Esper参考书

说明：

* 文档中字体颜色为橙色的部分是翻译有待确认的部分。

# 技术总览

## 复杂事件处理（CEP）和事件流分析

Esper引擎是为了满足分析和响应事件的应用而开发的。一些典型的应用示例如下：

* 业务流程的管理及自动化（流程监控、业务活动监控（BAM：Business Activity Monitoring）、异常报告）
* 金融（规则交易、欺诈侦查、风险管理）
* 网络和应用监控（闯入侦查、服务等级协议（SLA）监控）
* 传感器网络应用（射频识别（RFID）、流水线安排和控制、空中交通）

上述应用的共同点是对实时或准实时（近实时）事件处理的需求。它有时被称为复杂事件处理（CEP）或事件流分析。这一类型的应用考虑的关键因素是吞吐量、延迟、逻辑需求的复杂性。

* 高吞吐量-处理大量信息（每秒1000到100000）
* 低延迟-发生事件的实时响应（几毫秒到几秒）
* 复杂运算-在事件（事件关联）、事件过滤、时间或长度窗口的事件集、事件流交集、基于事件缺失的触发器等逻辑中匹配模式

Esper引擎就是为了更容易地创建和扩展复杂事件处理的应用而设计的。

## 复杂事件处理（CEP）和关系型数据库

关系型数据库和结构化查询语言（SQL）设计的目的是为了满足那些数据几乎都是静态的，或者很少有复杂查询的应用。另外，大部分数据库把所有数据存储在磁盘中（内在数据库除外），因此它们需要优化磁盘访问。

从数据库中获取数据，必须触发一次查询。如果一个应用每秒有10次需要某数据，那它必须每秒触发10次查询。当然还有可能需要每秒成百上千次的查询。

数据库触发器能用来响应数据库更新事件，然而，它却很慢，并且很难胜任复杂情况的检查和逻辑响应。

相比传统关系型数据库，内存数据库也许更适合复杂事件处理的应用，因为它们有很好的性能。然而，它们也不能满足复杂事件处理和事件流分析对实时查询结果的需求。

## 为复杂事件处理（CEP）而生的Esper引擎

Esper引擎的工作原理有点像数据库的反置。数据库往往保存数据，在这些数据中执行查询，而Esper引擎允许则是保存查询，将数据从查询中流过。当匹配查询的事件发生时，Esper引擎的响应是实时的。很明显，相对于只有提交查询才能有响应，这种运行模式更具有连续性。

Esper引擎提供了两种主要的方法（机制）来处理事件：事件模式（Event Patterns）和事件流查询（Event Stream Queries）。

Esper提供了一种事件模式语言来指定基于表达式的事件模式匹配。底层的模式匹配引擎是状态机的一种实现。这种事件处理方法，能匹配预期的事件出现或者缺失的序列或事件排列，包括事件基于时间的关联。

Esper还提供另外一种事件流查询机制，满足复杂事件处理应用对于事件流分析的需求。事件流查询机制提供了对于事件流的窗口、集合、交集和分析功能函数等用法。这些查询都遵循EPL语法。EPL被设计成类似SQL查询的语言，但不能于SQL从表中查询，EPL是从视图（Views）中查询。视图（Views）需要在事件流中安排数据并且从事件流中得到数据。

Esper通过同一套API，提供对上述这两种方法的支持。

## 第三方类库

Esper运行时需要如下第三方的类库：

* ANTLR是解析器生产者，用来解析模式（Patterns）和EPL语法。可以在<http://www.antlr.org>网站上了解更多此类库。这个类库仅在编译时需要。
* CGLIB是快速方法调用的代码生成类库。这个开源的软件是Apache组织下的。
* Apache commons logging是和LOG4J和其他日志API协同工作的日志处理API。Apache commons logging是必须的，但LOG4J不是必须的，可以换成SLF4J或者其他的日志处理工具。

Esper编译和测试时，需要如下第三方的类库：

* JUnit是一个强大的单元测试框架。这个类库仅在创建时（build-time）需要。
* MySQL连接类库，用来测试与数据库整合，在自动化测试时需要。

# 事件表现形式

这个章节讲解事件模型和事件表现形式。

Esper用事件类型*event type*来描述事件的有效信息。

你的应用可以在启动前配置预定义的事件类型或者在运行时通过API或者EPL语法动态地添加事件类型。启动前预定义详见15.4章节，“配置元素（Configuration Items）”，运行时配置API详见14.3.7章节，“运行时配置（Runtime Configuration）”。

EPL的create schema语法是在运行时定义事件类型，详见5.16章节，“定义事件类型：Create Schema（Declaring an Event Type:Create Schema）”。

## 事件包含的Java对象

事件（event）是已发生动作或状态改变的不可再变的记录。事件属性（event properties）描述事件的状态信息。

Esper中，事件可以包含以下Java对象

表格2.1：事件包含的Java对象

|  |  |
| --- | --- |
| **Java类** | **描述** |
| java.lang.Object | 遵循JavaBean惯例，有表getter方法的简单java对象（POJO）；不遵循JavaBean惯例的传统的java类也能作为事件类型。 |
| java.util.Map | Map事件是java.util.Map接口的实现，每一个Map.Entry是一个属性及其值。 |
| org.w3c.dom.Node | XML文档对象模型。 |
| org.apache.axiom.om.OMEocument or OMElement | EsperIO包提供。 |
| 应用程序的类（Application classes） | 通过扩展API添加的事件表现形式。 |

Epser提供了多种事件的表现形式，不需要再创建java类来表现事件。

Esper事件表现形式有如下共同点：

* 支持嵌套，索引，属性映射。如下所述，对嵌套层数没有限制；
* 提供事件类型元数据，包括嵌套的；
* 允许将事件本身或者事件中的部分属性组装成一个新的事件，包括嵌套的属性。Apache Axiom事件类型表现形式是一个例外，它暂时不允许转移事件属性到新的事件中，而是只能转移事件本身。
* Java对象，Map和对象数组能够识别类的继承关系。

Esper API对所有事件表现形式的处理都是一样的，除了一些少数的例外，本章节中会提到。

多种事件表现形式的好处如下：

* 对已经支持某种事件表现形式的应用来说，事件处理前，不必将事件转换成java对象；
* 事件表现形式可变性。当事件表现形式变化后，不必修改语句；
* 事件表现形式交互性。在相同或者不同的语句中，表现形式可以交互使用；
* 使得根据交易、易用性、扩展等做一些权衡？（The choice makes its possible to consciously trade-off performance, ease-of-use, the ability to evolve and effort needed to import or externalize events and use existing event type metadata.）。

