Oceanus 使用文档

五八同城信息技术有限公司

修订记录

修订日期	修订内容	修订人	修订版本号
2014/8/7	编写第一版文档	陈阳	0.0.1
2014/11/3	升级 0.0.2 版，增加报警、ORM	陈阳	0.0.2
2014/12/8	增加对 mybatis 和 hibernate 支持	陈阳	0.0.2
2014/12/27	兼容 jdk1.7，完善执行线程池功能	陈阳	0.0.2
2015/1/8	完善对 having 语句支持	陈阳	0.0.2

目录

1 引言

1.1 编写目的

Oceanus 是 58 同城的数据库中间件，这篇文档介绍 Oceanus 的使用方法和注意事项，由于底层服务极为重要，请严格按照文档要求进行接入

1.2 相关背景

1.2.1 数据积累，分库分表需求迫切

由于业务增长迅速，项目初期的单库单表已经效率低下，急需拆表来分担压力，提高效率。

1.2.2 分库分表逻辑与业务耦合

业务线各自实现的分库分表逻辑不一样，与自己的业务耦合过多，无法纳入到基础架构中给其他业务线使用

1.2.3 监控、连接池、Load Balance、HA、可扩展、可视化

数据访问层需要配套相关功能，使数据访问层可靠、可用、易用，其中每个功能点的意义不在这个文档中一一赘述

2 配置说明

configurations.xml 该配置文件说明了 Oceanus 初始化内容，文件名称不固定。root 节点为 <configurations> </configurations>，下面介绍配置节点：

2.1 threadpool

- 作用：

  Oceanus 内部执行器

- 属性：

  id – 唯一标识

  size – 线程池大小。根据DB操作并发量和服务器硬件情况调整

  qsize – 执行队列的长度

  timeout – 执行超时时间

- 示例：

  <threadpool id="default" size="100" qsize="1024" timeout="100000" />

2.2 bean

- 作用：

  类的声明，通过这里进行实例化

- 属性：

  id – 唯一标识

  class – 实现类

- 示例：

  <bean id="userShardFunc" class="com.bj58.oceanus.demo.shard.UserDynamicShardFunction" />

2.3 datanode

- 作用：

  配置数据源，有多少数据库实例就配置多少datanode

- 属性：

  id – 唯一标识

  parent – datanode 可继承，该属性标识该 datanode 继承自哪个 datanode

  slaves – HA功能，当有主从关系且该数据源为主时，在这里声明从库id

  alarm – 报警功能，当该数据源不可用和恢复正常时会调用其实现类的报警方法

- 子节点：

  url –连接数据库的 URL

  username – 登陆数据库所用的帐号

  password – 登陆数据库所用的密码

  maxActive – 连接池中可同时连接的最大的连接数，为0则表示没有限制，默认为8

  maxIdle – 连接池中最大的空闲的连接数（默认为8,设 0 为没有限制），超过的空闲连接将被释放，如果设置为负数表示不限制（maxIdle不能设置太小，因为假如在高负载的情况下，连接的打开时间比关闭的时间快，会引起连接池中idle的个数 上升超过maxIdle，而造成频繁的连接销毁和创建)

  minIdle – 连接池中最小的空闲的连接数（默认为0，一般可调整5），低于这个数量会被创建新的连接（该参数越接近maxIdle，性能越好，因为连接的创建和销毁，都是需要消耗资源的；但是不能太大，因为在机器很空闲的时候，也会创建低于minidle个数的连接）

  maxWait –超过时间会丢出错误信息 最大等待时间(单位为 ms)，当没有可用连接时，连接池等待连接释放的最大时间，超过该时间限制会抛出异常，如果设置-1表示无限等待（默认为-1，一般可调整为60000ms，避免因线程池不够用，而导致请求被无限制挂起）

