



OceanBase 客户端 用户指南

文档版本：01

发布日期：2013.7.30

支付宝（中国）网络技术有限公司·OceanBase 团队

前言

概述

本文档介绍OceanBase数据库客户端的架构和使用方法。

读者对象

本文档主要适用于：

- Java 开发工程师。
- C 开发工程师。
- 安装工程师。

通用约定

在本文档中可能出现下列各式，它们所代表的含义如下。

格式	说明
警告	表示可能导致设备损坏、数据丢失或不可预知的结果。
注意	表示可能导致设备性能降低、服务不可用。
小窍门	可以帮助您解决某个问题或节省您的时间。
说明	表示正文的附加信息，是对正文的强调和补充。
宋体	表示正文。
粗体	表示命令行中的关键字（命令中保持不变、必须照输的部分）或者正文中强调的内容。
斜体	用于变量输入。
{ a b ... }	表示从两个或多个选项中选取一个。
[]	表示用“[]”括起来的部分在命令配置时是可选的。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本。

联系我们

如果您有任何疑问或是想了解 OceanBase 的最新开源动态消息，请联系我们：

支付宝（中国）网络技术有限公司·OceanBase 团队

地址：杭州市万塘路 18 号黄龙时代广场 B 座；邮编：310099

北京市朝阳区东三环中路 1 号环球金融中心西塔 14 层；邮编：100020

邮箱：rongxuan.lc@alipay.com

新浪微博：<http://weibo.com/u/2356115944>

技术交流群（阿里旺旺）：853923637

目 录

1 概述.....	- 1 -
1.1 客户端简介.....	- 1 -
1.2 获取安装包	- 1 -
2 Java 客户端	- 3 -
2.1 客户端结构	- 3 -
2.2 功能模块	- 4 -
2.2.1 ObGroupDatasource	- 5 -
2.2.2 OceanbaseDataSourceProxy	- 6 -
2.2.3 OB Config.....	- 8 -
2.3 利用 OceanbaseDataSourceProxy 访问 OceanBase	- 9 -
2.3.1 初始化流程	- 9 -
2.3.2 示例	- 9 -
2.4 利用 ObGroupDataSource 访问 OceanBase	- 11 -
2.4.1 访问方式说明	- 11 -
2.4.2 示例	- 12 -
2.5 利用 Merge Server 访问 OceanBase	- 13 -
3 C 客户端.....	- 16 -
3.1 客户端结构	- 16 -
3.2 访问流程	- 17 -
3.3 安装 C 客户端.....	- 17 -
3.4 示例	- 19 -
4 附录.....	- 22 -
4.1 Java 客户端流量分配执行实现细节	- 22 -
4.2 Java 客户端定时 Merge Server 重新生效流程	- 22 -
4.3 Java 客户端日志梳理	- 23 -
4.4 C 客户端流量分配和重试设计.....	- 23 -
4.4.1 流量分配.....	- 23 -
4.1.2 重试.....	- 24 -

1 概述

主要介绍客户端的基础知识以及安装包的获取方法。

1.1 客户端简介

OceanBase 客户端主要用于开发人员编程时连接 OceanBase 数据库。

Oceanbase 内置了对 SQL 的支持，用户可以通过 libmysql、JDBC 等方式直接访问 Oceanbase，但由于 OceanBase 是一个分布式数据库，可以由多个节点（Merge Server）同时提供 SQL 服务。而 MySQL 客户端等都是针对单机系统，在连接 OceanBase 时，客户端会绑定其中一台 Merge Server 进行 SQL 操作，不能有效利用其他 Merge Server 资源。

为了实现了多集群间流量分配和多 Merge Server 间的负载均衡，并给应用开发人员提供一个简单接入方案，我们在 libmysql、JDBC 的基础上封装一个 OceanBase 客户端。

OceanBase 客户端可以根据“流量分配”选择 Merge Server。例如：主集群比例为 40，备集群比例为 60。那么在发送弱一致性读 SQL 语句时，将会有 40% 的概率发送至主集群中的 Merge Server，60% 的概率发送至备集群中的 Merge Server。

流量分配只涉及弱一致性读请求，所有非弱一致性读请求全部发送至主集群。

说明：

- 流量分配：主要指弱一致性读在主备集群中的分配比例。可以在内部表“__all_cluster”中设置。
- 弱一致性读请求包括：hint weak sql、无 hint 的 select，除此之外的 SQL 语句类型为非弱一致性读请求。

OceanBase 客户端主要有以下两种：

- C 客户端
为应用程序提供了“libobsql.so”动态库，这个库在二进制接口上与 mysql 的 libmysqlclient 库完全兼容。
- Java 客户端
提供了符合 Java 标准的 DataSource，Java 应用程序可以使用 Java 客户端获得与 OceanBase 服务器交互的连接。

1.2 获取安装包

OceanBase 客户端安装包的获取方式和说明如[表 1-1](#)所示。

表 1-1 安装包

类型	安装包	获取地址
C 客户端	<p>Linux 版本为 RedHat 5 的安装包：</p> <ul style="list-style-type: none"> • curl-7.29.0-1.el5.x86_64.rpm • oceanbase-devel-0.4.2.1-1193.el5.x86_64.rpm <p>Linux 版本为 RedHat 6 的安装包：</p> <ul style="list-style-type: none"> • curl-7.29.0-1.el6.x86_64.rpm • oceanbase-devel-0.4.2.1-1193.el6.x86_64.rpm <p><i>说明：</i> 您可以执行 <code>cat /etc/issue</code> 命令查看 Linux 版本号。</p>	<p>https://github.com/alibaba/oceanbase 的“oceanbase_0.4”分支下的“client_package/c”文件夹中。</p>
Java 客户端	<ul style="list-style-type: none"> • commons-lang-2.3.jar • commons-logging-1.1.jar • druid-0.2.12.jar • mysql-connector-java-5.1.14.jar • oceanbase-1.0.1.jar • oceanbase-core-1.1.0.jar 	<p>https://github.com/alibaba/oceanbase 的“oceanbase_0.4”分支下的“client_package/java”文件夹中。</p>