## 事件属性

事件属性包含了一个事件的状态信息。事件属性有simple（简单）、indexed（索引）、mapped（映射）、nested（嵌套）几种类型。下表列出了这几种类型的事件属性和他们在表达式中的使用语法。这种语法允许语句从java继承图、xml结构、map映射的事件中查询信息。

表格2.2：事件属性的类型

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **描述** | **语法** | **例子** |
| Simple | 有单一的值的简单属性，这个值可能会被检索。 | name | sensorId |
| Indexed | 索引属性保存了一个有顺序的对象的集合（相同类型）。通过一个整数、非负的索引（或者一个子表达式）进行访问。 | name[index] | sensor[0] |
| Mapped | 映射属性保存的是键值对的集合，通过键访问对应的对象。 | name(“key”) | sensor(“light”) |
| Nested | 嵌套属性保存了对象的另外的一个属性。 | name.nestedname | sensor.value |

上面类型的组合也是可用的，例如，一个合法的组合方式如下：

person.address(“home”).street[0].

可以使用任意合法的表达式作为映射属性的键或者索引属性的索引，可以在下面寻找到相应的例子。

### 转义符

如果你的应用中用Map，Object[]或者XML表现事件，那么事件属性名本身可能含有”.”点符号，那么在属性名中，”\”反斜线符号将做为点符号的转义符来使用。

如下例，某个EPL中可能使用part1.part2作为事件MyEvent的属性，那么相应的写法如下：

select part1\.part2 from MyEvent

有时，你的事件属性名与EPL语言中的关键字相同，或者包含有空格或者其他特殊的字符，这种情况下，”`”后撇符号可以用来做为属性名中的转义符号。

如下例，order作为事件Quote中的一个属性，而order也是EPL语言中的一个关键字，那么写法如下：

select `order` , price as `price.for.goods` from Quote

当使用”`”后撇符号作为映射、索引类型的属性时，要保证”`”后撇符号出现在键或者索引之外。

如下例，EPL语句要从事件中检索的事件属性中包含空格（candidate book），包含特殊符号（children’s books），索引属性（children’s books[0]），和包含关键字的映射属性（book select(‘isbn’)），则写法如下：

select `candidate book` ,`children’s books`[0],`book select`(‘isbn’) from MyEventType

### 表达式作为映射属性的键、索引属性的索引

作为键或者索引的表达式必须包含在”()”圆括号内，作为映射属性的键，表达式必须返回一个String类型的值；作为索引属性的索引，表达式必须返回一个int类型的数字。

下例表明如何使用该规则，该规则适用于所有事件表现形式。

定义一个类，注意所包含的属性。注：为了节约空间，类的getter方法未写出来，自己实现时，注意要有属性的getter方法。

public class MyEventType {

String myMapKey;

int myIndexValue;

int myInnerIndexValue;

Map<String, InnerType> innerTypesMap; // mapped property

InnerType[] innerTypesArray; // indexed property

}

public class InnerType {

String name;

int[] ids;

}

EPL写法如下：

select innerTypesMap('somekey'), // returns map value for 'somekey'

innerTypesMap(myMapKey), // returns map value for myMapKey value (an expression)

innerTypesArray[1], // returns array value at index 1

innerTypesArray(myIndexValue) // returns array value at index myIndexValue(an //expression)

from MyEventType

“.”句点操作符可以用来访问映射或索引对象返回对象的方法，使用方法详见章节8.6，“句点操作符（Dot Operator）”。

使用句点操作符的简单例子如下：

select innerTypesMap('somekey').ids[1],

innerTypesMap(myMapKey).getIds(myIndexValue),

innerTypesArray[1].ids[2],

innerTypesArray(myIndexValue).getIds(myInnerIndexValue)

from MyEventType

请注意以下限制

* 此时的索引属性的”[]”方括号语法也是合法的，可以传入表达式，或者一个常量索引值；
* 当使用句点操作符操作映射属性或者索引属性的属性时，必须遵循方法调用的语法。

## 动态事件属性

动态属性是在语句编译时未知的事件属性，它们将在运行时被识别。

在一些情况下，我们并不知道事件所有的属性。在编译时，可能会有一些附加的未知事件属性，而这些属性又有被检索的需求，这就是动态属性产生的意义，这种意义尤其适用于富面向对象的模型。

动态属性的语法包括属性名和”?”一个问号。索引，映射，嵌套属性也可以有动态属性。

表格2.3：事件属性类型

|  |  |
| --- | --- |
| **类型** | **语法** |
| Dynamic Simple | name? |
| Dynamic Indexed | name[index]? |
| Dynamic Mapped | name(“key”)? |
| Dynamic Nested | name?.nestedname |

动态属性将返回一个Object类型的对象，当然，如果运行行时处理事件时，不存在该属性，那么动态属性将返回null值。

一个简单的例子：OrderEvent事件中有一个”item”属性，它是一个Service或者Product实例的引用。假如Servcie和Product对象都有一个”price”属性。那么就可以通过动态属性获得item的price属性，无论item引用的是Serice实例还是Product实例。

select item.price? from OrderEvent

第二个例子：假设Service有一个“serviceName”属性，而Product没有该属性，那么，如果item引用的是Service实例时，下面这个查询语句将返回它的serviceName属性值，而当item引用的是Product实例时，它将返回一个null值。

select item.serviceName? from OrderEvent

另外一种情况：假设OrderEvent有多种实现类，有些实现类有timestamp属性，而有的实现类没有该属性，那么拥有该属性的实例将返回该属性的值，而没有些属性的实例将返回null值。

select timestamp? from OrderEvent

上述查询将返回名为”timestamp?”的列。

当动态属性被嵌套时，动态属性后面的所有属性都将是动态属性。下面例子：

select detail?.direction from OrderEvent

等效于

select detail?.direction? from OrderEvent

用动态属性连接中常常用到的函数：

* cast函数，将动态属性的值转换成指定类型的对象；
* exsits函数，检查某个动态属性是否存在。如果存在该名称的属性，则返回true，否则返回false；
* instanceof函数，检查动态属性是否为指定类型（指定多个）中的其中一种；
* typeof函数，返回动态属性的类型的类名。

