接受的第二个时间和后续事件，但跳过对这些事件的规则集处理。该规则为全状态规则。

eventAttribute 元素

<eventAttribute> 元素提供一种方法来将事件类型与事件属性关联，作为 <attributeAlias> 元素定义的属性别名的一部分。

属性

<eventAttribute> 具有以下属性:

表 34. <eventAttribute> 元素的属性

名称	描述	数据类型	必需?
类型	定义事件类型的名称。这与用于 <eventType> 元素的类型属性的名称相同。	xsd:NMTOKEN	是
属性名称	指定事件属性的标准名称，该名称正通过属性别名与其他事件属性相关联。该名称必须与在 act_event 变量中用于调用 getAttribute 方法的名称匹配。	xsd:string	是

包含于

<eventAttribute> 包含于以下元素中:

- <attributeAlias>

包含

<eventAttribute> 不包含元素。

eventCountThreshold 元素

<eventCountThreshold> 元素仅对阈值规则有效。该元素定义必须在某个时间段内符合事件选择条件的事件的数量。<eventCountThreshold> 元素还为时间窗口指定两个可能的时间间隔方式之一（固定或滑动）。

详细信息

固定时间间隔

固定时间间隔在接收到第一个符合事件选择条件的事件时开始，在发生以下某种情况时结束:

- 规则在指定的持续时间内符合其阈值。
- 指定的持续时间已过。

滑动时间间隔

滑动时间间隔在接收到第一个符合事件选择条件的事件时开始。然而，当规则尚未符合其阈值且指定的持续时间已过时，时间窗口将把开始时间调整（滑动）为新的“第一个”事件的接收时间，通常情况下为接受的下一个事件。在发生以下某种情况之前，滑动时间间隔将继续以该方式进行调整:

- 规则在指定的持续时间内符合其阈值。
- 在接收到启动时间窗口的事件后，指定的持续时间内未接收到后续事件。

启动时间窗口的事件（成为新的“第一个”事件）为具有符合以下条件的接收时间的事件: 接收时间加上规则的时间间隔持续时间大于当前时间。以下为公式形式的条件:

事件接收时间 + 规则的时间间隔持续时间 > 当前时间

当不存在此类事件时，滑动时间间隔将无法再调整时间，时间间隔将结束。

在符合阈值或该时间段结束之前，阈值规则将计算每个接受的事件的数量。随后它将相应地运行 <onDetection> 元素或 <onTimeOut> 元素中定义的操作。

<onDetection> 操作

这些操作在事件计数与 <eventCountThreshold> 元素的阈值属性所定义的值相同（表明符合该阈值）时运行。

<onTimeOut> 操作

这些操作运行的时间取决于时间间隔方式是固定的还是滑动的。

固定方式

使用固定方式，这些操作将在时间窗口到期时运行。

滑动方式

使用滑动方式，在接收到启动时间窗口的事件后且指定的持续时间内未接收到后续事件，这些操作将运行。换言之，不存在任何事件，其接收时间加上规则的时间间隔持续时间大于当前时间。

时间窗口的时间间隔方式由 <eventCountThreshold> 元素的 timeIntervalMode 属性来定义。以下场景将说明这两个可能的时间间隔方式的行为以及之间的区别。

说明固定和滑动方式的场景

假定规则接收 4 个符合事件选择条件的事件，分别在以下时间：8:00、8:04、8:06 和 8:07。事件计数阈值为 3，且时间窗口的持续时间为 5 分钟。

fixed方式的规则行为

使用该时间间隔方式，阈值规则在 8:00 开始处理，并在 8:05 运行 <onTimeOut> 操作（由于该规则在 5 分钟内仅接收到 2 个事件）。因此，在该时间窗口中不符合阈值。在 8:06 接收到第三个事件后，阈值规则再次开始处理，并在 8:11 运行 <onTimeOut> 操作（由于该规则在 5 分钟内仅接收到 2 个事件）。

固定方式是静态的。

sliding方式的规则行为

使用该时间间隔方式，阈值规则在 8:00 开始处理。在 8:05，届时已调度了时间窗口完成，规则 确定其仅接收到 2 个事件。随后，规则将废弃 8:00 接收的事件，并将持续时间重新计算为在 8:09 结束（因为第一个事件现在是 8:04 接收的那个事件）。当规则在 8:07 接收到事件时，由于目前已在最近的时间窗口（8:04 – 8:09）中符合其阈值（在 8:04、8:06 和 8:07 接收到 3 个事件），因此将运行 <onDetection> 操作。

滑动方式是动态的，因为它将不断调整（滑动）开始时间，以尝试在时间窗口中符合阈值。

现在，假定规则接收到 4 个符合事件选择条件的事件，分别在以下时间：8:00、8:04、8:06 和 8:10。事件计数阈值为 3，且时间窗口的持续时间为 5 分钟。

sliding方式的规则行为

在该情况下，阈值规则在 8:00 开始处理。在 8:05，届时已调度了时间窗口完成，规则 确定其仅接收到 2 个事件。随后，规则将废弃 8:00 接收的事件，并将持续时间重新计算为在 8:09 结束（因为第一个事件现在是 8:04 接收的那个事件）。

在 8:09（已调度时间窗口在此时完成），规则确定其仅接收到 2 个事件。随后，规则将废弃在 8:04 接收的事件，并将持续时间重新计算为在 8:11 结束（因为第一个事件现在是 8:06 接收的那个事件）。

在 8:11（已调度时间窗口在此时完成），规则确定其仅接收到 2 个事件。随后，规则将废弃在 8:06 接收的事件，并将持续时间重新计算为在 8:15 结束（因为第一个事件现在是 8:10 接收的那个事件）。

在 8:15（已调度时间窗口在此时完成），规则确定自 8:10 启动时间窗口的事件以来未接收到任何事件。规则将运行 <onTimeOut> 操作。

属性

<eventCountThreshold> 具有以下属性:

表 35. <eventCountThreshold> 元素的属性

名称	描述	数据类型	必需?
阈值	定义必须在某个时间段内符合事件选择条件的事件的数量。这是将符合的事件计数阈值。该值必须为正整数。	xsd:positiveInteger	是
timeIntervalMode	定义时间窗口的时间间隔是固定的还是滑动的。该属性的有效值为: • `fixed`（缺省值） • `sliding`	xsd:string	否

包含于

<eventCountThreshold> 包含于以下元素中:

- <thresholdRule>

包含

<eventCountThreshold> 不包含元素。

eventSelector 元素

<eventSelector> 元素定义由规则选择进行处理的事件。

详细信息

以下为选择事件的 3 种可能方式:

- 使用带有 <filteringPredicate> 元素的一个或多个 <eventType> 元素
- 使用不带有 <filteringPredicate> 元素的一个或多个 <eventType> 元素
- 使用不带有任何 <eventType> 元素的 <filteringPredicate> 元素

在希望规则处理所有事件的特殊情况下，可使用以下选项:

- 不编写 <eventSelector> 元素。
- 编写不包含元素的 <eventSelector> 元素。

不编写任何 <eventType> 元素可对系统性能造成负面影响。

假定要选择`Audit Failure` 类型的所有事件。可使用过滤谓词来进一步优化选择条件，以便仅包含事件属性具有特定值的事件。例如，可编写 <eventType> 元素以选择`Audit Failure`类型的所有事件，并编写 <filteringPredicate> 元素以仅选择主机名属性具有值`MyCriticalSystem`）的那些事件。

属性

<eventSelector> 具有以下属性:

表 36. <eventSelector> 元素的属性

名称	描述	数据类型	必需?
alias	该属性仅在序列规则中有效，序列规则是唯一具有多个 <eventSelector> 元素的规则。它唯一地指定了序列规则中某个事件选择器选择的事件。过滤谓词和操作可随后使用该别名访问该事件。	xsd:NMTOKEN	否

包含于

<eventSelector> 包含于以下元素中:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>

包含

<eventSelector> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 37. <eventSelector> 元素中包含的元素

元素	必需或可选?
<eventType>	可选。允许出现 0 次或多次。
<filteringPredicate>	可选。允许出现 0 或 1 次。

eventType 元素

<eventType> 元素定义选定事件的类型，该事件由某个规则处理，或者用于激活或停用该规则。

属性

<eventType> 具有以下属性:

表 38. <eventType> 元素的属性

名称	描述	数据类型	必需?
类型	定义事件类型。 对于符合通用基础事件（Common Base Event）规范的事件，该名称为 extensionName 属性的值。 对于 IBM Tivoli Enterprise Console® 事件，该名称为 BAROC 文件中定义的事件类名称。 基于其他格式的事件可能使用其他属性来指定事件类型。	xsd:NMTOKEN	是

包含于

<eventType> 包含于以下元素中:

- <activateOnEvent>
- <deactivateOnEvent>
- <eventSelector>

包含

<eventType> 不包含元素。

filteringPredicate 元素

<filteringPredicate> 元素包含表达式，该表达式进一步限制选择哪些事件来由规则处理，或者选择哪些事件来激活或停用规则。因此，相对于仅通过 <eventType> 元素按照事件类型进行过滤的方式，使用该元素可以更为全面地过滤事件。

详细信息

表达式定义条件并返回布尔值 `true`（符合条件）或 `false`（不符合条件）。

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<filteringPredicate> 具有以下属性:

表 39. <filteringPredicate> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是

包含于

<filteringPredicate> 包含于以下元素中:

- <activateOnEvent>
- <deactivateOnEvent>
- <eventSelector>

包含

<filteringPredicate> 不包含元素。

相关概念

第 17 页的『表达式』
表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

filterRule 元素

<filterRule> 元素根据过滤模式来定义规则。

属性

<filterRule> 具有以下属性:

表 40. <filterRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	确定规则是接收所有事件还是仅接收从其他规则转发的事件。缺省值为 `false`，表明规则将接收所有事件，包括从其他规则转发的事件。	xsd:boolean	否

包含于

<filterRule> 包含于以下元素中:

- <ruleBlock>

包含

<filterRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 41. <filterRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<eventSelector>	可选。允许出现 0 或 1 次。
<onDetection>	可选。允许出现 0 或 1 次。

相关概念

第 12 页的『过滤模式』
过滤规则由过滤模式定义。该规则在接受事件时将执行某个操作。它只对单个事件执行操作，因此是无状态规则。

groupingKey 元素

通常情况下，每个活动的规则都具有一个正在活动关联技术引擎中运行的规则实例或副本。然而，同一规则有时为不同的事件组所需，而不同的事件组一般与不同的资源组相关联。组密钥是一个或多个事件属性，或者是事件属性的组成部分，可用于将选定的事件分到不同的组中，以便按组进行不同处理。<groupingKey> 元素定义规则的组密钥。<groupingKey> 元素的用途是指定规则针对共享公共特性（如构成组密钥的属性的值所定义）的每组事件创建不同的规则实例（或其副本）。

详细信息

以下两个场景将说明组密钥的重要性。

场景 1:
发生 2 个事件，DB2down 事件和 DB2up 事件。DB2 程序在名为 A、B 和 C 的 3 台计算机上运行。序列规则被定义为将 DB2down 事件与 DB2up 事件相关，并在 DB2 程序停止且不重新启动时提醒操作员。

如果将该序列规则定义为不带有组密钥，且从计算机 A 接收到 DB2down 事件，则来自上述任一计算机的 DB2up 事件都将完成该序列，但这将不会实现预期目标。然而，如果已将组密钥定义为主机名属性，则将为选定事件中主机名属性的每个唯一的值创建规则的唯一副本或实例。活动关联技术引擎将把每个事件发送至正确的规则实例（该事件的主机名值的规则实例）。因此，如果从计算机 A 接收到 DB2down 事件，则

活动关联技术引擎将为计算机 A 创建一个规则实例。如果从计算机 B 接收到 DB2down 事件，则活动关联技术引擎将为计算机 B 创建第二个规则实例。当从计算机 B 接收到 DB2up 事件时，活动关联技术引擎将在这第二个规则实例中处理该事件。由于来自计算机 B 的 DB2down 和 DB2up 事件已正确关联，因此该序列完成，并且操作员接收到提醒。

场景 2:
在特定环境中，所有计算机上发生 `Incorrect login attempted` 消息的事件。该事件包含用户标识。阈值规则已定义为如果该事件在 5 分钟内发生次数超过 10 次，则将对操作员发出警告。

组密钥可定义为用户标识。然后，将为每个唯一的用户标识创建新的规则实例。将通过唯一的阈值规则实例来跟踪每个用户的登录尝试，每个实例对于该用户的登录尝试次数都具有单独的计数。如果任何用户标识 5 分钟内不正确登录的次数大于 10，操作员将接收到警告。

这一概念的其他变化形式包括:

- 组密钥可定义为主机名，而非用户标识。该选项可检测单台计算机上较多次数的不正确登录尝试。
- 组密钥可定义为主机名和用户标识的组合。该选项可检测特定用户标识对特定计算机的潜在入侵尝试。

