MySQL应用层的高可用方案

MySQL Router 和 MySQL Connector支持高可用的密诀

杜修文

议程

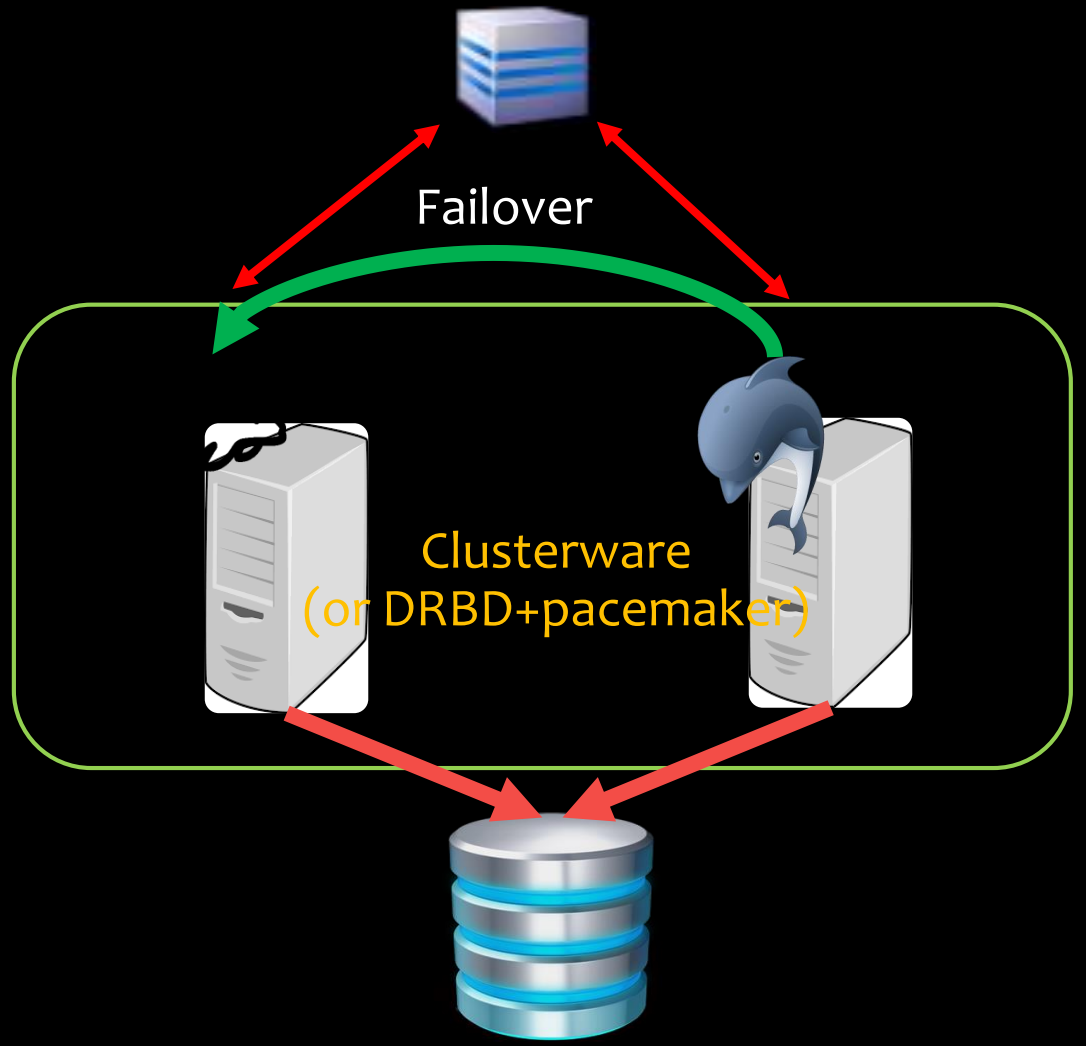
- MySQL高可用结构
- MySQL应用开发者的理想方案
- MySQL驱动器对高可用的支持
- 什么都别管 - MySQL Router释放您的限制
- 结论和发展方向

MySQL高可用结构

- 共享存储的主备MySQL结构
- MySQL主从复制,和半同步复制
- MySQL Group Replication
- MySQL Cluster (NDB Cluster)

