利用扩展事件进行调优和Troubleshooting

北京格瑞趋势 宋沄剑

DTCC

2015中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2015

大数据技术探索和价值发现

议程

什么是扩展事件
为什么使用扩展事件
扩展事件应用示例
扩展事件最佳实践

什么是扩展事件（类比）

出门上班这个过程

1、出门（触发事件1：锁门）

全局信息：时间
锁门事件信息：拿出钥匙的位置、用的哪个钥匙

2、走到停车位，解锁车（触发事件2：用钥匙开车门）

全局信息：时间
解锁车事件信息：停车位置

3、启动车（触发事件3：启动汽车）

全局信息：时间
启动汽车事件：无

TroubleShooting：我今天是否锁门了？

什么是扩展事件

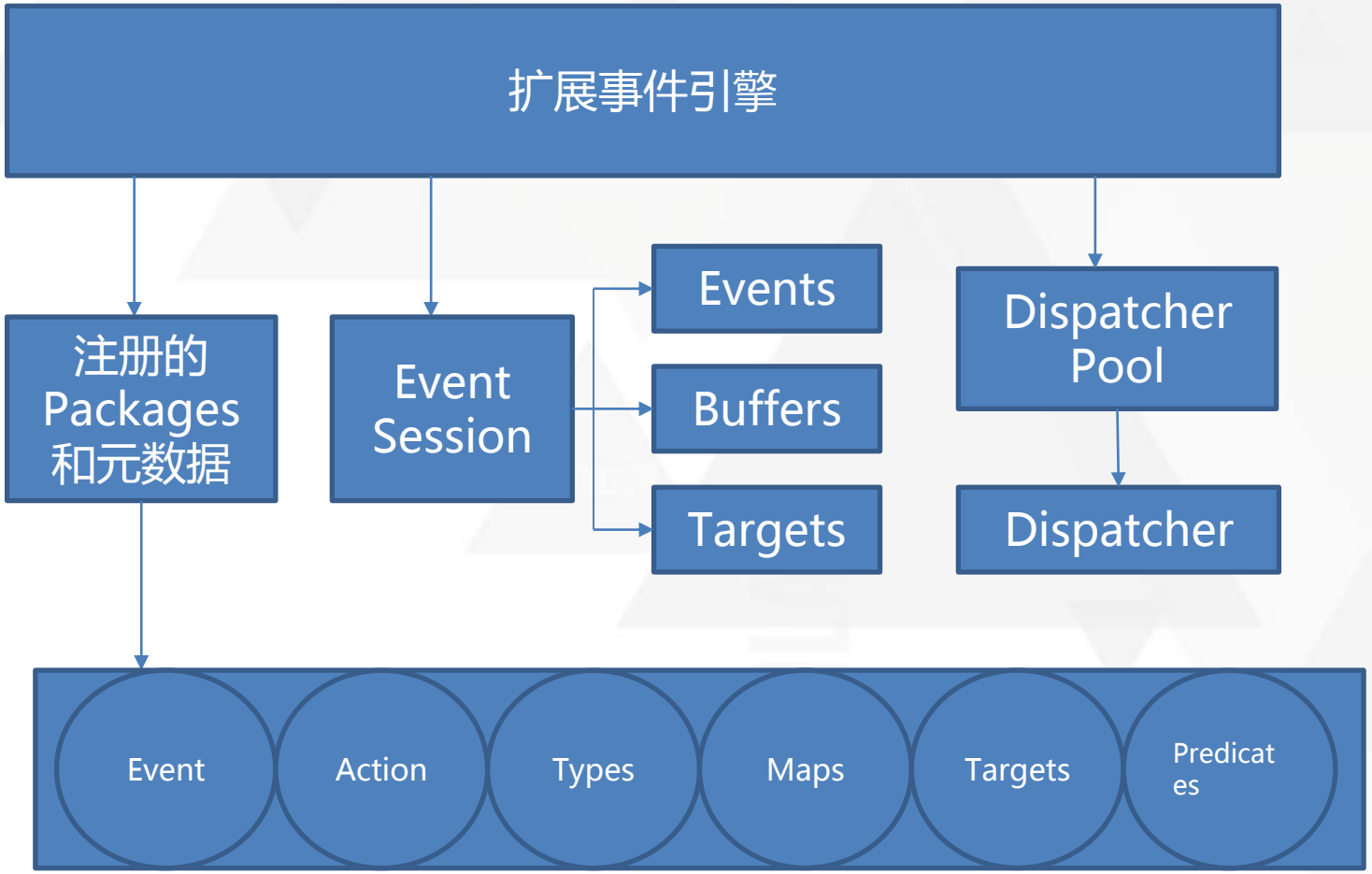
- 可扩展、可高度定制化的事件处理系统
- 扩展事件引擎寄存于sqlserver.exe
  - 扩展事件信息的消费者可以在该进程内，也可以是其他进程
  - 提供了ETW消费者，从而可以将SQL Server收集到的信息与Windows信息相结合
- 主要应用场景
  - 问题诊断&信息收集
  - 审核

