## 网易分库分表数据库DDB

互联网时代，也是关系型数据库独领风骚的时代，从早期的oracle独步天下，到现在MySQL蒸蒸日上，关系型数据库是大多数互联网应用在数据可靠性存储上的“命脉”。

随着互联网产品在体量和规模上的日益膨胀，无论是oracle还是mysql，都会第一时间面临来自磁盘，CPU以及内存等单机瓶颈，为此，产品方除了需要不断购买成本难以控制的高规格服务器，还要面临不断迭代的在线数据迁移。在这种情况下，无论是海量的结构化数据还是快速成长的业务规模，都迫切需要一种水平扩展的方法将存储成本分摊到成本可控的商用服务器上，同时，也希望通过线性扩容降低全量数据迁移对线上服务带来的影响，分库分表方案便应运而生。

分库分表的原理是将数据按照一定的分区规则sharding到不同的关系型数据库中，应用再通过中间件的方式访问各个shard中的数据。分库分表的中间件，隐藏了数据sharding和路由访问的各项细节，使应用大多数场景下可以像使用单机数据库一样使用分库分表后的分布式数据库。业界中，网易DDB，阿里TDDL，corbar，mycat以及hotdb等系统都是分库分表中间件中的佼佼者。

### 背景——十年一剑

DDB（全称Distributed database）是网易杭研院立项最早，应用最为广泛的后台产品之一，也是国内最早出现的数据库分库分表中间件。

最早可以追溯到2006年，网易杭研成立之初，为了应对网易博客这个日活超过800W的大体量应用，由现任杭研院院长的汪源带队主导开发了DDB这套分库分表数据库，伴随着博客的成长，DDB集群也从最早的20+节点，到40+节点，最后到现在云端100+个RDS实例。除了博客外，十年来DDB也见证了很多其他的大体量应用，如易信，云音乐，云阅读，考拉等。在大家耳熟能详的网易互联网产品中，几乎都可以看到DDB的身影。

经过10年的发展和演变，DDB的产品形态已全面趋于成熟，功能和性能得到了众多产品的充分验证，下面罗列一些大家比较关注的功能特性：

1. 与SQL92标准的兼容度达90%以上
2. 支持跨库join和跨库事务，支持大部分标量函数
3. 支持count，sum，avg，max，concat等常用聚合函数
4. 支持与MySQL高度一致的用户管理
5. 支持读写分离和数据节点高可用
6. 支持数据节点在线扩缩容，在线更改表分布
7. 提供有完善的数据库管理工具，WEB工具以及命令行工具
8. 数据节点支持oracle和mysql

目前DDB在网易内部有近50个产品使用，最大集群过百数据节点，大部分部署在云端，为应用提供透明，无侵入，MySQL标准协议的分库分表服务。

### DDB演变之路

十年来，DDB经历了三次服务模式的重大更迭，从最早的Driver模式，到后来的Proxy模式，再到近几年的云模式，DDB服务模式的成长也深刻反映着互联网流行架构的变迁。

#### Driver模式

Driver模式的特点在于应用通过DDB提供的JDBC Driver来访问DDB，类似与通过MySQL的JDBC驱动访问MySQL。而对于MySQL的驱动connector/J，只需要实现将SQL按照特定协议编码和转码即可。而DDB的驱动为了实现透明的分库分表，需要做很多额外的工作，如下图所示：