2 Java 客户端

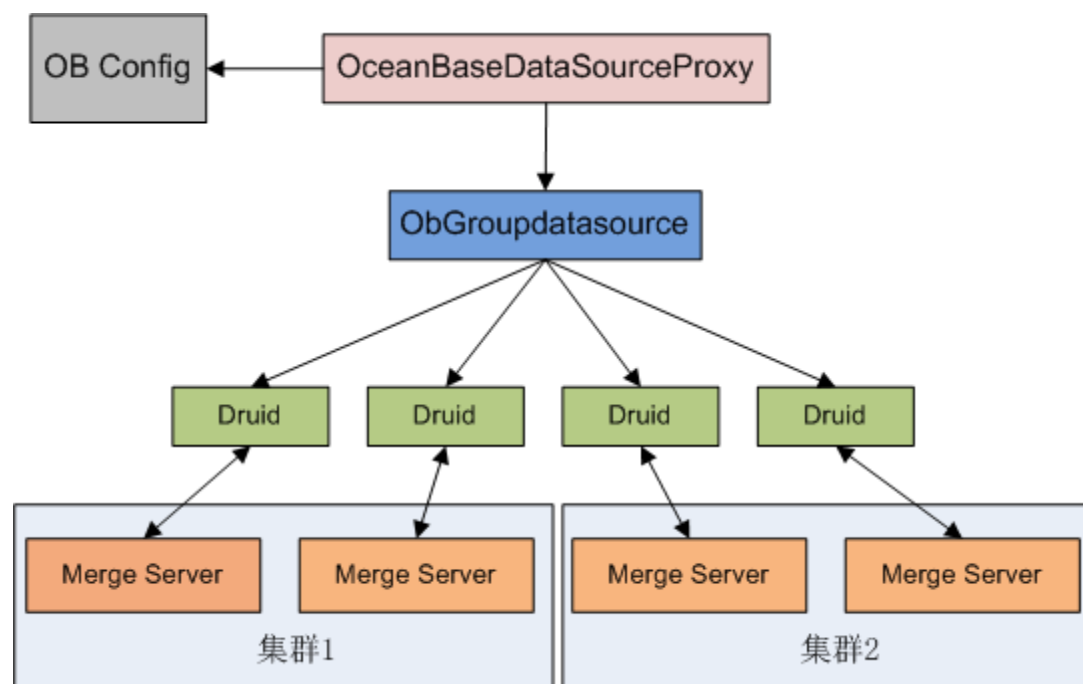
Java 客户端主要用于开发人员编写的 Java 程序连接 OceanBase 数据库。

Java 客户端是 `javax.sql.DataSource` 的一个实现，应用程序可以通过 Java 客户端调用 MySQL JDBC 驱动连接 OceanBase 读写数据。

2.1 客户端结构

JAVA 客户端的结构示意图如 [图 2-1](#) 所示。

图 2-1 结构示意图



JAVA 客户端的结构介绍如下：

- **Merge Sever**
OceanBase 中负责提供 SQL 服务的接口模块，用于处理客户端发送的 SQL 请求。
- **Druid**
数据源模块，是阿里巴巴开发的开源的数据库连接池。每个 Druid 数据源对应一个 Merge Server，用于维护到 Merge Server 的连接。关于 Druid 详细介绍请参见 [“<http://code.alibabatech.com/wiki/display/Druid/Home>”](http://code.alibabatech.com/wiki/display/Druid/Home)。

- **ObGroupDataSource**
也称为 **ObDataSource Core** 模块，在 **OceanBase** 集群中包含有多个 **Merge Server**，所以需要部署多个 **Druid** 来与之对应。
ObGroupDataSource 封装了这些 **Druid** 数据源，并且提供负载均衡、流量分配等功能。
- **OceanBaseDataSourceProxy**
封装了 **ObGroupDataSource**，是 **OceanBase** **JAVA** 客户端提供给用户使用的数据库连接访问的接口。
- **OB Config**
OceanBase 客户端的配置管理中心，实现客户端获取 **OceanBase** 集群地址的功能，管理多个版本客户端库和元数据，并通过配置项管理客户端的升级。用户只需要获取到 **OB Config** 的地址以及 **OceanBase** 数据库的用户名和密码就能获取到数据库的访问地址以及流量分配等信息，从而初始化整个 **JAVA** 客户端。

Java 客户端按功能主要可以分为两种使用场景，如[表 2-1](#)所示。

表 2-1 场景说明

功能	适用人员	所需模块	示例
自动升级或回滚客户端	阿里巴巴内部人员	<ul style="list-style-type: none"> • Merge Server • Druid • OceanBaseDataSourceProxy • OB Config 	请参见“2.3 利用 OceanbaseDataSourceProxy 访问 OceanBase”。
通过手动修改 pom 文件进行升级或回滚客户端	阿里巴巴内部人员和非阿里巴巴内部人员	<ul style="list-style-type: none"> • Merge Server • Druid • ObGroupDataSource 	请参见“2.4 利用 ObGroupDataSource 访问 OceanBase”。

说明：Java 应用连接 **OceanBase**，除了上述两种场景外，还可以直接通过 **Merge Server** 连接。该方式不通过 **Java** 客户端，详情请参见“2.5 利用 **Merge Server** 访问 **OceanBase**”。

2.2 功能模块

Java 客户端的功能模块主要包括 **ObGroupDatasource**、**OceanbaseDataSourceProxy** 和 **OB Config**。

对于非阿里巴巴内部的使用人员，您只需要关心 **ObGroupDatasource**。
ObGroupDatasource 封装了 **OceanBase** 数据库的流量分配、负载均衡的功能，你可以直接使用 **ObGroupDatasource** 提供的接口方便的访问 **OceanBase** 数据库，**ObGroupDatasource** 的代码将在近期开源。

OceanBaseDataSourceProxy 和 **OB Config** 属于 **OceanBase** 团队内部使用的客户端模块，在一段相当长的时间内不会将相关的资源进行开源。

2.2.1 ObGroupDatasource

ObGroupDatasource 也称为“ObDataSource Core”，其主要功能为实现集群流量分配、单集群内多 Merge Server 负载均衡、管理 Merge Server 上下线和集群 failover 处理。

ObGroupDatasource 的 jar 包形式为“oceanbase-core-XXX.jar”，其依赖的包为“druid-XXX.jar”，其中“XXX”为版本号。

* 集群流量分配

集群流量分配主要是为了控制每个集群的 SQL 请求量。根据内部表“__all_cluster”中的“cluster_flow_percent”字段值来分配主备集群中的 SQL 请求量。

