# 0.ORM概述

主要的几个核心点：ORM框架支持**多数据源，分库分表**、**事务夸库事务**，以及**自定义连接池和第三方连接池**的接入

**1.初始化读取配置元数据**：

初始化的时候主要涉及读取两个配置文件，一个是所有数据源配置文件，一个是数据模型对象所在包路径：在启动时会加载这些路径下的所有数据模型对象，其中这些数据模型对应的java bean中的字段信息，其中头部通过注解配置了：分库分表回调接口实现类，对应数据源配置的标志key以及**表名等信息**。

**2.读写分离&分库分表（多数据源）**：

分库分表的实现采用的是客户端编码方式实现的，主要是为每个数据模型java bean构建一个分库分表的回调实现类，当然如果分库分表实现逻辑是一样的，那么可以共用一个的。这里回调接口主要是6个，分别对应读写的数据源（RoutingStorage）、数据库、表的构建

而具体的回调时机是在构建SQL语句的时候来进行回调从而完成具体SQL的拼接。

**3.事务夸库事务**：

应用面上访问数据库的操作基本就是DQL和DML，针对查询语句，走一个查询的DQLExecutor实现。具体包含四个步骤，首先是通过初始化读取的配置文件中的元数据，以及回调分库分表接口的实现来确定具体的数据源库表，从而组装成完整的SQL语句；第二个是SQL语句的构建，包含了数据源连接池的初始化以及连接的获取；接着是执行阶段：核心就是在构建Connection组装Statement的时候，设置事务autoCommit为true，就是每个查询都是自动提交的。最后一步就是Connection的return到对应数据源的连接池中。

然后针对DML的插入和更新操作，走的是一个支持DML的DMLExecutor的实现，这里主要将一个DML分为4步，首先是通过初始化读取的配置文件中的元数据，以及回调分库分表接口的实现来确定具体的数据源库表，从而组装成完整的SQL语句；第二步就是构建SQL语句，分库分表的逻辑，第三步就是执行SQL，这一步不会对事物进行提交，第三步是Commit操作，分别去对多个Statement的Connection进行commit事务的提交。上述2,3,4三个步骤包在一个try catch中，一旦哪一步出错，就会在异常中对此次操作涉及到的多个事务分别进行回滚。

**4.自定义连接池和第三方连接池**：

策略模式+模板模式：

1.DataBasePool为具体数据源实现定义的统一接口：

1.1包含：获取Connetion接口和归还Connection接口

2.然后会有个抽象实现类实现DataBasePool来统一实现

2.1获取Datasource（ds = setupDataSource(key, rsa)）

2.2然后定义了一个设置Dataseource的抽象方法（setupDataSource），用来给不同数据源实现提供设置数据源的接口

3.每个具体的数据源实现需要去实现该接口

2.3比如使用HikariCP第三方数据源的话，则需要基于

具体数据源的获取步骤：

1.在构建Statement的时候，首先根据数据源配置文件中的DataBasePoolStyle来获取指定DB（作为缓存key）的数据源实现类。

2.然后根据model对应的数据源中的数据源name+数据库name 来构建的key去数据源缓存中具体的获取Datasource

3.拿到具体的DataSource之后，获取Conneciton

4.执行完成之后，关闭statment、ResultSet资源，释放链接到具体的连接池

**TODO：第三方数据源链接归还的问题**

HikariDataSource#ProxyConnection

"Overridden" java.sql.Connection

**@Override**

**public final void close() throws SQLException{**

**}**

**TODO：自定义连接池的实现**

**和springboot的整合**

非标准的自定义的springboot start

1. AlbianBootListener implements SpringApplicationRunListener

**contextPrepared**{

String albianConfig = context.getEnvironment().getProperty("albianj.config");

boolean startResult = AlbianBootService.start(albianConfig);

}

2.spring.factories

org.springframework.boot.SpringApplicationRunListener=org.albianj.spring.AlbianBootListener