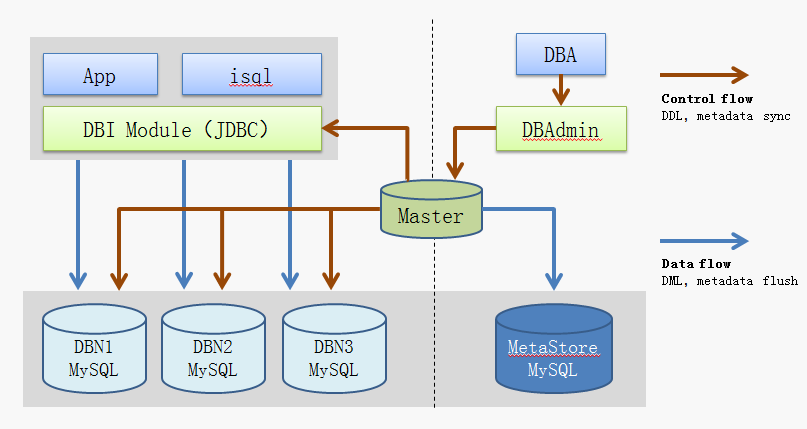


DBI Driver内部模块简图

当DDB D]river执行一条SQL时，会经历以下几个步骤：

1. 由语法解析器解析SQL，生成抽象语法树parsed tree，并根据是否PreparedStatement决定是否进入PTC（parsed tree cache），PTC保存了SQL模式到语法树的映射，对PreparedStatement SQL，会优先进入PTC中查询语法树
2. 根据语法树和启发式规则生成分布式执行计划，这个过程中会涉及到多个步骤的SQL转换和优化，如条件合并，join拆分，limit转化等
3. 由SQL执行器按照执行计划和语法树生成下发给每个数据节点的真实SQL，然后通过标准数据库驱动将SQL下发给各个数据节点，这个过程为并发执行。
4. 将各个数据节点返回的结果按照执行计划进行合并，并返回上层。具体的合并操作可能在应用调用结果时动态执行。

DBI模块作为DDB提供给应用的JDBC 驱动，包含了完整的透明分库分表逻辑，是DDB最为核心的组件，除此之外，DDB中还有用于元数据管理和同步的master组件，数据库管理工具dbadmin，以及命令行工具isql，DDB的Driver模式整体架构如下图所示：



DDB Driver模式架构

管理操作以建表为例：

DBA通过DBAdmin的窗口创建表，或者用isql执行建表语句之后，向master发起实际的建表请求，master完成用户认证和合法性校验之后，先在各个数据节点上创建新表，然后将新表元数据记录在系统库中，最后由master将新表元数据同步给各个DBI模块。

对于建表语句中DDB特有的语法，会由isql或dbadmin在解析DDL时完成相应处理，如自增ID的设置。

在DDB中，master用于元数据管理，同步以及报警监控。在DBI模块启动时，会第一时间向master注册，并拉取元数据，之后master对元数据的同步保障了DBI模块元数据的更新。在DBI执行SQL，以及创建DB连接的过程中，不会涉及到与master的交互。

在分库分表中间件中，与DDB Driver模式同样类型的还有阿里TDDL，优势是部署简单，成本较低，容易理解和上手。劣势也非常明显：只支持JAVA客户端，版本难以管理，问题难以追踪，DB连接难以归敛等，另外一点是，中间件与应用绑定在一起，对应用本身是个巨大侵入，而且分库分表的过程是比较耗费CPU资源的，所以在Driver模式下，无论是运维还是性能开销上都存在不可控的因素。

#### Proxy模式

相比于Driver模式在多语言，版本管理，运维风险上存在的问题，Proxy模式很好地弥补了这些缺陷。所谓Proxy，就是在DDB中搭建了一组代理服务器来提供标准的MySQL服务，在代理服务器内部实现分库分表的逻辑。本质上说，DDB Proxy作为一组独立服务，实现了MySQL标准通信协议，任何语言的MySQL驱动都可以访问，而在Proxy内部，依赖DBI组件实现分库分表，Proxy与DBI的关系如下所示：



DDB Proxy模块简图

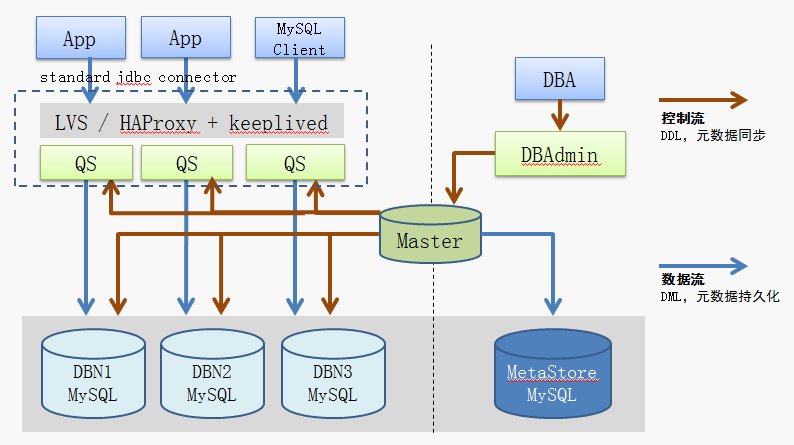
应用通过标准数据库驱动访问DDB Proxy，Proxy内部通过MySQL解码器将请求还原为SQL，并由DDB Driver，也就是DBI模块执行得到结果，最后通过MySQL编码器返回给应用。