流量分配只涉及弱一致性读请求，所有非弱一致性读请求全部发送至主集群。

流量的分配情况不是完全精确的分配，会出现偏差。主要原因是由于非弱一致性请求只能发往主集群，当主集群在执行非弱一致性读请求时，所有的弱一致性读请求将发送至备集群，直到主集群的非弱一致性请求处理完毕。

当 SQL 请求经过第一步的集群负载之后，就流入了某个具体的集群。在同一集群中的 Merge Server 选择方式由内部表“__all_cluster”中的“read_strategy”字段值决定：

- 0：随机策略。
- 1：一致性哈希策略。

* 管理 Merge Server 上下线

obdatasource core 的调度线程会定时扫描内部表“__all_server”中 Merge Server 列表，对于上线或下线的 Merge Server，将新增或者销毁数据源（Druid）。

* 集群 failover 处理

OceanBase 数据库异常主要有以下几种：

- 连接类异常
客户端和 OceanBase 建立底层 TCP 连接时，发生的异常。如：数据库主机故障、数据库服务故障、机房（网络设备）故障等。
- 特殊类异常
客户端和 OceanBase 能够建立 TCP 连接，但是由于 OceanBase 发生内部错误，导致在应用层所建立的连接失败。如：no memory, login error。
- 执行类异常
客户端和 OceanBase 正确建立连接之后，在执行 SQL 语句时发生的异常。如：SQL 语法异常、无执行权限等。

OceanBase 针对以上的三种异常类型，明确了三种不同的处理逻辑。

- 对于连接类异常，根据执行的 SQL 的类型以及集群状况来选择是否需要重试。
- 对于特殊类异常，将通知物理连接池回收掉腐化的连接。
- 对于执行类异常，不执行任何操作。

集群 failover 处理主要针对“连接类异常”：

- 如何判定 Merge Server 故障
当 Merge Server 的 SQL 处理请求异常时，为该 Merge Server 的 SQL 计数器“+1”。如果一分钟内计数器值累计大于 47.63，则判定该 Merge Server 为不可用。
判定一台 Merge Server 故障后，在其他可用的 Merge Server 中进行重新选择，直至选择到正常的 Merge Server。如果该集群已经没有可用的 Merge Server，则抛出异常。
- 如何判定集群故障
查看该集群下的所有 Merge Server 是否都为故障。如果全部故障，则判定该集群为故障。
当请求为弱一致性读请求时，主集群故障时，会重选集群，将请求发送至备集群。否则，则抛出异常。

2.2.2 OceanbaseDataSourceProxy

OceanbaseDataSourceProxy 的主要负责获取 OB Config 上的配置信息，并完成自身模块（配置中心交互模块、本地 cache 模块、自动升级模块、客户端汇报模块）和 ObGroupDatasource 初始化。

OceanbaseDataSourceProxy 属于 OceanBase 团队内部使用的客户端模块，在一段相当长的时间内不会将相关的资源进行开源。提供服务的 jar 包为“oceanbase-XXX.jar”，其中“XXX”为版本号。

* 配置中心交互模块

该模块主要负责从 OB Config 上获取数据源的配置项，并初始化 ObGroupDatasource。

说明：OB Config 上保存着“oceanbase-core”文件，即 ObGroupDatasource 模块。用于 Java 应用程序第一次初始化时，通过 OceanbaseDataSourceProxy 的“本地 cache 模块”从 OB Config 缓存到本地，而不需要手动导入“oceanbase-core-XXX.jar”。

OceanbaseDataSourceProxy 会定期与 OB Config 交互，对比本地与 OB Config 上 ObGroupDatasource 的版本以及升级配置项，从而确认是否需要升级。

配置中心交互模块向 OB Config 传入的信息有以下两种：

- 传入 url、username 和 password
主要获取 OceanBase 主集群的 Listner 地址等信息。然后

OceanbaseDataSourceProxy 就能够通过 Listner 读取内部表 “__all_cluster” 中的集群的流量分配比、集群的负载策略，以及 “__all_server” 中的所有 Merge Server 信息。

- 只传入 url
主要获取 OceanBase 主集群的 Listner 地址，以及连接 OceanBase 的用户名、密码等信息。然后 OceanbaseDataSourceProxy 就能够通过 Listner 读取内部表 “__all_cluster” 中的集群的流量分配比、集群的负载策略，以及 “__all_server” 中的所有 Merge Server 信息。

* 本地 cache 模块

本地 cache 模块将 OB Config 上的 ObGroupDatasource 包缓存到本地 “\$USER_HOME/.obdatasource/\$version”。

*说明：*其中 “\$USER_HOME” 为本地用户路径，“\$version” 为其版本号。

从配置中心获取 ObGroupDatasource，主要作以下考虑：

- 减少对 OceanBase 配置中心的访问，缓解 OceanBase 配置中心压力。
- 方便控制 ObGroupDatasource 升级。
- 方便控制 ObGroupDatasource 中所使用的加密密钥。

当 ObGroupDatasource 版本升级或者本地 cache 删除时，客户端会再次访问 OceanBase 配置中心，并重新获取并保存。

* 自动升级模块

该模块主要负责自动升级 ObGroupDatasource。自动升级模式主要有以下几种：

- 白名单升级
在 OceanBase 配置中心设置指定需要升级的客户端，其他客户端不升级。
例如：某业务系统集群的客户端机器列表为 “10.10.10.1，10.10.10.2，10.10.10.3，10.10.10.4”。在 OceanBase 配置中心设置白名单为：10.10.10.1，则单独为 10.10.10.1 升级。
- 百分比升级
在 OceanBase 配置中心设置设置集群升级的比例，完成该占比机器的升级。
例如：某业务系统集群的客户端机器列表为 “10.10.10.1，10.10.10.2，10.10.10.3，10.10.10.4”。在 OceanBase 配置中心设置升级比例为 50%，则将为 4 台客户端中的 2 台完成升级（理论值）。
同时，可以通过逐渐变化升级的比例。比如第一次设置 25%，完成集群的四分之一升级；然后通过设置比例为 50%，完成集群的二分之一升级；通过逐步提升，到最终完成集群的所有机器升级至指定版本。
- 回滚
恢复到上一次所使用的版本。

ObGroupDatasource 每次升级时，原数据源和新数据源同时存在，原数据源在 5 分钟后失效。例如：ObGroupDatasource 从 A 版本升级到 B 版本。

- 5 分钟内，A 版本和 B 版本的数据源同时存在，此时，ObGroupDatasource 可以直接回滚到版本 A，不能够升级到其他版本。
- 5 分钟后，A 版本失效，支持升级到其他版本。如需回滚到 A 版本，则视为从 B 版本“升级”到 A 版本。

