# SymmetricDS完全配置安装手册

概述

SymmetricDS 是一个基于 [Java](https://www.2cto.com/kf/ware/Java/) 的[数据库](https://www.2cto.com/database/" \t "_blank)同步框架。

我的环境

[root@localhost ~]# cat /proc/version

Linux version 3.10.0-229.el7.x86\_64 (builder@kbuilder.dev.centos.org) (gcc version 4.8.2 20140120 (Red Hat 4.8.2-16) (GCC) ) #1 SMP Fri Mar 6 11:36:42 UTC 2015

[root@localhost ~]# cat /etc/redhat-release

CentOS Linux release 7.1.1503 (Core)

[root@localhost ~]# java -version

java version "9.0.4"

Java(TM) SE Runtime Environment (build 9.0.4+11)

Java HotSpot(TM) 64-Bit Server VM (build 9.0.4+11, mixed mode)

mysql> status;

--------------

mysql Ver 14.14 Distrib 5.7.19-17, for Linux (x86\_64) using 6.2

Connection id: 812

Current database:

Current user: symmetric@localhost

SSL: Not in use

Current pager: stdout

Using outfile: ''

Using delimiter: ;

Server version: 5.7.18-log MySQL Community Server (GPL)

Protocol version: 10

Connection: Localhost via UNIX socket

Server characterset: utf8

Db characterset: utf8

Client characterset: utf8

Conn. characterset: utf8

UNIX socket: /var/lib/mysql/mysql.sock

Uptime: 1 day 23 hours 3 min 21 sec

Threads: 2 Questions: 328546 Slow queries: 0 Opens: 2523 Flush tables: 1 Open tables: 649 Queries per second avg: 1.939

[root@localhost jdk-9.0.4]# ssh -V

OpenSSH\_6.6.1p1, OpenSSL 1.0.1e-fips 11 Feb 2013

[root@localhost bin]# ./symadmin -h

SLF4J: The following set of substitute loggers may have been accessed

SLF4J: during the initialization phase. Logging calls during this

SLF4J: phase were not honored. However, subsequent logging calls to these

SLF4J: loggers will work as normally expected.

SLF4J: See also http://www.slf4j.org/codes.html#substituteLogger

SLF4J: org.jumpmind.symmetric.AbstractSymmetricEngine

symadmin version 3.9.2

Perform administration tasks with SymmetricDS.相关配置

这里需要你的 JDK 版本为 jdk1.6

防火墙

此处为测试环境，故设置了关闭防火墙

# service firewalld stop

# chkconfig firewalld off # 设置防火墙不随系统启动，这一条你可以选择性添加

SSH 免密码登录

关于 SSH 的免密码登录在这里不是很重要，只是本人在下发文件的时候用到了。你可以选择性配置。

创建 MySQL 新用户

这里为了避免使用 root 帐户，所以创建了新的数据库操作用户：symmetric

首先使用 root 帐户登录 MySQL，并执行如下 SQL 语句：

CREATE USER symmetric IDENTIFIED BY 'Aa@123456';   
grant all on \*.\* to 'symmetric'@'%';

show grants for ' symmetric ';

配置 MySQL 驱动

在 Java 中，如果需要连接 MySQL 数据库，需要使用 JDBC 的数据库连接技术。在使用 JDBC 的时候比较容易忽略的就是连接驱动的配置了。

这一点在 Windows 下还好，不过要在 [Linux](https://www.2cto.com/os/linux/) 下使用 JDBC 就需要配置 MySQL 的驱动程序了。

1. 首先上传 MySQL 的驱动 jar 包 mysql-connector-java-5.1.6-bin.jar 到 Linux 的合适位置

2. 配置环境变量

# vim /etc/profile

在文件末尾处添加如下内容：

export CLASSPATH=${CLASSPATH}: /root/mysql-connector-java-5.1.45/mysql-connector-java-5.1.45-bin.jar