  driverClassName – JDBC Driver Class

  removeAbandoned – 超过removeAbandonedTimeout时间后，是否进行没

用连接的回收（默认为false）

removeAbandonedTimeout –超过时间限制，回收无用的连接（默认为 300 秒），removeAbandoned 必须为 true

➢ 示例：

```
<datanode id="safe_db1" slaves="safe_db2" alarm="com.bj58.oceanus.demo.alarms.DefaultAlarm" >
        <url>
        <![CDATA[
    jdbc:mysql://localhost:3306/ppsafe58com_0?useUnicode=true&characterEncoding=UTF-8&zeroDate
TimeBehavior=convertToNull
        ]]>
        </url>
        <username>octester</username>
        <password>123456</password>
        <driverClassName>com.mysql.jdbc.Driver</driverClassName>
        <initialSize>4</initialSize>
        <maxActive>10</maxActive>
        <maxWait>5000</maxWait>
        <maxIdle>2</maxIdle>
        <minIdle>2</minIdle>
    </datanode>

 <datanode id="safe_db2" parent="safe_db1" slaves="safe_db3">
        <url>
        <![CDATA[
    jdbc:mysql://localhost:3306/ppsafe58com_2?useUnicode=true&characterEncoding=UTF-8&zeroDate
TimeBehavior=convertToNull
        ]]>
        </url>
    </datanode>
```

2.4 namenode

- 作用：

  同一类型数据源的簇，可配置多个datanode，便于扩展读写分离等功能

- 属性：

  id – 唯一标识

  loadbalance – 负载方式，可选值：POLL 表示轮询，POLL-WEIGHT 表示有权重的轮询，RANDOM 表示随机，RANDOM-WEIGHT 表示有权重的随机，HA-RANDOM 表示支持高可用的随机，HA-RANDOM-WEIGHT 表示支持高可用的有权重随机。默认是 HA-RANDOM-WEIGHT

- 子节点：

  datanodes – 该节点下面可配置多个datanode子节点，并声明每个datanode的读写模式和负载权重

- 示例：

```
<namenode id="safe_source1" loadbalance="HA-RANDOM-WEIGHT">
    <datanodes>
        <datanode ref="safe_db1" access-mode="READ-WRITE" weight="10" />
    </datanodes>
</namenode>
```

2.5 table

- 作用：

  描述具体的表规则

- 示例：

```
<table name="t_userdynamic" differ-name="false" shard-type="BY_DATABASE_TABLE"
threadpool="default">
    <columns>
        <column name="uid"/>
```

```
        </columns>
        <namenodes>
            <namenode ref="safe_source1" tablename="t_userdynamic"/>
            <namenode ref="safe_source2" tablename="t_userdynamic"/>
        </namenodes>
        <function ref="userShardFunc" />
        <trackers>
            <tracker type="EXECUTE_SQL" threshold="10"
    class="com.bj58.oceanus.demo.trackers.TableExecuteSqlTracker" />
        </trackers>
</table>
```

2.6 function

- 作用：

  作为 table 的子节点，用于声明路由规则

- 属性：

  ref – Function 类型的 bean id

- 示例：

  <function ref="userShardFunc" />

2.7 tracker

- 作用：

  关键功能的监控埋点，当指定位置的执行时间过长时，调用这里配置的类方法，可用于监控报警。Tracker可以作为configurations的子节点表示全局有效，也可以作为table的子节点表示只对匹配的表有效

- 属性：

  type – 埋点类型，目前有以下几种固定值：GET_CONNECTION（从连接池中获取连接）、CONNECTION_CONTEXT（一个 connection 的生命周

期）、EXECUTE_SQL（sql 执行耗时）、PARSE_SQL（sql 解析耗时）

threshold – 执行时间的阈值，大于这个阈值才会调用方法

class – 埋点回调方法实现类

- 示例：

```
<tracker type="EXECUTE_SQL" threshold="2" class="com.bj58.oceanus.demo.trackers.ExecuteSqlTracker" />
<tracker type="PARSE_SQL" threshold="5" class="com.bj58.oceanus.demo.trackers.ParseSqlTracker" />
```

2.8 include

- 作用：

  配置文件支持引用功能，file 属性为被引用文件的相对路径名称

- 示例：

```
<include file="configurations_demo_tables.xml"/>
```

3 编码调用

3.1 初始化

首先调用 Oceanus 初始化方法，加载配置进行各项的初始化：
Oceanus.init("d:/configurations_demo.xml");

3.2 获取链接

- 使用说明：通过静态方法 Oceanus.getConnection(); 获取一个链接，connection、statement、resultset 这些对象的释放也要调用 Oceanus 的静态方法，