* 客户端汇报模块

在 Java 应用程序第一次初始化时，将汇报客户端机器的 IP 以及所使用的 ObGroupDatasource 的版本号。如果 ObGroupDatasource 升级，则重新汇报。

2.2.3 OB Config

OB Config 是 OceanBase 客户端的配置管理中心，属于 OceanBase 团队内部使用的客户端模块。它实现了客户端获取 OceanBase 集群地址的功能，管理多个版本客户端库和元数据，并通过配置项管理客户端的自动升级。

OB Config 支持两种接口：

- HTTP GET：用于 Java 客户端获取 OceanBase 应用的 Listener 地址列表。
- HTTP POST：用于修改 OceanBase 应用的 listener 地址列表。

DBA 可以通过 curl 之类的工具直接修改 OceanBase 应用的 Listener 地址。

在 OceanBase 监控中心中提供了 OB Config 的 WEB 界面，可以直接通过该界面进行设置。OB Config 统一管理了多个 OceanBase 的应用集群，以 dataId 作为主键进行应用集群的区分。不同应用的 OB Config 的 URL 地址，只需要修改为对应的集群名即可。

OB Config 对外提供统一的域名访问（oceanbase.alibaba-inc.com）。客户端只需要配置一个 OB Config 服务的 DNS 地址，就会自动定位到某个机房的 OB Config 服务。OB Config 选择 DNS 容错的方式，在两个机房部署 OB Config 服务。这种方式实现比较简单，但是，如果某个机房的 OB Config 服务长期不可用，DNS 还是会查询到该地址。如果要修改 DNS，则需要走一遍“DNS 变更流程”，周期很长。另外，这种方式还有一个好处就是以后迁移 OB Config 到另外一个 IP 可以不需要客户端走重新发布流程。

OB Config 服务的功能如下：

- 管理 OceanBase 集群的 Listener 列表，提供查询和更改操作。
- 提供上传和下载“oceanbase-core”文件功能。将“oceanbase-core”文件上传到 Ob Config 服务器后，Ob Config 除了保存 jar 文件本身，还需要保存文件的 md5 值。另外，Ob Config 还需要存储最近上传的多个版本的 jar 文件（建议为 5 个），方便回滚。

说明：“oceanbase-core”文件即 ObGroupDatasource 模块，用于 Java 应用程序第一次初始化时，通过 OceanbaseDataSourceProxy 的“本地 cache 模块”从 OB Config 缓存到本地。而不需要手动导入“oceanbase-core-XXX.jar”。

- 提供读取最新版本的 oceanbase-core 文件的元数据的功能。包括最后更新时间、版本号、md5 值、百分比、允许升级的客户端机器 IP 白名单。
- 管理客户端的升级。当有新版本的客户端出现时，通过配置百分比和允许升级的客户端机器 IP 白名单的方式，通知客户端自动进行版本的升级。

Java 客户端首先从 OB Config 读取元数据，并根据版本号信息构造 HTTP 请求下载相应的“oceanbase-core”文件，检查“last_update_time_ms”是否比之前更新，执行 md5 校验。如果客户端需要升级，则替换自身依赖的“oceanbase-core”文件。

2.3 利用 OceanbaseDataSourceProxy 访问 OceanBase

由于 OceanBaseDataSourceProxy 和 OB Config 属于 OceanBase 团队内部使用的客户端模块，在一段相当长的时间内不会将相关的资源进行开源，因此，对于非阿里巴巴内部的使用人员，怎么搭建和使用 OB Config 是未知的。

2.3.1 初始化流程

在使用利用 OceanbaseDataSourceProxy 访问 OceanBase 时，JAVA 程序需要初始化 OceanBaseDataSourceProxy。在初始化过程中，OceanBaseDataSourceProxy 会完成如下几件事情：

1. OceanBaseDataSourceProxy 访问 OB Config 配置中心，从配置中心获取 OceanBase 数据库中 Listener 的位置信息，同时 OB Config 也会返回 ObGroupDataSource 的最新版本号、是否允许进行主动升级、升级的机器比例、升级白名单等信息。
2. OceanBaseDataSourceProxy 会根据这些信息来判断是否要主动升级 ObGroupDataSource 的版本。如果需要升级，则会将 ObGroupDataSource 的新版本缓存到 JAVA 程序的本地。
3. OceanBaseDataSourceProxy 访问 OceanBase 数据库中 Listener，从而获取所有可供使用的 Merge Server 的地址列表和流量分配信息，并为每个 MergeServer 建立一个 Druid 数据源。
4. OceanBaseDataSourceProxy 根据流量分配的原则，从一个 Druid 数据源中选择一个连接给 JAVA 程序使用。等到应用断开连接后，该连接重新回到 Druid 数据源中等待下一次使用。

说明：Listener 是 OceanBase 集群内一个特殊的 Merge Server，主要用来提供获取 Merge Server 列表、集群流量分配的进程。

2.3.2 示例

* 前提条件

- OceanBase 主备集群已经安装且正常运行。
- OB Config 搭建完成且正常运行。
- Java 开发环境以及部署完成，如已经安装 JDK、Eclipse 等。
- 已经获取 Java 客户端 Jar 包：“oceanbase-1.0.1.jar”和“druid-0.2.12.jar”。
- 已经获取依赖包：“commons-lang-2.3.jar”、“commons-logging-1.1.jar”和“mysql-connector-java-5.1.14.jar”。

* 案例操作流程

1. 连接 OceanBase 数据库。
2. 新建 PreparedStatement 语句，来查询__all_server 表中 svr_port 为某个值的 server 信息。
3. 指定查询__all_server 表中 svr_port 为 2600 的 server 信息。
4. 打印查询结果。

* 预期结果

Eclipse Console 界面将打印出所有端口号为 2600 的 server 信息。

* 举例

编辑 Java 代码，如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class OceanBaseDataSourceProxyExample {
    public static void main(String[] args) throws Exception {
        useDataSource();
        System.exit(0);
    }
    public static void useDataSource() throws Exception {
        OceanbaseDataSourceProxy datasource = new OceanbaseDataSourceProxy();
        datasource.setConfigURL("http://obconsole.test.alibaba-inc.com/ob-
                                config/config.co?dataId=alipay_junbian");
        datasource.setUsername("admin");
        datasource.setPassword("admin");
        datasource.init();
        Connection conn = datasource.getConnection();
        PreparedStatement pstmt = conn
            .prepareStatement("select * from __all_server where svr_port = ?");
```

