ZStack 技术白皮书精选

揭密首个面向 IaaS 的查询语言：

ZStack Query Language（ZQL）

扫一扫二维码，获取更多技术干货吧

版权声明

摘要

大道至简·极速部署，ZStack 致力于产品化私有云和混合云。

ZStack 是新一代创新开源的云计算 IaaS 软件，由英特尔、微软、CloudStack 等世界上最早一批虚拟化工程师创建，拥有 KVM、Xen、Hyper-V 等成熟的技术背景。

ZStack 创新提出了云计算 4S 理念，即 Simple （简单）、Strong （健壮）、Smart （智能）、Scalable （弹性），通过全异步架构，无状态服务架构，无锁架构等核心技术，完美解决云计算执行效率低，系统不稳定，不能支撑高并发等问题，实现 HA 和轻量化管理。

ZStack 发起并维护着国内最大的自主开源 IaaS 社区——zstack.io，吸引了 6000 多名社区用户，对外公开的 API 超过 1000 个。基于这 1000 多个 API，用户可以自由组装出自己的私有云、混合云，甚至利用 ZStack 搭建公有云对外提供服务。

ZStack 拥有充足的知识产权储备，积极申报多项软著和专利，参与业内标准、白皮书的撰写，入选云计算行业方案目录，还通过了工信部云服务能力认证和信通院可信云认证。ZStack 面向企业用户提供基于 IaaS 的私有云和混合云，是业内唯一一家实现产品化，并领先业内首家推出同时打通数据面和控制面无缝混合云的云服务商。选择 ZStack，用户可以官网直接下载、1 台 PC 也可上云、30 分钟完成从裸机的安装部署。

目前已有 1000 多家企业用户选择了 ZStack 云平台。

揭密首个面向 IaaS 的查询语言：ZStack Query Language

背景

IaaS 管理着海量的数据中心资源，如何对这些资源进行灵活快速的查询是运维人员面临的一个难题。在以往的 IaaS 软件中，往往只对单个资源的某些字段提供有限的 API 查询支持，例如可以通过虚拟机的 IP 字段查询，这不足够也不灵活。运维人员在做复杂查询时往往得绕开 IaaS 软件直接查询其后端数据库，这既要求运维人员要了解 IaaS 资源的内部关系，又带来了数据库误操作的风险。

从 ZStack 正式发布的第一个版本 ZStack0.6 开始，我们就致力在 API 层面提供跟数据库级别的查询功能，ZStack 的每个资源都包含一个 Query API，能够通过资源的自身字段以及资源的关联资源字段进行查询。例如

```
QueryVmInstance name~=web-vm state=Running
```

这里查询所有名字包含 web-vm 字符串，正在运行中的 VM。又例如

```
QueryVmInstance vmNics.eip.vip.ip='22.22.22.22'
```

EIP 是虚拟机的关联资源，这里查询网卡绑定了 EIP 为 22.22.22.22 的虚拟机。

Query API 功能强大：

用户可以通过 count 参数返回满足查询条件资源数量，类似 SQL 的 select count(*)；

通过 fields 参数指定要返回的字段，类似 SQL 的 select uuid,name from;

通过 sortBy、sortDirection 参数指定排序的字段和方向，类似 SQL 的 order by;

通过 start、limit 参数实现分页查询，类似 SQL 的 limit 和 offset。

Query API 除了使用方便外，定义也很简单。程序员在 ZStack 中增加了一种新资源后，只需要在代码中定义如下 class:

```
@AutoQuery(replyClass = APIQueryVmInstanceReply.class,
inventoryClass = VmInstanceInventory.class)
public class APIQueryVmInstanceMsg extends APIQueryMessage {
}
```

不需要写任何实现，对应资源就具有了 Query API。

ZStack 内部包含一个 Query Service 负责处理所有资源的 Query API，将他们翻译成相应的 SQL 语句，在查询条件中包含关联资源条件时会生成对应的 Join 子句。

基于 Query API， ZStack0.6 版本就包含了超过 400 万个单项查询条件，组合查询条件数为 400 万的阶乘。极大的方便了运维和复杂 UI 的设计。但 Query API 仍然包含一些缺陷：

