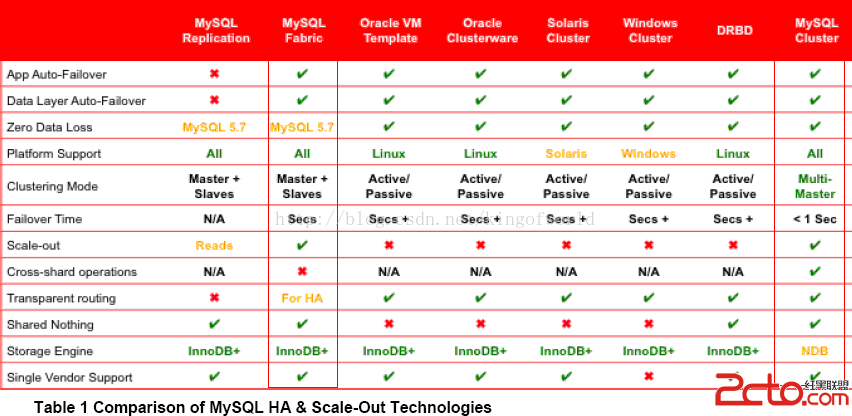
**实战体验几种MySQLCluster方案**

**1.背景**

MySQL的cluster方案有很多官方和第三方的选择，选择多就是一种烦恼，因此，我们考虑MySQL[数据库](http://www.2cto.com/database/)满足下三点需求，考察市面上可行的解决方案：

高可用性：主服务器故障后可自动切换到后备服务器可伸缩性：可方便通过脚本增加DB服务器负载均衡：支持手动把某公司的数据请求切换到另外的服务器，可配置哪些公司的数据服务访问哪个服务器

需要选用一种方案满足以上需求。在MySQL官方网站上参考了几种解决方案的优缺点：



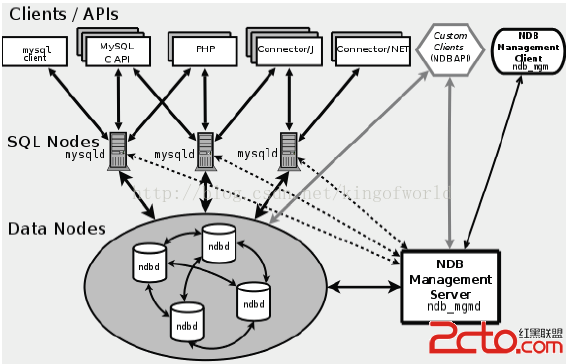
综合考虑，决定采用MySQL Fabric和MySQL Cluster方案，以及另外一种较成熟的集群方案Galera Cluster进行预研。

**2.MySQLCluster**

**简介：**

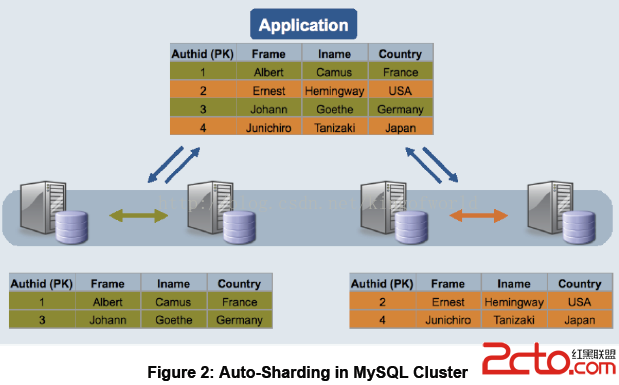
MySQL Cluster 是MySQL 官方集群部署方案，它的历史较久。支持通过自动分片支持读写扩展，通过实时备份冗余数据，是可用性最高的方案，声称可做到99.999%的可用性。