```

        pstmt.setInt(1, 2600);
        ResultSet rs = pstmt.executeQuery();
        ResultSetMetaData rSetMetaData = rs.getMetaData();
        for (int i = 1; i <= rSetMetaData.getColumnCount(); i++) {
            System.out.print("|" + rSetMetaData.getColumnName(i) + "");
        }
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= rSetMetaData.getColumnCount(); i++) {
                System.out.print("|" + rs.getString(i) + "");
            }
            System.out.println();
        }
        if (rs != null) {
            rs.close();
        }
        if (pstmt != null) {
            pstmt.close();
        }
        if (conn != null) {
            conn.close();
        }
        datasource.destroy();
    }
}

```

如果 Eclipse Console 界面输出如下信息则表明运行正常。其中第一段为“__all_server”表的 schema 信息，而第二段才是真正的输出结果行。

```

|gm_create|gm_modify|cluster_id|svr_type|svr_ip|svr_port|inner_port|svr_role|svr_version
|2013-05-27 21:54:30.08001|2013-05-29
17:54:45.81237|0|chunkserver|10.209.144.31|2600|0|0|0.4.1.2_13189M(Apr 10 2013
12:40:32)

```

2.4 利用 ObGroupDataSource 访问 OceanBase

由于 ObGroupDataSource 的 jar 包开放计划正在筹划当中，您可能暂时无法获取 jar 包，敬请关注最新版本。

2.4.1 访问方式说明

应用程序会根据传入的 Listener 的地址从集群中获取 MergeServer 的列表，并且根据集群流量分配信息从中选择一个 Merge Server 来执行程序中的 SQL 请求。

与“利用 OceanbaseDataSourceProxy 访问 OceanBase”相比较，“利用 ObGroupDataSource 访问 OceanBase”只是不能使用 ObGroupDatasource 的自动升级、回滚等功能。您可以通过手动修改 pom.xml 中 ObGroupDatasource 的版本，来达到升级、回滚的目的，如下所示：

```
<dependency>
  <groupId>com.alipay.oceanbase</groupId>
  <artifactId>oceanbase-core</artifactId>
  <version>0.0.1</version>
</dependency>
```

2.4.2 示例

* 前提条件

- OceanBase 主备集群已经安装且正常运行。
- Java 开发环境以及部署完成，如已经安装 JDK、Eclipse 等。
- 已经获取 Java 客户端 Jar 包：“oceanbase-core-1.1.0.jar”和“druid-0.2.12.jar”。
- 已经获取依赖包：“commons-lang-2.3.jar”、“commons-logging-1.1.jar”和“mysql-connector-java-5.1.14.jar”。

* 案例操作流程

1. 连接 OceanBase 数据库。
2. 新建 PreparedStatement 语句，来查询__all_server 表中 svr_port 为某个值的 server 信息。
3. 指定查询__all_server 表中 svr_port 为 2600 的 server 信息。
4. 打印查询结果。

* 预期结果

Eclipse Console 界面将打印出所有端口号为 2600 的 server 信息。

* 举例

假设搭建的 OceanBase 数据库包含两个集群，两个集群中的 Listener 的地址分别为：10.232.36.29:15847,10.232.36.30:15847

说明：使用 obGroupDataSource 来访问 Oceanbase 数据库的时候，访问请求会跟进两个集群的流量分配准备来将请求发送给其中一个集群的 MergeServer 上面。

编辑 Java 代码，如下所示：

```
import java.io.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.*;
import javax.sql.DataSource;
```



```

public class ObGroupDataSourceExample {
    public static void main(String[] args) throws Exception {
        // 初始化 OceaBase DataSource 。
        OBGroupDataSource obGroupDataSource = new OBGroupDataSource();
        obGroupDataSource.setUsername("admin");
        obGroupDataSource.setPassword("admin");
        obGroupDataSource.setLMS("10.232.36.29:15847,10.232.36.30:15847");
        obGroupDataSource.init();

        Connection conn = obGroupDataSource.getConnection();
        PreparedStatement pstmt = conn
            .prepareStatement("select * from __all_server where svr_port = ?");
        pstmt.setInt(1, 2600);
        ResultSet rs = pstmt.executeQuery();
        ResultSetMetaData rSetMetaData = rs.getMetaData();
        for (int i = 1; i <= rSetMetaData.getColumnCount(); i++) {
            System.out.print("|" + rSetMetaData.getColumnName(i) + "");
        }
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= rSetMetaData.getColumnCount(); i++) {
                System.out.print("|" + rs.getString(i) + "");
            }
            System.out.println();
        }
        if (rs != null) {
            rs.close();
        }
        if (pstmt != null) {
            pstmt.close();
        }
        if (conn != null) {
            conn.close();
        }
    }
}

```

如果 Eclipse Console 界面输出如下信息则表明运行正常。其中第一段为“__all_server”表的 schema 信息，而第二段才是真正的输出结果行。

```

|gm_create|gm_modify|cluster_id|svr_type|svr_ip|svr_port|inner_port|svr_role|svr_version
|2013-05-27 21:54:30.08001|2013-05-29
17:54:45.81237|0|chunkserver|10.209.144.31|2600|0|0|0.4.1.2_13189M(Apr 10 2013
12:40:32)

```

2.5 利用 Merge Server 访问 OceanBase

该方式没有通过 Java 客户端连接 OceanBase，因此，无需导入 Java 客户端的 jar 包。适用于非阿里内部人员和阿里内部人员。

利用 MergeServer 访问 OceanBase 的过程最为直接，你只要知道一台 OceanBase 集群中 MergeServer 的地址和端口，就能直接进行访问，你所发出的所有 SQL 请求也都将由这台 MergeServer 进行处理。

* 前提条件

- OceanBase 主备集群已经安装且正常运行。
- Java 开发环境以及部署完成，如已经安装 JDK、Eclipse 等。
- 已经获取依赖包：“commons-lang-2.3.jar”、“commons-logging-1.1.jar”和“mysql-connector-java-5.1.14.jar”。

* 案例操作流程

1. 连接 OceanBase 数据库。
2. 新建 PreparedStatement 语句，来查询__all_server 表中 svr_port 为某个值的 server 信息。
3. 指定查询__all_server 表中 svr_port 为 2600 的 server 信息。
4. 打印查询结果。

* 预期结果

Eclipse Console 界面将打印出所有端口号为 2600 的 server 信息。

* 举例

编辑 Java 代码，如下所示：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;

public class OceanBaseMergeServerExample {
    public static void main(String[] argv) {
        executeByPreparedStatement();
    }

    public static void executeByPreparedStatement() {
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(
                "jdbc:mysql://10.232.36.29:15847", "admin", "admin");
```

```

        pstmt = conn.prepareStatement("select * from __all_server where svr_port = ?");
        pstmt.setInt(1, 2600);
        rs = pstmt.executeQuery();
        ResultSetMetaData rSetMetaData = rs.getMetaData();
        for (int i = 1; i <= rSetMetaData.getColumnCount(); i++) {
            System.out.print("|" + rSetMetaData.getColumnName(i) + "");
        }
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= rSetMetaData.getColumnCount(); i++) {
                System.out.print("|" + rs.getString(i) + "");
            }
            System.out.println();
        }
    } catch (Exception e) {
    } finally {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException ex) {
            }
        }
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (SQLException ex) {
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException ex) {
            }
        }
    }
}