什么是扩展事件

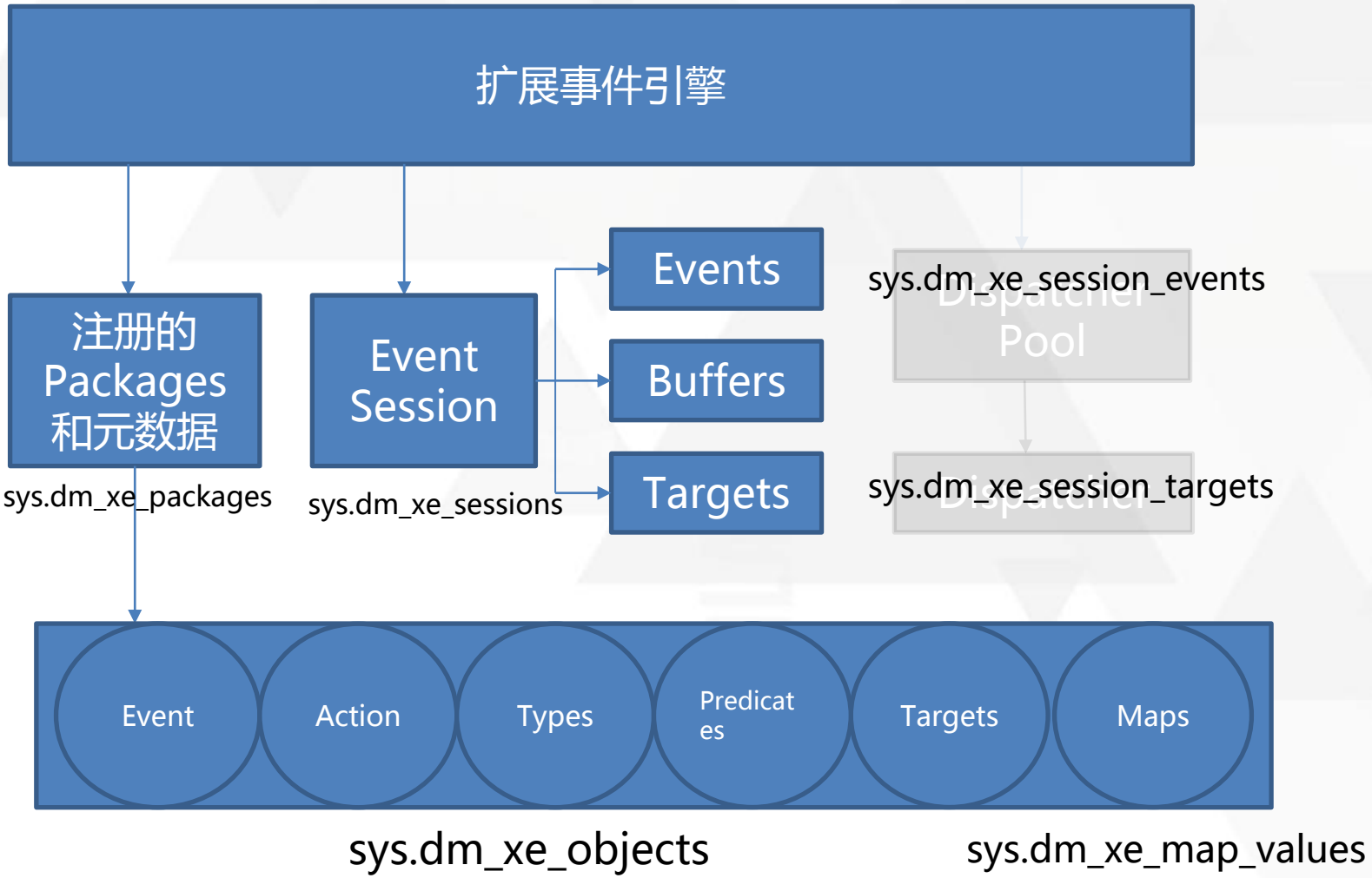
- 在SQL Server 2008被引入
  - 没有UI支持！
  - 捕捉结果为XML，解析需要使用Xquery！
  - 支持的事件少，很多地方有BUG