设置配置立即生效

# source /etc/profile

SymmetricDS

首先上传 symmetric-server到 192.168.70.211 节点自定义的一个目录下，再重命名移动到用户根目录。

# mv symmetric-server-3.9.2 ~/symmetric

另一台机器也运行

配置 corp

这里的 corp 也就是我的 192.168.70.211 节点。

复制 symmetric 目录下的 samples/corp-000.properties 文件到 engines 目录下：

# cp samples/corp-000.properties engines/

并修改此文件内容

# vim engines/corp-000.properties

修改后的内容如下：

engine.name=corp-000

db.driver=com.mysql.jdbc.Driver

db.url=jdbc:mysql://192.168.70.211/sample

db.user=symmetric

db.password=Aa@123456

registration.url=

sync.url=https://192.168.70.211:8080/sync/corp-000

group.id=corp

external.id=000

job.purge.period.time.ms=7200000

job.routing.period.time.ms=5000

job.push.period.time.ms=10000

job.pull.period.time.ms=10000

initial.load.create.first=true

这里我只保留了实际的内容，不包含注释。

配置 store

这里的 store 也就是我的 192.168.70.68 节点。

复制 symmetric 目录下的 samples/store-001.properties 文件到 engines 目录下：

# cp samples/store-001.properties engines/

并修改此文件内容

# vim engines/store-001.properties

修改后的内容如下：

engine.name=store-001

db.driver=com.mysql.jdbc.Driver

db.url=jdbc:mysql://192.168.70.68/sample

db.user=symmetric

db.password=Aa@123456

registration.url=https://192.168.70.211:8080/sync/corp-000

group.id=store

external.id=001ZZ

job.routing.period.time.ms=5000

job.push.period.time.ms=10000

job.pull.period.time.ms=10000

创建数据库

# mysql -usymmetric -pAa@123456

mysql> create database sample;

创建数据表

自动化创建

在 symmetric 中默认已经给出了自动化的数据创建过程，不过这种自动化的创建有一些不足的地方，那就是数据不够完整。这样就会致使在后面自动化插入数据时出现找不到数据表的情况，所以这里我比较推荐你使用下面手动创建的方式。自动化的创建方式只作为参考：

创建指令

# ./bin/symadmin --engine corp-000 create-sym-tables

创建结果

mysql> show tables;

+-----------------------------+

| Tables\_in\_sample |

+-----------------------------+

| sym\_channel |

| sym\_conflict |

| sym\_data |

| sym\_data\_event |

| sym\_data\_gap |

| sym\_extension |

| sym\_extract\_request |

| sym\_file\_incoming |

| sym\_file\_snapshot |

| sym\_file\_trigger |

| sym\_file\_trigger\_router |

| sym\_grouplet |

| sym\_grouplet\_link |

| sym\_incoming\_batch |

| sym\_incoming\_error |

| sym\_load\_filter |

| sym\_lock |

| sym\_node |

| sym\_node\_channel\_ctl |

| sym\_node\_communication |

| sym\_node\_group |

| sym\_node\_group\_channel\_wnd |

| sym\_node\_group\_link |

| sym\_node\_host |

| sym\_node\_host\_channel\_stats |

| sym\_node\_host\_job\_stats |

| sym\_node\_host\_stats |

| sym\_node\_identity |

| sym\_node\_security |

| sym\_outgoing\_batch |

| sym\_parameter |

| sym\_registration\_redirect |

| sym\_registration\_request |

| sym\_router |

| sym\_sequence |

| sym\_table\_reload\_request |

| sym\_transform\_column |

| sym\_transform\_table |

| sym\_trigger |

| sym\_trigger\_hist |

| sym\_trigger\_router |

| sym\_trigger\_router\_grouplet |

+-----------------------------+

42 rows in set (0.00 sec)

手动创建

手动创建虽然比较繁琐，但是数据比较完整，整个过程也很顺利。

SQL

item

CREATE TABLE item(

item\_id INTEGER PRIMARY KEY,

name VARCHAR(100)

);

CREATE TABLE item\_selling\_price(

item\_id INTEGER REFERENCES item(item\_id),

store\_id VARCHAR(5),

price DECIMAL(10,2),

cost DECIMAL(10,2),

PRIMARY KEY(item\_id, store\_id)

);

CREATE TABLE sale\_transaction(

tran\_id INTEGER PRIMARY KEY,

store\_id VARCHAR(5) NOT NULL,

workstation VARCHAR(3) NOT NULL,

day VARCHAR(10) NOT NULL,

seq INTEGER NOT NULL

);