```

如果 Eclipse Console 界面输出如下信息则表明运行正常。其中第一段为“__all_server”表的 schema 信息，而第二段才是真正的输出结果行。

```

|gm_create|gm_modify|cluster_id|svr_type|svr_ip|svr_port|inner_port|svr_role|svr_version
|2013-05-27 21:54:30.08001|2013-05-29
17:54:45.81237|0|chunkserver|10.209.144.31|2600|0|0|0.4.1.2_13189M(Apr 10 2013
12:40:32)

```

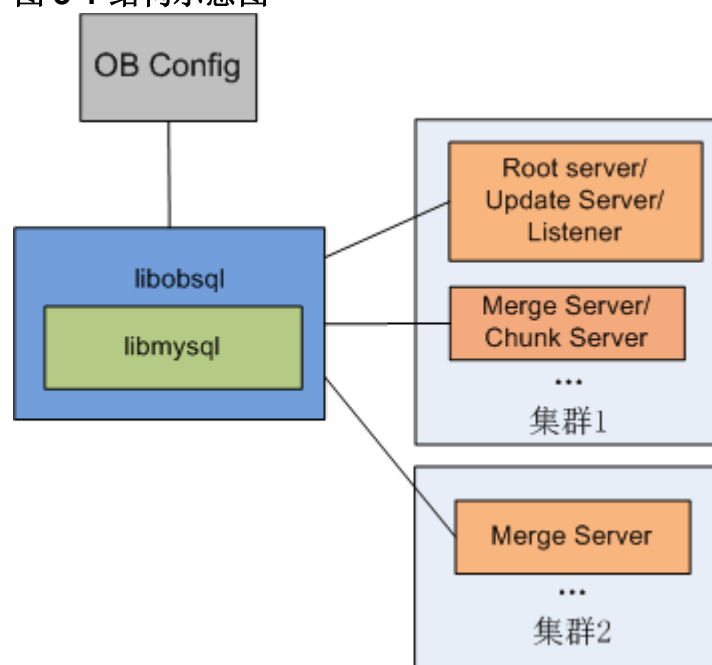
3 C 客户端

C 客户端主要用于开发人员编写 C 程序连接 OceanBase 数据库。它通过动态库“libobsql.so”的形式提供给用户使用。

3.1 客户端结构

C 客户端的结构示意图如[图 3-1](#)所示。

图 3-1 结构示意图



C 客户端的结构介绍如下：

- **OB Config**
OceanBase Client 的配置管理中心，实现客户端获取 OceanBase 集群地址的功能，管理多个版本客户端库和元数据，并通过配置项管理客户端的升级。用户只需要获取到 OB Config 的地址以及 OceanBase 数据库的用户名和密码就能获取到数据库的访问地址以及流量分配等信息，从而初始化整个 C 客户端。
- **Libmysqlclient**
MySQL 提供的 C 语言客户端库，用来访问 MySQL 数据库。
- **Libobsql**
OceanBase 提供的 C 语言客户端库。libobsql 封装了 libmysqlclient，它实现了所有 libmysqlclient 的接口。用户可以通过使用“LD_PRELOAD”

环境变量，调用“libmysqlclient”动态库中的接口，但是实际上是调用的 libobsql 的接口。

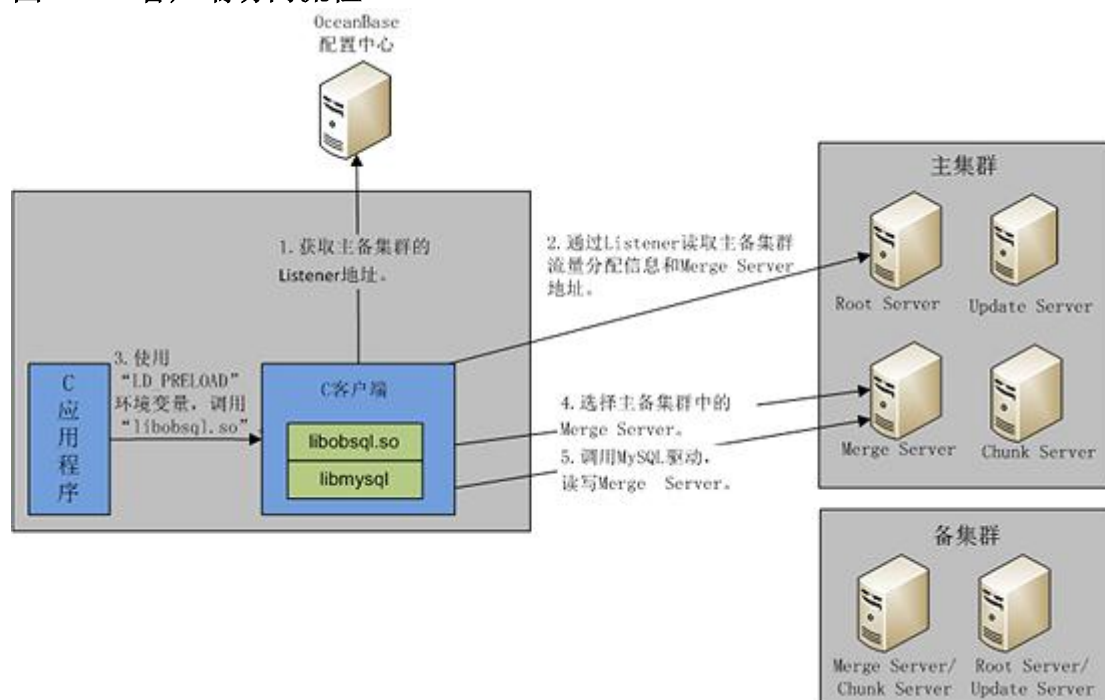
- Root Server/Update Server/Merge Server/Chunk Server
OceanBase 集群的组成模块，具体功能介绍请参见[《OceanBase 架构》](#)。
- Listener
OceanBase 集群内的特殊角色，是客户端获取访问 OceanBase 中 MergeServer 列表、集群流量分配的接口。