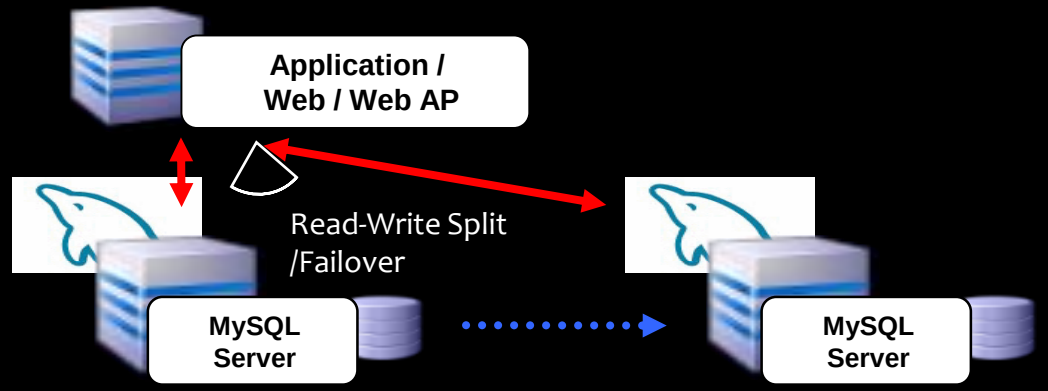
MySQL高可用结构

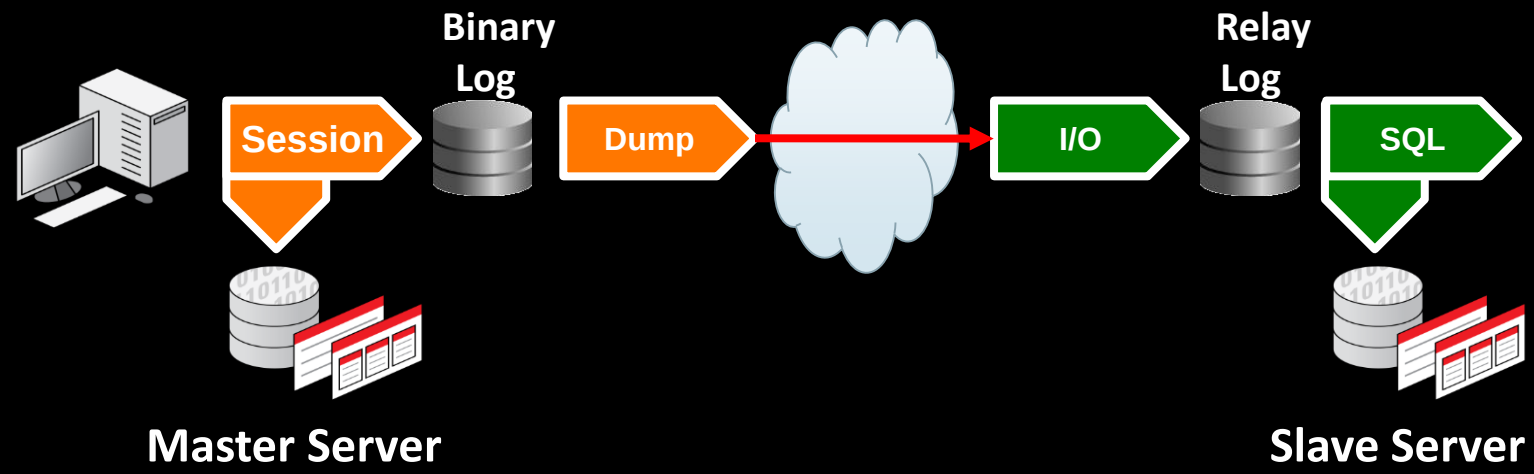
- **共享存储的主备MySQL结构**
- MySQL主从复制,和半同步复制
- MySQL Group Replication
- MySQL Cluster (NDB Cluster)



MySQL高可用结构

- 共享存储的主备MySQL结构
- **MySQL主从复制,和半同步复制**
- MySQL Group Replication
- MySQL Cluster (NDB Cluster)