动态属性支持以下所有事件表现形式：java对象，基于Map的，基于对象数组的，基于XML文档的事件类型。

## 分段和分段类型（Fragment and Fragment Type）

有时，事件属性是事件本身。Esper用Fragment和Fragment Type来表现这些事件片段。一个典型的例子：一个模式，匹配两个或两个以上事件，并且输出事件包含匹配的事件。换一种说法：输出事件可以是复杂的事件，包含更即将到来的事件和当前片段？

此章节翻译不通，请详见官方英文文档。

## 简单的java对象事件

简单的java对象事件通过javaBean风格的getter方法得到实例对象的属性值。事件类或者接口，并不一定完全遵循javaBean的规范，但是对于Esper引擎，为了得到事件的属性，必须提供getter方法，如果没有提供getter方法，而想获得属性值，则需要额外的配置

Esper事件类型支持事件类继承父类，或者实现一个或多个接口，并且Esper的Pattern和EPL也可以引用接口或者抽象类型。

用来表现事件的类要设计成不可改变的，因为事件是过去发生的变化和动作状态的记录，相关联的属性也要设计成不可变的。这一点不难实现，并且esper引擎也能接受可变的事件。

hashCode方法和equals方法没有必要实现。这两个方法的实现也不会影响esper引擎。

详见第15章，“配置（Configuration）”，事件类型的命名选项。那些没有完全遵照javaBean规范，或者公共字段、方法命名没规范的java类，需要额外的配置才能来表现事件。通过配置，还可以设定属性名大小写敏感，相关的章节15.4.1.3，“非JavaBean和传统Java事件类（Non-JavaBean and Legacy Java Event Classes）”。

### Java对象事件属性

如上所述，JavaBean规范支持不同类型的属性，但有一些属性是Esepr独家支持的：

* 简单属性，拥有一个可获取的单独的值，属性类型可以是一个java语言的原始类型、引用类型或者复杂的java对象；
* 索引属性，有顺序的对象的集合，通过索引获取对应的值；
* 映射属性，基础JavaBean规范的扩展，Esper可以接受任何以String类型为key的属性为映射属性；
* 嵌套属性，包含另外一个对象的属性，而被包含的对象本身也是事件的属性。

如下例子：假设NewEmployeeEvent是事件类。例子中的映射属性和索引属性能返回java对象或者java原始类型的值。Address和Employee属性本身也有自己的属性，它们自己或许也有name属性，而它们的name属性就是所谓的嵌套属性。

public class NewEmployeeEvent {

public String getFirstName();

public Address getAddress(String type);

public Employee getSubordinate(int index);

public Employee[] getAllSubordinates();

}

简单属性需要getter方法获得属性的值，例子中的getFirstName()方法就是获得firstName属性值的方法。

索引属性需要以下任意一种的getter方法：需要一个int类型的索引值获得属性值的方法（例子中的getSubordinate方法）、获得数组类型的方法、实现遍历方法iterator。例如可以有一个getSubordinates()方法，返回数组或者一个Iterable。在EPL或者Pattern中语句中，访问索引属性都是通过property[index]语法。

映射属性需要通过一个String类型的key得到属性值的getter方法。例如例子中的getAddress方法。在EPL和Pattern语句中，访问映射属性通过property(‘key’)语法。

嵌套属性需要返回嵌套对象的getter方法。例子中的getAddress和getSubordinate是映射或索引属性返回嵌套对象的方法。在EPL和Pattern语句中，通过property.nestedProperty语法获得嵌套属性的值。

EPL和Pattern语句中允许各种属性出现在可以出现的地方，下例展示了各种属性的复合使用方法（每一行是一个EPL）：

every NewEmployeeEvent(firstName='myName')

every NewEmployeeEvent(address('home').streetName='Park Avenue')

every NewEmployeeEvent(subordinate[0].name='anotherName')

every NewEmployeeEvent(allSubordinates[1].name='thatName')

every NewEmployeeEvent(subordinate[0].address('home').streetName='Wat Street')

相同的，属性名也可以出现在EPL语句如select语句、where语句、join语句中，如下例：

select firstName, address('work'), subordinate[0].name, subordinate[1].name

from NewEmployeeEvent

where address('work').streetName = 'Park Ave'

### 属性名

属性名的命名要遵循java基础，可以通过java.beans.Introspector类的getBeanInfo方法根据getter方法获得正确的属性名。另外，Esper可以通过配置设置属性名的大小写是否敏感。下表是列举一些例子：

表格2.4：JavaBean风格的getter方法和属性命名

|  |  |  |
| --- | --- | --- |
| **方法** | **属性名** | **例子** |
| getPrice() | price | select price from MyEvent |
| getNAME() | NAME | select NAME from MyEvent |
| getItemDesc() | itemDesc | select itemDesc from MyEvent |
| getQ() | q | select q from MyEvent |
| getQN() | QN | select QN from MyEvent |
| getqn() | qn | select qn from MyEvent |
| gets() | s | select s from MyEvent |

### 常量和枚举类型

常量是java对象的public static final字段，在EPL和Pattern语句中也可以使用，例如：

select \* from MyEvent where property=MyConstantClass.FIELD\_VALUE

事件属性是枚举类型时，可以用作与枚举值的比较，例如：

select \* from MyEvent where enumProp=EnumClass.ENUM\_VALUE\_1

或者，另外一个选择是静态方法。在枚举类里写一个静态方法获得枚举值，例如：

select \* from MyEvent where enumProp=EnumClass.valueOf('ENUM\_VALUE\_1')

如果你的应用没有通过配置导入包含枚举类的包，那么在使用它时，必须指定该类包含包名的全名。如果该枚举类是内部类，则使用时需要使用” $”符号引用。例如：

select \* from MyEvent(enumProp=org.myorg.MyEvent$Color.GREEN).std:firstevent()

实例的方法也能在EPL和Pattern语句中被调用。例如：

select myevent.computeSomething() as result from MyEvent as myevent

关联实例的方法调用也是被支持的。例如：

select myevent.getComputerFor('books', 'movies').calculate() as result

from MyEvent as myevent

### 参数化类型

如果getter方法或者配置的访问方法返回的是一个参数类型（例如：方法返回的是Iterable<MyEventData>的索引属性或者Map<String,MyEventData>的映射属性），那么，属性表达式中，可以直接引用作为参数化类型的类。例如：

这是一个有参数化类型的属性的类：

public class NewEmployeeEvent {

public String getName();

public Iterable<EducationHistory> getEducation();

public Map<String, Address> getAddresses();