3.2 访问流程

应用程序可以使用 libmysqlclient 提供的 API 接口访问 OceanBase，通过 libmysqlclient 编译的应用程序无需重新编译即可连接 OceanBase。

C 客户端的访问流程如[图 3-2](#)所示。

图 3-2 C 客户端访问流程



说明：OB Config 属于 OceanBase 团队内部使用的客户端模块，暂不开源。非阿里巴巴内部人员可以通过配置文档传入主备集群的 Listener。如果您需要详细了解 OB Config 请参见[“2.2.3 OB Config”](#)。

3.3 安装 C 客户端

假设本地计算机的用户为 **sqluser**。安装 C 客户端的操作步骤如下：

1. 以 **sqluser** 用户登录本地计算机。
2. 执行以下命令，检出 Oceanbase 源码。
git clone https://github.com/alibaba/oceanbase oceanbase_install

3. 执行以下命令，进入“oceanbase_0.4”分支。
git checkout oceanbase_0.4
4. 执行以下命令，进入 C 客户端安装包的存放目录。
cd ~/oceanbase_install/client_package/c
5. 执行以下命令，安装依赖库 curl。
sudo rpm -Uvh curl-7.29.0-1.el6.x86_64.rpm
6. 执行以下命令，安装 C 客户端。
sudo rpm -Uvh oceanbase-devel-0.4.2.1-1193.el6.x86_64.rpm
7. 使用 vi 编辑器在“~/.bashrc”文件中添加以下内容。

```
export LD_PRELOAD=/home/admin/oceanbase/lib/libobsql.so.0.0.0
export OB_SQL_CONFIG_DIR=/home/admin/oceanbase/etc/
```
8. 执行以下命令，使环境变量生效。
source ~/.bashrc
9. 使用 vi 编辑器修改“~/oceanbase/etc/libobsql.conf”文件，参数说明如表 3-1 所示。

```
logfile=/tmp/obsql.log
#initurl=http://10.232.102.182:8080/diamond-server/config.co?dataId=fangji197
loglevel=DEBUG
minconn=2
maxconn=50
ip=10.10.10.2
port=2828
username=admin
passwd=admin
```

表 3-1 参数说明

参数名称	说明
logfile	客户端日志路径。
initurl	OB Config 的 URL 地址，向 DBA 申请获取。需要搭建 OB Config，暂时仅支持阿里巴巴内部人员使用。 使用 OB Config 获取主备集群的 Listener 时，需要配置。
minconn	客户端到 Merge Server 的最小连接数。
maxconn	客户端到 Merge Server 的最大连接数。
ip	主集群中主 Root Server 的 IP。 使用 OB Config 获取主备集群的 Listener 时，不需要配置。

参数名称	说明
port	Listener 端口号。 使用 OB Config 获取主备集群的 Listener 时，不需要配置。
username	连接 OceanBase 的用户名。 缺省值：admin
passwd	连接 OceanBase 的密码。 缺省值：admin

3.4 示例

编写一段 C 程序，使用 C 客户端进行如下操作：

1. 连接 OceanBase 数据库。
2. 创建表 “test”：c1 列为主键，int 类型；c2 列为 varchar 类型。
3. 插入两行数据 “1”、“hello ob” 和 “2”、“hello java”。
4. 查询表 “test” 内容。
5. 删除表 “test”。

前提条件

- OceanBase 主备集群已经安装且正常运行。
- 配置 “__all_cluster” 中主备集群的流量分配 “cluster_flow_percent”，主集群为 “0”，备集群为 “100”。
- C 开发环境已经部署完成，如已安装 g++，gcc 等。
- 客户端服务器用户：**sqluser**。

预期结果

根据客户端特性，create table、insert 和 drop table 在主集群中执行，select 操作在备集群中执行。

操作步骤

1. 编写一段 SQL 的 C 代码，保存为 “test.c” 文件，并上传至客户端服务器的 “/home/sqluser” 目录。

说明： API 的完整说明请参考 [《MySQL C API 文档》](#)。

```

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <mysql/mysql.h> //注意，此处为“mysql.h”
int main(int argc, char *argv[])
{
    (void)argc;
    (void)argv;
    const char* HOST = "127.0.0.1";
    int PORT = 2828;

    // 1. 初始化 libmysqlclient
    if (0 != mysql_library_init(0, NULL, NULL))
    {
        fprintf(stderr, "could not init mysql library\n");
        exit(1);
    }
    MYSQL my_;
    mysql_init(&my_);

    // 2. 连接 OceanBase SQL 服务
    fprintf(stderr, "Connecting server %s:%d...\n", HOST, PORT);
    mysql_real_connect(&my_, HOST, "admin", "admin", "", PORT, NULL, 0);

    // 3. 建表
    int ret = 0;
    ret = mysql_query(&my_, "drop table if exists test");
    if (0 != ret)
    {
        fprintf(stderr, "%sn", mysql_error(&my_));
    }
    ret = mysql_query(&my_, "create table if not exists test (c1 int primary key, c2
        varchar)");
    if (0 != ret)
    {
        fprintf(stderr, "%sn", mysql_error(&my_));
    }

    //建表后预留用于主备同步的时间。
    sleep(60);

    //4. 插入数据
    ret = mysql_query(&my_, "insert into test (c1, c2) values (1, 'hello ob'), (2, 'hello
        java')");
    if (0 != ret)
    {
        fprintf(stderr, "%sn", mysql_error(&my_));
    }

    // 5. 查询数据
    ret = mysql_query(&my_, "select * from test");
    if (0 != ret)
    {