CREATE TABLE sale\_return\_line\_item(

tran\_id INTEGER PRIMARY KEY REFERENCES sale\_transaction(tran\_id),

item\_id INTEGER NOT NULL REFERENCES item(item\_id),

price DECIMAL(10,2) NOT NULL,

quantity INTEGER NOT NULL,

returned\_quantity INTEGER

);

于是创建了如下 4 张数据表：

mysql> show tables;

+-----------------------+

| Tables\_in\_sample |

+-----------------------+

| item |

| item\_selling\_price |

| sale\_return\_line\_item |

| sale\_transaction |

+-----------------------+

修改 HOST

由于之前的配置文件中设置了 HOST 为 192.168.70.211，所以这里需要修改用户 symmetric 的 HOST。

mysql> UPDATE user SET HOST='%' WHERE user='symmetric';

mysql> FLUSH privileges;

生成分销系统表

在 symmetric 中默认已经给出了自动化生成分销系统表的方式，只要一句指令即可：

指令

# ./bin/symadmin --engine corp-000 create-sym-tables

检查结果

查看 MySQL 中的数据表

mysql> show tables;

+-----------------------------+

| Tables\_in\_sample |

+-----------------------------+

| item |

| item\_selling\_price |

| sale\_return\_line\_item |

| sale\_transaction |

| sym\_channel |

| sym\_conflict |

| sym\_data |

| sym\_data\_event |

| sym\_data\_gap |

| sym\_extension |

| sym\_extract\_request |

| sym\_file\_incoming |

| sym\_file\_snapshot |

| sym\_file\_trigger |

| sym\_file\_trigger\_router |

| sym\_grouplet |

| sym\_grouplet\_link |

| sym\_incoming\_batch |

| sym\_incoming\_error |

| sym\_load\_filter |

| sym\_lock |

| sym\_node |

| sym\_node\_channel\_ctl |

| sym\_node\_communication |

| sym\_node\_group |

| sym\_node\_group\_channel\_wnd |

| sym\_node\_group\_link |

| sym\_node\_host |

| sym\_node\_host\_channel\_stats |

| sym\_node\_host\_job\_stats |

| sym\_node\_host\_stats |

| sym\_node\_identity |

| sym\_node\_security |

| sym\_outgoing\_batch |

| sym\_parameter |

| sym\_registration\_redirect |

| sym\_registration\_request |

| sym\_router |

| sym\_sequence |

| sym\_table\_reload\_request |

| sym\_transform\_column |

| sym\_transform\_table |

| sym\_trigger |

| sym\_trigger\_hist |

| sym\_trigger\_router |

| sym\_trigger\_router\_grouplet |

+-----------------------------+

46 rows in set (0.00 sec)

填充数据

虽然上面说了一堆东西，但是还只是一堆空的数据表，如下我们就把 symmetric 自带的数据填充到数据表中。操作过程如下：

指令

# ./bin/dbimport --engine corp-000 ./samples/insert\_sample.sql

结果检查

mysql> select \* from item\_selling\_price;

+----------+----------+-------+------+

| item\_id | store\_id | price | cost |

+----------+----------+-------+------+

| 11000001 | 001 | 0.20 | 0.10 |

| 11000001 | 002 | 0.30 | 0.20 |

+----------+----------+-------+------+

2 rows in set (0.00 sec)

启动 SymmetricDS

因为这里是两个节点之间的数据同步，所以我们需要启动两个节点：corp 和 store

start corp node

corp 设置开启的端口为 8888

# ./bin/sym --engine corp-000 --port 8888

start store node

store 设置开启的端口为 9090

# ./bin/sym --engine store-001 --port 9090

上面两个窗口的启动之后，会进入等待状态，所以如果有其他操作，就需要另外打开窗口了。

注册节点

这里还有一步需要处理就是注册节点，注册操作在 corp 和 store 启动之后进行。

# ./bin/symadmin --engine corp-000 open-registration store 001

Log output will be written to /home/hadoop/symmetric/logs/symmetric.log

[] - AbstractCommandLauncher - Option: name=engine, value={corp-000}

[corp-000] - JdbcDatabasePlatformFactory - Detected database 'MySQL', version '5', protocol 'mysql'

[corp-000] - JdbcDatabasePlatformFactory - The IDatabasePlatform being used is org.jumpmind.db.platform.mysql.MySqlDatabasePlatform