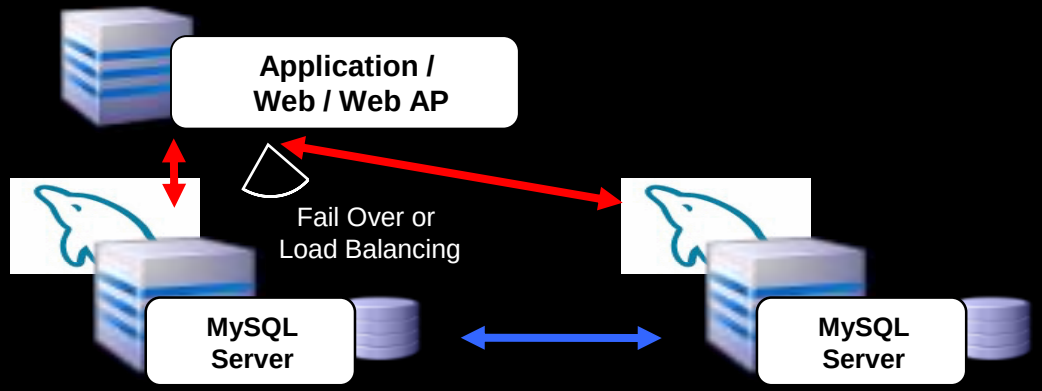


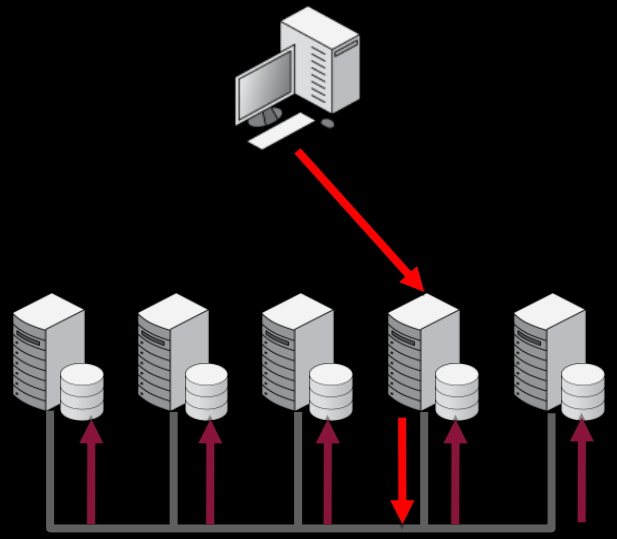


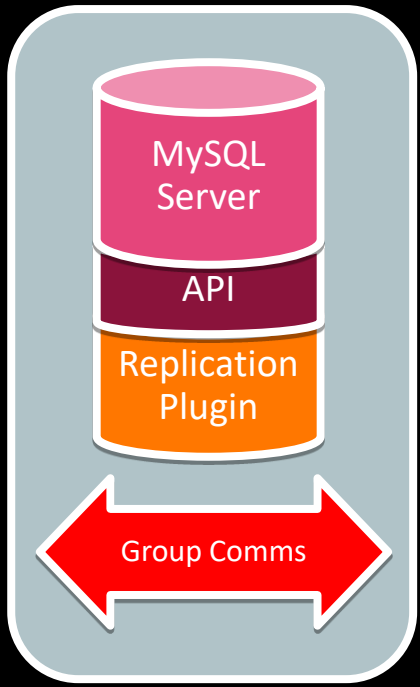
- ReplicaSet 的出现让配置更简单

MySQL高可用结构

- 共享存储的主备MySQL结构
- MySQL主从复制,和半同步复制
- **MySQL Group Replication**
- MySQL Cluster (NDB Cluster)