如果为规则指定的所有事件类型中存在同一属性，则使用 <attributeName> 元素来定义组密钥是最简单且最常见的方式。

属性

<groupingKey> 具有以下属性:

表 42. <groupingKey> 元素的属性

名称	描述	数据类型	必需?
missingAttributeHandling	定义规则在以下某个情况下必须执行的操作: • 当选定事件具有参与组密钥中的属性但该属性的值缺失时 • 当 <computedValue> 元素中的表达式返回空值时。规则将把该空值视为缺失属性值。 missingAttributeHandling 属性的有效值为: • `ignoreEvent`（缺省值），表明规则将忽略该事件并对其不执行操作。 • `ignoreAttribute`，表明规则将接受该事件，但忽略带有缺失值的属性。活动关联技术引擎随后将包含该属性的替换值。	xsd:string	否

包含于

<groupingKey> 包含于以下元素中:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <sequenceRule>
- <thresholdRule>

包含

<groupingKey> 包含以下元素。

表 43. <groupingKey> 元素中包含的元素

元素	必需或可选?
<attributeAlias>	以下某个元素是必需的。编写以下元素中的多个元素是可选的。允许所有这 3 个元素多次出现。可按任何顺序编写这些元素。
<attributeName>	
<computedValue>	

import 元素

<import> 元素中包含表达式，该表达式指定要导入供规则内的其他表达式使用的外部模块（例如 Java 类）。

详细信息

表达式代码为 <import> 元素中的字符串。活动关联技术编译器将使用 <import> 元素提供的导入语句来编译规则中调用外部方法的表达式代码。

属性

<import> 具有以下属性:

表 44. <import> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是

包含于

<import> 包含于以下元素中:

- <ruleSet>
- <ruleBlock>

包含

<import> 不包含元素。

相关概念

第 18 页的『导入和访问外部模块和对象』

本示例说明如何使外部代码（例如 Java 类）和外部对象可由表达式访问。外部对象是应用程序创建用来与表达式进行通信的对象。

inactiveWhenLoaded 元素

<inactiveWhenLoaded> 元素指定当活动关联技术引擎装入某个规则时，该规则处于不活动状态。在使用其他方法激活该规则之前，该规则将保持不活动状态。

属性

<inactiveWhenLoaded> 不具有属性。

包含于

<inactiveWhenLoaded> 包含于以下元素中:

- <start>

包含

<inactiveWhenLoaded> 不包含元素。

lifeCycleActions 元素

<lifeCycleActions> 元素包含定义操作的元素，这些操作在规则的生命周期中的 4 个主要阶段执行。

详细信息

为装入和激活阶段定义的操作在实际装入或激活规则之后但在该规则开始任何处理之前进行调用。为停用和卸装阶段定义的操作在实际停用或卸装规则之前立即进行调用。

属性

<lifeCycleActions> 不具有属性。

包含于

<lifeCycleActions> 包含于以下元素中:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>

- <timerRule>

包含

<lifeCycleActions> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 45. <lifeCycleActions> 元素中包含的元素

元素	必需或可选?
<onLoad>	可选。允许出现 0 或 1 次。
<onActivation>	可选。允许出现 0 或 1 次。
<onDeactivation>	可选。允许出现 0 或 1 次。
<onUnload>	可选。允许出现 0 或 1 次。

never 元素

<never> 元素指定在某个特定时间始终不停用某个规则。但仍可通过某个事件或其他方法来停用规则。

属性

<never> 不具有属性。

包含于

<never> 包含于以下元素中:

- <stop>

包含

<never> 不包含元素。

onActivation 元素

<onActivation> 元素指定当激活规则时执行的操作或操作集合。<onActivation> 操作在激活规则之后但在规则开始任何处理之前进行调用。

详细信息

如果规则集包含在相同日期和时间（或者由相同事件）激活的多个规则，且具有相同时间窗口，则运行这些规则的以下操作的时间并不完全相同:

- <onTimeOut> 和 <onTimeWindowComplete> 元素中的规则响应操作
- <onActivation> 和 <onDeactivation> 元素中的生命周期操作

这些操作按照任一顺序连续运行。它们未必按照在规则集中对它们进行编写的顺序运行。由于必须在完成每个操作之后，才能开始序列中的下一操作，所以这些操作的运行时间不同。

属性

<onActivation> 不具有属性。

包含于

<onActivation> 包含于以下元素中:

- <lifeCycleActions>

包含

<onActivation> 包含以下元素:

表 46. <onActivation> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onDeactivation 元素

<onDeactivation> 元素指定当停用规则时执行的操作或操作集合。 <onDeactivation> 操作在停用规则之前立即调用。

详细信息

如果规则集包含在相同日期和时间（或者由相同事件）激活的多个规则，且具有相同时间窗口，则运行这些规则的以下操作的时间并不完全相同:

- <onTimeOut> 和 <onTimeWindowComplete> 元素中的规则响应操作
- <onActivation> 和 <onDeactivation> 元素中的生命周期操作

这些操作按照任一顺序连续运行。它们未必按照在规则集中对它们进行编写的顺序运行。由于必须在完成每个操作之后，才能开始序列中的下一操作，所以这些操作的运行时间不同。

属性

<onDeactivation> 不具有属性。

包含于

<onDeactivation> 包含于以下元素中:

- <lifeCycleActions>

包含

<onDeactivation> 包含以下元素:

表 47. <onDeactivation> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onDetection 元素

<onDetection> 元素仅对于重复规则、过滤规则、序列规则和阈值规则有效。该元素指定当检测到规则模式时执行的操作或操作集合。

详细信息

表 48 描述在 <onDetection> 操作有效时，如何为每个规则类型检测规则模式。

表 48. 如何根据规则类型检测规则模式

规则类型	如何检测规则模式
重复	当接收到第一个符合事件选择条件的事件时，表示检测到该规则模式。
过滤	当接收到任意一个符合事件选择条件的事件时，表示检测到该规则模式。
序列	在时间窗口中按照适当顺序接收到一组符合事件选择条件的事件时，表示检测到该规则模式。
阈值	在时间窗口中，在达到阈值的情况下接收到符合事件选择条件的事件时，表示检测到该规则模式。

属性

<onDetection> 不具有属性。

包含于

<onDetection> 包含于以下元素中:

- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>

包含

<onDetection> 包含以下元素:

表 49. <onDetection> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onLoad 元素

<onLoad> 元素指定正在运行的活动关联技术引擎中装入（或部署）规则时执行的操作或操作集合。<onLoad> 操作在装入规则之后但在规则开始任何处理之前进行调用。