从上图可以看出，Proxy在DBI上架设了MySQL编解码模块，从而形成独立标准的MySQL服务，而在MySQL编解码模块之上，DDB Proxy也提供了很多特色命令支持，例如：

1. show processlist：查看Proxy所有连接状态，与MySQL相关命令高度一致
2. show connection\_pool：查询Proxy到数据节点的连接池状态
3. show topsql..：查询按照SQL模式聚合的各项统计结果，如执行次数，平均执行时间
4. count.. from..：查询过去各个时间段内的吞吐量

此外，DDB Proxy内还提供了slow log等辅助功能，给运维带来很大的便利。

DDB Proxy模式完整架构如下所示：



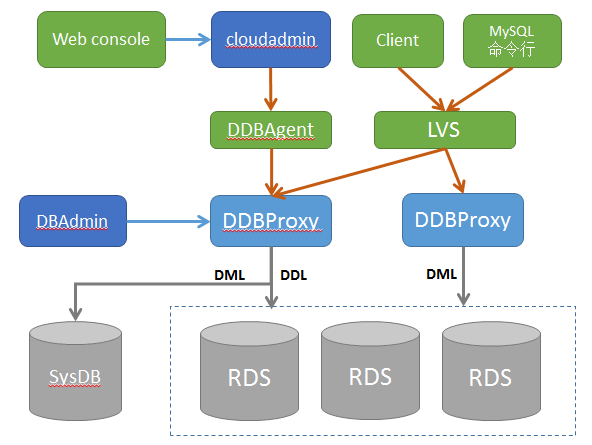
DDB Proxy模式架构

与Driver模式架构相比，除了QS（DDBProxy的内部称谓，下同）取代了DBI的位置，还在多个QS节点之上部署了LVS或haproxy + keepalived的组合做负载均衡，从而实现多个DDBProxy节点的热备，由于DDBProxy无状态，或者说状态统一由Master同步，在数据库节点没有达到瓶颈时，可以通过简单地增设QS服务器实现服务线性扩展。

#### 私有云模式

在网易私有云项目启动之前，DDB一直以一个个独立集群为不同业务提供服务，不同DDB各自为政毫不相干，这样的好处是业务之间完全隔离，互不影响，不好之处在于随着使用DDB的产品数目不断增多，一个DBA往往同时运维数个甚至数十个DDB集群，而之前我们一直缺乏一个平台化的管理系统，在DBA在各个集群之间应接不暇时，我们没有平台化的统筹运维帮助应用及早发现问题，或是优化一些使用方法。例如版本管理，2013年我们在一个大版本中做了个hotfix，并通知所有DBA将相关的版本进行升级，但是最后由于管理疏漏，有个别集群没有及时上线，为业务带来了损失。当时如果我们有平台化的管理方案，可以提供一些运维手段帮助和提醒运维人员及时更新所有有问题集群，另外，平台化的管理工具也可以定制一些自动化功能，如自动备份，报警组等。

网易私有云的出现为DDB的思变提供了契机，从12年开始，我们就在基于网易私有云开发一套平台化的管理工具cloudadmin，为此，我们将DDB中原先master的功能打散，一部分分库相关功能集成到Proxy中，如分库管理，表管理，用户管理等，一部分中心化功能集成到cloudadmin中，如报警监控，此外，cloudadmin中提供了一键部署，自动和手动备份，版本管理等平台化功能。私有云DDB的整体架构如下图所示：



私有云DDB架构

在云DDB解决方案中，还打包了网易私有云LVS服务，cloudadmin通过DDBAgent实现一键部署和报警监控。到目前为止，网易80%以上的DDB集群都已部署云端，云DDB的出现极大减轻了运维人员的负担。

### DDB特性介绍

#### 分布式执行计划

分布式执行计划定义了SQL在分库分表环境中各个数据库节点上执行的方法，顺序以及合并规则，是DDB实现中最为复杂的一环。

如SQL：select \* from user order by id limit 10 offset 10;