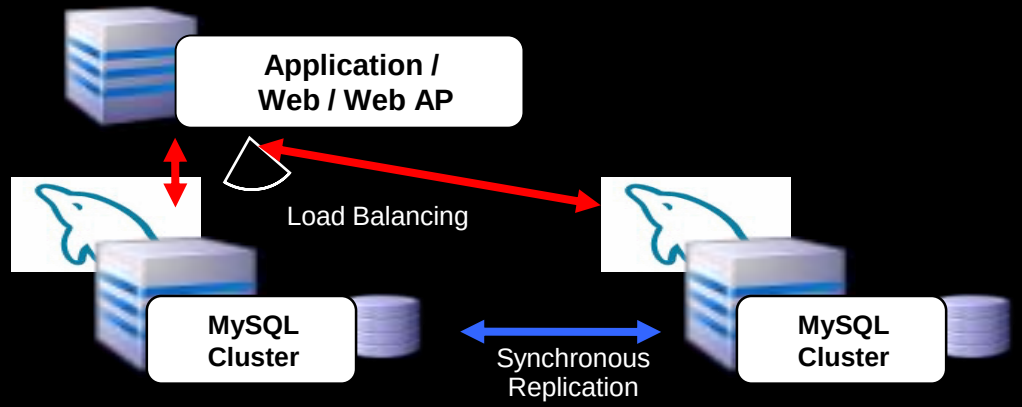


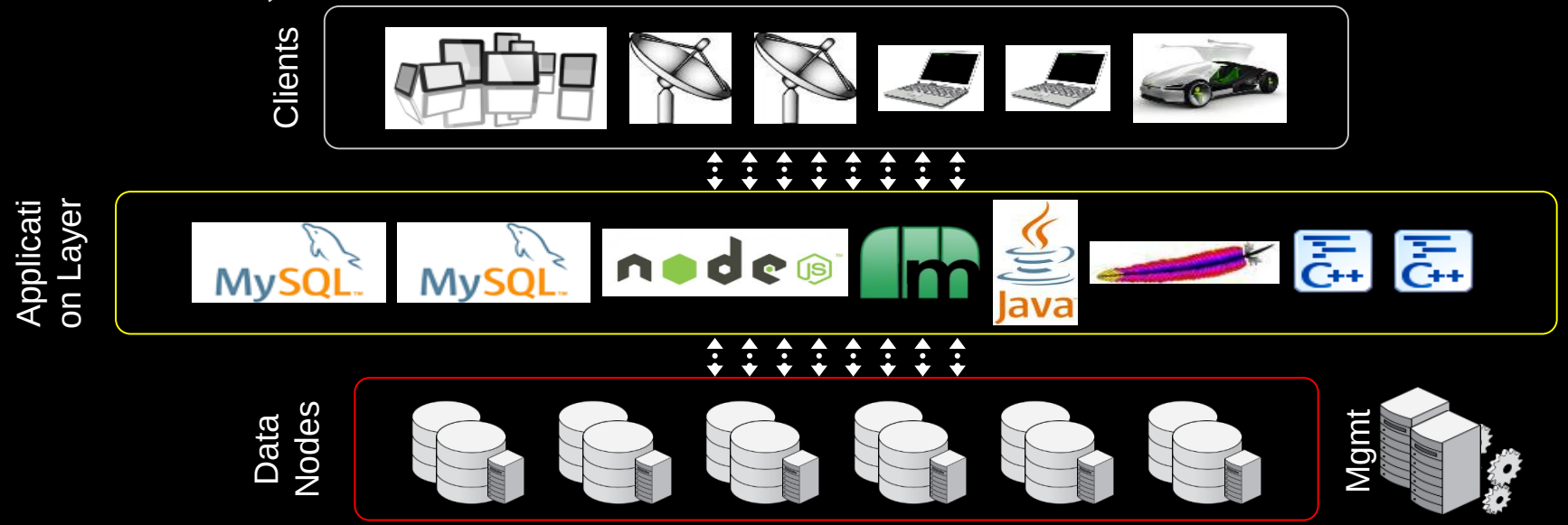




MySQL高可用结构

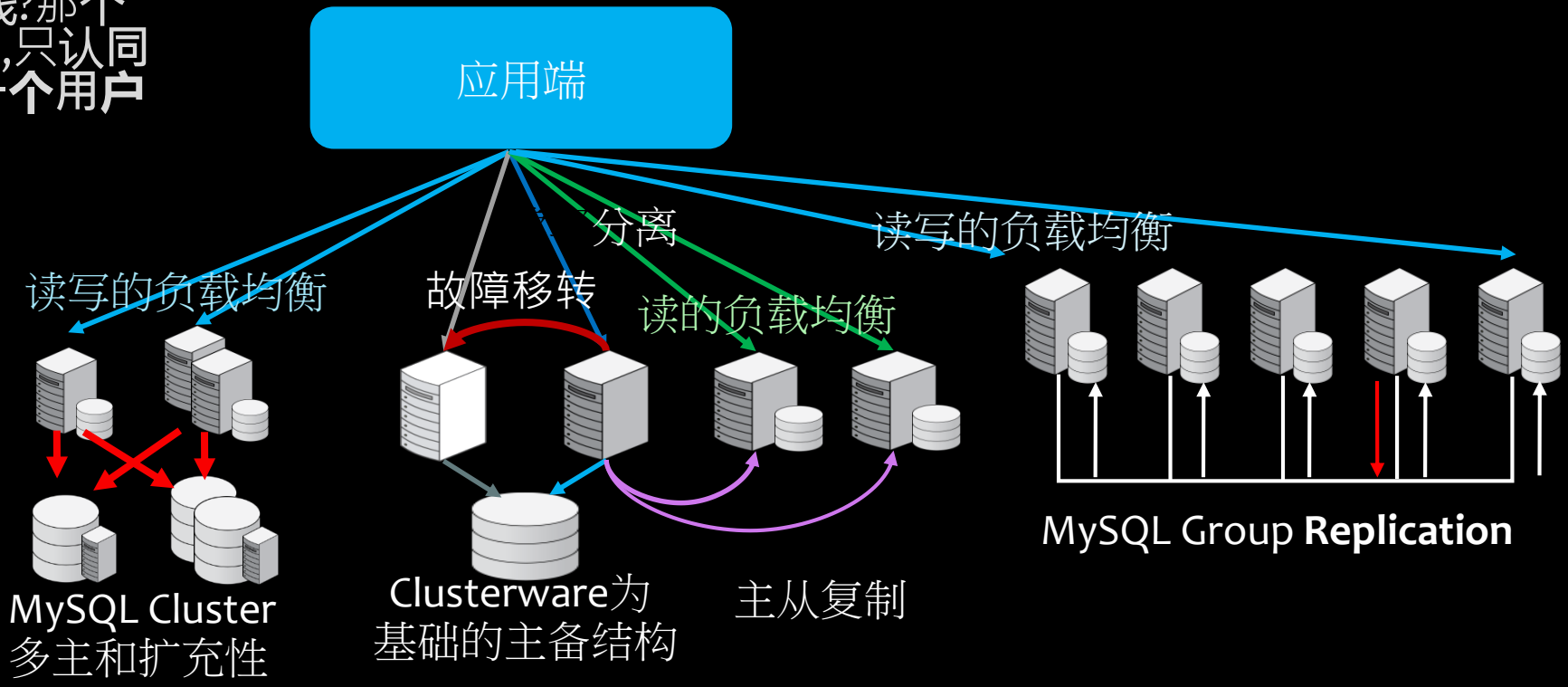
- 共享存储的主备MySQL结构
- MySQL主从复制,和半同步复制
- MySQL Group Replication
- **MySQL Cluster (NDB Cluster)**





应用程序怎么看数据库的高可用?