- 查询条件之间只能是 AND 逻辑，无法执行 OR 逻辑，条件之间也无法加括号实现复杂逻辑组合
- 不支持类似 SQL 中的 sub query 子句
- 单个 API 只能查询一种资源，查询多种资源时需要调用多个 API
- 不支持跟监控系统的查询语言整合

随着 ZStack UI 的场景越来越丰富，Query API 的限制使得 UI 端的工作越来越多，很多场景需要多次调用 Query API 进行数据组合。例如在监控 Top 5 页面（用于检测系统中 CPU、内存、磁盘、网络等资源使用率最高 5 个资源的

页面），需要先采用 Query API 将虚拟机、物理机等资源信息查询回来，再调用监控系统 ZWatch 的 API 查询对应的监控数据。

Query API 在未来的 ZStack 版本中会一直保留并维护，其后端实现已经从原来的 Query Service 替换成 ZQL

ZStack Query Language

使用过著名 issue 管理系统 JIRA 的开发者都知道 JIRA 在进行高级搜索的时候提供一种查询语言 JQL (JIRA Query Language)，能够使用一种类似 SQL 的 DSL（Domain Specific Language）对 JIRA 中 ticket 的各个字段进行高效的查询。ZQL 跟 JQL 类似，也是一种类似 SQL 的 DSL，先来看一个例子：

```
query vminstance where name='webvm' or
vmnics.ip='192.168.0.10' or (vmnics.eip = '172.20.100.100' and
(cpuNum >= 8 or clusterUuid in
('fe13b725c80e45709f0414c266a80239','73ca1ca7603d454f8fa7f3bb570
97f80')))
```

在这个简单例子中，可以看到很多熟悉的 SQL 元素，例如 and/or 条件、括号、>=/in 操作符等。ZQL 可以看作 SQL 的一个子集外加 ZStack 根据自身需求进行的增强的查询语言。它的基本结构如下：

```
QUERY queryTarget (WHERE condition+)? restrictBy?
returnWith? groupBy? orderBy? limit? offset? filterBy? namedAs?
```

query 关键词

一条 ZQL 语句通常以 query 关键字开头，queryTarget 表示要查询的资源或资源字段的集合。前面的例子中 vminstance 代表虚拟机，例如 host 代表物理机、zone 代表区域，所有可被查询的资源都有自己的名称。如果不希望返回资源的所有字段，只希望获得资源的一个或多个字段，实现类似 SQL 的 select uuid,name from ...的功能，可以在资源名后指定字段名，多个字段名用逗号隔离，例如：

```
query vminstance.uuid,name,cpuNum
```

该查询返回所有虚拟机的 UUID、名称以及 CPU 数量。

除了 query 关键字，查询也能以 count 和 sum 关键字开头，前者返回满足查询条件资源的总数，后者可以对资源的某个字段进行求和。例如：

```
count vminstance where cpuNum > 8
```

返回系统中 CPU 数量超过 8 核的虚拟机的总数。

```
sum vminstance.memorySize by name where cpuNum > 8
```

用虚拟机名字对 CPU 核数超过 8 个的虚拟机进行分组，对它们的 memorySize 字段进行求和。如果系统中有两个 10CPU8G 的虚拟机都名为 webvm，则求和后返回 webvm 虚拟机总内存使用数为 16G。翻译成 SQL 则为：

```
select sum(memorySize) from vminstance where cpuNum > 8
group by name
```

WHERE 从句

ZQL 的 WHERE 从句跟 SQL 的 WHERE 从句类似，支持 and/or 逻辑操作符、括号组合，条件的比较符支持=, !=, >, >=, <, <=, like, not like, is null,

is not null, in, not in，查询条件名为资源的字段名。跟 SQL 不一样的地方在于，ZQL 的查询条件可以是关联资源的字段，例如：

```
query vminstance where vmNics.eip.vip.ip='22.22.22.22'
```

注意 where 从句前无需写类似 SQL 的 from xx 从句，因为 query vminstance 已经限定了被查询的资源

这里 vip 跟 eip 关联，eip 跟 vmnic 关联，vmnic 又跟 vminstance 关联，则我们可以指定 vip 的 IP 作为查询条件。这正是 ZQL 的强大之处，对于多个关联资源的查询，无需调用多次 API 在应用端组合数据，也无需像 SQL 一样写复杂的 join 从句，只需要像编程一样通过点号（.)引用另一个资源即可， ZQL 的翻译器会自动将跨资源引用翻译成对应的 SQL join 从句。

