链滴

记一次 redis 和 mysql 断连接问题

作者：ken2105x
原文链接：https://ld246.com/article/1568879980546
来源网站：链滴

数据库连接周期性断开问题

背景

项目在测试环境部署的时候，测试人员经常反馈访问超时，app 端跑满了设置的最大超时时间 60s 依没有返回信息

部署环境

- 前端：app
- 后端：阿里云容器服务
- 后台管理： 阿里云容器服务
- MySQL：独立的阿里云服务器
- Redis：独立的阿里云服务器

版本及配置

- 后端：
  - MySQL-connector-Java:5.1.34
  - druid:1.0.26
  - Spring-boot-start-data-Redis:2.1.2.RELEASE
- 后台管理系统
  - MySQL-connector-Java:8.0.13
  - HikariCP:3.2.0
- MySQL:5.6.39
- Redis:3.2.12

```
# 后端
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: *****
    username: *****
    password: *****
    initialSize: 10   # 初始化大小
    minIdle: 10       # 最小
    maxActive: 50    # 最大
    maxWait: 10000    # 获取连接等待超时的时间
    timeBetweenEvictionRunsMillis: 60000    # 配置间隔多久才进行一次检测，检测需要关闭的空连接，单位是毫秒
    minEvictableIdleTimeMillis: 300000      # 配置一个连接在池中最小生存的时间，单位是毫秒
    validationQuery: SELECT 1 FROM DUAL
    validationQueryTimeout: 10000
```

```
    testOnConnect: true
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    poolPreparedStatements: true                    # 打开PSCache
    maxPoolPreparedStatementPerConnectionSize: 20   # 指定每个连接上PSCache的大小
    filters: stat,wall,log4j                        # 配置监控统计拦截的filters，去掉后监控界面sql无法统计，wall'用于防火墙
    connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000  # 通过connetProperties属性来打开mergeSql功能；慢SQL记录

# 后台
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: *****
    username: *****
    password: *****
    type: com.zaxxer.hikari.HikariDataSource
    hikari:
      auto-commit: true # 此属性控制从池返回的连接的默认自动提交行为,默认值：true
      connection-timeout: 30000 # 待连接池分配连接的最大时长（毫秒），超过这个时长还没可用连接则发生SQLException， 缺省:30秒
      idle-timeout: 60000 # 一个连接idle状态的最大时长（毫秒），超时则被释放（retired），缺省:0分钟
      minimum-idle: 10 # 最小空闲连接数量
      maximum-pool-size: 10 # 连接池中允许的最大连接数。缺省值：10；推荐的公式：((core_count * 2) + effective_spindle_count)
      max-lifetime: 1800000 # 一个连接的生命时长（毫秒），超时而且没被使用则被释放（retired），缺省:30分钟，建议设置比数据库超时时长少30秒，参考MySQL wait_timeout参数（show variables like '%timeout%';)
      connection-test-query: SELECT 'x' FROM DUAL
```

现象

查询后台日志，发现后台周期性的报连接错误

MySQL

```
2019-08-22 15:31:19.880  WARN [ms-auth,da0c7cd96a7425eb,129248695007c355,true] 6 --- [http-nio-2030-exec-2] o.s.s.o.provider.endpoint.TokenEndpoint  : Handling error: InternalAuthenticationServiceException,
### Error querying database.  Cause: com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link failure

The last packet successfully received from the server was 1,891,990 milliseconds ago.  The last packet sent successfully to the server was 1,892,020 milliseconds ago.
### The error may exist in com/qivan/issmp/dal/appuser/mapper/AppUserMapper.java (best guess)
### The error may involve com.qivan.issmp.dal.appuser.mapper.AppUserMapper.findByUsernme-Inline
### The error occurred while setting parameters
### SQL: select * from app_user where username = ?
### Cause: com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link fail
```

re

Redis

org.springframework.dao.QueryTimeoutException: Redis command timed out; nested excepti
n is io.lettuce.core.RedisCommandTimeoutException: Command timed out

问题排查

微服务交互

*第一次没有仔细查看日志，以为是服务之间连接的问题，对 feign 客户端做了饿加载，并使用 k8s 的 svc 替代了 erueka。虽然减少的第一次访问的连接时间。