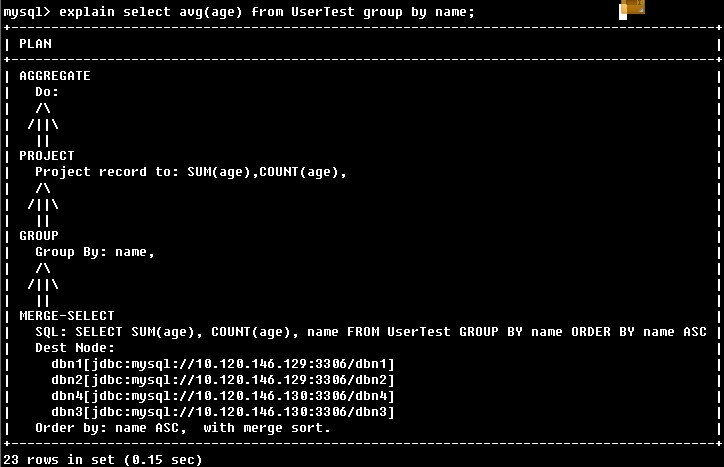
这个SQL要查询id排名在10—20之间的user信息，这里涉及到两个合并操作：全局id排序和全局limit offset。对全局id排序，DDB的做法是将id排序下发给各个数据库节点，在DBI层再进行一层归并排序，这样可以充分利用数据库节点的计算资源，同时将中间件层的排序复杂度降到最低，例如一些需要用到临时文件的排序场景，如果在中间件做全排序会导致极大开销。

对全局limit offset，DDB的做法是将offset累加到limit中下发，因为单个数据节点中的offset是没有意义的，且会造成错误的数据偏移，只有在中间件层的全局offset才能保证offset的准确性。

所以最后下发的给各个DBN的SQL变为：select \* from user order by id limit 20。

又如SQL：select avg(age) from UserTet group by name

可以通过explain语法得到SQL的执行计划，如下图所示：



explain示例

上述SQL包含group by分组和avg聚合两种合并操作，与全局order by类似，group by也可以下发给数据节点，中间件层做一个归并去重，但是前提要将group by的字段同时作为order by字段下发，因为归并的前提是排序。对avg聚合，不能直接下发，因为得到所有数据节点各自的平均值，不能求出全局平均值，需要在DBI层把avg转化为sum和count再下发，在结果集合并时再求平均。

DDB执行计划的代价取决于DBI中的排序，过滤和连接，在大部分场景下，排序可以将order by下发来简化为一次性归并排序，这种情况下代价较小，但是对group by和order by同时存在的场景，需要优先下发group by字段的排序，以达到归并分组的目的，这种情况下，就需要将所有元素做一次全排序，除非group by和order by的字段相同。

DDB的连接运算有两种实现，第一种是将连接直接下发，若连接的两张表数据分布完全相同，并且在分区字段上连接，则满足连接直接下发的条件，因为在不同数据节点的分区字段必然没有相同值，不会出现跨库连接的问题。若不满足连接下发条件，会在DBI内部执行nest loop算法，驱动表的顺序与from表排列的次序一致，此时若出现order by表次序与表排列次序不一致，则不满足order by下发条件，也需要在DBI内做一次全排序。

分库分表的执行计划代价相比单机数据库而言，更加难以掌控，即便是相同的SQL模式，在不同的数据分布和分区字段使用方式上，也存在很大的性能差距，DDB的使用要求开发者和DBA对执行计划的原理具有一定认识。

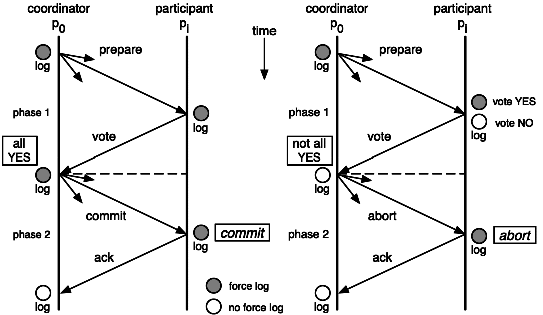
如分库分表在分区字段的使用上很有讲究：一般建议应用中80%以上的SQL查询通过分区字段过滤，使SQL可以单库执行。对于那些没有走分区字段的查询，需要在所有数据节点中并行下发，这对线程和CPU资源是一种极大的消耗，伴随着数据节点的扩展，这种消耗会越来越剧烈。另外，基于分区字段跨库不重合的原理，在分区字段上的分组，聚合，distinct，连接等操作，都可以直接下发，这样对中间件的代价往往是最小的。