WHERE 从句可以包含子查询，类似于 SQL 的 sub query 功能，例如：

```
query vminstance where vmNics.l3NetworkUuid in (query
l3network.uuid where ipRanges.networkCidr='10.1.0.0/24')
```

这里找出所有 CIRD 为 10.1.0.0/24 的三层网络上运行的虚拟机。

上面这个例子也可以用更简单的方法实现：query vminstance where vmNics.l3network.ipRanges.networkCidr='10.1.0.0/24'，这里只是为了演示子查询功能

GROUP BY、ORDER BY、LIMIT、OFFSET 子句

跟 SQL 一样，ZQL 支持 GROUP BY、ORDER BY、LIMIT、OFFSET 关键字，以实现分组、排序、分页等功能。

GROUP BY：

通过虚拟机的区域 UUID 和集群 UUID 分组，统计各区域中各集群中虚拟机的数量。

```
count vminstance group by zoneUuid,clusterUuid
```

ORDER BY：

查询所有虚拟机，使用 cpuNum 字段降序排序。

```
1.    query vminstance order by cpuNum desc
```

LIMIT、OFFSET：

使用 limit 和 offset 实现分页：

```
query vminstance limit 100 offset 10
```

多资源查询

对于多个资源的查询，可以通过多条 query 查询语句实现，语句之间使用分号分隔，例如：

```
query vminstance where name = 'my-vm';

query host where cpuNum > 10;

query zone;
```

则一次调用即可返回三种资源的查询结果。由于返回的结果是一个 map 的 JSON 结构，为了方便获得对应语句的查询结果，可以使用 named as 关键字对查询语句命名，例如：

```
query vminstance where name = 'my-vm' named as 'vm';

query host where cpuNum > 10 named as 'host';

query zone named as 'zone';
```

则在返回的 JSON map 中，可以通过 vm、host、zone 作为 key 获得对应语句的查询结果。

合并监控查询（return with 从句）

在 ZStack 中使用了两种数据库：关系数据库存放元数据，时序数据库存放监控数据。由于不同数据库查询方式不一样，在 ZQL 之前，用户要查询一个资源的监控数据，需要先通过 Query API 获得该资源的元数据，再通过 ZWatch 的查询 API 获得其监控数据。例如要查询一个名为 webvm 虚拟机的 CPU 使用率监控数据，要执行如下 API：

```
QueryVmInstance fields=uuid name=webvm

GetMetricData namespace=ZStack/VM metricName=CPUUsedUtilization
labels=VMUuid=QueryVmInstance 返回的 UUID
offsetAheadOfCurrentTime=60
```

ZQL 通过 return with 子句解决这个问题。return with 是一种插件机制，它允许子系统 通过插件将自身的查询条件注入 ZQL 中，ZQL 会先执行关系数据库查询，将满足条件资源的原数据查询出来后，再将资源的主键(primary key)作为输入条件调用实现 return with 子句的插件，最后将插件的查询结果一并返回给 ZQL 的调用者。

上述查询虚拟机监控数据的需求可以通过一条 ZQL 语句实现：

```
    query vminstance.hostUuid,uuid where name = 'webvm' return
with (zwatch{resultName='webvm-
cpu',metricName='CPUAllUsedUtilization',offsetAheadOfCurrentTime
=60})
```

返回：

```
{
    "results": [
        {
            "inventories": [
                {
                    "hostUuid":
"f8271f58468b4281a212a43e530b5535",
                    "uuid":
"05781209d24341ac84fc055ae71820ac"
                }
            ],
            "returnWith": {
                "webvm-cpu": [
                    {
                        "labels": {
                            "VMUuid":
"05781209d24341ac84fc055ae71820ac"
                        },
                        "time": 1533280402,
                        "value": 0.8
                    },
```

```
                {
                    "labels": {
                        "VMUuid":
"05781209d24341ac84fc055ae71820ac"
                    },
                    "time": 1533280462,
                    "value": 0.8
                }
            ]
        }
    }
],
"success": true
}
```