```



```

    fprintf(stderr, "%sn", mysql_error(&my_));
}
MYSQL_RES *result = mysql_store_result(&my_);

// 6. 删表
ret = mysql_query(&my_, "drop table if exists test");
if (0 != ret)
{
    fprintf(stderr, "%sn", mysql_error(&my_));
}

// 7. destroy
mysql_close(&my_);
mysql_library_end();
return 0;
}

```

2. 以 **sqluser** 用户登录客户端服务器。
3. 执行以下命令，编译 “test.c” 。
g++ atest.c -o atest -I /usr/include -L /usr/lib64/mysql -lmysqlclient
4. 执行以下命令，运行 “test.c” 。
./atest
5. 分别登录主备集群中安装 Merge Server 的服务器，查看 Merge Server 的日志文件 “/home/admin/oceanbase/log/mergeserver.log” 。

如果出现如下日志，则说明客户端运行正常。

- 主集群

```

[2013-06-08 10:51:16.467612] INFO ob_mysql_server.cpp:1054 [139693026940672]
start query: "create table test (c1 int primary key, c2 varchar)" real_query_len=50,
peer=10.14.19.17:53472
[2013-06-08 10:52:16.764349] INFO ob_mysql_server.cpp:1054 [139693440083712]
start query: "insert into test values (1, 'hello ob'), (2, 'hello java')" real_query_len=58,
peer=10.14.19.17:53472
[2013-06-08 10:52:16.810628] INFO ob_mysql_server.cpp:1054 [139693062612736]
start query: "drop table test" real_query_len=15, peer=10.14.19.17:53472

```

- 备集群

```

[2013-06-08 10:52:16.797199] INFO ob_mysql_server.cpp:1056 [140177548736256]
start query: "select * from test" real_query_len=18, peer=10.14.19.17:53527

```

4 附录

4.1 Java 客户端流量分配执行实现细节

Java 客户端流量分配执行实现细节如下：

1. 初始化长度为 100 的数组，按照主备集群的比例填充数组。假设 40 个元素为主集群；60 个元素为备集群。填充完，随机将顺序打乱。
2. 全局 SQL 请求计数器，在 SQL 请求到来时，自增一，对数组长度求余后再求绝对值，最终获得数组的下标，并定位值数组元素，决定此次访问的集群。
 - 如果是主集群，查看主集群非弱一致性请求计数器是否小于或等于 0
 - 小于或等于 0：则发送至主集群。
 - 大于 0：则重新选择。
 - 如果是备集群，直接发送。

4.2 Java 客户端定时 Merge Server 重新生效流程

Java 客户端会启动一个定时线程，该线程的主要任务为：扫描内部表“__all_cluster”和“__all_server”，并查看这两个表中的数据是否发生了变化。如果变化，则将生效新的流量规则并上下线 Merge Server。

实现细节如下：

1. 任务调度周期 60-120s（考虑是，将调度时间分散，避免在某个集中时刻，对配置中心的访问，导致配置中心发生异常）。

每次调度时，将首先访问 OceanBase 配置中心，获取访问入口。如果配置中心发生故障，则无法获取访问入口，调度任务将失败。但是，不影响客户端的处理外部请求的功能。
2. 获得到访问入口后，将读取内部表“__all_cluster”和“__all_server”的内容。
3. 与当前使用的信息进行对比，如果发现改变，则开始生效。
 - 流量发生变化，则对决定流量的数组进行重新分配。
 - Merge Server 列表发生变化，则对底层的 Druid 数据源进行创建或销毁。

4.3 Java 客户端日志梳理

Java 客户端日志主要包以下内容：

- 将每个 Merge Server 的 druid 数据源连接池的状态进行输出，其中包括：当前正在使用的连接个数、历史创建的连接总个数、历史销毁的连接总个数、历史最大连接使用个数、最大可创建的个数等信息。
- 输出 Java 客户端与 Merge Server 连接异常信息以及此次异常所完成的动作（重试 Merge Server，直接抛出等）。
- 完全独立 ob-datasource 的日志信息。后续将考虑从日志信息中获取并将信息展示至监控中心。

4.4 C 客户端流量分配和重试设计

4.4.1 流量分配

流量分配只涉及弱一致性读请求，非弱一致性读请求全部发送至主集群，所以 C 客户端的流量分配方案采用了一个记账的模型：

每次更新集群信息时都会根据最新的流量比列，构造出一个定长数组，并且用变量“OFFSET”表述数据的当前偏移，变量“MASTER”记录当前一致性请求的次数。

1. 在每次发起读请求时，如果是一致性请求，则直接发往主集群（不查询上述构造的数组），且“MASTER++”，“OFFSET”不变；如果是非一致性请求，则继续步骤 2。
2. 查询上述构造的数组，如果当前偏移是主集群，则继续步骤 3；如果是备集群则直接发送，且“OFFSET++”。
3. 查看 MASTER 值，如果“MASTER==0”，则直接发送请求到该主集群，且“OFFSET++”，“MASTER”不变；否则“MASTER—”，“OFFSET++”，并重复步骤 2。

由于强一致性请求固定发往主集群，弱一致性通过查询数组按流量比发送，因此会造成主集群流量偏大。通过 MASTER 这个变量就有效抵消了强一致性请求到主集群的流量，从而达到服务端设置的流量比列。

Ex: 假设有集群 A 和 B，A 是主集群，B 是备集群，流量比列是“4:6”，请求序列“cq1 Nq1 Cq2 Nq2 Nq3 Nq4”。其中“Cq”表示一致性请求，“Nq”表示非一致性请求。

构造的一个定长数组如下：

A	B	B	A	B	A	B	B	A	B
---	---	---	---	---	---	---	---	---	---

MASTER = 0, OFFSET=0

则流量分配如下：

请求	发往的集群	MASTER	OFFSET
Cq1	A	1	0
Nq1	B	0	2
Cq2	A	1	2
Nq2	B	1	3
Nq3	B	0	5
Nq4	A	0	6

4.1.2 重试

OceanBase 数据库为分布式系统，单点失效时需要保持对用户透明，所以在客户端需要在访问失败的时候增加重试策略。

对于普通请求，当客户端访问某 **Merge Server** 失败时，就给它打上标签并持续一段时间，然后随机重试该集群中的另一台 **Merge Server** 处理。重试过程中不修改内部的 **OBGroupDataSource**。

对于绑定变量的请求的，访问失败后并不重试，直接把错误交由应用处理，因为在客户端内部无法得知连接上绑定过哪些 **statement**。