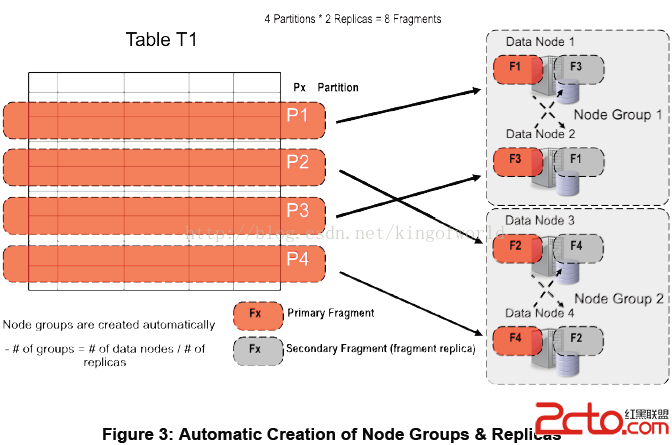
架构及实现原理：

****

MySQL cluster主要由三种类型的服务组成：

NDB Management Server：管理服务器主要用于管理cluster中的其他类型节点（Data Node和SQL Node），通过它可以配置Node信息，启动和停止Node。 SQL Node：在MySQL Cluster中，一个SQL Node就是一个使用NDB引擎的[mysql](http://www.2cto.com/database/MySQL/" \t "_blank) server进程，用于供外部应用提供集群数据的访问入口。Data Node：用于存储集群数据；[系统](http://www.2cto.com/os/" \t "_blank)会尽量将数据放在内存中。

****

****

缺点及限制：

对需要进行分片的表需要修改引擎Innodb为NDB，不需要分片的可以不修改。NDB的事务隔离级别只支持Read Committed，即一个事务在提交前，查询不到在事务内所做的修改；而Innodb支持所有的事务隔离级别，默认使用Repeatable Read，不存在这个问题。外键支持：虽然最新的Cluster版本已经支持外键，但性能有问题（因为外键所关联的记录可能在别的分片节点中），所以建议去掉所有外键。Data Node节点数据会被尽量放在内存中，对内存要求大。

数据库系统提供了四种事务隔离级别：  
A.Serializable（串行化）：一个事务在执行过程中完全看不到其他事务对数据库所做的更新（事务执行的时候不允许别的事务并发执行。事务串行化执行，事务只能一个接着一个地执行，而不能并发执行。）。  
B.Repeatable Read（可重复读）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，但是不能看到其他其他事务对已有记录的更新。  
C.Read Commited（读已提交数据）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，而且能看到其他事务已经提交的对已有记录的更新。  
D.Read Uncommitted（读未提交数据）：一个事务在执行过程中可以看到其他事务没有提交的新插入的记录，而且能看到其他事务没有提交的对已有记录的更新。

**3.MySQL Fabric**

简介：

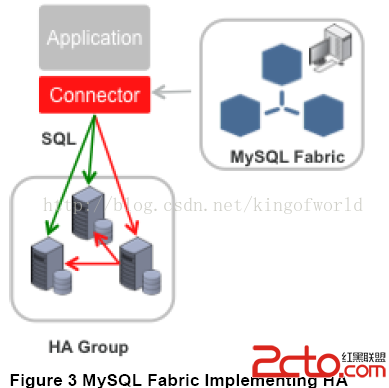
为了实现和方便管理MySQL 分片以及实现高可用部署，[Oracle](http://www.2cto.com/database/Oracle/" \t "_blank)在2014年5月推出了一套为各方寄予厚望的MySQL产品 -- MySQL Fabric, 用来管理MySQL 服务，提供扩展性和容易使用的系统，Fabric当前实现了两个特性：高可用和使用数据分片实现可扩展性和负载均衡，这两个特性能单独使用或结合使用。