#### 分布式事务

分布式事务是个历久弥新的话题，对分库分表，分布式事务的目的是保障分库数据一致性，而跨库事务会遇到各种不可控制的问题，如个别节点永久性宕机，如此像单机事务一样的ACID是无法奢望的。另外，业界著名的CAP理论也告诉我们，对分布式系统，需要将数据一致性和系统可用性，分区容忍性放在天平上一起考虑。

两阶段提交协议（简称2PC）是实现分布式事务较为经典的方案，适用于中间件这种数据节点无耦合的场景。2PC的核心原理是通过提交分阶段和记日志的方式，记录下事务提交所处的阶段状态，在组件宕机重启后，可通过日志恢复事务提交的阶段状态，并在这个状态节点重试，如coordinator重启后，通过日志可以确定提交处于prepare还是prepareAll状态，若是前者，说明有节点可能没有prepare成功，或所有节点prepare成功但是还没有下发commit，状态恢复后给所有节点下发rollback；若是prepareAll状态，需要给所有节点下发commit，数据库节点需要保证commit幂等。与很多其他一致性协议相同，2PC保障的是最终一致性。

2PC整个过程如下图所示：



2PC过程

在DDB中，DBI和Proxy组件都作为coordinator存在，2PC实现时，记录prepare和prepareAll的日志必须sync，以保障重启后恢复的状态是正确的，而coordinator最后的commit日志主要作用是回收之前日志，可异步执行。

由于2PC要求coordinator记日志，事务吞吐率受到磁盘IO性能的约束，为此DDB实现了group io优化，可极大程度提升2P C的吞吐率。2PC本质上说是一种阻塞式协议，两阶段提交过程需要大量线程资源，因此CPU和磁盘都有额外消耗，与单机事务相比，2PC在响应时间和吞吐率上会相差很多，从CAP的角度出发，可以认为2PC在一定程度上成全了C，牺牲了A。

另外，目前MySQL最流行的5.5和5.6版本中，XA事务日志无法replicate到从节点，这意外着主库一旦宕机，切换到从库后，XA的状态会丢失，可能造成数据不一致，这方面MySQL 5.7已经有所改善。

虽然2PC有诸多不足，我们依然认为在DDB中有实现价值，DDB作为中间件，其迭代周期要比数据库这种底层服务频繁很多，若没有2PC，一次更新或重启就可能造成应用数据不一致。从应用角度看，分布式事务的现实场景常常是无法规避的，在有能力给出其他解决方案前，2PC也是一个不错的选择。

对购物转账等电商和金融业务，中间件层的2PC最大的问题在于业务不可见，一旦出现不可抗力或意想不到的一致性破坏，如数据节点永久性宕机，业务难以根据2PC的日志进行补偿。金融场景下，数据一致性是命根，业务需要对数据有百分之百的掌控力，建议使用TCC这类分布式事务模型，或基于消息队列的柔性事务框架，这两种方案都实现在业务层，业务开发者具有足够掌控力，可以结合SOA框架来架构。原理上说，这两种方案都是大事务拆小事务，小事务变本地事务，最后通过幂等的retry来保障最终一致性。

#### 弹性扩缩容

分库分表数据库中，在线数据迁移也是核心需求，会用在以下两种场景：

1. 数据节点弹性扩容

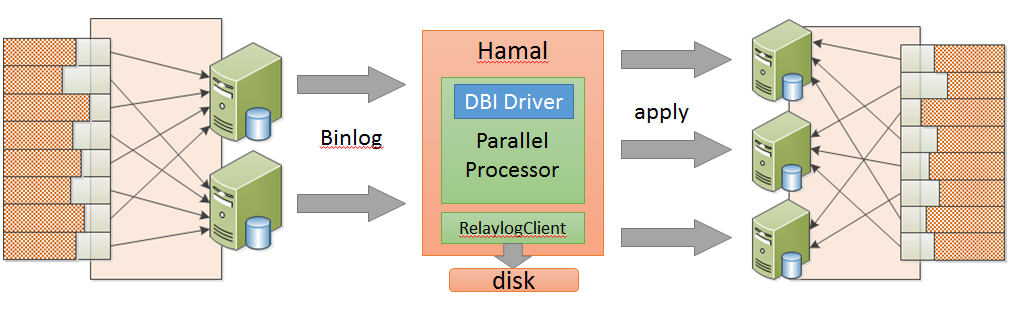
随着应用规模不断增长，DDB现有的分库可能有一天不足以支撑更多数据，要求DDB的数据节点具有在线弹性扩容的能力，而新节点加入集群后，按照不同的sharding策略，可能需要将原有一些数据迁入新节点，如hash分区，也有可能不需要在线数据迁移，如一些场景下的range分区。无论如何，具备在线数据迁移是DDB支持弹性扩容的前提。

