数据同步的终极解决方案，阿里巴巴开源的Canal框架当之无愧！！

[架构师社区](javascript:void(0);) 今天

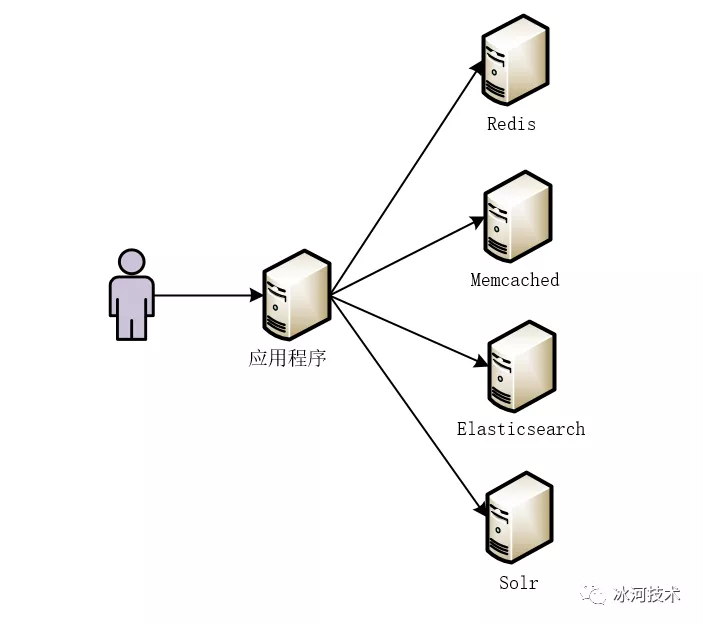
**写在前面**

在当今互联网行业，尤其是现在分布式、微服务开发环境下，为了提高搜索效率，以及搜索的精准度，会大量使用Redis、Memcached等NoSQL数据库，也会使用大量的Solr、Elasticsearch等全文检索服务。那么，这个时候，就会有一个问题需要我们来思考和解决：那就是数据同步的问题！如何将实时变化的数据库中的数据同步到Redis/Memcached或者Solr/Elasticsearch中呢？

**互联网背景下的数据同步需求**

在当今互联网行业，尤其是现在分布式、微服务开发环境下，为了提高搜索效率，以及搜索的精准度，会大量使用Redis、Memcached等NoSQL数据库，也会使用大量的Solr、Elasticsearch等全文检索服务。那么，这个时候，就会有一个问题需要我们来思考和解决：那就是数据同步的问题！如何将实时变化的数据库中的数据同步到Redis/Memcached或者Solr/Elasticsearch中呢？

例如，我们在分布式环境下向数据库中不断的写入数据，而我们读数据可能需要从Redis、Memcached或者Elasticsearch、Solr等服务中读取。那么，数据库与各个服务中数据的实时同步问题，成为了我们亟待解决的问题。

试想，由于业务需要，我们引入了Redis、Memcached或者Elasticsearch、Solr等服务。使得我们的应用程序可能会从不同的服务中读取数据，如下图所示。

本质上讲，无论我们引入了何种服务或者中间件，数据最终都是从我们的MySQL数据库中读取出来的。那么，问题来了，如何将MySQL中的数据实时同步到其他的服务或者中间件呢？

**注意：为了更好的说明问题，后面的内容以MySQL数据库中的数据同步到Solr索引库为例进行说明。**

**数据同步解决方案**

**1.在业务代码中同步**

在增加、修改、删除之后，执行操作Solr索引库的逻辑代码。例如下面的代码片段。

public ResponseResult updateStatus(Long[] ids, String status){  
    try{  
        goodsService.updateStatus(ids, status);  
        if("status\_success".equals(status)){  
            List<TbItem> itemList = goodsService.getItemList(ids, status);  
            itemSearchService.importList(itemList);  
            return new ResponseResult(true, "修改状态成功")  
        }  
    }catch(Exception e){  
        return new ResponseResult(false, "修改状态失败");  
    }  
}

**优点：**

操作简便。

**缺点：**

业务耦合度高。

执行效率变低。

**2.定时任务同步**

在数据库中执行完增加、修改、删除操作后，通过定时任务定时的将数据库的数据同步到Solr索引库中。

定时任务技术有：SpringTask，Quartz。

哈哈，还有我开源的mykit-delay框架，开源地址为：https://github.com/sunshinelyz/mykit-delay。

**这里执行定时任务时，需要注意的一个技巧是：第一次执行定时任务时，从MySQL数据库中以时间字段进行倒序排列查询相应的数据，并记录当前查询数据的时间字段的最大值，以后每次执行定时任务查询数据的时候，只要按时间字段倒序查询数据表中的时间字段大于上次记录的时间值的数据，并且记录本次任务查询出的时间字段的最大值即可，从而不需要再次查询数据表中的所有数据。**