MySQL Fabric 使用了一系列的python脚本实现。

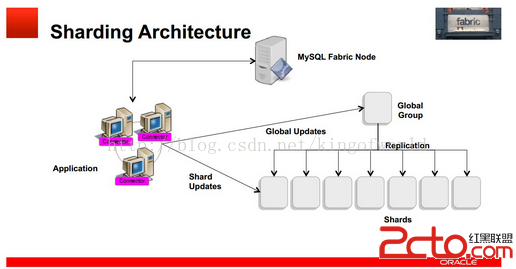
应用案例：由于该方案在去年才推出，目前在网上暂时没搜索到有大公司的应用案例。

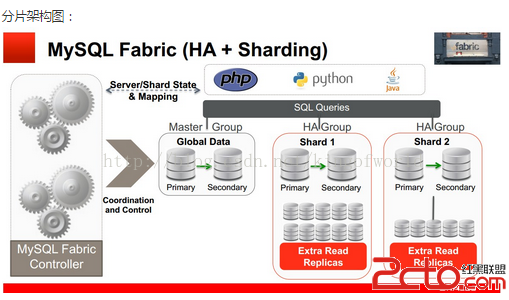
架构及实现原理：

Fabric支持实现高可用性的架构图如下：



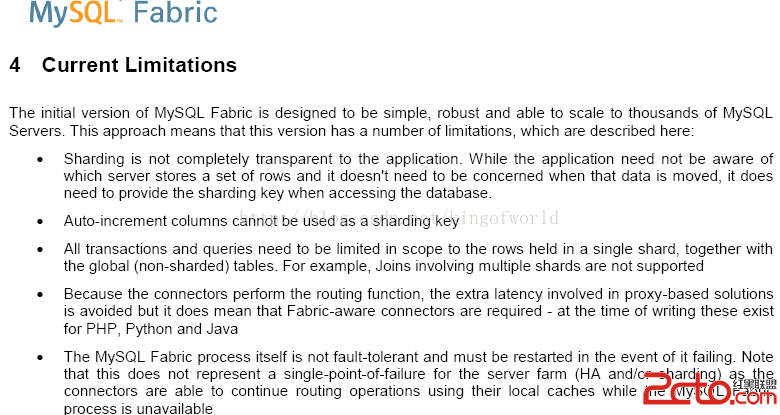
Fabric使用HA组实现高可用性，其中一台是主服务器，其他是备份服务器， 备份服务器通过同步复制实现数据冗余。应用程序使用特定的驱动，连接到Fabric 的Connector[组件](http://www.2cto.com/kf/all/zujian/)，当主服务器发生故障后，Connector自动升级其中一个备份服务器为主服务器，应用程序无需修改。  
  
Fabric支持可扩展性及负载均衡的架构如下：





使用多个HA 组实现分片，每个组之间分担不同的分片数据（组内的数据是冗余的，这个在高可用性中已经提到）  
应用程序只需向connector发送query和insert等语句，Connector通过MasterGroup自动分配这些数据到各个组，或从各个组中组合符合条件的数据，返回给应用程序。  
  
缺点及限制：  
影响比较大的两个限制是：

自增长键不能作为分片的键；事务及查询只支持在同一个分片内，事务中更新的数据不能跨分片，查询语句返回的数据也不能跨分片。



**测试高可用性**

服务器架构：

|  |  |  |
| --- | --- | --- |
| 功能 | IP | Port |
| Backing store(保存各服务器配置信息) | 200.200.168.24 | 3306 |
| Fabric 管理进程（Connector） | 200.200.168.24 | 32274 |
| HA Group 1 -- Master | 200.200.168.23 | 3306 |
| HA Group 1 -- Slave | 200.200.168.25 | 3306 |

安装过程省略，下面讲述如何设置高可用组、添加备份服务器等过程

首先，创建高可用组，例如组名group\_id-1，命令：

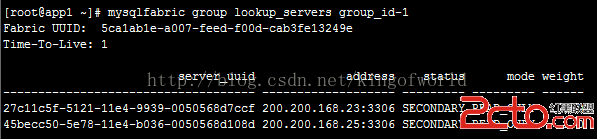
mysqlfabric group create group\_id-1

往组内group\_id-1添加机器200.200.168.25和200.200.168.23：

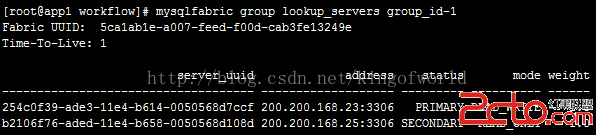
mysqlfabric group add group\_id-1 200.200.168.25:3306

mysqlfabric group add group\_id-1 200.200.168.23:3306

然后查看组内机器状态：



由于未设置主服务器，两个服务的状态都是SECONDARY  
提升其中一个为主服务器：  
mysqlfabric group promote group\_id-1 --slave\_id 00f9831f-d602-11e3-b65e-0800271119cb  
然后再查看状态：



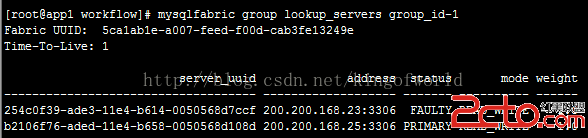
设置成主服务器的服务已经变成Primary。  
另外，mode属性表示该服务器是可读写（READ\_WRITE)，或只读(READ\_ONLY)，只读表示可以分摊查询数据的压力；只有主服务器能设置成可读写（READ\_WRITE)。  
这时检查25服务器的slave状态：