1. 数据重分布

开发者在使用DDB过程中，有时会陷入困局，比如一些表的分区字段一开始没考虑清楚，在业务已经初具规模后才明确应该选择其他字段。又如一些表一开始认为数据量很小，单节点分布足以，而随着业务变化，需要转变为多节点sharding。这两种场景都体现了开发者对DDB在线数据迁移功能的潜在需求。

无论是弹性扩容，还是表重分布，都可当做DDB以表或库为单位的一次完整在线数据迁移。可分为两个阶段：全量迁移和增量迁移，全量迁移是将原库或原表中需要迁移的数据dump出来，并使用工具按照分区策略 apply到新库新表中。增量迁移是要将全量迁移过程中产生的增量数据更新按照分区策略apply到新库新表。

全量迁移的方案相对简单，使用DDB自带工具按照特定分区策略dump和load即可。对增量迁移，DDB实现了一套独立的迁移工具hamal来订阅各个数据节点的增量更新，hamal内部又依赖DBI模块将增量更新apply到新库新表，如下图所示：



数据迁移工具hamal

Hamal作为独立服务，与Proxy一样由DDB统一配置和管理，每个hamal进程负责一个数据节点的增量迁移，启动时模拟slave向原库拉取binlog存储本地，之后实时的通过DBI模块apply到新库新表，除了基本的迁移功能外，Hamal具备以下两个特性：

1. 并行复制：hamal的并行复制组件，通过在增量事件之间建立有向无环图，实时判断哪些事件可以并行执行，hamal的并行复制与MySQL的并行复制相比快10倍以上。
2. 断点续传：hamal的增量apply具有幂等性，在网络中断或进程重启之后可以断点续传

#### 全局表

考虑一种场景：city表记录了国内所有城市信息，应用中有很多业务表需要与city做联表查询，如按照城市分组统计一些业务信息。假设city的主键和分区键都是cityId，若连接操作发生在中间件层，代价较高，为了将连接操作下发数据节点，需要让联接的业务表同样按照cityId分区，而大多数业务表往往不能满足这个条件。

联接直接下发需要满足两个条件，数据分布相同和分区键上联接，除此之外，其实还有一种解法，可以把city表冗余到所有数据节点中，这样各个数据节点本地联接的集合，便是所求结果。DDB将这种类型的表称之为全局表。

全局表的特点是更新极少，通过2PC保障各个节点冗余表的一致性。可以通过在建表语句添加相关 hint指定全局表类型，在应用使用DDB过程中，全局表的概念对应用不可见。

### 未来——独立平台，与云共舞

DDB作为网易浓缩了10年技术经验与精华的分库分表数据库，近一两年除了满足内部产品使用外，也渐渐开始帮助外部企业客户解决海量结构化数据存储的难题。随着公有云技术的大力发展和日趋成熟，各种IaaS和PaaS平台如雨后春笋层出不穷，如网易蜂巢的推出，为应用开发，部署和运维提供了极大便利，而随着IaaS层和PaaS平台的普及，各种SaaS服务也会慢慢为广大开发者所接纳，未来DDB也将重点为网易蜂巢客户打包DDB的SaaS服务，与蜂巢一同构建一套更加丰富的数据存储生态系统。

我们对DDB的SaaS服务化无比坚定，同时DDB的公有云之路绝非私有云的生搬硬套，在与蜂巢一同帮助企业客户解决分库分表难题的同时，未来我们也会更加注重平台独立，首先要做的是将DDB的SaaS层与底层PaaS和IaaS层解耦，实现将DDB平台所依赖的PaaS和IaaS以插件方式注入。这样一来可以为客户提供更灵活的服务方式，二来可以极大程度降低DDB平台本身的开发和运维成本：一套平台管理工具，适用所有内外部DDB用户，这是我们正在进行并将持续优化的目标。