- 示例：

```
private static final String sql_insert = "INSERT INTO t_userdynamic(uid, pwds,
ver) VALUES(?, ?, ?)";
    public boolean addUserDynamic(final UserDynamic userDynamic) throws
Exception {
        Connection connection = null;
        PreparedStatement ps = null;
        try {
            connection = Oceanus.getConnection();
            ps = connection.prepareStatement(sql_insert);
            ps.setLong(1, userDynamic.getUid());
            ps.setString(2, userDynamic.getPwds());
            ps.setString(3, userDynamic.getVersion());

            return ps.execute();
        } finally {
            Oceanus.closeStatement(ps);
            Oceanus.closeConnection(connection);
        }
    }
```

3.3 事务的写法

- 规则说明：事务不可以跨库，只能在同一数据源进行事务操作，全过程分为三个阶段：初始化、提交、释放。

  初始化方法：Oceanus.beginTransaction(connection);
  提交方法：Oceanus.endTransaction(connection);
  释放方法：Oceanus.releaseTransaction();

- 示例：

```
public void doTransaction() throws Exception{
    Connection connection = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
        connection = Oceanus.getConnection();
        Oceanus.beginTransaction(connection);
        stmt = connection.createStatement();
        stmt.executeUpdate("update t_userdynamic set ver='try' where
uid=1");
        rs = stmt.executeQuery("select * from t_userdynamic where uid=1");
        stmt.executeUpdate("update t_userdynamic set ver='yu' where uid=5");

        Oceanus.endTransaction(connection);
    } finally {
        Oceanus.closeResultSet(rs);
        Oceanus.closeConnection(connection);
        Oceanus.releaseTransaction();
    }
}
```

注：正如要保证资源一定能释放一样，事务也需要在 finally 中释放

3.4 自定义路由逻辑函数

- 实现接口：

```
com.bj58.oceanus.core.shard.Function
```

- 实现方法：

```
public int execute(int size, Map<String, Object> parameters)
```

- 参数说明：

  size：表示该 table 中配置的 datanode 数量

  parameters：key 为字段名，大写开头，value 为字段在 sql 中的值

- 返回值：
  namenode 的下标

- 示例。(例子中根据 uid 取模来返回下标，这不是通用方法，方法取决于 uid 的生成规则，非顺序递增的逻辑会造成数据分布不均)：

```
public class UserDynamicShardFunction implements Function {

    public int execute(int size, Map<String, Object> parameters) {
        long uid = Long.parseLong(parameters.get("UID").toString());
        return (int)(uid % size);
    }

}
```

3.5 自定义埋点函数

- 实现接口：
  com.bj58.oceanus.core.timetracker.Tracker

- 实现方法：
  `public void doTrack(TrackResult trackResult)`

- 参数说明：
  trackResult：当对应埋点执行时间超过配置时间，Oceanus 会调用该方法
  trackResult.getTableName() 有 table 时获取对应的 table 名字
  trackResult.getSql() 有 sql 时获取对应的 sql 内容
  trackResult.getCostTime() 获取执行耗时

- 示例：

```
public class ExecuteSqlTracker extends Tracker{
    @Override
    public void doTrack(TrackResult trackResult) {
        trackResult.getCostTime();
        trackResult.getSql();
        trackResult.getTableName();
        System.err.println("in custom ExecuteSqlTracker:" +
trackResult.toString());
    }
}
```

5 报警

报警扫描周期是 Oceanus 内置的参数，不对外提供。两种方式实现：

5.1 实现 com.bj58.oceanus.core.alarm.Alarm 接口

将实现类名配置在 datanode 节点属性 alarm 中，实现其中的 void excute(AlarmType type, String dataNodeId) 方法，每当需要报警时就会调用 excute 方法

（注：这种报警调用会很频繁，根据各自需求进行选择实现）

5.2 继承 com.bj58.oceanus.core.alarm.PeriodicAlarm 抽象类

实现其中的三个方法：

1. long getAlarmCycle()　返回报警时间间隔
2. TimeUnit getAlarmUnit()　指定报警间隔时间单位
3. void excuteAlarm(AlarmType type, String dataNodeId)　对报警内容的实现，目前报警类型包括 DB_NOTAVALIABLE：表示当前 datanode 不可用，DB_AVALIABLE：表示 datanode 恢复