}

该事件合法的属性表达式可以是这样的：

select name, education, education[0].date, addresses('home').street

from NewEmployeeEvent

### 索引和映射属性的setter方法

EPL语句可以更新索引和映射属性的属性值，事件类需要暴露对应的setter方法。

索引属性更新属性值的setter方法，必须名为setPropertyName，并且需要两个参数，一个是int类型的索引值，一个是新的属性值。

映射属性更新属性值的setter方法，必须名为setPropertyName，并且需要两个参数，一个是String类型key值，一个是新的属性值。

下面的例子就是给索引和映射属性提供setter方法的事件类：

public class MyEvent {

private Map props = new HashMap();

private Object[] array = new Object[10];

public void setProps(String name, Object value) {

props.put(name, value);

}

public void setArray(int index, Object value) {

array[index] = value;

}

// ... also provide regular JavaBean getters and setters for all properties

}

更新属性值的语句如下：

update istream MyEventStream set props('key') = 'abc', array[2] = 100

### 已知的限制

Esper利用字节码生成机制来快速访问事件属性，当字节码生成不成功时，引擎会记录一个警告级的日志，并利用java的反射机制来获取事件的属性。

一个已知的限制：如果一个接口有一个常见类型的属性，而实际的事件对象返回该属性类型的一个子类型实例时，引擎将记录一个警告级日志，并利用java反射机制获取该属性的值。

## Map事件

### 概述

实现Map接口的对象也是Esper事件的一种表现形式，Map事件属性的获得是通过Map接口的get方法。

和对象数组一样，Map类型的事件是Esper事件类型系统的组成部分之一，它解除了java对象作为事件类型的必要性，因此使得在运行时改变类型或者从其他资源生成类型信息变得更加容易。

指定的Map类型的事件可以有一个或者多个父类事件类型（父类必须也是Map类型），所有父类的属性和该类型的所有属性都是可用的。并且，在EPL的任何地方，父类的属性名都可用。父类和子类的所有类型事件都会来匹配表达式。

应用可以在运行时通过配置操作updateMapEventType方法为事件类型添加属性。嵌套的属性也能添加。注：事件类型只能添加属性，而不能修改和删除属性。然而，运行时，可能移除Map事件类型，然后再添加一个新的类型信息。

通过指定事件类型名配置一个Map事件类型后，在定义其他的Map或者对象数组事件类型时，就可以用该类型名作为事件属性的类型了。

Map事件类型中，一对多的关系往往通过数组来表现。Map事件类型中的属性可以是原始类型数组、引用类型数组、Map数组或者对象数组（Object-Array）数组。

Esper引擎通过EPRuntime接口的sendEvent(Map map,String eventTypeName)方法处理Map类型的事件。Map中的实体代表事件的属性。Map的key必须是String类型，Esper引擎才能根据EPL和Pattern指定的属性名去查找。

引擎不去验证Map事件中的属性名和属性值，所以你的应用应该保证作为属性传入的对象必须与create schema语句中的属性名和类型或者配置的事件属性和类型匹配。

### Map事件的属性

Map事件属性可以是任意类型。可以是java对象，Map（或者Map数组），对象数组（或者对象数组（Object-Array）数组）。这一特性拥有以下功能：

* Java对象属性可能通过嵌套、索引、映射和动态属性的语法（如前述）进行检索；
* Map类型的属性允许深度的嵌套，所以它能表现非常复杂的信息。嵌套、索引、动态属性的语法可被用来检索Map或者Map数组的属性。
* 对象数组属性也允许深度的嵌套。有Map一样的好外。

为了使用Map事件，Map事件类型名，属性名和属性类型必须为引擎实例所知，可以通过配置或者通过create schema的EPL语法。详见章节5.16，“定义事件类型：Create Schema（Declaring an Event Type:Create schema）”和章节15.4.2，“Map表现事件（Events represented by java.util.Map）”。

正面的代码片段定义了一个Map事件类型并将它注册到引擎中。这个例子通过运行时的接口定义了CarLocUpdateEvent事件类型（也可以通过create shcame或者静态配置）；

// Define CarLocUpdateEvent event type (example for runtime-configuration

interface)

Map<String, Object> def = new HashMap<String, Object>;

def.put("carId", String.class);

def.put("direction", int.class);

epService.getEPAdministrator().getConfiguration().addEventType("CarLocUpdateEvent", def);

定义完成后，就可以在语句中使用它了：

select carId from CarLocUpdateEvent.win:time(1 min) where direction = 1

之后，就可以向引擎中发送些类型的事件了：

// Create a CarLocUpdateEvent event and send it into the engine for processing

Map<String, Object> event = new HashMap<String, Object>();

event.put("carId", carId);

event.put("direction", direction);

epRuntime.sendEvent(event, "CarLocUpdateEvent");

通过Map属性的嵌套，就可以查询Map属性中的java对象的属性了。如下例，Map事件有一个Account类型的属性：

Map event = new HashMap();

event.put("txn", txn);

event.put("account", account);

epRuntime.sendEvent(event, "TxnEvent");

它的查询语句就可以这样写：

select account.id, account.rate \* txn.amount

from TxnEvent.win:time(60 sec)

group by account.id

### Map父类

在引擎初始化时或者通过管理接口配置Map事件类型时，可以为Map事件类型可以定义一个或者多个父类。

父类也必须是Map类型。父类所有的属性和属性类型都可在子类中使用，并且相同名字的属性将会被覆盖。并且，任意属性名被EPL使用，那么所有父类都将会来匹配这个表达式。

下面这个例子，假设BaseUpdate事件类型被定义成AccoutUpdate事件类型的父类：

epService.getEPAdministrator().getConfiguration().

addEventType("AccountUpdate", accountUpdateDef,

new String[] {"BaseUpdate"});

应用中的EPL语中可以从BaseUpdate中检索，语句将匹配BaseUpdate事件和Accountupdate事件，并且匹配BaseUpdate所有父类和子类事件。

// Receive BaseUpdate and any subtypes including subtypes of subtypes

select \* from BaseUpdate

Map事件类型可以多继承，但不允许循环继承。如果在运行时配置，子类添加父类时，该父类必须已经存在。

详见章节15.4.2，“Map表现事件（Events represented by java.util.Map）”。

### 高级Map属性类型

#### 嵌套属性

Map-Map强类型嵌套事件可以用来创建丰富的，类型安全的事件类型。通过Configuration或者ConfigurationOperations初始化或者通过运行时的定义，或者通过Crate schema的EPL语法。