- 最理想是应用层对数据库高可用结构完全透明
- 不管现在数据库有多少服务器?那个是主?那个是从?那个在线?那个下线?应用代码完全不用管,只认同**一个主机,同一个端口,同一个用户帐号**
- 如何对应主备数据库?
- 如何对应异步数制?
- 如何对应组复制?



应用端的方案

- MySQL Connectors
- MySQL Router
- DNS
- VIP

应用端的方案

- **MySQL Connectors**
- MySQL Router
- DNS
- VIP

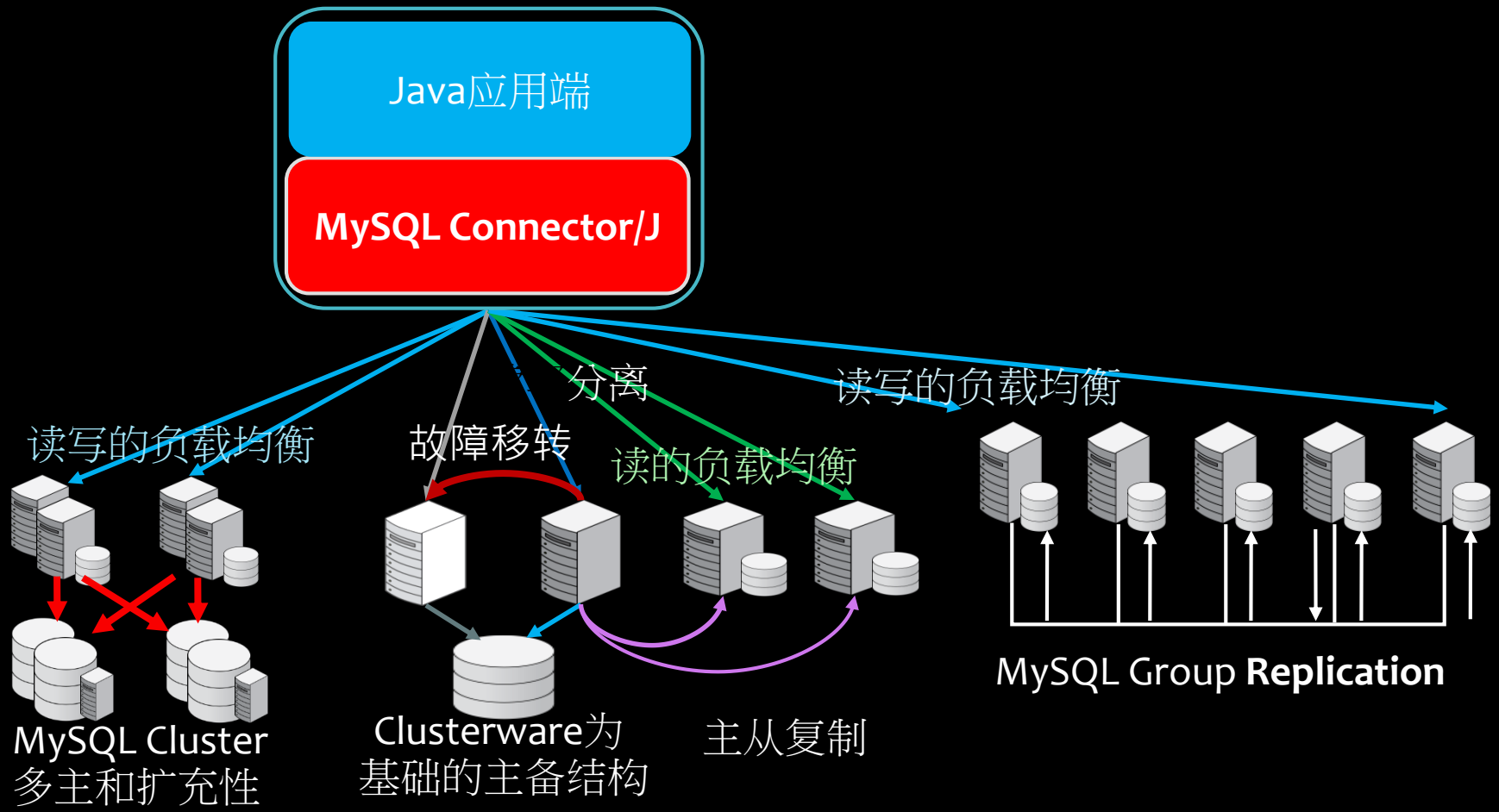
MySQL Connector的高可用

Connector	Failover	Load Balancing	透明的Error Handling
Connector/J	√	√	在Exception Session中处理就不会影响用户
Connector/NET	√	√	在Exception Session中处理就不会影响用户
Connector/Python	√		在Exception Session中处理就不会影响用户
Connector/NodeJS	√ (用poolCluster)	√ (用poolCluster)	Try {…} catch (<mysql error>) {…}
Connector/PHP	使用 mysqlnd的 PHP mysql extension (在PHP5.3以后的版本)支持MySQL复制		