- SQL Server 2012&2014增加了大量事件，提供了UI支持，修复了很多Bug，提供了对SQL Azure、AlwaysOn、PowerShell的支持。

扩展事件架构


扩展事件引擎
注册的 Packages 和元数据
Event Session
Events
Buffers
Targets
Dispatcher Pool
Dispatcher
Event
Action
Types
Maps
Targets
Predicates

扩展事件架构


扩展事件引擎
注册的 Packages 和元数据
Event Session
Events
Buffers
Targets
Dispatcher Pool
sys.dm_xe_session_events
sys.dm_xe_session_targets
sys.dm_xe_packages
sys.dm_xe_sessions
Event
Action
Types
Predicates
Targets
Maps
sys.dm_xe_objects
sys.dm_xe_map_values

Package（包）

- Package在运行时由各自的Module载入
  - 默认的Package0由扩展事件引擎装载，其中包含与Module无关的通用对象
- Package是对象和其定义的容器
- 不同Package中包含的对象可以交叉使用。

  Package示例: sqlservr.exe, sqlos.dll

Event（事件）

- 事件对应代码中的特定点
  - 例：事务结束、发生死锁
    事件本身包含基本的信息
  - 某些事件可能会包含一些可选（自定义）列，这些列只有在指定时才会收集
  - 事件总是返回所有非自定义列
  - 所有的Trace都可以找到对应匹配事件（http://t.cn/RAXFkvM）

Predicates（筛选器）

- 筛选器是用于定义事件是否触发的布尔表达式
  - 筛选器支持short-circuit评估
- 筛选器结果为False会阻止事件发生
- 筛选器表达式类似SQL，可以是简单的比较表达式，也可以更复杂的文本比较
- 筛选器可以作用于事件列，也可以作用于全局列
  - 当筛选器作用于全局列时，需要同步收集数据
- 筛选器可以存储状态
  例： WHERE **package0.divides_by_uint64(package0.counter,5)**)

Predicates（筛选器）

所选事件(E):

名称

事件配置选项(O):

全局字段(操作) | 筛选器(谓词) | 事件字段

```
ADD EVENT sqlos.wait_info(
    ACTION(package0.callstack,sqlserver.session_id,sqlserver.sql_text)
    WHERE ([duration]>(15000) AND
        ([wait_type]>(31) AND
        ([wait_type]>(47) AND
        [wait_type]<(54) OR
        [wait_type]<(38) OR [wait_type]>(63)
         AND [wait_type]<(70)
         OR [wait_type]>(96) AND
         [wait_type]<(100) OR
         [wait_type]=(111) OR
         [wait_type]=(117) OR
         [wait_type]>(178) AND
         [wait_type]<(183) OR
          [wait_type]=(190) OR
          [wait_type]=(214) OR [wait_type]=(276)
       )
      OR [duration]>(30000) AND [wait_type]<(22)
      ))
```

Action（操作）

- 只有在筛选器结果为True时，才会执行操作（Action）
- 在线程触发事件时，操作执行一些额外的步骤
- 操作可以收集额外的状态数据，并附加到事件上
- 一些非收集信息的事件可以执行操作
- 类似Memory Dump之类的操作会产生副作用（DBA不应该使用）
- 任何事件都可以使用任何操作。