值得注意的几点：

* 普通java类型可能作为Map事件类型的属性；
* 可以用之前注册的Map事件类型的名字作为事件属性的类型；
* 嵌套的深度没有限制；
* Map类型的属性中keys可以通过动态属性的语法访问；
* 访问嵌套结构的路径时，如果实体不存在，引擎将会返回null值。

下面的例子是一个多重嵌套的例子。AccountUpdate是顶层的事件类型，它有一个UpdateFieldType类型的属性，UpdateFieldType类型中又多咱类型的属性。事件类型定义如下：

Map<String, Object> updatedFieldDef = new HashMap<String, Object>();

updatedFieldDef.put("name", String.class);

updatedFieldDef.put("addressLine1", String.class);

updatedFieldDef.put("history", UpdateHistory.class);

epService.getEPAdministrator().getConfiguration().

addEventType("UpdatedFieldType", updatedFieldDef);

Map<String, Object> accountUpdateDef = new HashMap<String, Object>();

accountUpdateDef.put("accountId", long.class);

accountUpdateDef.put("fields", "UpdatedFieldType");

// the latter can also be: accountUpdateDef.put("fields", updatedFieldDef);

epService.getEPAdministrator().getConfiguration().

addEventType("AccountUpdate", accountUpdateDef);

以下代码片段展示向引擎中发送事件：

Map<String, Object> updatedField = new HashMap<String, Object>();

updatedField.put("name", "Joe Doe");

updatedField.put("addressLine1", "40 Popular Street");

updatedField.put("history", new UpdateHistory());

Map<String, Object> accountUpdate = new HashMap<String, Object>();

accountUpdate.put("accountId", 10009901);

accountUpdate.put("fields", updatedField);

epService.getEPRuntime().sendEvent(accountUpdate, "AccountUpdate");

以下语句就是查询语句：

select accountId, fields.name, fields.addressLine1, fields.history.lastUpdate

from AccountUpdate

#### 一对多关系

为了定义多重的属性，你可以用数组类型来定义事件属性。可以使用原始类型、引用类型数组，也可以使用之前定义的Map事件类型、对象数组（Object-Array）数组。

当使用之前定义的Map类型数组作为属性时，一定要带”[]”方括号（见下例）。

下面的这个例子中，Sale事件类型中，有多种类型数组做为事件属性。假设SalesPerson类存在，并且之前已经定义了名为OrderItem的Map类型的事件：

Map<String, Object> sale = new HashMap<String, Object>();

sale.put("userids", int[].class);

sale.put("salesPersons", SalesPerson[].class);

sale.put("items", "OrderItem[]"); // The property type is the name itself

//appended by []

epService.getEPAdministrator().getConfiguration().addEventType("SaleEvent", sale);

该事件类型中的三个属性分别是int数组，SalesPerson数组和OrderItem（之前定义的Mpa事件类型）数组。

查询语句可以这样写：

select userids[0], salesPersons[1].name,

items[1], items[1].price.amount from SaleEvent

## 对象数组（Object[]）事件

### 概述

对象数组（Object-Array）也可以作为事件类型的表现形式。作为事件类型的对象类型只能有一个父类，父类也必须是对象数组事件类型。父类所有的属性子类也可用。并且其父类和子类都将匹配对应EPL语句。

运行时，可以通过配置操作updateObjectArrayEventType来添加事件属性，而不能删除和修改。嵌套的属性也能被添加。当然可以通过API将该事件类型移除，再作为一个新的事件类型进行添加。

事件类型被定义后，这个事件类型的名称就可以在其他事件类型中引用作为事件属性的类型。

对象数组事件类型的一对多关系用数组来表现，对象数组事件类型中的属性，可以是原始类型，引用类型，Map类型以及对象数组类型。

引擎通过调用sendEvent(Objcet[] array,String eventTypeName)来处理对象数组类型的事件。

引擎不会去校验数组的长度和值的类型，所以需要自己保证数组中的值与定义时的类型和数量匹配。

### 对象数组属性

对象数组属性的类型可以是任何类型，Java对象，Map，对象数组。它的好处与章节2.6.2Map类型属性的好处相似。

为了使用对象数组的事件，事件类型名、事件属性名及属性类型都要在EPL引用之前定义好。可以通过配置或者通过创建语名。

正面的代码片段定义一个事件类型，生成一个该类型的事件并发送到引擎中，例子通过运行时报配置接口定义了CarLocUpdateEvent事件类型。

// Define CarLocUpdateEvent event type (example for runtime-configuration

interface)

String[] propertyNames = {"carId", "direction"}; // order is important

Object[] propertyTypes = {String.class, int.class}; // type order matches name

order

epService.getEPAdministrator().getConfiguration().addEventType("CarLocUpdateEvent", propertyNames, propertyTypes);

然后，就可以在EPL语句中引用该事件类型了：

select carId from CarLocUpdateEvent.win:time(1 min) where direction = 1

向引擎中发送该类型事件：

// Send an event

Object[] event = {“testCarId”, 2};

epRuntime.sendEvent(event, "CarLocUpdateEvent");

### 对象数组父类

对象数组作为事件类型，其父类也需要是已定义的对象数组。相同的属性名将会被覆盖。并且查询语句会匹配所有的父类和子类。

### 高级的对象数组属性类型

#### 嵌套属性

对象数组的嵌套可以构建丰富的，类型安全的事件类型。可以通过初始化时的静态配置、运行时的配置接口或者create schema语法来创建。

值得注意的几点：

* 普通的java对象可以作为对象数组事件类型的属性；
* 可以用之前定义的对象数组或者对象数组数组作为对象数组事件的属性；
* 嵌套的深度没有限制；
* 可以用动态属性；
* 嵌套的结构中，如果访问路径下的实体不存在，引擎将返回null值。

下面的例子中，AccountUpdate是顶级的事件类型，它含有一个UpdatedFeildType类型的属性。这种类型的属性中又包含了多种类型的属性，包括普通的java对象。

String[] propertyNamesUpdField = {"name", "addressLine1", "history"};

Object[] propertyTypesUpdField = {String.class, String.class,

UpdateHistory.class};

epService.getEPAdministrator().getConfiguration().

addEventType("UpdatedFieldType", propertyNamesUpdField,

propertyTypesUpdField);