MySQL Connector/J 支持高可用

支持**负载均衡,高可用读写分离**

- Connector/J支持负载均衡,故障移转,和读写分离
- MySQL Cluster和Group Replication多主模式,可用Connector/J的负载均衡模式
- 主从复制和Group Replication单主模式,可用Connecdtor/J的复制/读写分离模式
- 读写分离时,只读时需要设connection object 为read-only



Connector/J 的JDBC上的故障移转

- 用ReplicationDriver
- JDBC URL format，第一个主机为master，第二个以后为 backup:

```
import com.mysql.jdbc.ReplicationDriver;
jdbc:mysql://[primary-host][:port],[secondary-host1][:port][,[secondary-
  host2][:port]]...[/[database]][?propertyName1=propertyValue1[&propertyName2=
  propertyValue2]...]
```

- Connection的faileover属性
  - failOverReadOnly :为true时，faileover后设Connection.setReadOnly(false); 第二个host仍为read-only mode
  - secondsBeforeRetryMaster
  - queriesBeforeRetryMaster
  - retriesAllDown
  - autoReconnect
  - autoReconnectForPools

Connector/J 的JDBC上的Load Balancing

```
String URL = "jdbc:mysql:loadbalance://" +"host1:3306,host2:3306/test?" +
   "loadBalanceConnectionGroup=first&loadBalanceEnableJMX=true";
Class.forName("com.mysql.jdbc.Driver");
Connection conn DriverManager.getConnection(URL, "root", “secret");

conn.setAutoCommit(false);

Statement stmt = conn.createStatement();

stmt.executeQuery("SELECT SLEEP(1) /* Connection: " + conn + ", transaction: " +
   trans + " */");

conn.commit();
```

Connector/J 的JDBC上的Read Write Split

```
conn =
   DriverManager.getConnection("jdbc:mysql:replication://master,slave1,slave2,slav
   e3/test?" + "user=<username>&password=<pwd>");

conn.setReadOnly(true);

ps = conn.prepareStatement(

 "select emp_no, first_name,  last_name, d.name, i.variable_value   from
    employees.employees e, information_schema.global_variables i,
    employees.departments d where e.department_id = d.department_id and
    i.variable_name='port' and emp_no = ?");
```

支持Multiple-Master Replication Topographies

- 一个以上的master时，多个address，指定type为master或slave

```
jdbc:mysql://address=(type=master)(host=<master1host>),
  address=(type=master)(host=<master2host>),
  address=(type=slave)(host=<slave1host>)/database
```

- 设allowMasterDownConnections=true使Connection objects能在没有 master 可以连上时也能创建，此时这个Connection Object为read only