Target（目标）

- 目标是事件消费者
  - 处理单个事件或者缓冲区满处理
- 存在同步或异步目标
- 基本目标
  - Event File
  - Ring Buffer
- 基于条件聚合目标数据的目标
  - Event Bucketizer (提供直方图)
  - Event Counter
  - Event Pairing (用于匹配事件)

Target（目标）示例

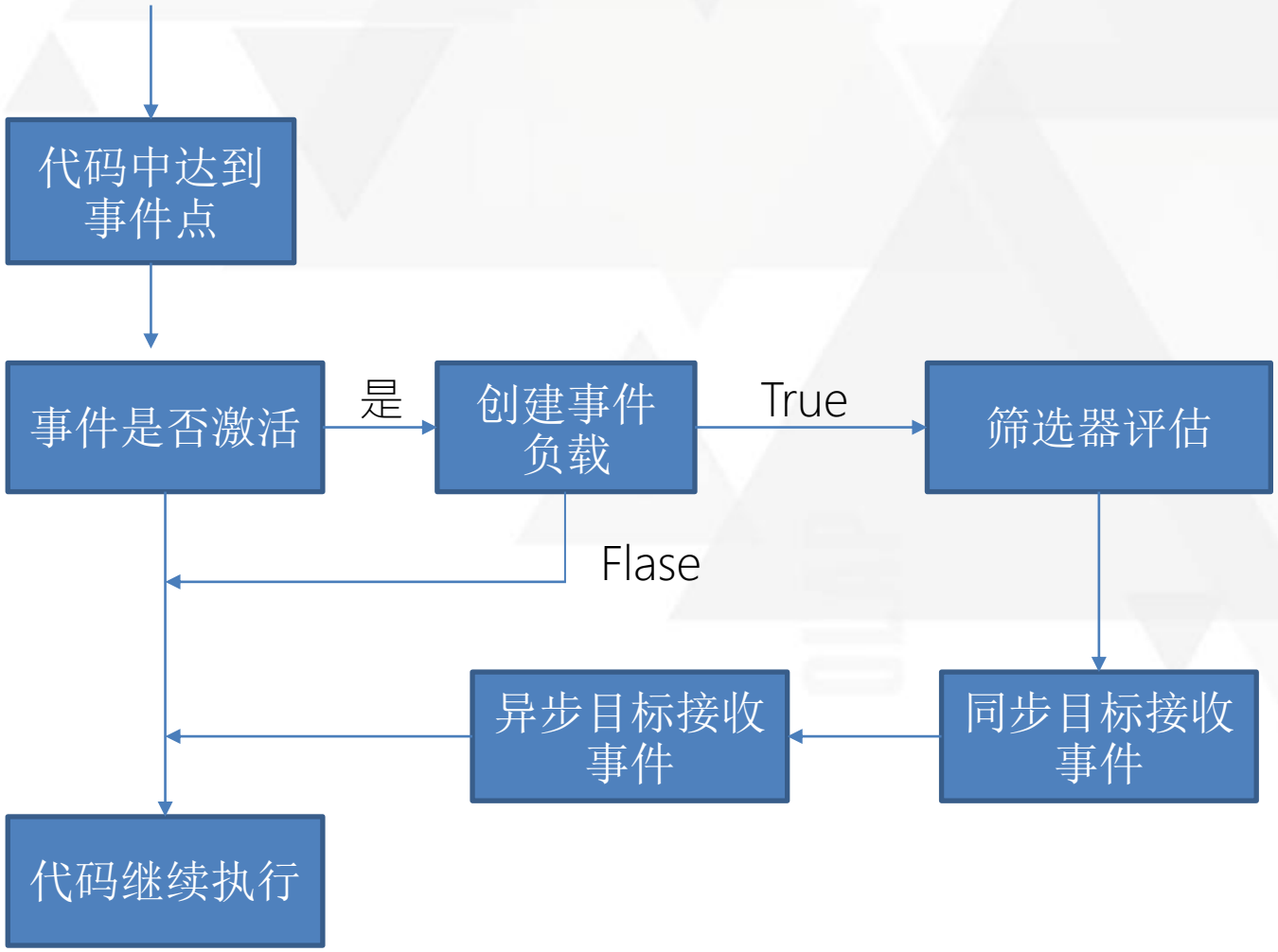
```
CREATE EVENT SESSION [DatabaseUsage] ON SERVER ADD EVENT sqlserver.
    lock_acquired ( WHERE owner_type = 4 --
    AND resource_type = 2 -- 数据库级别的锁
    AND database_id > 4 -- 非系统数据库
    AND sqlserver. is_system = 0 -- 用户进程
) ADD TARGET package0.histogram ( SET slots = 32, --分组数量，根据数据库数量设置
    filtering_event_name='sqlserver.lock_acquired', -- 将获得锁作为聚合列
    source_type=0, -- 事件数据，而非操作（Action）数据
    source='database_id' -- 根据Database id做聚合
)
```