这里我们用一条 ZQL 语句中即返回了我们感兴趣的元数据字段：uuid 和 hostUuid，也返回了该虚拟机的监控数据。细心的读者已经注意到我们在 ZWatch 查询字段中指定了参数 resultName='webvm-cpu'，并且在返回的 JSON map 中监控数据的 key 也是 webvm-cpu。跟 named as 关键字一样，这是为了执行多条 ZWatch 查询子句时方便检索返回结果准备的。 ZStack UI 使用非常复杂的 ZQL 查询语句，例如在 TOP 5 页面，一条 ZQL 查询包含多达 13 个 ZWatch 查询：

```
ZQLQuery zql="query vmInstance.uuid,name where
zoneUuid='89e148fb667c404dbc5309a2e956fa28' and
hypervisorType='KVM' and type='UserVm' and state='Running'
return with
(zwatch{resultName='CPUAllUsedUtilization',metricName='CPUAllUse
dUtilization',offsetAheadOfCurrentTime=60,period=6,functions='av
erage(groupBy=\"VMUuid\")',functions='top(num=5)'},zwatch{result
Name='MemoryUsedInPercent',metricName='MemoryUsedInPercent',offs
etAheadOfCurrentTime=60,period=6,functions='average(groupBy=\"VM
Uuid\")',functions='top(num=5)'},zwatch{resultName='MemoryFreeIn
Percent',metricName='MemoryFreeInPercent',offsetAheadOfCurrentTi
me=60,period=6,functions='average(groupBy=\"VMUuid\")',functions
='top(num=5)'},zwatch{resultName='DiskAllReadOps',metricName='Di
skAllReadOps',offsetAheadOfCurrentTime=60,period=6,functions='av
erage(groupBy=\"VMUuid\")',functions='top(num=5)'},zwatch{result
Name='DiskAllWriteOps',metricName='DiskAllWriteOps',offsetAheadO
fCurrentTime=60,period=6,functions='average(groupBy=\"VMUuid\")'
,functions='top(num=5)'},zwatch{resultName='DiskAllReadBytes',me
tricName='DiskAllReadBytes',offsetAheadOfCurrentTime=60,period=6
,functions='average(groupBy=\"VMUuid\")',functions='top(num=5)'}
,zwatch{resultName='DiskAllWriteBytes',metricName='DiskAllWriteB
ytes',offsetAheadOfCurrentTime=60,period=6,functions='average(gr
oupBy=\"VMUuid\")',functions='top(num=5)'},zwatch{resultName='Ne
tworkOutBytes',metricName='NetworkOutBytes',offsetAheadOfCurrent
Time=60,period=6,functions='average(groupBy=\"VMUuid,NetworkDevi
ceLetter\")',functions='top(num=5)'},zwatch{resultName='NetworkI
nBytes',metricName='NetworkInBytes',offsetAheadOfCurrentTime=60,
period=6,functions='average(groupBy=\"VMUuid,NetworkDeviceLetter
```

\")',functions='top(num=5)'},zwatch{resultName='NetworkOutPackets',metricName='NetworkOutPackets',offsetAheadOfCurrentTime=60,period=6,functions='average(groupBy=\"VMUuid,NetworkDeviceLetter\")',functions='top(num=5)'},zwatch{resultName='NetworkInPackets',metricName='NetworkInPackets',offsetAheadOfCurrentTime=60,period=6,functions='average(groupBy=\"VMUuid,NetworkDeviceLetter\")',functions='top(num=5)'},zwatch{resultName='NetworkOutErrors',metricName='NetworkOutErrors',offsetAheadOfCurrentTime=60,period=6,functions='average(groupBy=\"VMUuid,NetworkDeviceLetter\")',functions='top(num=5)'},zwatch{resultName='NetworkInErrors',metricName='NetworkInErrors',offsetAheadOfCurrentTime=60,period=6,functions='average(groupBy=\"VMUuid,NetworkDeviceLetter\")',functions='top(num=5)'})"