应用端的方案

- MySQL Connectors
- MySQL Router
- DNS
- VIP

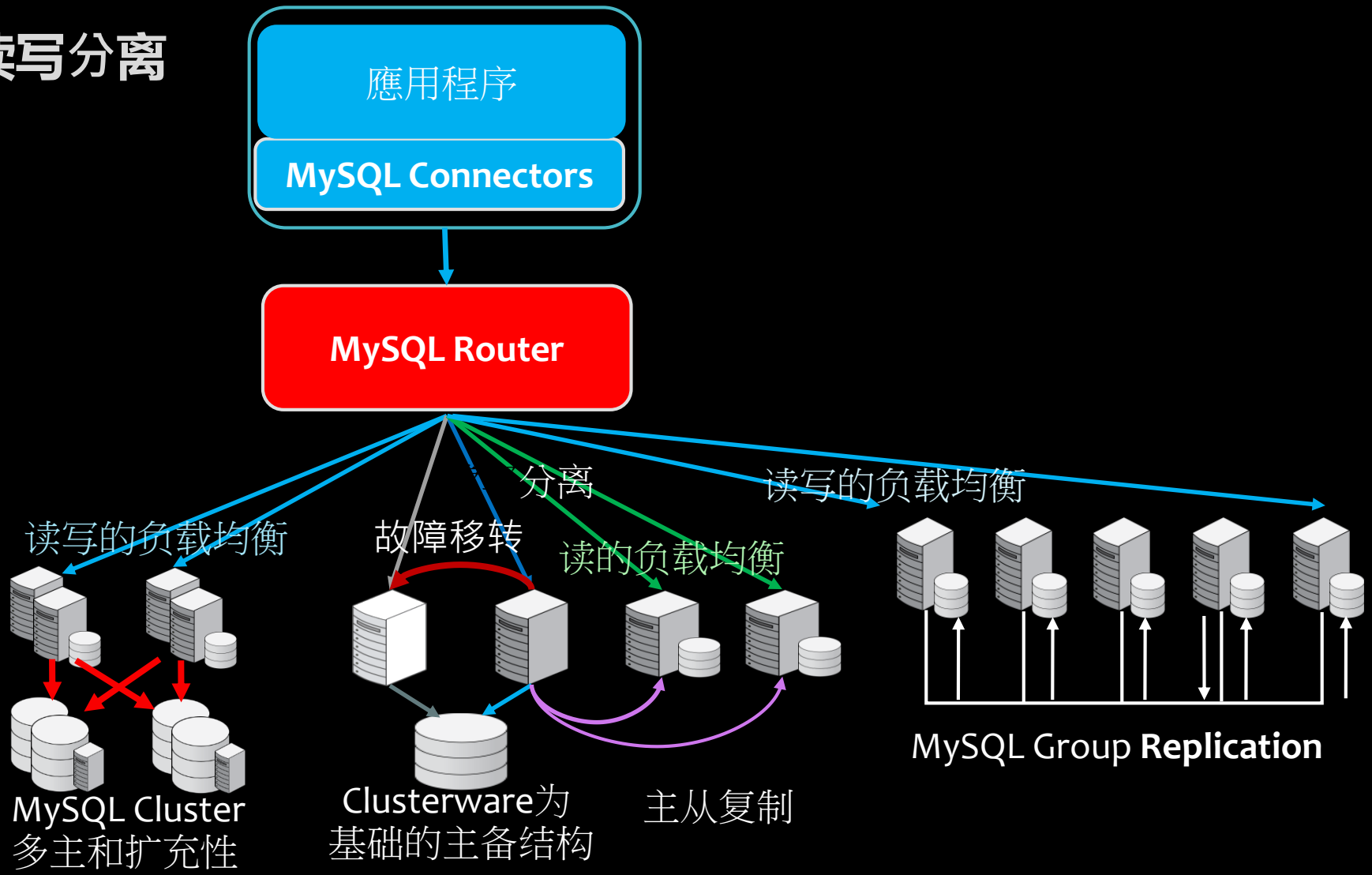
应用端的方案

- MySQL Connectors
- **MySQL Router**
- DNS
- VIP

MySQL Router 支持高可用

支持负载均衡,高可用读写分离

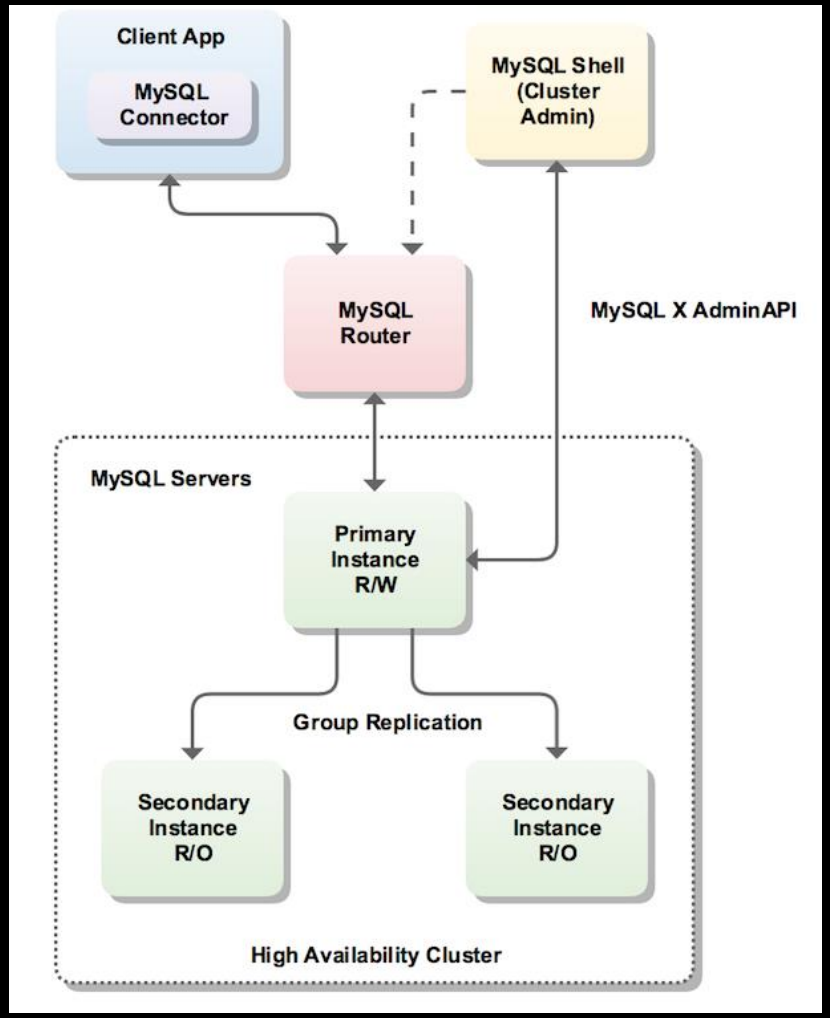
- 编辑mysqlrouter.conf文件,手动指定高可用结构
- 配合MySQL InnoDB Cluster或 MySQL ReplicaSet,由数据库的mysql_innodb_cluster_metadata 中了解数据库的高可用结构,而能自动选择主和故障移转