3.启动的时候：jvm运行环境系统变量-D指定配置文件目录

-Dalbianj.config=D:\innergitjava\newSpring\AuthorWeb\config\DEV\albianj

4.引入到web项目中

<dependency>

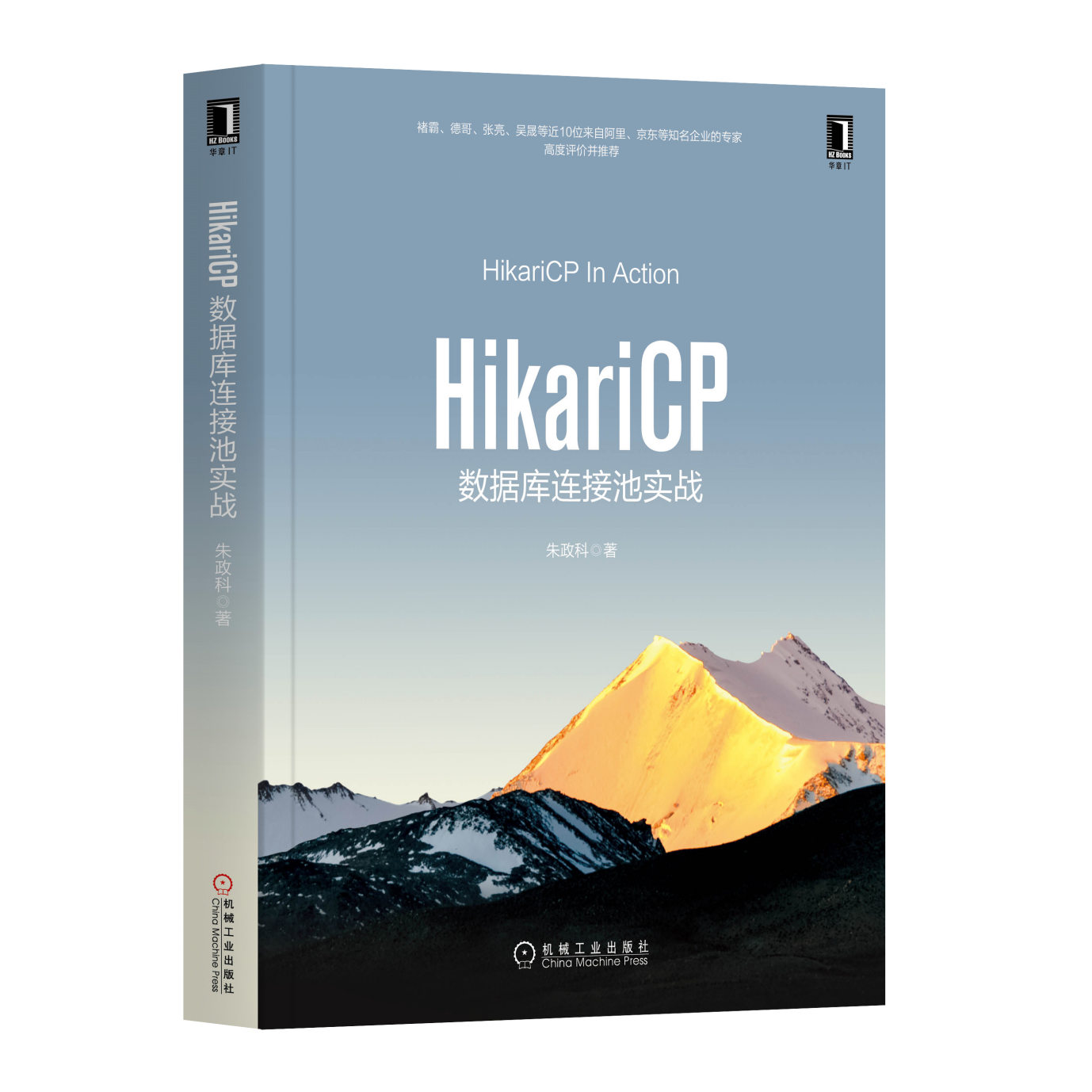
<groupId>albianj-framework</groupId>

<artifactId>albianj-framework-spring-starter</artifactId>

<version>${albianj-starter.verson}</version>

</dependency>

<https://developer.aliyun.com/article/726362>



# 1.一个查询请求的执行流程

**1.初始化读取配置元数据**：

初始化的时候主要涉及读取两个配置文件，一个是所有数据源配置文件，一个是数据模型对象所在包路径：在启动时会加载这些路径下的所有数据模型对象，其中这些数据模型对应的java bean中的字段信息，其中头部通过注解配置了：分库分表回调接口实现类，对应数据源配置的标志key以及**表名等信息**。