100 %

结果 消息

	Count	Database
1	261	
2	92	
3	88	
4	38	
5	38	
6	19	
7	19	
8	19	
9	19	

扩展事件生命周期


代码中达到事件点
事件是否激活
是
创建事件负载
True
筛选器评估
Flase
异步目标接收事件
同步目标接收事件
代码继续执行

默认的系统Session

- 预定义的Event Session
- 收集常见问题的相关信息
  - Severity>20的错误信息
  - 内存错误
  - Non-yielding Problems
  - Deadlocks
  - >15秒的latch
  - >30秒的lock
  - 抢占式等待或外部等待
  - 该Event Session的目标是Ring Buffer

- AlwaysOn_Health

议程

什么是扩展事件
为什么使用扩展事件
扩展事件应用示例
扩展事件最佳实践

Trace&Profiler

扩展事件

为什么使用扩展事件（1）

- 相比服务端跟踪，开销更低（见http://t.cn/RAtKKho ）
- 扩展事件框架可扩展
  - 任何事件产生的结果都可以由任意Cosumer使用
  - 可以添加新的事件，立即可用
- 在事件触发时允许采取Action（可以收集额外信息，甚至是采用操作）

- 丰富的过滤系统
  - 允许更加强大的筛选操作

- 由于Trace已被标记为过时，所以不再会有任何新功能的开发。所以尽早转移到扩展事件里（AlwaysOn相关事件就无法用Trace跟踪，只能是用扩展事件）

- 可以在Microsoft Azure SQL Database中使用扩展事件

为什么使用扩展事件（2）

- 跟踪所有SQL Server 2012和之后版本新功能的唯一方式！

- 我强调过Trace已经过时了吗？

议程

什么是扩展事件
为什么使用扩展事件
扩展事件应用示例
扩展事件最佳实践

DEMO：监测消耗较高的SQL语句

```
--创建Event Session
CREATE EVENT SESSION [RunningAwaySQL] ON SERVER
ADD EVENT sqlserver.sql_statement_completed(SET collect_statement=(1) --可选事件，是否收集SQL语句
    ACTION(package0.collect_system_time, --系统时间
    sqlserver.client_app_name, --客户端程序名称
    sqlserver.client_hostname, --客户端主机名称
    sqlserver.database_id,  --数据库ID
    sqlserver.plan_handle,  --Plan_handle
    sqlserver.session_id,  --Spid
    sqlserver.username)  --登录的用户名称

 WHERE ([duration]>=(2000000)))  --Demo，收集执行时间大于20秒的语句

ADD TARGET package0.event_file(SET filename=N'D:\XeventResut1\RunningAwaySQL.xel'),
ADD TARGET package0.ring_buffer
```