第二种是周期性的报警，约定周期内不会对相同 datanode 相同报警类型做重复的报警调用

6 对象关系映射(ORM)

Oceanus 定位于分布式存储数据的中间件解决方案，分布式环境中对数据存储的结构和读写方式都有较高的要求，举例说明：ebay 早期使用了 Oracle, 后来重构成 MySQL 集群，把业务逻辑尽量从底层数据库剥离，数据库仅仅作为存放数据和简单查询的容器，以方便横向扩展。所以 Oceanus 中的 ORM 只做从 ResultSet 到对象的一对一映射。

6.1 持久化对象(PO)注解

1. @NotColumn 注解在成员变量上，表示该变量不做映射。PS：默认情况下所有的变量都要做映射
2. @Column 注解在成员变量上，该注解有三个参数可选：
   name : 声明 table 的列名，默认与成员变量名相同
   setFuncName : 定义特殊 set 方法名，默认是 ”set”+成员变量首字母大写
   getFuncName : 定义特殊 get 方法名，默认是 ”get”+成员变量首字母大写

6.2 数据访问层继承 com.bj58.oceanus.client.orm.BaseDao

其中有三个方法供子类调用：

1. boolean com.bj58.oceanus.client.orm.BaseDao.excute(String sql, Object... objects) throws Exception

2. int com.bj58.oceanus.client.orm.BaseDao.excuteUpdate(String sql, Object... objects) throws Exception

3. <T> List<T> com.bj58.oceanus.client.orm.BaseDao.excuteQuery(Class<T> clazz, String sql, Object... objects) throws Exception

第三个方法会将查询结果中的 ResultSet 结果映射成传入的 Class<T>对象，封装成集合返回

7 组件补充说明

7.1 链接池

Oceanus 底层的连接池使用 DBCP，例举几个重要的相关配置如下，原文参见：
http://commons.apache.org/proper/commons-dbcp/configuration.html

参数名	DBCP 默认值	Oceanus 默认值	描述说明
minEvictableIdleTimeMillis	1000 * 60 * 30	1000 * 60 * 10	连接在池中保持空闲，而不被空闲连接回收器线程回收的最小时间值
numTestsPerEvictionRun	3	5	在每次空闲连接回收器线程运行时检查的连接数量
removeAbandoned	false	true	是否对连接池中的链接做超时清理
removeAbandonedTimeout	300	10	如果该连接超过 removeAbandonedTimeout 设置的 n 秒，认为该连接可以被清理，物理断开并删除链接
timeBetweenEvictionRunsMillis	1000 * 60 * 30	1000 * 60 * 5	每 timeBetweenEvictionRunsMillis 毫秒检查一次连接池中空闲的连接,把空闲时间超过 minEvictableIdleTimeMillis 毫秒的连接断开,直到连接池中的连接数到 minIdle 为止
testOnBorrow	true	false	获取链接的时候检查链接是否可用
testOnReturn	false	false	归还链接的时候检查链接是否可用

testWhileIdle	false	true	链接空闲的时候检查链接可用性
validationQuery	空	SELECT 1	检查链接可用性的 SQL，这里必须是一条非空结果的查询语句

7.2 对 mybatis 的支持

1. POM 依赖：
   <groupId>com.bj58</groupId>
   <artifactId>oceanus-plugins-mybatis</artifactId>
   <version>0.0.2-SNAPSHOT</version>
2. 事务管理器需要配置为 JDBC，如：<transactionManager type="JDBC"/>
3. DataSource 的工厂实现类为 com.bj58.oceanus.plugins.mybatis.datasource.OceanusDataSourceFactory

DEMO 可见 oceanus-plugins-mybatis 的单元测试代码

7.2 对 hibernate 的支持

1. POM 依赖：
   <groupId>com.bj58</groupId>
   <artifactId>oceanus-plugins-hibernate</artifactId>
   <version>0.0.2-SNAPSHOT</version>
2. 链接生产者为 com.bj58.oceanus.plugins.hibernate.datasource.OceanusConnectionProvider

DEMO 可见 oceanus-plugins-hibernate 的单元测试代码