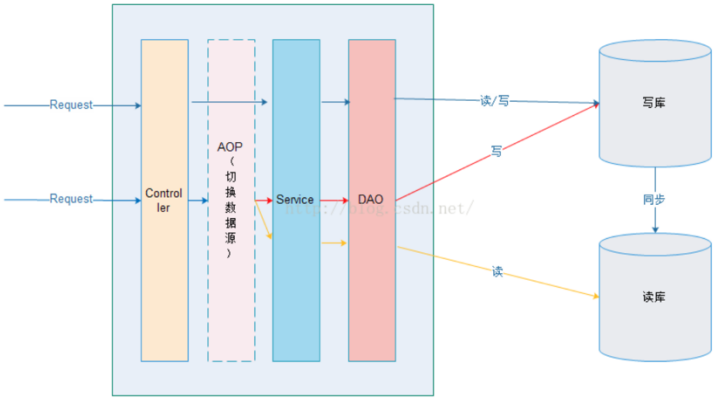
Mysql 读写分离以及 nginx 负载均衡方案

5140379026 徐瑾卿

一 基本思想

对数据库的操作读多写少，读操作对数据库的压力很大，因此对 mysql 做读写分离，其中一个主库负责写，其余的从库负责读。要求主库与从库数据一致，读数据到读库，写数据到写库。

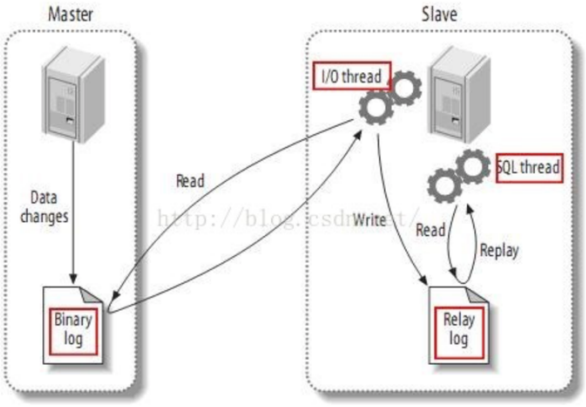


使用 AOP 对 service 做区分，根据 service 的命名来区分，带有 query，get 的走读库，带有 update，insert，delete 的走写库。

二 搭建 mysql 集群

使用主机下的数据库做主库，虚拟机下的做从库。

1 Mysql 主从备份原理：



1、master 将数据改变记录到二进制日志(binarylog)中,也即是配置文件 log-bin 指定的文件(这些记录叫做二进制日志事件，binary log events)
2、slave 将 master 的 binary logevents 拷贝到它的中继日志(relay log)
3、slave 重做中继日志中的事件,将改变反映它自己的数据(数据重演)

2 主库配置

允许远程访问：

```
mysql -u root -p
mysql>use mysql;
mysql>update user set host = '%' where user = 'root';
mysql>select host, user from user;
```

在 my.ini 里添加：

```
#开启主从复制，主库的配置
log-bin= mysql3306-bin
#指定主库serverid
server-id=101
#指定同步的数据库，如果不指定则同步全部数据库
binlog-do-db=mydb
```

在主库下创建同步账户：

```

#授权用户slave01使用123456密码登录mysql
grant replication slave on *.* to 'slave01'@'127.0.0.1'identified by '123456';
flush privileges;
```

3 从库配置

在 my.ini 修改：

#指定 serverid，只要不重复即可

server-id=102

以下执行 SQL：

```

CHANGEMASTER TO
 master_host='127.0.0.1',
 master_user='slave01',
 master_password='123456',
 master_port=3306,
 master_log_file='mysql3306-bin.000006',
 master_log_pos=1120;

#启动slave同步
STARTSLAVE;

#查看同步状态
SHOWSLAVE STATUS;

```

三 配置数据源

1 mysql 连接配置文件 jdbc.properties

```
jdbc.master.driver=com.mysql.jdbc.Driver
jdbc.master.url=jdbc:mysql://127.0.0.1:3306/mydb
jdbc.master.username=slave01
jdbc.master.password=123456


jdbc.slave01.driver=com.mysql.jdbc.Driver
jdbc.slave01.url=jdbc:mysql://127.0.0.1:3307/mydb
jdbc.slave01.username=slave01
jdbc.slave01.password=123456
```

2 定义动态数据源 使用 threadlocal 保证线程安全

```
public class DynamicDataSourceHolder {
    //写库对应的数据源key
    private static final String MASTER = "master";
    //读库对应的数据源key
    private static final String SLAVE = "slave";
    //使用ThreadLocal记录当前线程的数据源key
    private static final ThreadLocal<String> holder = new ThreadLocal<String>();
    public static void putDataSourceKey(String key) {
        holder.set(key);
    }
    public static String getDataSourceKey() {
        return holder.get();
    }
    public static void markMaster(){
        putDataSourceKey(MASTER);
    }
    public static void markSlave(){
        putDataSourceKey(SLAVE);
    }
}
```

```
import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;
public class DynamicDataSource extends AbstractRoutingDataSource{

    @Override
    protected Object determineCurrentLookupKey() {
        // 使用DynamicDataSourceHolder保证线程安全，并且得到当前线程中的数据源key
        return DynamicDataSourceHolder.getDataSourceKey();
    }

}
```