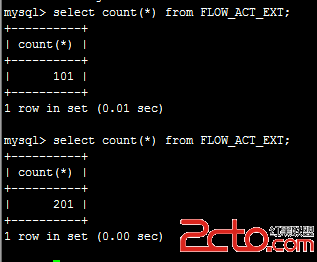
可以看到它的主服务器已经指向23  
  
  
然后激活故障自动切换功能：  
mysqlfabric group activate group\_id-1  
激活后即可测试服务的高可以性  
首先，进行状态测试：  
停止主服务器23

\

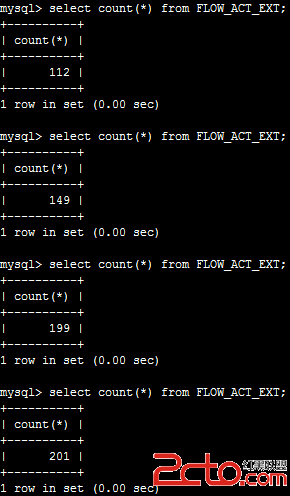
然后查看状态：



可以看到，这时将25自动提升为主服务器。  
但如果将23恢复起来后，需要手动重新设置23为主服务器。  
  
  
**实时性测试：**  
目的：测试在主服务更新数据后，备份服务器多久才显示这些数据  
测试案例：使用java代码建连接，往某张表插入100条记录，看备份服务器多久才能同步这100条数据  
测试结果：  
表中原来有101条数据，运行程序后，查看主服务器的数据条数：



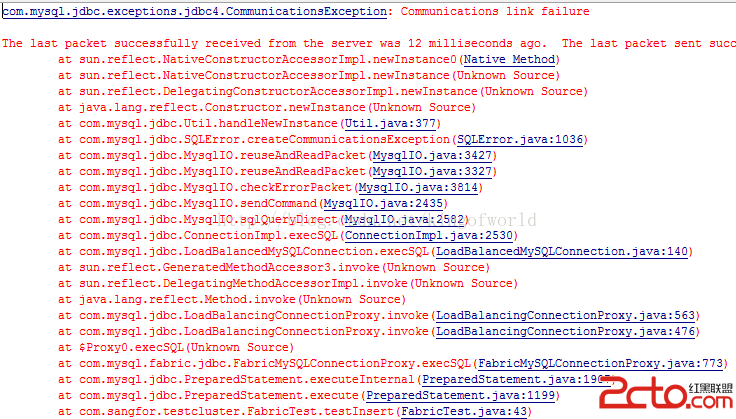
可见主服务器当然立即得到更新。  
  
查看备份服务器的数据条数：



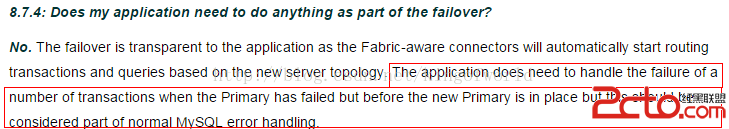
但备份服务器等待了1-2分钟才同步完成（可以看到fabric使用的是异步复制，这是默认方式，性能较好，主服务器不用等待备份服务器返回，但同步速度较慢）

对于从服务器同步数据稳定性问题，有以下解决方案：

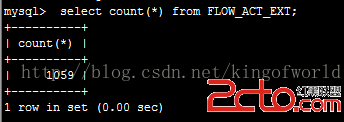
使用半同步加强数据一致性：异步复制能提供较好的性能，但主库只是把binlog日志发送给从库，动作就结束了，不会验证从库是否接收完毕，风险较高。半同步复制会在发送给从库后，等待从库发送确认信息后才返回。可以设置从库中同步日志的更新方式，从而减少从库同步的延迟，加快同步速度。 安装半同步复制：  
在mysql中运行  
install plugin rpl\_semi\_sync\_master soname 'semisync\_master.so';  
install plugin rpl\_semi\_sync\_slave soname 'semisync\_slave.so';  
SET GLOBAL rpl\_semi\_sync\_master\_enabled=ON;  
SET GLOBAL rpl\_semi\_sync\_slave\_enabled=ON;  
修改my.cnf ：  
rpl\_semi\_sync\_master\_enabled=1  
rpl\_semi\_sync\_slave\_enabled=1  
sync\_relay\_log=1  
sync\_relay\_log\_info=1  
sync\_master\_info=1  
  