上例是在 ZStack CLI 中执行时的例子，使用\对引号转义

当资源特别多时，时序数据库查询性能可能成为多条 ZWatch 查询的性能瓶颈，故 return with 会通过并发的方式执行插件，默认并发度为 10。例如上述例子中的 13 条 ZWatch 查询会在 10 个线程中并发执行。用户可以通过全局配置 zql.returnWith.concurrency 更改并发度，例如

```
UpdateGlobalConfig category=query
name=zql.returnWith.concurrency value=15
```

限制查询 （restrict by 从句）

ZStack 的企业管理模块包含一个功能，可以对管理绑定某个区域，使得该管理员只能管理该区域内的资源，这就要求我们的 ZQL 对该管理员的查询请求只返回与其绑定区区中的资源。

对于虚拟机这样的资源，其元数据本身就带 zoneUuid 字段用于标识所在区域。但对于 eip 这样的资源，其元数据并无任何字段表示区域属性，区域属性是由其所在的三层网络或绑定的虚拟机确定的。例如要查询某个区域内的 eip，可以使用：

```
# 通过与虚拟机的绑定关系查询

query eip where vmNic.vmInstance.zoneUuid =
'52fdad0a2c0d4131a6c0fc6c1b7141a6'
```

或

```
# 通过所在三层网络确定

query eip where vip.l3Network.zoneUuid =
'52fdad0a2c0d4131a6c0fc6c1b7141a6'
```

无论那种方式，都需要调用者了解知道 eip 跟 zone 之间的关联关系，这对 API 的使用者提出了非常苛刻的要求。ZQL 通过 restrict by 从句解决这个问题。跟 return with 从句类似，restrict by 也是个插件框架，它允许其它服务通过插件解读 restrict by 从句中指定的条件，向生成的 SQL 中注入额外条件。例如上面的 eip 例子通过 restrict by 从句可以写成：

```
query eip restrict by
(zone.uuid='52fdad0a2c0d4131a6c0fc6c1b7141a6')
```

这里调用者无需知道 eip 跟 zone 之间的逻辑关系，restrict by 的路径插件会自动计算两者的逻辑关系，并生成对应的 SQL join 从句。这里 eip 既可以通过所在三层网络，也可以通过绑定的虚拟机确定和区域的关系，插件会自动计算路径权重，使用权重最高的路径生成 SQL 语句。

对于 eip 这个例子，插件会选取通过三层网络的关系生成 SQL 语句。因为 eip 可能没有跟虚拟机绑定，但其一定处于某个三层网络，故三层网络这条路径的权重更高。

restrict by 支持多个条件，通过逗号分隔，多个条件之间是 AND 关系。

除了给 ZQL 调用者使用外，restrict by 插件在 ZStack 内部也被其它服务广泛使用。例如账号系统会通过插件在普通账户调用 ZQL 的时候注入跟账号关联的 SQL 语句，使得普通账号只能查询到属于该账号的资源；又例如 SNS 服务会通过插件注入语句让 ZQL 只能查询到非系统类型的接收端。

未来

ZQL 为 ZStack 提供了一种类似 SQL 的 IaaS 查询语言，并且能够通过 return with 插件框架跟其它非关系数据库系统进行查询整合。在未来的版本中我们还会继续丰富其功能，目前有两个方向：

filter by 从句

虽然 return with 的 ZWatch 插件能让我们在查询资源元数据的同时查询其监控数据，但还不能将监控数据作为元数据的查询条件，例如无法通过一条 ZQL 实现查询某个集群中所有 CPU 使用率超过 90%的虚拟机。这在未来版本中会通过 filter by 从句实现，例如：

```
query vminstance where clusterUuid =
'33e26bd547d149fbb190436cc9aca824' filter by
(zwatch{metricName='CPUAllUsedUtilization',
offsetAheadOfCurrentTime=60, threshold>90})
```

同样，filter by 从句会实现成类似 return with 的插件框架，用于整合非关系数据库的查询条件。

智能 CLI

ZQL 有大量的从句，每个 ZStack 又有大量的可查询字段，目前 ZStack CLI 可以对 Query API 的可查询字段进行补全，但 ZQL 还暂时无法补全。未来版本中，我们会对 CLI 进行在增强，使其对所有查询条件可以进行提示和补全。