String[] propertyNamesAccountUpdate = {"accountId", "fields"};

Object[] propertyTypesAccountUpdate = {long.class, "UpdatedFieldType"};

epService.getEPAdministrator().getConfiguration().addEventType("AccountUpdate", propertyNamesAccountUpdate,propertyTypesAccountUpdate);

下正面的代码片段构建了一个该类型的事件并将其发送给引擎。

Object[] updatedField = {"Joe Doe", "40 Popular Street", new UpdateHistory()};

Object[] accountUpdate = {10009901, updatedField};

epService.getEPRuntime().sendEvent(accountUpdate, "AccountUpdate");

查询语句就可以这样写了：

select accountId, fields.name, fields.addressLine1, fields.history.lastUpdate

from AccountUpdate

#### 一对多的关系

一对多的关系，用数组来表现。如果属性类型是之前定义的一种事件类型，那第引用它的数组类型时，一定要带上”[]”方括号。

下面的例子是对象数组事件类型中一对多关系的属性定义方法，其中OrderItem是之前定义的事件类型：

String[] propertyNames = {"userids", "salesPersons", "items"};

Object[] propertyTypes = {int[].class, SalesPerson[].class, "OrderItem[]");

epService.getEPAdministrator().getConfiguration().addEventType("SaleEvent", propertyNames, propertyTypes);

查询语句就可以这样写了：

select userids[0], salesPersons[1].name,items[1], items[1].price.amount from SaleEvent

## XML文档事件

org.w3c.dom.Node实例也能来表现事件。通过EPRuntime或者EventSender的sendEvent发送给引擎。注意，需要通过配置，让引擎知道事件类型名和文档中的元素名（事件属性名），详见章节15，“配置（Configuration）”。

如果XML概要文档（XSD文件）作为配置的一部分可用，那么，Esper就可以读取该概要文档并生成事件类型的元数据，并且验证用到此事件类型的语句。详见章节2.8.1，“有概要文档的XML事件（Schema-Provided XML Events）”。

如果没有提供概要文档，XML事件仍然可以被检索，但是返回的类型和属性表达式返回的类型只能是String类型的，并且除了显示配置的属性外，没有事件类型的元数据。详见章节2.8.2，“无概要文档的XML事件（No-Schema-Provided） XML Events”。

Esper允许配置明确的XPath表达式作为事件属性。可以任意地指定XPath函数和表达式，提供属性名和类型名，这些都可以在EPL语句中使用了。详见章节2.8.3，“显示配置属性（Explicitly-Configuration Properties）”。

XML事件也支持嵌套、映射、索引属性，所以可以通过属性表达式语法方便地查询XML树。

一个根元素名只能配置一种事件类型。引擎根据根元素名来识别事件类型，否则，只能通过EventSender来发送事件。

请看下面的例子，XML文档内容：

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

<Sensor xmlns="SensorSchema">

<ID>urn:epc:1:4.16.36</ID>

<Observation Command="READ\_PALLET\_TAGS\_ONLY">

<ID>00000001</ID>

<Tag>

<ID>urn:epc:1:2.24.400</ID>

</Tag>

<Tag>

<ID>urn:epc:1:2.24.401</ID>

</Tag>

</Observation>

</Sensor>

它的概要文档如下：

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="Sensor">

<xs:complexType>

<xs:sequence>

<xs:element name="ID" type="xs:string"/>

<xs:element ref="Observation" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="Observation">

<xs:complexType>

<xs:sequence>

<xs:element name="ID" type="xs:string"/>

<xs:element ref="Tag" maxOccurs="unbounded" />

</xs:sequence>

<xs:attribute name="Command" type="xs:string" use="required" />

</xs:complexType>

</xs:element>

<xs:element name="Tag">

<xs:complexType>

<xs:sequence>

<xs:element name="ID" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### 有概要文档的XML事件

如果你的XML事件有XSD概要文档，它会带来如下好处：

* EPL语句中用到的事件属性将会根据概要文档进行验证；
* 作为EventType接口一部分的事件类型元数据可用，可以用来被检索。

#### 开始（Getting Started）

引擎根据你提供的URL来读取XSD概要文档。要保证在概要文档以及其中关联导入的概要文档能被正确找到。

引擎配置接受概要文档的URL，如下：

URL schemaURL = this.getClass().getClassLoader().getResource("sensor.xsd");

下面是利用运行API配置的例子：

epService = EPServiceProviderManager.getDefaultProvider();

ConfigurationEventTypeXMLDOM sensorcfg = new ConfigurationEventTypeXMLDOM();

sensorcfg.setRootElementName("Sensor");

sensorcfg.setSchemaResource(schemaURL.toString());

epService.getEPAdministrator().getConfiguration().addEventType("SensorEvent", sensorcfg);

需要提供一个根元素的名称，这个名称将被sendEventt方法用来查找事件类型。如果根据文档的根元素查找或者元素名得不到，那么EventSender是一个很有用的选择。

添加事件类型后，就可以写EPL语句了：

select ID, Observation.Command, Observation.ID,

Observation.Tag[0].ID, Observation.Tag[1].ID

from SensorEvent

如例，属性表达式可以从XML文档中的属性中获得属性值。

有多种方法根据XML文件信息获得XML文档的实例，如下例：

InputSource source = new InputSource(new StringReader(xml));

DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();

builderFactory.setNamespaceAware(true);

Document doc = builderFactory.newDocumentBuilder().parse(source);

然后，就可以把文档实例作为事件发送到引擎中处理了。

epService.getEPRuntime().sendEvent(doc);

#### 属性表达式和命名空间

计算形如Oberservation.Tag[0].ID的属性表达式值的方式，是Esper提供的一种fast DOM-walker的实现。这种实现默认情况下对命名空间不敏感。

如果想对命名空间敏感，需要把esper的xpath-property-expr配置选项设置成true。设置成true后，代替DOM-walker的方式，Esper会把每一个属性表达式生成空间敏感的属性表达式。把这个选项设置成true，还需要设置一下命名空间前缀，如下文描述。

在和XSD概要文档中的信息进行属性名匹配时，引擎会判断文档中的属性或元素是否提供了值。实现的算法先检查属性名再检查元素名，并且只返回第一个匹配的属性名。

#### 属性表达式XPath重写

通过设置xpath-property-expr选项为true，可以把每一个表达式重写为xpath表达式，比基本的XPath实现的性能更优越。JVM内置的XPath实现以及其他的例如Jaxen也都可以使用。