MySQL Router with InnoDB Cluster

- 完全在MySQL shell上操作
- 以MySQL Shell 创建Group Replication
  - > cluster=dba.createCluster('myCluster')
  - > cluster.addInstance('<user>@<host1>:<port1>')
  - > cluster.addInstance('<user>@<host2>:<port2>')
- Bootstrap router, 打开router
  - $ mysqlrouter --bootstrap=<user>@<hosto>:<porto>' --directory=~/mysqlrouterdata

  ### 生成~/mysqlrouterdata/mysqlourter.conf
  - > mysqlrouter -c ~/mysqlrouterdata/mysqlrouter.conf



编辑 mysqlrouter.config
- RPM安装默认在/etc/mysqlrouter/mysqlrouter.conf)

- 只能依固定顺序故障移转,在InnoDB Cluster之下不会自动找新的主
- Config [section_name:option_key_name],这些section包括:

[default]
[routing] # 手动定如何送命令到数据库
- 以bind_address(默认为0.0.0.0)和bind_port指定用那个IP地址和端口
- 以destinations, 指定后台数据库的地址和端口
- 以mode指定是读写还是只读
- 以routine_strategy指定如何找后台的服务器-first-available, round-robin/round-robin-with-fallback

[metada_cache]
- # 自MySQL metadata中找分配消息,
- 以bootstrap_server_address指定找那个库,可为以",”区隔的多个库,
- 以cluster_type指定是group replication (gr),还是replicaSet(rs)
- 以auth_cache_refresh_interval和auth_cache_tll决定多久重新以指定之user装入一次

[http_server]
[http_auth_backend]
[logger]

Bootstrap生成的MySQL Router 配置文件

```
[DEFAULT]
logging_folder=/home/mysql/mysqlrouter_data/log
runtime_folder=/home/mysql/mysqlrouter_data/run
data_folder=/home/mysql/mysqlrouter_data/data
keyring_path=/home/mysql/mysqlrouter_data/data/keyring
master_key_path=/home/mysql/mysqlrouter_data/mysqlrouter.key
connect_timeout=15
read_timeout=30
dynamic_state=/home/mysql/mysqlrouter_data/data/state.json
[logger]
level = INFO
[metadata_cache:myCluster]
cluster_type=gr
router_id=3
user=mysql_router3_nqhu6ocqg948
metadata_cluster=myCluster
ttl=0.5
use_gr_notifications=0
[routing:myCluster_rw]
bind_address=0.0.0.0
bind_port=6446
destinations=metadata-cache://myCluster/?role=PRIMARY
routing_strategy=first-available
protocol=classic
[routing:myCluster_ro]
bind_address=0.0.0.0
bind_port=6447
destinations=metadata-cache://myCluster/?role=SECONDARY
routing_strategy=round-robin-with-fallback
protocol=classic
[http_server]
[rest_api]
[rest_router]
require_realm=somerealm
[http_auth_realm:somerealm]
backend=somebackend
method=basic
name=Some Realm
[http_auth_backend:somebackend]
backend=file
filename=/home/mysql/mysqlrouter_data/mysqlrouter.pwd
require_realm=somerealm
```

支持REST API

- 以 [http_server]指定开缺省8081端口,[rest_api]开swagger.json
- 查swagger.json确认router版本等消息

http://127.0.0.1:8081/api/20190715/swagger.json

- 查版本,数据库版本,,主机名称等消息
- http://127.0.0.1:8081/api/20190715/router/status

Demo

在VM OL7Server64之下:
1.dba.tart.sandboxInstance(3310); dba.tart.sandboxInstance(3320); dba.tart.sandboxInstance(3330); \c root@192.168.56.101:3310; cluster.rebootClusterFromCompleteOutage(); cluster=dba.getCluster('myCluster');cluster.status()
2. ~/mysqlrouter_data/start.sh
3. ~/testrw.sh for read-write, stop 3310 and see how the port change
4. ~/testro.sh for read-only, check the port while 3310 is stopped, and see hoe port changed when bring back 3310
5. check REST API by port http://127.0.0.1:8081/api/20190715/router/status and http://192.168.56.101:8081/api/20190715/swagger.json on firefox
6. On Windows – multiple-primary, Java create connection with multi-host

MySQL Router 的注意事项

- 要给服务器一个名字 – name resolution with DNS or hosts table
- 无状况
- 不是完全透明的故障移转
- 单独在应用服务器之外运行要注意是否有,和瓶颈
- 现在可达65536

结论和发展

- 让应用代码对数据库高可用架构更透明
- 对数据分片的支持
- 对云的支持
- K8S operator支持

欢迎您提出您的看法

谢谢，请您指教

如果您需要完善数据库安全，更好的备份工具，和更大的并发量 – 请考虑MySQL企业版