属性

<onLoad> 不具有属性。

包含于

<onLoad> 包含于以下元素中:

- <lifeCycleActions>

包含

<onLoad> 包含以下元素:

表 50. <onLoad> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onNextEvent 元素

<onNextEvent> 元素仅对于重复规则有效。该元素指定重复规则在指定的时间窗口中接收到符合事件选择条件的第二个以及每个后续事件时，所执行的操作或操作集合。

详细信息

对于重复规则，活动关联技术引擎将跳过对在指定时间窗口中与事件选择条件匹配的第二个和每个后续事件的规则集处理。因此，编写 <onNextEvent> 操作的唯一原因在于为第二个和每个后续事件指定备用处理。

属性

<onNextEvent> 不具有属性。

包含于

<onNextEvent> 包含于以下元素中:

- <duplicateRule>

包含

<onNextEvent> 包含以下元素:

表 51. <onNextEvent> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onTimeOut 元素

<onTimeOut> 元素仅对于序列规则和阈值规则有效。该元素指定规则的时间窗口到期时执行的操作或操作集合。

详细信息

第 91 页的表 52 描述对于每个规则类型，在 <onTimeOut> 操作有效时，时间窗口如何到期。

表 52. 时间窗口如何根据规则类型到期

规则类型	时间窗口如何到期
序列	如果在时间窗口中接受了一个或多个事件但未接收到完整序列的事件，则时间窗口到期。
阈值	如果在时间窗口中接受了一个或多个事件但未达到阈值，则时间窗口到期。

如果规则集包含在相同日期和时间（或者由相同事件）激活的多个规则，且具有相同时间窗口，则运行这些规则的以下操作的时间并不完全相同:

- <onTimeOut> 和 <onTimeWindowComplete> 元素中的规则响应操作
- <onActivation> 和 <onDeactivation> 元素中的生命周期操作

这些操作按照任一顺序连续运行。它们未必按照在规则集中对它们进行编写的顺序运行。由于必须在完成每个操作之后，才能开始序列中的下一操作，所以这些操作的运行时间不同。

属性

<onTimeOut> 不具有属性。

包含于

<onTimeOut> 包含于以下元素中:

- <sequenceRule>
- <thresholdRule>

包含

<onTimeOut> 包含以下元素:

表 53. <onTimeOut> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onTimeWindowComplete 元素

<onTimeWindowComplete> 元素仅对于收集规则、计算规则、重复规则和定时器规则有效。该元素指定当规则的时间窗口结束时执行的操作或操作集合。

详细信息

如果规则集包含在相同日期和时间（或者由相同事件）激活的多个规则，且具有相同时间窗口，则运行这些规则的以下操作的时间并不完全相同:

- <onTimeOut> 和 <onTimeWindowComplete> 元素中的规则响应操作
- <onActivation> 和 <onDeactivation> 元素中的生命周期操作

这些操作按照任一顺序连续运行。它们未必按照在规则集中对它们进行编写的顺序运行。由于必须在完成每个操作之后，才能开始序列中的下一操作，所以这些操作的运行时间不同。

属性

<onTimeWindowComplete> 不具有属性。

包含于

<onTimeWindowComplete> 包含于以下元素中:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <timerRule>

包含

<onTimeWindowComplete> 包含以下元素:

表 54. <onTimeWindowComplete> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

onUnload 元素

<onUnload> 元素指定从正在运行的活动关联技术引擎中卸装（或除去）规则时执行的操作或操作集合。<onUnload> 操作在卸装规则之前立即调用。

属性

<onUnload> 不具有属性。

包含于

<onUnload> 包含于以下元素中:

- <lifeCycleActions>

包含

<onUnload> 包含以下元素:

表 55. <onUnload> 元素中包含的元素

元素	必需或可选?
<action>	可选。允许出现 0 次或多次。

ruleBlock 元素

<ruleBlock> 元素用于将相关规则进行分组并将规则组织为层次结构。

属性

<ruleBlock> 具有以下属性:

表 56. <ruleBlock> 元素的属性

名称	描述	数据类型	必需?
name	标识规则块。该标识在包含该规则块的规则集或规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是

包含于

<ruleBlock> 包含于以下元素中:

- <ruleSet>
- <ruleBlock>

包含

<ruleBlock> 包含以下元素。

如已编写，则必须按显示的顺序编写 <comment>、 <import> 和 <variable> 元素。剩余元素可按任意顺序进行编写。

表 57. <ruleBlock> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<import>	可选。允许出现 0 次或多次。
<variable>	可选。允许出现 0 次或多次。
<ruleBlock>	可选。允许出现 0 次或多次。
<collectionRule>	可选。允许出现 0 次或多次。
<computationRule>	可选。允许出现 0 次或多次。
<duplicateRule>	可选。允许出现 0 次或多次。
<filterRule>	可选。允许出现 0 次或多次。
<sequenceRule>	可选。允许出现 0 次或多次。
<thresholdRule>	可选。允许出现 0 次或多次。
<timerRule>	可选。允许出现 0 次或多次。

ruleSet 元素

由 act:ruleSet 定义的 <ruleSet> 元素为活动关联技术规则语言的根元素。所有其他元素均包含于该 <ruleSet> 元素中。

详细信息

由活动关联技术语言模式（act:ruleSet）和活动关联技术基础规则集模式（br:ruleSet）定义的 <ruleSet> 元素是相同的。不过，在创建规则集时，必须在 <ruleSet> 元素中指定以下名称空间: act:ruleSet。

属性

<ruleSet> 具有以下属性:

表 58. <ruleSet> 元素的属性

名称	描述	数据类型	必需?
name	标识规则集。该标识必须是唯一的。不得包含句点。	xsd:NMTOKEN	是

包含于

由于 <ruleSet> 为规则语言的根元素，因此不包含于任何元素中。

包含

<ruleSet> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 59. <ruleSet> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<import>	可选。允许出现 0 次或多次。
<variable>	可选。允许出现 0 次或多次。
<ruleBlock>	可选。允许出现 0 次或多次。

runUntilDeactivated 元素

<runUntilDeactivated> 元素指定在停用规则前，时间窗口将一直保持打开。因此，该规则的时间窗口将在该规则开始处理时启动，且在停用该规则、从规则集除去该规则或活动关联技术引擎关闭之前不停止。

详细信息

包含 <runUntilDeactivated> 元素的规则的特定行为取决于规则类型。表 60 描述每个规则类型（其中 <timeWindow> 元素有效并包含 <runUntilDeactivated> 元素）的规则行为。