**2.为不同操作SQL构建不同的操作实例去执行**

第二步是也就是我们业务层面针对读写不同的操作去获取针对不写不同的操作实现类

这一步的前提是通过提供的SQLBuilder以代码的方式构建SQL，支持的不多只支持简单的增删改查

针对读写操作分别构建的是读写对应的不同的执行实现类：

DQLSqlExecutorService和DMLSqlExecutorService

这两个执行类包含了整个操作的所有流程

**3.以写为例，也就是DMLSqlExecutorService为例：**

**首先**：构造task（数据库上下文类），也就是该写操作涉及到：写数据源信息的信息，和封装拼接之后的SQL语句。

这一步的重点是通过回调路由实现类完成分库分表中的库表名的构造

在SQL语句的封装中会回调模型类上设置的**路由实现类：这个路由实现类包含三个方法-数据源名构造、库名构造、表名构造（分库分在了不同数据库服务器节点上，分库分在了同一个数据库服务器节点上）**

在SQL语句的封装中更具分库分表的信息去获取对应的数据源信息，然后根据分库分表字段来构造其所在库名和表名，比如XXXDB01 和XXXtable02.

**接着**：进入到DMLSqlExecutorService中具体execute方法中进行执行

这里分为四步来执行：preExecute，Execute，commit以及exceptionRollback和returnConnection

**1.preExecute:**

这一步的重点是：构造数据源连接池和Connection和Statement

构造数据源连接池：

一个database名称对应一个数据库连接池的实现类，而具体的数据库连接池实现类，根据数据源配置信息中的poolstyle字段来加载初始化不同的数据源连接池实现类，比如支持C3P0、DBCP、HIKARICP等以及自定义实现类（从缓存（Map,key为xxxDB01，value是实现类）中获取，没有就new一个然后放入map缓存中）。构造完成之后的数据源实现类放入到之前的taskinfo中去。

Connection：

然后就是从具体的数据源连接池中获取Connection，这里有个关键是，我们从JDBC的DataSource中获取到的Connection，都是自动提交的，我们这里需要显示的设置为false，是为了事务的回滚

PreparedStatement：

然后就是一个或者多个Statement的构造，设置参数

**2.Execute**:

这一步就是for循环去执行PreparedStatement#executeUpdate()方法

**3.commit**:

这一步就是Connection的commit方法进行事务提交，但是连接不关闭

**4.exceptionRollback:**

获取到所有taskInfo的Conneciton，然后调用rollback方法进行回归。

**5.returnConnection**

关闭Statement

归还Connection到数据库连接池中

## 结果集映

EntityMetadata 中包含这个数据库模型类的所有Fields，而Fields是个map，其中value是对应字段的Field

~~接口，Class，Type~~

~~然后会为Field设置setArrcessible为true~~

~~字段是private修饰时isAccessible()得到的值是false，此时不允许通过反射访问该字段~~

~~所以f.setAccessible(true)的作用就是让我们在用反射时访问私有变量。~~

结果集ResultSet#getObject(fAttr.getPropertyName()获取具体的结果Object实例

然后通过ResultConvert类来进行类型装换：逻辑类似下面这样

if (long.class.isAssignableFrom(cls) || Long.class.isAssignableFrom(cls)) {

return Long.parseLong(o.toString());