如果需要，例如你有多个命名空间或者元素重名了，就可以设置这个属性为true。

下表列出几个简单的例子，展示重写后的效果，为了省空间，生成的xpath表达式（省略了命名空间）。

|  |  |
| --- | --- |
| **属性表达式** | **等效的XPath** |
| Observeration.ID | /Sensor/Observation/ID |
| Observeration.Command | /Sensor/Observation/@Command |
| Observeration.Tag[0].ID | /Sensor/Observation/Tag[position() =1]/ID |

对于以name(‘key’)语法来指定的映射属性，算法会找到名为id的属性，并产生一个开如mapped[@id=’key’]的xPath的表达式。

下面是个简单的例子，用到了各种类型的属性：

select nested.mapped('key').indexed[1].attribute from MyEvent

它的xPath表达式如下：

/n0:rootelement/n0:nested/n0:mapped[@id='key']/n0:indexed[position() = 2]/@attribute

#### 数组属性

在概要文档中指定允许出现次数大于1的元素就是索引属性，需要索引才能检索。

在同一个父元素里重复出现的多个子元素就被表现为数组元素。

例如下面的XML文档:

<item>

<book sku="8800090">

<author>Isaac Asimov</author>

<author>Robert A Heinlein</author>

</book>

</item>

这里，book.author是String数组类型的属性，book.author[0]就是一个单独的值了。

#### 动态属性

动态属性不会验证概要文档信息，它的返回值都是org.w3c.Node类型的。你可以使用自定义的功能来处理返回Node类型的动态属性。

下面的例子将会从文档中检索Origin元素下包含的名为LocationCode的元素或属性。

select Origin?.LocationCode from SensorEvent

#### 传递属性

对于提供概要文档的XML文档，默认配置允许将本身的复杂元素传递到事件流中。这一特性可以通过auto-fragment属性进行配置。

例如下面的查询语句：

insert into ObservationStream

select ID, Observation from SensorEvent

Observation是SensorEvent事件的复杂类型的属性，可以从该事件流中检索出来并插入名为ObervationStream的事件流中。

例如下面的语句：

select Observation.Command, Observation.Tag[0].ID from ObservationStream

再看下面的例子，可以用通配符查询某一属性下的属性元素：

insert into TagListStream

select ID as sensorId, Observation.\* from SensorEvent

下例中的TagListStream有一个String类型的ID属性，一个Command属性，和一个Tag数组类型的复杂属性：

select sensorId, Command, Tag[0].ID from TagListStream

请注意如下的限制：

* XPath标准规定，遵守org.w3c.Node的XPath表达式，是基于节点所属文档进行计算的，所以XPath不是对于当前节点的相对路径，而是节点所属文档的绝对路径。因为Esper不会为传输的节点创建document实例，所以当xpath属性表达式标志被设置时，传输属性不可用。
* 拥有简单属性值和复杂子元素的复杂元素不能被传输。这样做是为了确保它们的属性值不会被隐藏。可以用一个显示的XPath表达式来传输这些属性。

Esper会自动为传输属性注册一个新的事件类型。它会为这个新的XML事件类型创建名字，依据是这个XML事件类型名和在表达式中所使用的属性名。概要是：type\_name.property\_name[.property\_name…].type\_name可以在使用EventSender时检索出来或者提前创建。

#### 事件发送器（EventSender）

EventSender将指定类型的事件发送给引擎，根据元素名存一个事件类型的检索。

通过EventSender发送事件的简短例子如下：

EventSender sender = epRuntime.getEventSender("SensorEvent");

sender.sendEvent(node);

XML文档的事件发送器在处理事件前，会检查文档的根元素名称。设置“event-sender-validates-root”属性可以禁用校验。禁用以后，会强制引擎根据预定义的事件类型处理XML文档，而不用校验根元素的名称。

#### 限制

Esper引擎对于概要文档的解析依赖Sun的java运行时包中的Xerces，所以你的应用不能替换Xerces版本来使用XML概要文档，除非在加载引擎的配置前，你的应用已经注册了DOM的实现。注册方法如下：

System.setProperty(DOMImplementationRegistry.PROPERTY,"com.sun.org.apache.xerces.internal.dom.DOMXSImplementationSourceImpl");

### 没有概要文档的XML事件

没有概要文档的XML事件也可能会被检索，但是这样的XML事件的元数据（metadata）会有一些重要的差异，所以属性表达式的结果也就会存在差异，这些不同，将会在下面列出来：

* 没有概要文档的XML事件类型的属性表达式都被假定为合法，除了语法的正确性校验外，没有属性表达式的校验。在运行时期间，属性表达式将返回一个String类型的值。如果表达式没有匹配的元素或者属性值，将会返回null。
* 如果要得到事件类型的所有属性名或者属性元数据，将会返回一个空的数组。

### 显式配置的属性

对于XML文档事件类型，不管有没有提供XSD的概要文档，都可以通过Xpath表达式，反过来设置显式的属性。

关于Xpath的更多资料，请查阅Xpath标准和其他在线文档。决定使用Jaxen还是Apache Axiom。例如，为了获得更快的Xpath计算，java 虚拟机（JVM）内置的Xpath就是不错的选择。

#### 简单的显式属性

下面的例子就是通过Xpath配置显式属性并且定义例句空间前缀。

epService = EPServiceProviderManager.getDefaultProvider();

ConfigurationEventTypeXMLDOM sensorcfg = new ConfigurationEventTypeXMLDOM();

sensorcfg.addXPathProperty("countTags", "count(/ss:Sensor/ss:Observation/

ss:Tag)",

XPathConstants.NUMBER);

sensorcfg.addNamespacePrefix("ss", "SensorSchema");

sensorcfg.setRootElementName("Sensor");

epService.getEPAdministrator().getConfiguration().addEventType("SensorEvent", sensorcfg);

现在，countTags属性就可以被检索了：

select countTags from SensorEvent

Xpath表达式count(……)是Xpath内置的功能，用来计算节点的数目。例子文档中的结果就是2。

#### 显式属性的转换和解析

Esper引擎可以将Xpath表达式的结果转换或者解析成为你想要的类型。属性的配置需要指定你想的目标类型，如下：

sensorcfg.addXPathProperty("countTags", "count(/ss:Sensor/ss:Observation/

ss:Tag)",XPathConstants.NUMBER, "int");