表 60. 编写了 <runUntilDeactivated> 时的规则行为

规则类型	编写了 <runUntilDeactivated> 时的规则行为
收集	收集规则将接受符合其事件选择条件的第一个事件，并在停用该规则之前继续接受并处理事件，当该规则停用时，<onTimeWindowComplete> 元素中定义的操作将运行，且 <onDeactivation> 元素中定义的操作也将紧随其后运行。
计算	计算规则将接受符合其事件选择条件的第一个事件，并在停用该规则之前继续接受并处理事件，当该规则停用时，<onTimeWindowComplete> 元素中定义的操作将运行，且 <onDeactivation> 元素中定义的操作也将紧随其后运行。

表 60. 编写了 <runUntilDeactivated> 时的规则行为 (续)

规则类型	编写了 <runUntilDeactivated> 时的规则行为
重复	重复规则将接受符合其事件选择条件的第一个事件，并在停用该规则之前继续接受并处理事件，当该规则停用时，<onTimeWindowComplete> 元素中定义的操作将运行，且 <onDeactivation> 元素中定义的操作也将紧随其后运行。
序列	序列规则将接受符合其事件选择条件的第一个事件，并在发生以下任一情况之前继续接受并处理事件: • 检测到序列模式。当发生该情况时，<onDetection> 元素中定义的操作将运行，且规则将返回至其初始状态。该规则将再次开始处理事件，且在停用该规则前，该过程可重复多次。 • 规则在处理事件时停用。当发生该情况时，<onTimeOut> 元素中定义的操作将运行，且 <onDeactivation> 元素中定义的操作将紧随其后运行。
阈值	阈值规则将接受符合其事件选择条件的第一个事件，并在发生以下任一情况之前继续接受并处理事件: • 检测到阈值模式。当发生该情况时，<onDetection> 元素中定义的操作将运行，且规则将返回至其初始状态。该规则将再次开始处理事件，且在停用该规则前，该过程可重复多次。 • 规则在处理事件时停用。当发生该情况时，<onTimeOut> 元素中定义的操作将运行，且 <onDeactivation> 元素中定义的操作将紧随其后运行。
定时器	在定时器规则处于活动状态后直至将其停用前，该规则将不执行任何操作，该规则停用时，<onTimeWindowComplete> 元素中定义的操作将运行，且 <onDeactivation> 元素中定义的操作将紧随其后运行。<timerRule> 元素中的 repeat 属性将被忽略。

属性

<runUntilDeactivated> 无属性。

包含于

<runUntilDeactivated> 包含于以下元素中:

- <timeWindow>

包含

<runUntilDeactivated> 不包含元素。

sequenceRule 元素

<sequenceRule> 元素将根据序列模式来定义规则。序列规则是唯一允许多个事件选择器的规则。最少需要两个事件选择器。

属性

<sequenceRule> 具有以下属性:

表 61. <sequenceRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	确定规则是接收所有事件还是仅接收从其他规则转发的事件。缺省值为 `false`，表明规则将接收所有事件，包括从其他规则转发的事件。	xsd:boolean	否
arrivalOrder	定义事件是否必须按为规则编写 <eventSelector> 元素代码的顺序到达。有效值为: • `inOrder`（缺省值） • `randomOrder`	xsd:string	否

如果 arrivalOrder 属性的值为 `randomOrder`，则编写 <eventSelector> 元素的顺序十分重要。带有最为具体的事件选择条件的 <eventSelector> 元素应在带有不太具体的事件选择条件的 <eventSelector> 元素之前进行编写。否则，应检测到序列时检测不到序列。

例如，假定以下情况:

- 定义了 3 个 <eventSelector> 元素。
- 第 1 个 <eventSelector> 元素将检查事件 `eventA`。
- 第 2 个 <eventSelector> 元素将检查任一事件。
- 第 3 个 <eventSelector> 元素将检查事件 `eventB`。
- 系统在指定的时间窗口中存在下列事件: `eventA`、 `eventB`、`eventC`。

规则行为如下，结果应检测到序列时检测不到序列:

1. 第 1 个 <eventSelector> 元素接受第 1 个事件 `eventA`。
2. 第 2 个 <eventSelector> 元素接受第 2 个事件 `eventB`。
3. 忽略第 3 个事件 `eventC`。

假定以下情况，其中已正确编写 <eventSelector> 元素代码，并且最具体的事件选择条件列在不太具体的事件选择条件之前:

- 第 1 个 <eventSelector> 元素将检查事件 `eventA`。
- 第 2 个 <eventSelector> 元素将检查事件 `eventB`。
- 第 3 个 <eventSelector> 元素将检查任一事件。

规则行为如下，结果检测到序列:

1. 第 1 个 <eventSelector> 元素接受第 1 个事件 `eventA`。
2. 第 2 个 <eventSelector> 元素接受第 2 个事件 `eventB`。
3. 第 3 个 <eventSelector> 元素接受第 3 个事件 `eventC`。

包含于

<sequenceRule> 包含于以下元素中:

- <ruleBlock>

包含

<sequenceRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 62. <sequenceRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<eventSelector>	对于序列规则，要求该元素出现 2 次。允许出现更多次。
<groupingKey>	可选。允许出现 0 或 1 次。
<timeWindow>	必需。仅允许出现 1 次。
<onDetection>	可选。允许出现 0 或 1 次。
<onTimeOut>	可选。允许出现 0 或 1 次。

相关概念

第 12 页的『序列模式』
序列规则是由序列模式定义的。它将检测某个时间间隔内是否到达了一系列特定事件。序列可有序，也可随机。序列规则为全状态规则。

start 元素

<start> 元素定义是在特定日期的特定时间激活规则，还是在活动关联技术引擎装入规则时激活规则。

详细信息

如果没有编写 <start> 元素，则缺省开始时间与 <whenLoaded> 元素定义的时间相同。

属性

<start> 不具有属性。

包含于

<start> 包含于以下元素中:

- <activationTime>

包含

<start> 包含以下元素:

表 63. <start> 元素中包含的元素

元素	必需或可选?
<dateTime>	必须含有这些元素的其中 1 个，且选定元素仅允许出现 1 次。
<whenLoaded>	
<inactiveWhenLoaded>	

stop 元素

<stop> 元素定义是在特定日期的特定时间停用规则、在特定持续时间之后停用规则，还是在特定时间始终不停用。

详细信息

如果没有编写 <stop> 元素，则缺省停止时间与 <never> 元素定义的时间相同。

属性

<stop> 元素不具有属性。

包含于

<stop> 元素包含于以下元素中:

- <activationTime>

包含

<stop> 元素包含以下元素:

表 64. <stop> 元素中包含的元素

元素	必需或可选?
<dateTime>	必须含有这些元素的其中 1 个，且选定元素仅允许出现 1 次。
<never>	
<after>	

stopAfter 元素

<stopAfter> 元素指定由 <groupingKey> 元素定义的规则实例在开始处于活动状态后继续保持该状态的持续时间。在该持续时间过后，将停用该规则实例。

属性

<stopAfter> 元素具有以下属性:

表 65. <stopAfter> 元素的属性

名称	描述	数据类型	必需?
duration	指定持续时间的长度。该属性的数据类型取决于 unit 属性的值。	• 如果 unit 属性的值为 ISO-8601，则该数据类型为 xsd:duration。 • 如果 unit 属性的值为 milliseconds，则该数据类型为 xsd:positiveInteger。	是
unit	指定要使用的时间单位。该属性的有效值为: • ISO-8601 • milliseconds	xsd:string	是

对持续时间使用 ISO 8601 标准

将 ISO-8601 编码为 unit 属性的值，则表明根据 ISO 8601 标准对 duration 属性的值进行编码，以将持续时间指定为一个字符串。标准 XML 模式数据类型规范将使用 ISO 8601 来提供名为 duration 的数据类型。将在 http://www.w3.org/TR/xmlschema-2/#duration 中对该数据类型作详细描述。

标准 XML 模式中的 duration 数据类型的格式为以下字符串:

P*n*Y*n*M*n*DT*n*H*n*M*n*S

- P 为字符，字符串始终以该字符开头。
- *n*Y 代表年数。 1 年等同于 365 天。因此，编码为 1Y 等同于编码为 365D。
- *n*M 代表月数。 1 个月等同于 30 天。因此，编码为 1M 等同于编码为 30D。
- *n*D 代表天数。
- T 为分隔符，用于将日期单位（年、月、日）与时间单位（小时、分、秒）分隔开。时间单位始终跟随在 T 之后。
- *n*H 代表小时数。
- *n*M 代表分钟数。
- *n*S 代表秒数。

以下为该格式的示例:

- P5DT12H 为 5.5 天。
- PT59M59S 为 59 分 59 秒。
- P1M 为 1 个月。

包含于

<stopAfter> 包含于 <activateOnEvent> 元素 中（仅当 <activateOnEvent> 在 <activationByGroupingKey> 元素中时）。

包含

<stopAfter> 不包含元素。

thresholdRule 元素

<thresholdRule> 元素根据阈值模式来定义规则。

属性

<thresholdRule> 具有以下属性:

表 66. <thresholdRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	确定规则是接收所有事件还是仅接收从其他规则转发的事件。缺省值为 `false`，表明规则将接收所有事件，包括从其他规则转发的事件。	xsd:boolean	否

包含于

<thresholdRule> 包含于以下元素中:

- <ruleBlock>

包含

<thresholdRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 67. <thresholdRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<eventSelector>	可选。允许出现 0 或 1 次。
<groupingKey>	可选。允许出现 0 或 1 次。
<booleanThreshold>	必须含有这些元素的其中 1 个，且选定元素仅允许出现 1 次。
<computedThreshold>	
<eventCountThreshold>	
<timeWindow>	必需。仅允许出现 1 次。
<onDetection>	可选。允许出现 0 或 1 次。

表 67. <thresholdRule> 元素中包含的元素 (续)

元素	必需或可选?
<onTimeOut>	可选。允许出现 0 或 1 次。

相关概念

第 14 页的『阈值模式』
阈值规则是由阈值模式定义的。该规则将在某个时间间隔内收集一组选定的事件，并在 接收到每个事件后确定是否已符合阈值条件。该规则为全状态规则。

timeInterval 元素

<timeInterval> 元素指定时间窗口的持续时间。

属性

<timeInterval> 具有以下属性:

表 68. <timeInterval> 元素的属性

名称	描述	数据类型	必需?
duration	指定持续时间的长度。该属性的数据类型取决于 unit 属性的值。	• 如果 unit 属性的值为 ISO-8601，则该数据类型为 xsd:duration。 • 如果 unit 属性的值为 milliseconds，则该数据类型为 xsd:positiveInteger。	是
unit	指定要使用的时间单位。该属性的有效值为: • ISO-8601 • milliseconds	xsd:string	是

对持续时间使用 ISO 8601 标准

将 ISO-8601 编码为 unit 属性的值，则表明根据 ISO 8601 标准对 duration 属性的值进行编码，以将持续时间指定为一个字符串。标准 XML 模式数据类型规范将使用 ISO 8601 来提供名为 duration 的数据类型。将在 http://www.w3.org/TR/xmlschema-2/#duration 中对该数据类型作详细描述。

标准 XML 模式中的 duration 数据类型的格式为以下字符串:

P*n*Y*n*M*n*DT*n*H*n*M*n*S

- P 为字符，字符串始终以该字符开头。
- *n*Y 代表年数。 1 年等同于 365 天。因此，编码为 1Y 等同于编码为 365D。
- *n*M 代表月数。 1 个月等同于 30 天。因此，编码为 1M 等同于编码为 30D。
- *n*D 代表天数。
- T 为分隔符，用于将日期单位（年、月、日）与时间单位（小时、分、秒）分隔开。时间单位始终跟随在 T 之后。
- *n*H 代表小时数。

- *n*M 代表分钟数。
- *n*S 代表秒数。

以下为该格式的示例:

- P5DT12H 为 5.5 天。
- PT59M59S 为 59 分 59 秒。
- P1M 为 1 个月。

包含于

<timeInterval> 包含于以下元素中:

- <timeWindow>

包含

<timeInterval> 不包含元素。

timerRule 元素

<timerRule> 元素将根据定时器模式来定义规则。

属性

<timerRule> 具有以下属性:

表 69. <timerRule> 元素的属性

名称	描述	数据类型	必需?
name	标识规则。该标识在包含该规则的规则块中必须是唯一的。不得包含句点。	xsd:NMTOKEN	是
processOnlyForwardedEvents	由于定时器规则不处理事件,因此忽略该属性。	xsd:boolean	否
repeat	定义在停用定时器规则之前该规则是否重复运行。有效值为: • true(缺省值) • false 如果将该值设置为 false,则该规则在其时间间隔内仅运行一次,并在相应的时间窗口完成后执行规则响应操作,然后停止。 如果定时器规则的 <timeWindow> 元素包含 <runUntilDeactivated> 元素,则忽略 repeat 属性。	xsd:boolean	否

包含于

<timerRule> 包含于以下元素中:

- <ruleBlock>

包含

<timerRule> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 70. <timerRule> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<variable>	可选。允许出现 0 次或多次。
<activationInterval>	可选。允许出现 0 或 1 次。
<lifeCycleActions>	可选。允许出现 0 或 1 次。
<timeWindow>	必需。仅允许出现 1 次。
<onTimeWindowComplete>	可选。允许出现 0 或 1 次。

相关概念

第 16 页的『定时器模式』
定时器规则由定时器模式定义。该规则按照规定的时间间隔启动操作。该规则为全状态规则。虽然定时器规则不处理事件，但事件可对其进行激活或停用。

timeWindow 元素

<timeWindow> 元素包含定义规则进行处理的时间间隔的元素。

详细信息

例如，重复规则的时间窗口定义规则必须检查事件的时间长度，这些事件为所接受的第一个事件的重复事件。如果该时间窗口为 30 秒，则重复规则将处理它接受第一个事件的 30 秒内发生的所有重复事件。

属性

<timeWindow> 不具有属性。

包含于

<timeWindow> 包含于以下元素中:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

包含

<timeWindow> 包含以下元素:

表 71. <timeWindow> 元素中包含的元素

元素	必需或可选?
<timeInterval>	必须含有这些元素的其中 1 个，且选定元素仅允许出现 1 次。
<runUntilDeactivated>	

variable 元素

<variable> 元素定义变量，并按可由表达式引用的格式来包含信息。变量可在规则集、规则块或规则级别定义。

详细信息

规则集变量

全局应用于规则集并可由该规则集中的任意表达式引用。

规则块变量

仅在规则块（和所有包含的规则块）中应用，并可由该规则块中的任意表达式引用。

规则变量

仅应用于该规则中的表达式。

在规则层次结构中，不同级别的变量可具有相同的名称。当访问某个变量时，将使用该变量最为具体的定义。例如，如果使用相同的名称以规则集、规则块和规则的级别来定义变量，则该规则中的表达式将使用位于规则级别的变量定义。

如果以规则集或规则块的级别来定义变量，则多个规则将在不同时间获取和设置这些变量。因此，为了确保正确维护变量值，请对如何编写规则集中变量间的交互加以注意。

如果在规则集或规则块级别定义变量，则在规则模式匹配后不会重新初始化该变量。

在以下任一情况中，请对规则集和规则块变量的获取和设置使用锁定功能，以避免变量值设置不正确:

- 如果定时器规则在 <onTimeOut> 操作期间获取或设置变量
- 如果活动关联技术引擎所嵌入至的应用程序为多线程

如果使用组密钥定义某个规则，则 <variable> 元素定义的规则变量在生命周期操作或 <filteringPredicate> 元素（包含于 <activateOnEvent> 元素或 <deactivateOnEvent> 元素，这两个元素包含于 <activationInterval> 元素）中无效。这是由于在这种情况下，规则变量仅应用于规则实例，而在这些表达式运行时不存在规则实例。

属性

<variable> 具有以下属性:

表 72. <variable> 元素的属性

名称	描述	数据类型	必需?
name	标识特定变量。将名称引用变量。	xsd:NMTOKEN	是
dataType	标识变量所含信息的类型。必须为标准数据类型，例如 java.lang.String。	xsd:NMTOKEN	是

变量的名称限制

变量名称具有某些限制。因此，<variable> 元素上名称属性的值具有以下限制:

- 它仅可包含以下字符:
  - 大写 ASCII 拉丁字母 A-Z。Unicode 表示法为 \u0041-\u005a。
  - 小写 ASCII 拉丁字母 a-z。Unicode 表示法为 \u0061-\u007a。
  - ASCII 下划线（_）。 Unicode 表示法为 \u005f。
  - 美元符号（$）。Unicode 表示法为 \u0024。
  - ASCII 数字 0-9。Unicode 表示法为 \u0030-\u0039。
- 它不得为空。
- 不得为空字符串。
- 不得包含任何空格。
- 不得包含句点。
- 不得以任何形式（大写、小写或大小写混合形式） 的 act_ 开头。

包含于

<variable> 包含于以下元素中:

- <ruleSet>
- <ruleBlock>
- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

包含

<variable> 包含以下元素。

必须将这些元素按所显示的顺序进行编码。如果某个元素为可选元素，则可以不对其进行编码，但所有已编码的元素都必须按照正确的顺序。

表 73. <variable> 元素中包含的元素

元素	必需或可选?
<comment>	可选。允许出现 0 或 1 次。
<varInitializer>	必需。允许出现 1 次。

相关概念

第 21 页的『变量』
在规则语言中，某些变量用于存储涉及不同事件或规则的事件相关信息。 随后可通过规则内的表达式来访问这些与事件相关的信息。某些类型的变量由规则编写者定义，而其他类型的变量则由活动关联技术提供。某些类型可在表达式中直接访问，而其他类型则只能通过由活动关联技术提供的方法来访问。

varInitializer 元素

<varInitializer> 元素包含为关联的 <variable> 元素中定义的变量提供初始值的表达式。

详细信息

由于变量可为任意类型，因此该表达式代码可返回数组对象或任何其他由活动关联技术引擎存储的复杂的、特定于实现的对象。

有关可在表达式中使用的变量的信息，请参阅第 21 页的『变量』。某些变量的使用要取决于表达式的上下文。

属性

<varInitializer> 具有以下属性:

表 74. <varInitializer> 元素的属性

名称	描述	数据类型	必需?
expressionLanguage	标识编写表达式所使用的编程语言。由于 Java 编程语言是唯一受支持的表达式语言，因此该属性的唯一有效值为 `java`。	xsd:NMTOKEN	是

包含于

<varInitializer> 包含于以下元素中:

- <variable>

包含

<varInitializer> 不包含元素。

相关概念

第 17 页的『表达式』
表达式是包含定制逻辑的代码，可将该逻辑添加至规则。 表达式还可访问活动关联技术引擎外部的代码。在规则语言中，表达式仅在特定上下文或规则语言元素 中有效。

whenLoaded 元素

<whenLoaded> 元素指定当活动关联技术引擎装入某个规则时，将激活该规则。

属性

<whenLoaded> 不具有属性。

包含于

<whenLoaded> 包含于以下元素中:

- <start>

包含

<whenLoaded> 不包含元素。

第 6 章 词汇表

本词汇表包含活动关联技术中重要概念的术语和定义。

表达式（expression）
包含定制逻辑的代码，可将该逻辑添加至规则。规则编写者可将表达式用于不同用途；例如，变量的初始化、事件选择条件的定义或者规则响应操作和生命周期操作的指定。