}

# 2.和Spring的整合

**ORM和Spring的整合，其实涉及二个方面**：

1.一个是如何将ORM的启动初始化加入到SpringBoot的启动流程中

2.另一个就是如何使用我们定义好的那些Dao层的操作实现类，也及时将这些实例加入到IOC容器中

针对第一步的具体实现是这样：

首先：

我们抽了一个starter orm的jar包，jar里面就一个类ORMStarter类，该类继承了SpringApplicationRunListener

在contextPrepared方法处，

先获取上下文的Environment，然后获取orm启动的配置路径属性值（在启动脚本中-D指定）

这个配置路径我们放在项目的config目录下，所以指定的是相对路径了

然后：就是调用ORM的start方法来进行初始化启动

接着：

在META-INF中新增spring.factories文件，新增

**org.springframework.boot.SpringApplicationRunListener**=**org.albianj.spring.AlbianBootListener**

最后：就是pom文件中引入依赖

针对第二步的具体实现：

有几个方案，

最简单的就是加入@Component注解，在启动配置类中通过注解ComponentScan来将其扫描进来。我们采用的也是这种方案

继承ImportBeanDefinitionRegistrar

或者

继承BeanDefinitionRegistryPostProcessor

但都不是太完美，所以就是使用了上述的直接在Dao的实现类上加上@Component注解注入到IOC容器中

# 3.如何设计一个ORM

主要设计4个部分：

1.数据操作api层：提供增删改查

2.数据处理层：参数解析，SQL语句构造，SQL执行，SQL结果集映射到实体类

3.框架支持层：分库分表路由支持，连接池管理，夸库事务的支持

4.启动引导层：数据源配置读取，数据模型实体类配置读取（会初始化构建相应的路由实现类，通过实体类上的配置）

# ======Other============

# O1.启动时资源加载

1.加载资源-数据库模型bean基础信息

AlbianEntityMetadata

2.加载资源-数据源配置基础信息

# O2.连接池

## 1.概述

通过storage.xml构建DataSource，放入缓存

通过DataSource获取Connection

支持可插拔的连接池实现：

体现在storage.xml配置中的PoolStyle标签，在该标签内指定具体需要使用的是哪个数据源的实现

## 2.内部提供的自己实现的一个数据库连接池

# O3.事务实现

## 1：DQL和DML下的isAutoCommit（回滚）

在从DateBasePool中获取Connection的时候，根据是查询操作还是更新插入操作的不同

其中的一个参数isAutoCommit设置不同：

**查询**操作的时候，isAutoCommit设置为false，不需要回滚

**更新插入**操作的时候，isAutoCommit设置为true，为了实现事务回滚

## 2：DQLSqlExecuteImpl 和DMLSqlExecuteImpl

查询走的是一个查询的Execute的实现类，内部只走了：

preExecute和executing以及executed三个步骤

插入更新走的是另外一个Execute的实现类，内部走了：

preExecute和executeHandler和commit以及exceptionHandler

## 3：当初为了实现事务回滚，查询这里没有设置自动提交

4.事务回滚

1.事务-自动回滚

首先，在数据源处获取到的Connection的autoCommit属性设置为false，不自动提交。

其次，将涉及到多个库表操作（可以不同库也可以同库），放在一个writejob中

接着，是构造多个操作对应的preprastatement

然后，for循环操作，一旦某个失败，则异常抛出

再，执行入口处catche异常，更具执行的不同阶段来进行不同的回滚操作（注意关键点是：多个statement的Connection的还没有commit，这个时候就可以调用Connection的rollback来一个一个的进行回滚）

说明以及可能存在的问题：

1.特比注意这是一个长事务

2.每个接口调用比如save update等都是一个writeJob或者updateJob都是，该大Job会封装多个小的task，每个task中封装了一个更新或者insert操作的全部相关信息，包括sql，参数，对应的PrepareStatement等信息。

2.事务-手动设置回滚

貌似好像不行：

INSERT INTO `book\_novels01` (`CBID`) VALUES (? )

DELETE FROM `book\_novels01` WHERE 1=1 AND `CBID` = ?

下面这样好像回滚反操作好像不行啊！

public boolean insert(String sessionid, IBookNovel novel) {

// TODO Auto-generated method stub

IAlbianDataAccessService dao = AlbianServiceRouter.getSingletonService(IAlbianDataAccessService.class, IAlbianDataAccessService.Name);

IDataAccessContext accessContext = dao.newDataAccessContext();

accessContext.add(AlbianDataAccessOpt.Create, novel);

accessContext.setRollBackOnError();

return accessContext.commit(sessionid);

}

=====================================

=============================================

public boolean batchUpdate(String sessionId,List<IBookNovel> novels){

IAlbianDataAccessService dao = AlbianServiceRouter.getSingletonService(IAlbianDataAccessService.class, IAlbianDataAccessService.Name);

//如果是更新操作且数据库对象未做任何更改操作会导致失败

return dao.save(sessionId, novels);

}

=============================================

public boolean insert(String sessionid, IBook\_Novels book\_Novels, IBook\_Novels\_Exp book\_Novels\_Exp,

IBook\_Novels\_Sclink book\_Novels\_Sclink) {

// TODO Auto-generated method stub

IDataAccessContext accessContext = getDao().newDataAccessContext();

accessContext.add(AlbianDataAccessOpt.Create, book\_Novels);

accessContext.add(AlbianDataAccessOpt.Create, book\_Novels\_Exp);

accessContext.add(AlbianDataAccessOpt.Create, book\_Novels\_Sclink);

return accessContext.commit(sessionid);

}

3.源码分析

FreePersistenceTransactionClusterScope#excute

writerJob.setWriterJobLifeTime(WriterJobLifeTime.NoStarted);

//TODO 这里会设置一个weiterJob中的多个库表的Connection的 setAutoCommit(boolean autoCommit)赋给值为false