	servername	posttime	dbid	cpu_s	duration_s	reads_k	writes	username	client	sql_text	plan_handle
1		2015-04-02 01:17:36.613	16	399	1056	206630	176175				<plan handle='0x06001000E7E3923840411
2		2015-04-02 01:18:02.697	49	125	224	6157	1524				<plan handle='0x06003100D53E131540618
3		2015-04-02 01:20:57.047	16	37	48	18063	38			tistics]	<plan handle='0x06001000D7DDCA034061E
4		2015-04-02 01:21:27.093	19	947	1286	552801	10			emp=DATEADD(...	<plan handle='0x0600130065B89323408111
5		2015-04-02 01:22:33.397	49	109	270	6220	1563				<plan handle='0x06003100FAD4E11440E19
6		2015-04-02 01:25:19.277	2	177	40	1666	0			dbo.VendorNews...	<plan handle='0x06000200ACCEF9284021
7		2015-04-02 01:28:16.573	19	12	495	8411	179862			NowTime=getdate...	<plan handle='0x06001300E6904E2B4001F
8		2015-04-02 01:38:36.020	19	944	1028	56519	33527			emp=DATEADD(...	<plan handle='0x0600130065B89323408111
9		2015-04-02 01:40:40.977	16	27	30	18063	12			tistics]	<plan handle='0x06001000D7DDCA034041
10		2015-04-02 01:49:14.090	19	608	638	41096	3645			emp=DATEADD(...	<plan handle='0x0600130065B89323408111
11		2015-04-02 02:00:38.310	53	8	37	390	0				<plan handle='0x06003500EAF1C30440418
12		2015-04-02 02:00:45.673	16	0	45	60	357				<plan handle='0x0600100096 38B10E40A1E
13		2015-04-02 02:00:47.263	51	3	46	976	316			vs]	<plan handle='0x0600330062349D0040816
14		2015-04-02 02:01:07.170	51	24	66	268	670				<plan handle='0x060033003E2FCA2B40C16
15		2015-04-02 02:01:09.493	16	39	54	18063	38			tistics]	<plan handle='0x06001000D7DDCA03404E

DEMO：PairMatching
目标的使用

Pair_Matching目标

- 特征
  - 利用一个或多个数据列匹配两个事件
  - 如果两个事件匹配，则丢弃事件
- 参数
  - Max_Orphans----未匹配事件的FIFO列表，默认值未10000
  - 用于匹配的列或操作
  - 当面临内存压力时，停止跟踪新的孤立事件

记录Timeout查询

```
CREATE EVENT SESSION FindTimeoutConnection ON SERVER
ADD EVENT sqlserver.sql_statement_starting
(    ACTION(sqlserver.session_id, sqlserver.tsql_stack)
),
ADD EVENT sqlserver.sql_statement_completed
(    ACTION(sqlserver.session_id, sqlserver.tsql_stack)
)
ADD TARGET package0.pair_matching
(    SET begin_event = 'sqlserver.sql_statement_starting',
       begin_matching_actions = 'sqlserver.session_id, sqlserver.tsql_stack',
       end_event = 'sqlserver.sql_statement_completed',
       end_matching_actions = 'sqlserver.session_id, sqlserver.tsql_stack',
       respond_to_memory_pressure = 0,
       max_orphans=1000 --最大孤立事件储存条数,默认为10000
)
WITH (MAX_DISPATCH_LATENCY=5 SECONDS, TRACK_CAUSALITY=ON)
```

```
SELECT * FROM charge a INNER hash JOIN dbo.charge b
ON a.charge_code = b.charge_code
```

00 %

消息

消息 -2，级别 11，状态 0，第 0 行
Timeout 时间已到。在操作完成之前超时时间已过或服务器未响应。

记录Timeout查询

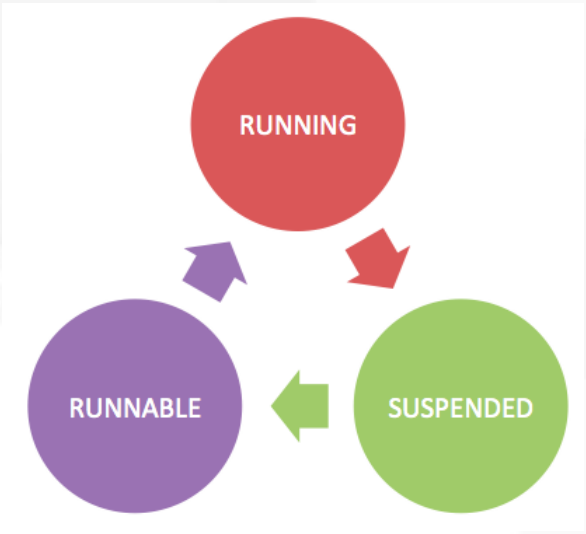
不刷新目标数据。 右键单击表可手动刷新或设置自动刷新间隔。

ackage_nam	event_name	timestamp	attach_activ	attach_activ	line_number	offset	offset_end	session_id	state	statement	tsql_stack
lserver	sql_statement_starting	2015-04-...	AE26D583...	D8578F7C...	1	0	-1	65	Normal	SELECT *...	<frames>...
lserver	sql_statement_starting	2015-04-...	2C3BC714...	D8578F7C...	1	0	-1	61	Normal	SELECT t...	<frames>...