==**问题仍然存在**==*

MySQL 服务器超时时间

*在日志中查找到对应的错误信息，发现为连接断开问题，百度了一波后，修改了 MySQL 服务器端置。

==**问题仍然存在**==*

```
# 默认8小时，修改成1年
# interactive_timeout针对交互式连接，wait_timeout针对非交互式连接。所谓的交互式连接，即在
ysql_real_connect()函数中使用了CLIENT_INTERACTIVE选项。
# 说得直白一点，通过mysql客户端连接数据库是交互式连接，通过jdbc连接数据库是非交互式连接。

wait_timeout=31536000
interactive_timeout=31536000
```

应用连通性测试

*修改 testOnBorrow=true ，在获取连接时测试连通性

==**问题仍然存在**==*

更换连接池

更换连接池，使用 hikari。然而突然想起来后台管理就是用的 hikari，也有相同的问题

==**问题仍然存在**==

网络

在开发过程中，发现 MySQL 客户端在连接数据库的时候，也经常断开。一开始因为是因为 Linux 系上不稳定，后来仔细想想，这就和应用是一个现象。换个方向，从网络原因的方向查询

==**问题就在这里**==

问题原因

主要原因是阿里云会断开长时间闲置的 TCP 连接，不给两头发 FIN or RST 包

参考：http://blog.itpub.net/31556440/viewspace-2637027/

解决思路

由于 tcp 连接长时间闲置，考虑如下 2 个方向

keepalive

使用 keepalive，简单说通过心跳保持双方连接

参考：https://www.cnblogs.com/wangjq19920210/p/8440824.html

最大存活时间

修改连接池中连接最大存活时间，短于 tcp 连接断开的时间

处理

MySQL

Hikari

后台管理系统使用了 hikari 连接池。

修改 hikari 配置文件，修改 max-lifetime=180000 ，即 3 分钟。经测试，tcp 超时时间在 5-10 分钟左，3 分钟就销毁连接，可以避免使用无效连接。

```
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: *****
    username: *****
    password: *****
    type: com.zaxxer.hikari.HikariDataSource
    hikari:
      auto-commit: true # 此属性控制从池返回的连接的默认自动提交行为,默认值：true
      connection-timeout: 30000 # 待连接池分配连接的最大时长（毫秒），超过这个时长还没可用连接则发生SQLException， 缺省:30秒
      idle-timeout: 60000 # 一个连接idle状态的最大时长（毫秒），超时则被释放（retired），缺省:0分钟
      minimum-idle: 10 # 最小空闲连接数量
      maximum-pool-size: 10 # 连接池中允许的最大连接数。缺省值：10；推荐的公式：((core_count * 2) + effective_spindle_count)
      max-lifetime: 180000 # 一个连接的生命时长（毫秒），超时而且没被使用则被释放（retired）缺省:30分钟，建议设置比数据库超时时长少30秒，参考MySQL wait_timeout参数（show variables like '%timeout%';)
      connection-test-query: SELECT 'x' FROM DUAL
```