applicationContext.xml 中配置：

```
    <bean id="masterDataSource" class="com.jolbox.bonecp.BoneCPDataSource"
        destroy-method="close">
        <!-- 数据库驱动 -->
        <property name="driverClass" value="${jdbc.master.driver}" />
        <!-- 相应驱动的jdbcUrl -->
        <property name="jdbcUrl" value="${jdbc.master.url}" />
        <!-- 数据库的用户名 -->
        <property name="username" value="${jdbc.master.username}" />
        <!-- 数据库的密码 -->
        <property name="password" value="${jdbc.master.password}" />
    </bean>


    <bean id="slave01DataSource" class="com.jolbox.bonecp.BoneCPDataSource"
        destroy-method="close">
        <!-- 数据库驱动 -->
        <property name="driverClass" value="${jdbc.slave01.driver}" />
        <!-- 相应驱动的jdbcUrl -->
        <property name="jdbcUrl" value="${jdbc.slave01.url}" />
        <!-- 数据库的用户名 -->
        <property name="username" value="${jdbc.slave01.username}" />
        <!-- 数据库的密码 -->
        <property name="password" value="${jdbc.slave01.password}" />
    </bean>
```

master slave 与 dynamicdatasourceholder 中的 key 一致。

```
    <!-- 定义数据源，使用自己实现的数据源 -->
    <bean id="dataSource" class="com.module.dataSource.DynamicDataSource">
        <!-- 设置多个数据源 -->
        <property name="targetDataSources">
            <map key-type="java.lang.String">
                <!-- 这个key需要和程序中的key一致 -->
                <entry key="master" value-ref="masterDataSource"/>
                <entry key="slave" value-ref="slave01DataSource"/>
            </map>
        </property>
        <!-- 设置默认的数据源，这里默认走写库 -->
        <property name="defaultTargetDataSource" ref="masterDataSource"/>
    </bean>
```

采用配置的方法与代码解耦，可以添加多个 slave

3 定义 AOP 切片

```
import org.apache.commons.lang3.StringUtils;
import org.aspectj.lang.JoinPoint;
public class DataSourceAspect {
    public void before(JoinPoint point) {
        // 获取到当前执行的方法名
        String methodName = point.getSignature().getName();
        if (isSlave(methodName)) {
            // 标记为读库
            DynamicDataSourceHolder.markSlave();
        } else {
            // 标记为写库
            DynamicDataSourceHolder.markMaster();
        }
    }
    private Boolean isSlave(String methodName) {
        // 方法名以query、get开头的方法名走从库
        return StringUtils.startsWithAny(methodName, ...searchStrings: "query", "get");
    }


}
```

利用 AOP，遇到负责读的 service 时，将 key 标记为 slave，对应从数据库，遇到负责写的 service 时，将 key 标记为 master，对应主数据库。

applicationContext.xml:

定义 transaction 注入 datasource

```
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>
```

配置事务：

```
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            //读
            <tx:method name="query*" read-only="true" />
            <tx:method name="get*" read-only="true" />
            //写
            <tx:method name="insert*" propagation="REQUIRED" />
            <tx:method name="update*" propagation="REQUIRED" />
            <tx:method name="delete*" propagation="REQUIRED" />

            <!--其他方法使用默认事务策略 -->
            <tx:method name="*" />
        </tx:attributes>
    </tx:advice>
```

配置 AOP 切面：

```
    <bean class="cn.itcast.usermanage.spring.DataSourceAspect" id="dataSourceAspect" />

    <aop:config>
        <!-- 定义切面，所有的service的所有方法 -->
        <aop:pointcut id="txPointcut" expression="execution(* xx.xxx.xxxxxxx.service.*.*(..))" />
        <!-- 应用事务策略到Service切面 -->
        <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>

        <!-- 将切面应用到自定义的切面处理器上，-9999保证该切面优先级最高执行 -->
        <aop:aspect ref="dataSourceAspect" order="-9999">
            <aop:before method="before" pointcut-ref="txPointcut" />
        </aop:aspect>
    </aop:config>
```

四 nginx 做负载均衡

当请求数过大时，仅有一台服务器会负载过大。因此考虑将服务器做分布式。利用 nginx，设置一台服务器做负载均衡，另外 3 台服务器处理请求。

有 3 种方式负载均衡：

（1） IP 哈希策略

优点是同一个客户端的请求会被分配到同一台服务器，可以会话保持。

缺点是同一台服务器可能负载过大。

（2） 加权轮询策略

优点是不依赖客户端的信息，更合理地调度请求。

缺点是无法会话保持。

(3) 轮询

无法真正做到公平

Bookstore 里有购物车功能，需要会话保持，并且多个客户端访问服务器的可能性远高于同一个客户端反复访问服务端，因此应该选用 IP 哈希策略。

1 负载均衡器 PA 配置 nginx.conf 文件

```
upstream bookstore {
    server 192.168.2.3:8081;
    server 192.168.2.4:8082;
    server 192.168.2.5:8083;
    ip_hash;
}
2 负载均衡器PA配置Server
server{
    listen 8080;
    server_name bookstore;
    location / {
        proxy_pass http://localhost;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
3 三个worker 配置server
server{
    listen 8081;
    server_name bookstore;
    index index.html;
    root /
}
server{
    listen 8082;
    server_name bookstore;
    index index.html;
    root /;
}
server{
    listen 8083;
    server_name bookstore;
    index index.html;
    root /;
}
```