DEMO：SpinLock（自旋锁）性能调优

背景知识-什么是自旋锁

- 锁
  - 锁作用于事务，用于保护不同用户之间对数据的读取和更改
- 栓锁（Latch）
  - 用于内存结构中的线程同步，生命周期只在代码临界区有效



背景知识-什么是自旋锁

- SpinLock（自旋锁）
  - 自旋锁用于同步线程对内存结构的访问
  - 线程不放弃CPU（无上下文切换，保持Running状态，也被称为Busy waiting）
  - 线程在获得SpinLock之前，不会结束
  - 主要用于保护非常繁忙的数据结构，持续时间通常较短
    - 例：Lock_Manager(Lock_Hash)

- 自旋锁竞争症状
  - 高CPU，但没做实际工作
  - sys.dm_os_spinlock_stats中backoffs值高

捕捉自旋锁

```
CREATE EVENT SESSION SpinlockContention ON SERVER
ADD EVENT sqlos.spinlock_backoff
(
    ACTION
    (
        package0.callstack
    )
)
ADD TARGET package0.histogram
(
    SET source = 'package0.callstack', source_type = 1
)
GO

-- 解析callstack
DBCC TRACEON(-1, 2592, 3656)
GO
```

sqldk.dll!XeSosPkg::spinlock_backoff::Publish+0x138
sqldk.dll!SpinlockBase::Sleep+0xc5
sqlmin.dll!Spinlock<129,7,1>::SpinToAcquireWithExponentialBackoff+0x169
sqlmin.dll!lck_lockInternal+0x841
sqlmin.dll!XactWorkspaceImp::GetSharedDBLockFromLockManager+0x18d
sqlmin.dll!XactWorkspaceImp::GetDBLockLocal+0x15b
sqlmin.dll!XactWorkspaceImp::GetDBLock+0x5a
sqlmin.dll!lockdb+0x4a
sqlmin.dll!DBMgr::OpenDB+0x1ec
sqlmin.dll!sqlusedb+0xeb
sqllang.dll!usedb+0xb3
sqllang.dll!LoginUseDbHelper::UseByMDDatabaseId+0x93
sqllang.dll!LoginUseDbHelper::FDetermineSessionDb+0x3e1
sqllang.dll!FRedoLoginImpl+0xa1b
sqllang.dll!FRedoLogin+0x1c1
sqllang.dll!process_request+0x3ec
sqllang.dll!process_commands+0x4a3
sqldk.dll!SOS_Task::Param::Execute+0x21e
sqldk.dll!SOS_Scheduler::RunTask+0xa8
sqldk.dll!SOS_Scheduler::ProcessTasks+0x279
sqldk.dll!SchedulerManager::WorkerEntryPoint+0x24c
sqldk.dll!SystemThread::RunWorker+0x8f
sqldk.dll!SystemThreadDispatcher::ProcessWorker+0x3ab
sqldk.dll!SchedulerManager::ThreadEntryPoint+0x226

议程

什么是扩展事件
为什么使用扩展事件
扩展事件应用示例
扩展事件最佳实践

扩展事件最佳实践

- 目标（Target）– 同步或异步
- 筛选器（Predicate） – 对于执行频繁的事件，不要创建过于复杂的谓词，且Short-circuit使得尽量将简单短小的谓词置于前
- 尽量将Action List保持到最小
- 保留期 – No_Event_Loss 或 Allow_Single_Event_Loss/Allow_multiple_Event_Loss
- 使用SSMS2012构建Session，生成代码后应用到2008


ChinaUnix
ITPUB
THANKS