Druid

后端服务使用了 druid 连接池，通过查看官方文档，在 1.0.28 版本后加入了 keepalive，建议使用 1.1.16 及以后版本

参考https://github.com/alibaba/druid/wiki/KeepAlive_cn

当前使用版本为 1.0.26，果断升级，并手写了个 DruidDataSource

重点在于 keepAlive: true

```
## 代码
@Configuration
public class DruidConfig {

    @Value("${spring.datasource.url}")
    private String url;

    @Value("${spring.datasource.username}")
    private String username;

    @Value("${spring.datasource.password}")
    private String password;

    @Value("${spring.datasource.initialSize}")
    private int initialSize;

    @Value("${spring.datasource.minIdle}")
    private int minIdle;

    @Value("${spring.datasource.maxActive}")
    private int maxActive;

    @Value("${spring.datasource.maxWait}")
    private long maxWait;

    @Value("${spring.datasource.timeBetweenEvictionRunsMillis}")
    private long timeBetweenEvictionRunsMillis;

    @Value("${spring.datasource.minEvictableIdleTimeMillis}")
    private long minEvictableIdleTimeMillis;

    @Value("${spring.datasource.validationQuery}")
    private String validationQuery;

    @Value("${spring.datasource.testWhileIdle}")
    private boolean testWhileIdle;

    @Value("${spring.datasource.testOnBorrow}")
    private boolean testOnBorrow;

    @Value("${spring.datasource.testOnReturn}")
    private boolean testOnReturn;

    @Value("${spring.datasource.poolPreparedStatements}")
    private boolean poolPreparedStatements;

    @Value("${spring.datasource.maxPoolPreparedStatementPerConnectionSize}")
```

```
    private int maxPoolPreparedStatementPerConnectionSize;

    @Value("${spring.datasource.keepAlive}")
    private boolean keepAlive;

    @Value("${spring.datasource.connectionProperties}")
    private String connectionProperties;

    @Bean
    public DruidDataSource dataSource() {
        DruidDataSource dataSource = new DruidDataSource();
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        dataSource.setInitialSize(initialSize);
        dataSource.setMinIdle(minIdle);
        dataSource.setMaxActive(maxActive);
        dataSource.setTimeBetweenEvictionRunsMillis(timeBetweenEvictionRunsMillis);
        dataSource.setMinEvictableIdleTimeMillis(minEvictableIdleTimeMillis);
        dataSource.setValidationQuery(validationQuery);
        dataSource.setTestWhileIdle(testWhileIdle);
        dataSource.setTestOnBorrow(testOnBorrow);
        dataSource.setTestOnReturn(testOnReturn);
        dataSource.setPoolPreparedStatements(poolPreparedStatements);
        dataSource.setMaxPoolPreparedStatementPerConnectionSize(maxPoolPreparedStateme
tPerConnectionSize);
        dataSource.setKeepAlive(keepAlive);
        dataSource.setConnectionProperties(connectionProperties);
        return dataSource;
    }
}
```

配置

```
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://101.132.34.244:33008/issmp?useUnicode=true&characterEncoding=utf-8
autoReconnect=true&useSSL=false
    username: issmp
    password: Issmp123!@#
    initialSize: 10   # 初始化大小
    minIdle: 10        # 最小
    maxActive: 50    # 最大
    maxWait: 60000    # 获取连接等待超时的时间
    timeBetweenEvictionRunsMillis: 60000    # 配置间隔多久才进行一次检测，检测需要关闭的空
连接，单位是毫秒
    minEvictableIdleTimeMillis: 30000      # 配置一个连接在池中最小生存的时间，单位是毫秒
    validationQuery: SELECT 1 FROM DUAL
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    poolPreparedStatements: true                    # 打开PSCache
    maxPoolPreparedStatementPerConnectionSize: 20   # 指定每个连接上PSCache的大小
```

```
    keepAlive: true # 保持连接 默认为120秒心跳
    filters: stat,wall,log4j                    # 配置监控统计拦截的filters，去掉后监控界面sql无法统计，wall'用于防火墙
    connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000  # 通过connetProperties属性来打开mergeSql功能；慢SQL记录
```

Redis

Redis 使用的是 Spring-boot-start-data-Redis:2.1.2.RELEASE

首先在修改过程中，发现 Redis 的配置写错了，springboot2.×使用的配置有修改，之前一直使用 sprigboot1.×的配置，导致未使用连接池。

注意的是，连接池依赖 commons-pool2

先修改下 Redis 配置

```
spring:
  redis:
    # Redis数据库索引（默认为0）
    database: 0
    # Redis服务器地址
    host: 47.101.137.129
    # Redis服务器连接端口
    port: 6379
    # Redis服务器连接密码（默认为空）
    password: 666666
    # 连接超时时间（毫秒）
    timeout: 5000
    lettuce:
      pool:
        # 连接池最大连接数（使用负值表示没有限制）
        max-active: 8
        # 连接池最大阻塞等待时间（使用负值表示没有限制）
        max-wait: -1
        # 连接池中的最大空闲连接
        max-idle: 8
        # 连接池中的最小空闲连接
        min-idle: 0
```

发现 Redis 配置中没有超时时间，keepalive 等相关的配置选项

百度了一波发现 Redis 在服务器端有 keepalive 的，由于使用 docker 的默认部署，未加载配置文件

默认的配置文件，修改"tcp-keepalive 60"，60 秒心跳间隔，重新部署

```
# redis-server /usr/local/etc/redis/redis.conf 加载配置文件启动
docker run --name myredis -p 6379:6379 -v /docker/redis/data:/data -v /docker/redis/conf/rdis.conf:/usr/local/etc/redis/redis.conf  -d redis:3.2 redis-server /usr/local/etc/redis/redis.conf -appendonly yes --requirepass "666666"
```

总结

1. 查询问题还是要从日志入手，不能主观判断，多做无用功
2. 做开发的同时，需要对网络有一定了解，今后面向互联网的开发会越来越多