**稳定性测试：**  
测试案例：使用java代码建连接，往某张表插入1w条记录，插入过程中将其中的master服务器停了，看备份服务器是否有这1w笔记录  
测试结果，停止主服务器后，java程序抛出异常：



但这时再次发送sql命令，可以成功返回。证明只是当时的事务失败了。连接切换到了备份服务器，仍然可用。  
翻阅了mysql文档，有章节说明了这个问题：

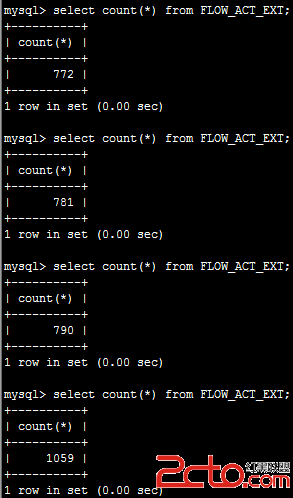


里面提到：当主服务器当机时，我们的应用程序虽然是不需做任何修改的，但在主服务器被备份服务器替换前，某些事务会丢失，这些可以作为正常的mysql错误来处理。  
  
**数据完整性校验：**  
测试主服务器停止后，备份服务器是否能够同步所有数据。  
重启了刚才停止主服务器后，查看记录数



可以看到在插入1059条记录后被停止了。

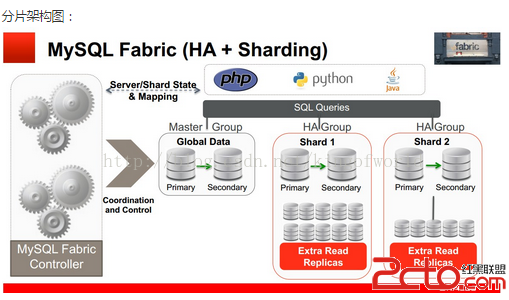
现在看看备份服务器的记录数是多少，看看在主服务器当机后是否所有数据都能同步过来



大约经过了几十秒，才同步完，数据虽然不是立即同步过来，但没有丢失。

**1.2、分片：如何支持可扩展性和负载均衡**

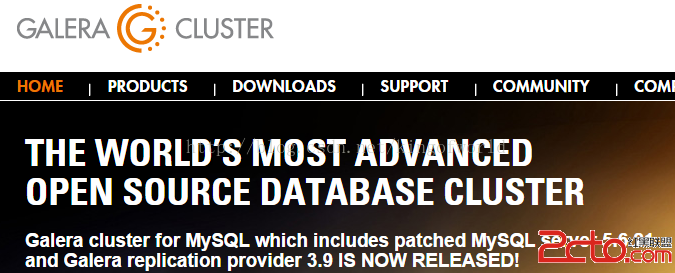
fabric分片简介：当一台机器或一个组承受不了服务压力后，可以添加服务器分摊读写压力，通过Fabirc的分片功能可以将某些表中数据分散存储到不同服务器。我们可以设定分配数据存储的规则，通过在表中设置分片key设置分配的规则。另外，有些表的数据可能并不需要分片存储，需要将整张表存储在同一个服务器中，可以将设置一个全局组（Global Group）用于存储这些数据，存储到全局组的数据会自动拷贝到其他所有的分片组中。



**4.Galera Cluster**

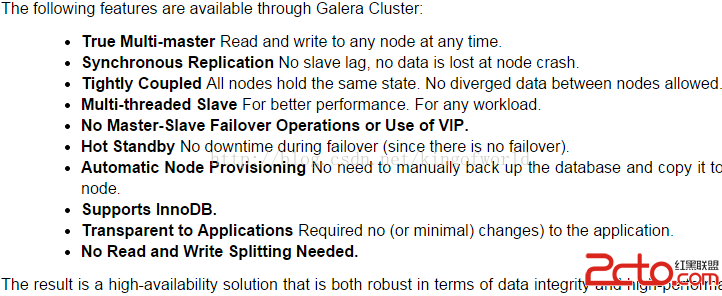
简介：

Galera Cluster号称是世界上最先进的开源数据库集群方案

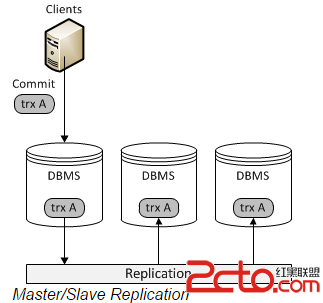


主要优点及特性：

真正的多主服务模式：多个服务能同时被读写，不像Fabric那样某些服务只能作备份用同步复制：无延迟复制，不会产生数据丢失热备用：当某台服务器当机后，备用服务器会自动接管，不会产生任何当机时间自动扩展节点：新增服务器时，不需手工复制数据库到新的节点支持InnoDB引擎对应用程序透明：应用程序不需作修改