属性配置的类型，必须是内置的类型。内置的数组类型也是被支持的，需要“XPathConstants.NODESET”返回类型。如下：

sensorcfg.addXPathProperty("idarray", "//ss:Tag/ss:ID",XPathConstants.NODESET, "String[]");

“//ss:Tag/ss:ID”这个Xpath表达式，将返回Tag节点下的所有ID节点。不管这个元素位于节点树下的什么位置。例子文档中的结果就是包含2个元素的数组：urn:epc:1:2.24.400和urn:epc:1:2.24.40。

#### 节点和节点集的显式属性

显式属性将会返回XPathConstants.NODE或者XPathConstants.NODESET，和此属性的预配置事件类型的事件类型名称。添加此类属性的方法名为addXPathPropertyFragment。

下面的代码段示例了添加两个显式属性，并给他们分别赋予事件类型。

sensorcfg.addXPathPropertyFragment("tagOne", "//ss:Tag[position() = 1]",

XPathConstants.NODE, "TagEvent");

sensorcfg.addXPathPropertyFragment("tagArray", "//ss:Tag",

XPathConstants.NODESET, "TagEvent");

上面代码中引用的“TagEvent”事件类型，必须已经被配置过。“//”前缀意味着从概要文档中检索定义的事件类型。上述事件类型的定义如下：

ConfigurationEventTypeXMLDOM tagcfg = new ConfigurationEventTypeXMLDOM();

tagcfg.setRootElementName("//Tag");

tagcfg.setSchemaResource(schemaURL);

epAdministrator.getConfiguration().addEventType("TagEvent", tagcfg);

现在，tagOne和tagArray属性就可以用在以后的检索语句中了：

insert into TagOneStream select tagOne.\* from SensorEvent

从这个事件流中查询：

select ID from TagOneStream

下面的例子是有索引的属性：

insert into TagArrayStream select tagArray as mytags from SensorEvent

从事件流中查询：

select mytags[0].ID from TagArrayStream

## 附加的事件表现形式

附加的事件表现形式是Esper扩展及插件特性的一部分。Esper允许应用根据可用的信息，静态地或者在EPL创建后动态地创建事件类型和事件实例。可以参阅章节17.8，“事件类型和事件对象”，了解详情。

如果你的程序拥有现成的java类，包含事件元数据或者事件属性数据，但是你不想（或者不能）将这些数据提取到内置的事件类型表现形式（简单的java对象、Map、对象数组或者XML文档）中去，此时，自定义一个事件类型表现形式将非常有用。

自定义事件类型表现形式更深入的用处是提供更迅速更快捷的事件数据访问途径。例如，通过XML流API访问保存在XML文件中的事件数据。甚至通过网络检索，或者针对动态事件数据的动态解决方案。

目前，EsperIO提供以下附加的事件类型表现形式：

* Apache Axiom:XML(Stax)实现的API。

可以详阅EsperIO文档。

章节17.8，“事件类型和事件对象”描述了如何创建自定义的事件表现形式。

## 事件的更新、合并和版本管理

概要地说，事件是过去发生的动作或者状态变化的不可变记录，事件属性记录了事件的有用信息。

事件发送到处理引擎的时间长度（延迟时间）依赖于你的EPL语句，尤其是数据窗口、模式以及EPL语句中输出率限制语法。

在上述的延迟时间中，事件的信息可能会发生变化，例如增加了一些附加属性或者现有属性的变化。Esper提供了三种概念来处理事件的变化。

第一种途径是章节5.20，“更新插入流：update istream语法”中所说的“update istream”语法。这种方法用在更新事件流中的事件，在它们未被相应的消费语句（sonsuming statement）处理之前。

第二种方法是通过on-merge和on-update语法。它们只能用在命名窗口中，将在章节5.15.9，“利用on-merge语法触发Upsert”和章节5.15.7，“更新命名窗口：on-update语法”中详解。On-merge语法类似于SQL中的merge语法，提供我们所知道的“upsert”操作。在一个原子操作中，更新现有的记录，如果不存在，就插入一条新记录。On-update语法，用来更新命名窗口中事件的个别属性。

第三种方法是修订事件类型。也是只能用于命名窗口。章节5.15.11，“命名窗口中事件类型的版本控制和修订”中详解。使用修订，可以声明（只能通过配置）多个不同的事件类型。这些事件类型包含了它所合并的所有事件类型的所有属性，并且引擎会得到所有它收到的合并的事件，而不用额外的EPL语句。

请注意：模式不能不能反应已发生事件的改变。

## 粗糙事件

你的应用的事件可能包含非常全面的粗糙结构或文档。例如在企业对企业的整合场景中，XML文档或者事件对象可能是深层嵌套的事件属性图。

为了从粗糙事件中提取信息或者在事件属性图中执行批量操作，Esper提供了一种方便的途径：当你在模式（Pattern）或者select语句中指定过滤器时，它能包含一个包含事件选择语法，章节5.19，“包含事件选择”中将有详解。

## 事件对象的实例化和insert into填充

Insert into语法能够直接从select语法的结果中实例化新的java对象、Map、对象数组等事件实例进行填充。详见章节5.10，“合并事件流以及连续的插入：insert into语法”。

如果通过一个表达式，能够得到一个现成的java对象的实例，例如通过单行结果功能（single-row function）、静态方法的调用，你就能将这个表达式的结果直接传递到一个事件流中。详见章节5.10.7，“表达式结果的传递”和章节9.4，“select语法传递功能”。

Select和insert into语法中的字段名，必须匹配要被填充的事件对象中的可用且可写的属性。Select语法表达式结果中的类型，也要兼容目标事件类型的属性类型。

如果要填充一个基于简单java对象的事件类型，并且这个类提供了匹配的构造方法，那么select语法表达式结果的类型，还必须兼容构造方法中参数的顺序。

请看下面的示例语句：

insert into com.mycompany.NewEmployeeEvent

select fname as firstName, lname as lastName from HRSystemEvent

# 处理模式

# 上下文和上下文分区

# EPL参考：语句（Clauses）

# EPL参考：模式（Patterns）

# EPL参考：匹配/识别

# EPL参考：操作

# EPL参考：功能函数

# EPL参考：列举方法

# EPL参考：日期-时间方法

# EPL参考：视图（Views）

# EPL参考：数据流

# API参考

# 配置

# 开发周期

# 集成和扩展

# 脚本支持

# 示例、教程、安全学习

# 性能

# 引用