表达式语言（expression language）
编写表达式所使用的编程语言。

操作（action）
作为规则响应的一部分运行或者在装入、卸装、激活或停用规则时运行的表达式。

导入（import）
特定于编程语言的方式，使表达式可访问外部代码。

定时器模式（timer pattern）
规则模式，用于定义规则按照固定时间间隔启动操作。由定时器模式定义的规则为全状态规则。虽然定时器规则不处理事件，但事件可对其进行激活或停用。

重复模式（duplicate pattern）
规则模式，用于定义规则计算指定时间间隔内接受的第二个和后续的事件，但跳过对这些事件的规则集处理。由重复模式定义的规则为全状态规则。

规则（rule）
用于识别事件之间的关系并运行相应规则响应的关联单元。一个规则是 7 种规则模式之一的一种实施，按功能将规则组织成规则块，规则块是规则集的一部分。如果某个事件符合事件选择条件，则规则将接受该事件以进行处理。

规则集（rule set）
活动关联技术规则语言的规则执行单元。规则集包含规则，组织成规则块，由活动关联技术引擎来执行。在某个给定时间，引擎仅对一个规则集执行操作。

规则块（rule block）
规则集中根据功能将规则分组到域中的组织单元。规则块不仅可包含规则，而且可包含其他规则块。

规则模式（rule pattern）
事件相关情境（例如，阈值条件或复制事件检测）的表示法。活动关联技术规则语言包括以下规则模式：收集、计算、重复、过滤、序列、阈值和定时器。当发生某个规则定义的情境时，将与该规则的模式匹配。当该模式匹配时，该规则将执行相应的规则响应操作来结束其处理。在规则处于活动状态时，该规则模式可发生多次匹配。

规则实例（rule instance）
在组密钥的上下文中，为规则的副本。

规则响应（rule response）
当活动关联技术引擎识别出已符合规则条件时所运行的表达式。规则响应由一个或多个操作组成。

规则响应操作（rule response action）
请参阅操作。

过滤模式（filter pattern）
规则模式，用于定义规则在接受某个事件时执行特定操作。由过滤模式定义的规则仅在单个事件上执行操作，因此为无状态规则。

过滤谓词（filtering predicate）
用于定义特定条件的表达式，规则根据该条件接受事件以进行处理。过滤谓词为事件选择器的一部分。过滤谓词将返回布尔值。

活动关联技术
通过规则提供事件相关的 IBM 技术。

活动关联技术编译器（Active Correlation Technology compiler）
活动关联技术组件，用于对规则集及其所含代码进行语法分析，以生成活动关联技术引擎所需的内部数据结构。

活动关联技术规则构建器（Active Correlation Technology rule builder）
GUI，用于使用活动关联技术规则语言来编写关联规则。

活动关联技术规则语言（Active Correlation Technology rule language）
基于 XML 的语言，用于编写关联事件的规则。随后可将这些规则部署至活动关联技术运行时环境。

活动关联技术引擎（Active Correlation Technology engine）
活动关联技术组件，用于根据活动关联技术编译器的输出来处理事件。

活动关联技术运行时环境（Active Correlation Technology runtime environment）
活动关联技术引擎嵌入至的应用程序，带有或不带有编译器。

计算模式（computation pattern）
规则模式，它定义的规则将在某个时间间隔内接收到每个事件时（通过表达式）将计算应用至收集的事件。由计算模式定义的规则为全状态规则。

节点（node）
规则层次结构中的对象，可通过个别而独立的方式将该对象添加至规则集、从中除去或在其中进行替换。具体而言，以下对象为节点:

- 规则集变量
- 规则
- 规则块
- 规则块变量

由于无法在规则级别以下以个别而独立的方式对某个对象进行操作，因此规则变量不是节点。

内部事件（internal event）
由活动关联技术引擎中正在运行的规则所创建的事件。该事件可能被转发至其他规则。

片段（snippet）
源代码的节选。

全状态规则（stateful rule）

保留状态信息（有关规则实例特性的信息）的规则，用于在某个时间段内对一组事件执行操作。以下任何规则模式定义的规则均为全状态规则：收集、计算、重复、序列、阈值或定时器。

生命周期操作（life cycle action）

在装入、卸装、激活或停用规则时运行的表达式。

事件提供程序（event provider）

用于生成由活动关联技术处理的事件的任何软件。

事件选择器（event selector）

事件选择的条件。这些条件将确定由规则接受以进行处理的事件。事件选择器包括事件类型和过滤谓词。

收集模式（collection pattern）

规则模式，定义用于在某个时间间隔内收集一组选定事件的规则。由收集模式定义的规则为全状态规则。

外部对象（external object）

应用程序创建用来与表达式通信的对象。

外部事件（external event）

活动关联技术引擎从其外部的源所接收的事件。

谓词（predicate）

请参阅过滤谓词。

无状态规则（stateless rule）

不保留状态信息的规则，因此在某个时间仅可对一个事件执行操作。由过滤模式定义的规则为无状态规则。

响应（response）

请参阅规则响应。

序列模式（sequence pattern）

规则模式，用于定义规则来检测某个时间间隔内是否存在特定的事件序列。序列可有序，也可随机。由序列模式定义的规则为全状态规则。

域（domain）

一组规则根据自身的功能所应用至的类别。例如，域可代表具体的地区、IT 管理规程（例如，安全性检测或网络事件相关）或业务机构（例如，具体的公司或分公司）。

阈值模式（threshold pattern）

规则模式，用于定义规则在某个时间间隔内收集一组选定事件，并在接收到每个事件后确定是否已符合阈值条件。由阈值模式定义的规则为全状态规则。

组密钥（grouping key）

该方法用于指定规则针对每组共享公共特性的事件创建不同的规则实例（或其副本）。

ACT 请参阅活动关联技术。

附录. 声明

本信息是为在美国提供的产品和服务编写的。IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可证查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区:

International Business Machines Corporation “按现状” 提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。

某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本出版物的新版本中。IBM 可以随时对本资料中描述的产品和／或程序进行改进和／或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

商标

DB2、IBM、IBM 徽标、Tivoli、Tivoli 徽标、Tivoli Enterprise Console 和 WebSphere 是 International Business Machines Corporation 在美国和／或其他国家或地区的商标或注册商标。

Java 和所有基于 Java 的商标与徽标是 Sun Microsystems, Inc. 在美国和／或其他国家或地区的商标或注册商标。

其他公司、产品和服务名称可能是其他公司的商标或服务标记。

Printed in China