**this.preExecute(writerJob);**

writerJob.setWriterJobLifeTime(WriterJobLifeTime.Opened);

**this.executeHandler(writerJob);**

writerJob.setWriterJobLifeTime(WriterJobLifeTime.Runned);

**this.commit(writerJob);**

writerJob.setWriterJobLifeTime(WriterJobLifeTime.Commited);

# O4.分库分表

Node1.在数据库model bean上的【路由】【数据源】定义

其中Name为数据源配置集合的key

StorageName为数据库名，真正用的是需要加上具体的分库数字的

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

@AlbianObjectRant(

Interface = XXXX.class,//TODO

DataRouter = TableDataRouter.class,

**ReaderRouters** = @AlbianObjectDataRouterRant(Name = "xxxDBrr", StorageName = "yyyyDBstorager", ***TableName*** = "table"),

**WriterRouters** = @AlbianObjectDataRouterRant(Name = "xxxDBwr", StorageName = "yyyyDBstoragew", ***TableName*** = "table")

)

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

StorageName:对应storage.xml配置文件中的<

StorageName标签对应的是<Name标签>：

最后应该解析成yyyyDBstorager1或者yyyyDBstoragew1，表示的是数据库服务器1。从配置获取获取到具体数据库服务器的相关配置

上面StorageName明白之后，对于DataBase和Table的配置只需要加上不同分库分表sharding后缀即可。而需要注意的是Database在Storage配置文件中，tablename在具体的数据库表对应bean的分库分表注解配置上，合情合理！

-----------

目前是读写分离两个库，主写库和从读库分别在一个数据库服务器上，然后主库这个数据库服务器上有多个数据库实例，写库这个数据库服务器上有多个写库实例

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

=====================================================

<Storage>

<**Name**>yyyyDBstorager1</Name>

<DatabaseStyle>MySql</DatabaseStyle>

<Server>x.x.x.xr1</Server>

<Port>3306</Port>

<**Database**>yyyyDB</Database>

<User>root</User>

<Password>db2202</Password>

<Pooling>true</Pooling>

<MinPoolSize>15</MinPoolSize>

<MaxPoolSize>15</MaxPoolSize>

<Timeout>1200</Timeout>

<Charset>utf8</Charset>

<Transactional>true</Transactional>

<PoolStyle>SpxDBCP</PoolStyle>

</Storage>

<Storage>

<**Name**>yyyyDBstorager2</Name>

<DatabaseStyle>MySql</DatabaseStyle>

<Server>x.x.x.xr2</Server>

<Port>3306</Port>

<**Database**>yyyyDB</Database>

<User>root</User>

<Password>db2202</Password>

<Pooling>true</Pooling>

<MinPoolSize>15</MinPoolSize>

<MaxPoolSize>15</MaxPoolSize>

<Timeout>1200</Timeout>

<Charset>utf8</Charset>

<Transactional>true</Transactional>

<PoolStyle>SpxDBCP</PoolStyle>

</Storage>

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

<Storage>

<**Name**>yyyyDBstoragew1</Name>

<DatabaseStyle>MySql</DatabaseStyle>

<Server>x.x.x.xw1</Server>

<Port>3306</Port>

<**Database**>xxxxxxDB</Database>

<User>root</User>

<Password>db2202</Password>

<Pooling>true</Pooling>

<MinPoolSize>15</MinPoolSize>

<MaxPoolSize>15</MaxPoolSize>

<Timeout>1200</Timeout>

<Charset>utf8</Charset>

<Transactional>true</Transactional>

<PoolStyle>SpxDBCP</PoolStyle>

</Storage>

<Storage>

<**Name**>yyyyDBstoragew2</Name>

<DatabaseStyle>MySql</DatabaseStyle>

<Server>x.x.x.xw2</Server>

<Port>3306</Port>

<**Database**>xxxxxxDB</Database>

<User>root</User>

<Password>db2202</Password>

<Pooling>true</Pooling>

<MinPoolSize>15</MinPoolSize>

<MaxPoolSize>15</MaxPoolSize>

<Timeout>1200</Timeout>

<Charset>utf8</Charset>

<Transactional>true</Transactional>

<PoolStyle>SpxDBCP</PoolStyle>

</Storage>

====================================================

->上面这样的是支持单台服务器中的过个写库和读库

那么如果有多个写库和读库部署在不同机器上，怎么办尼？

# O5.与spring-boot的整合