[corp-000] - MySqlSymmetricDialect - The DbDialect being used is org.jumpmind.symmetric.db.mysql.MySqlSymmetricDialect

[corp-000] - ExtensionService - Found 0 extension points from the database that will be registered

[corp-000] - StagingManager - The staging directory was initialized at the following location: /home/hadoop/symmetric/tmp/corp-000

[corp-000] - ClusterService - This node picked a server id of 192.168.70.211

[corp-000] - ExtensionService - Found 0 extension points from the database that will be registered

[corp-000] - ClientExtensionService - Found 9 extension points from spring that will be registered

[corp-000] - RegistrationService - Registration was reopened for 001

Opened registration for node group of 'store' external ID of '001'

发送初始负载

发送初始负载的目的是将最开始的时候 corp 节点数据表中的数据同步到 store 节点中。如果你开始的数据表中没有这些数据，那么就可以不用发送此负载了。

指令

# ./bin/symadmin --engine corp-000 reload-node 001

结果

Log output will be written to /home/hadoop/symmetric/logs/symmetric.log

[] - AbstractCommandLauncher - Option: name=engine, value={corp-000}

[corp-000] - JdbcDatabasePlatformFactory - Detected database 'MySQL', version '5', protocol 'mysql'

[corp-000] - JdbcDatabasePlatformFactory - The IDatabasePlatform being used is org.jumpmind.db.platform.mysql.MySqlDatabasePlatform

[corp-000] - MySqlSymmetricDialect - The DbDialect being used is org.jumpmind.symmetric.db.mysql.MySqlSymmetricDialect

[corp-000] - ExtensionService - Found 0 extension points from the database that will be registered

[corp-000] - StagingManager - The staging directory was initialized at the following location: /home/hadoop/symmetric/tmp/corp-000

[corp-000] - ClusterService - This node picked a server id of 192.168.70.211

[corp-000] - ExtensionService - Found 0 extension points from the database that will be registered

[corp-000] - ClientExtensionService - Found 9 extension points from spring that will be registered

Successfully enabled initial load for node 001

测试验证

这里的验证为在 corp(即 192.168.70.211) 节点上修改一条数据，观察两个节点中数据表的变化。

在 corp 节点的数据库中执行相应 SQL 语句：

更新

mysql> UPDATE item\_selling\_price SET price='0.21' WHERE item\_id='11000001' AND store\_id='001';

corp

mysql> select \* from item\_selling\_price;

+----------+----------+-------+------+

| item\_id | store\_id | price | cost |

+----------+----------+-------+------+

| 11000001 | 001 | 0.21 | 0.10 |

| 11000001 | 002 | 0.30 | 0.20 |

+----------+----------+-------+------+

2 rows in set (0.00 sec)

store

mysql> select \* from item\_selling\_price;

+----------+----------+-------+------+

| item\_id | store\_id | price | cost |

+----------+----------+-------+------+

| 11000001 | 001 | 0.21 | 0.10 |

+----------+----------+-------+------+

1 row in set (0.00 sec)

插入

注意这里不要只添加一条记录，因为涉及到了数据库的触发器，我们虚拟了一条交易记录，如下：

mysql> INSERT INTO item (item\_id, name) VALUES (110000055, 'Soft Drink');

mysql> INSERT INTO item\_selling\_price (item\_id, store\_id, price) VALUES (110000055, '001', 0.65);

mysql> INSERT INTO item\_selling\_price (item\_id, store\_id, price) VALUES (110000055, '002', 1.00);

当一条交易被完成之后，我们再去检查一下 store 节点中的数据信息：

mysql> select \* from item\_selling\_price;

+-----------+----------+-------+------+

| item\_id | store\_id | price | cost |

+-----------+----------+-------+------+

| 11000001 | 001 | 0.22 | 0.10 |

| 110000055 | 001 | 0.65 | NULL |

+-----------+----------+-------+------+

2 rows in set (0.00 sec)

mysql> select \* from item;

+-----------+------------+

| item\_id | name |

+-----------+------------+

| 11000001 | Yummy Gum |

| 110000055 | Soft Drink |

+-----------+------------+

2 rows in set (0.00 sec)

数据的确是有增加的，验证完成。