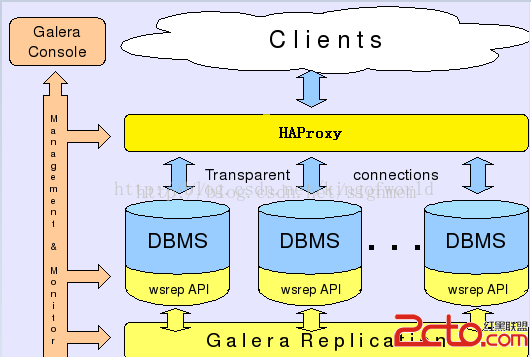


**架构及实现原理：**  
首先，我们看看传统的基于mysql Replication（复制）的架构图：



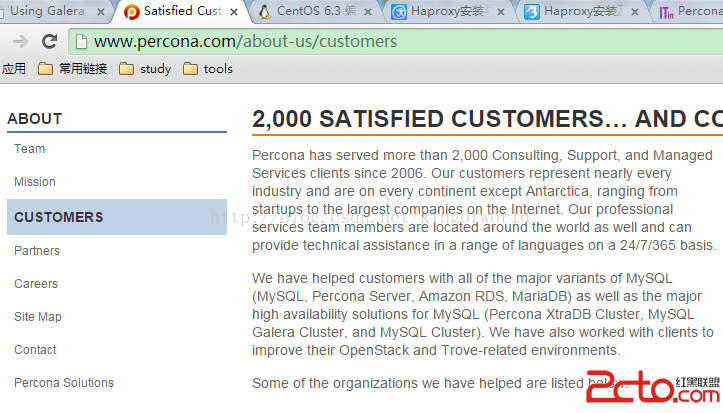
Replication方式是通过启动复制线程从主服务器上拷贝更新日志，让后传送到备份服务器上执行，这种方式存在事务丢失及同步不及时的风险。Fabric以及传统的主从复制都是使用这种实现方式。

而Galera则采用以下架构保证事务在所有机器的一致性：



客户端通过Galera Load Balancer访问数据库，提交的每个事务都会通过wsrep API 在所有服务器中执行，要不所有服务器都执行成功，要不就所有都回滚，保证所有服务的数据一致性，而且所有服务器同步实时更新。  
  
  
缺点及限制：

由于同一个事务需要在集群的多台机器上执行，因此网络传输及并发执行会导致性能上有一定的消耗。所有机器上都存储着相同的数据，全冗余。若一台机器既作为主服务器，又作为备份服务器，出现乐观锁导致rollback的概率会增大，编写程序时要小心。不支持的SQL：LOCK / UNLOCK TABLES / GET\_LOCK(), RELEASE\_LOCK()…不支持XA Transaction  
目前基于Galera Cluster的实现方案有三种：Galera Cluster for MySQL、Percona XtraDB Cluster、MariaDB Galera Cluster。  
我们采用较成熟、应用案例较多的Percona XtraDB Cluster。  
应用案例：  
超过2000多家外国企业使用：



包括：

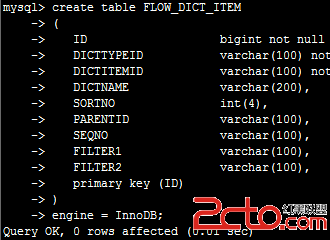


集群部署架构：

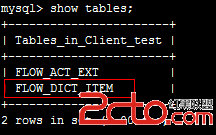
|  |  |  |
| --- | --- | --- |
| 功能 | IP | Port |
| Backing store(保存各服务器配置信息) | 200.200.168.24 | 3306 |
| Fabric 管理进程（Connector） | 200.200.168.24 | 32274 |
| HA Master 1 | 200.200.168.24 | 3306 |
| HA Master 2 | 200.200.168.25 | 3306 |
| HA Master 3 | 200.200.168.23 | 3306 |

**4.1、测试数据同步**

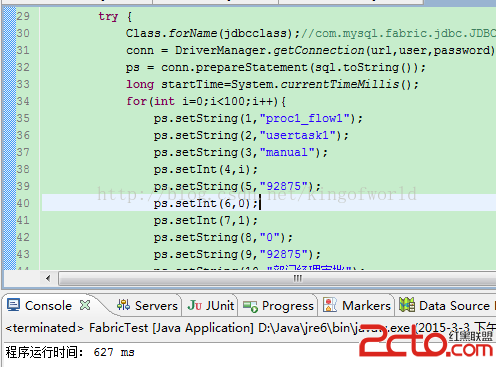
在机器24上创建一个表：



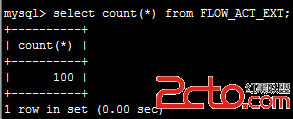
立即在25 中查看，可见已被同步创建



使用[Java](http://www.2cto.com/kf/ware/Java/" \t "_blank)代码在24服务器上插入100条记录



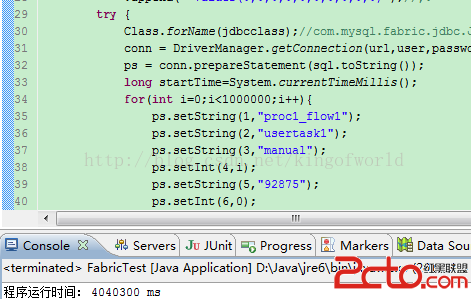
立即在25服务器上查看记录数



可见数据同步是立即生效的。  
  
**4.2、测试添加集群节点**  
添加一个集群节点的步骤很简单，只要在新加入的机器上部署好Percona XtraDB Cluster，然后启动，系统将自动将现存集群中的数据同步到新的机器上。  
  
现在为了测试，先将其中一个节点服务停止：

\

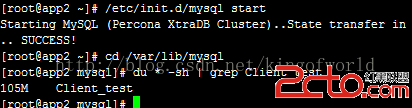
然后使用java代码在集群上插入100W数据



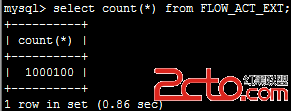
查看100w数据的数据库大小：

\

这时启动另外一个节点，启动时即会自动同步集群的数据：



启动只需20秒左右，查看数据大小一致，查看表记录数，也已经同步过来

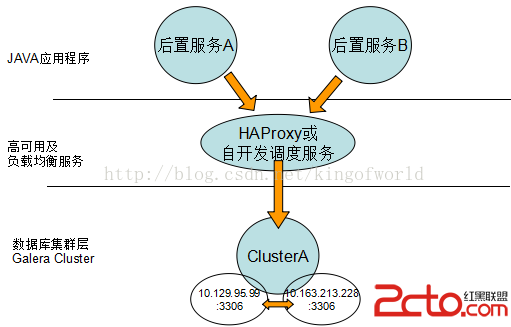


**5.对比总结**

|  |  |  |
| --- | --- | --- |
|  | MySQL Fabric | Galera Cluster |
| 使用案例 | 2014年5月才推出，目前在网上暂时没搜索到有大公司的应用案例 | 方案较成熟，外国多家互联网公司使用 |
| 数据备份的实时性 | 由于使用异步复制，一般延时几十秒，但数据不会丢失。 | 实时同步，数据不会丢失 |
| 数据冗余 | 使用分片，通过设置分片key规则可以将同一张表的不同数据分散在多台机器中 | 每个节点全冗余，没有分片 |
| 高可用性 | 通过Fabric Connector实现主服务器当机后的自动切换，但由于备份延迟，切换后可能不能立即查询数据 | 使用HAProxy实现。由于实时同步，切换的可用性更高。 |
| 可伸缩性 | 添加节点后，需要先手工复制集群数据 | 扩展节点十分方便，启动节点时自动同步集群数据，100w数据（100M）只需20秒左右 |
| 负载均衡 | 通过HASharding实现 | 使用HAProxy实现负载均衡 |
| 程序修改 | 需要切换成jdbc:mysql:fabric的jdbc类和url | 程序无需修改 |
| 性能对比 | 使用java直接用jdbc插入100条记录，大概2000+ms | 跟直接操作mysql一样，直接用jdbc插入100条记录，大概600ms |

**6.实践应用**

综合考虑上面方案的优缺点，我们比较偏向选择Galera 如果只有两台数据库服务器，考虑采用以下数据库架构实现高可用性、负载均衡和动态扩展：



如果三台机器可以考虑：