**注意：这里所说的时间字段指的是标识数据更新的时间字段，也就是说，使用定时任务同步数据时，为了避免每次执行任务都会进行全表扫描，最好是在数据表中增加一个更新记录的时间字段。**

**优点：**

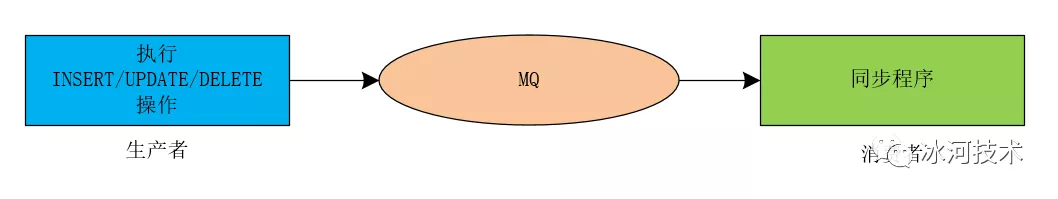
同步Solr索引库的操作与业务代码完全解耦。

**缺点：**

数据的实时性并不高。

**3.通过MQ实现同步**

在数据库中执行完增加、修改、删除操作后，向MQ中发送一条消息，此时，同步程序作为MQ中的消费者，从消息队列中获取消息，然后执行同步Solr索引库的逻辑。

我们可以使用下图来简单的标识通过MQ实现数据同步的过程。

我们可以使用如下代码实现这个过程。

public ResponseResult updateStatus(Long[] ids, String status){  
    try{  
        goodsService.updateStatus(ids, status);  
        if("status\_success".equals(status)){  
            List<TbItem> itemList = goodsService.getItemList(ids, status);  
            final String jsonString = JSON.toJSONString(itemList);  
            jmsTemplate.send(queueSolr, new MessageCreator(){  
                @Override  
                public Message createMessage(Session session) throws JMSException{  
                    return session.createTextMessage(jsonString);  
                }  
            });  
        }  
        return new ResponseResult(true, "修改状态成功");  
    }catch(Exception e){  
        return new ResponseResult(false, "修改状态失败");  
    }  
}

**优点：**

业务代码解耦，并且能够做到准实时。

**缺点：**

需要在业务代码中加入发送消息到MQ的代码，数据调用接口耦合。

**4.通过Canal实现实时同步**

Canal是阿里巴巴开源的一款数据库日志增量解析组件，通过Canal来解析数据库的日志信息，来检测数据库中表结构和数据的变化，从而更新Solr索引库。

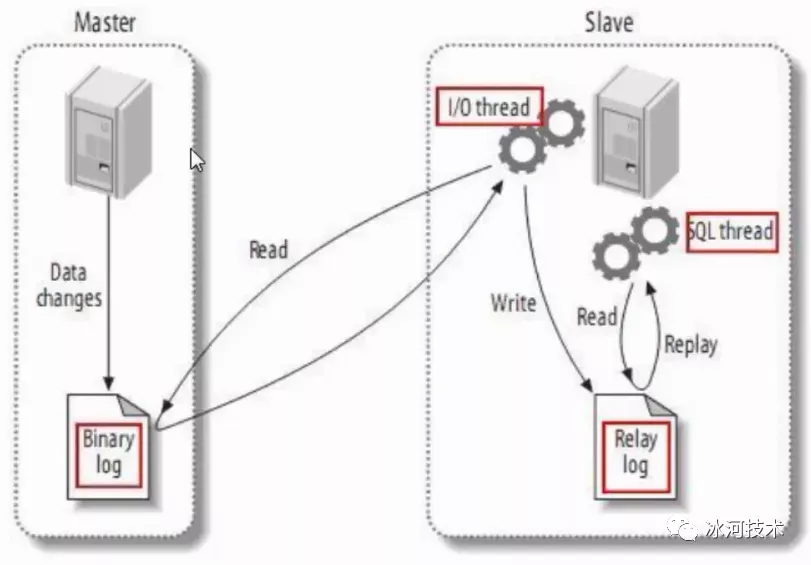
使用Canal可以做到业务代码完全解耦，API完全解耦，可以做到准实时。

**Canal简介**

阿里巴巴MySQL数据库binlog增量订阅与消费组件，基于数据库增量日志解析，提供增量数据订阅与消费，目前主要支持了MySQL。

Canal开源地址：https://github.com/alibaba/canal。

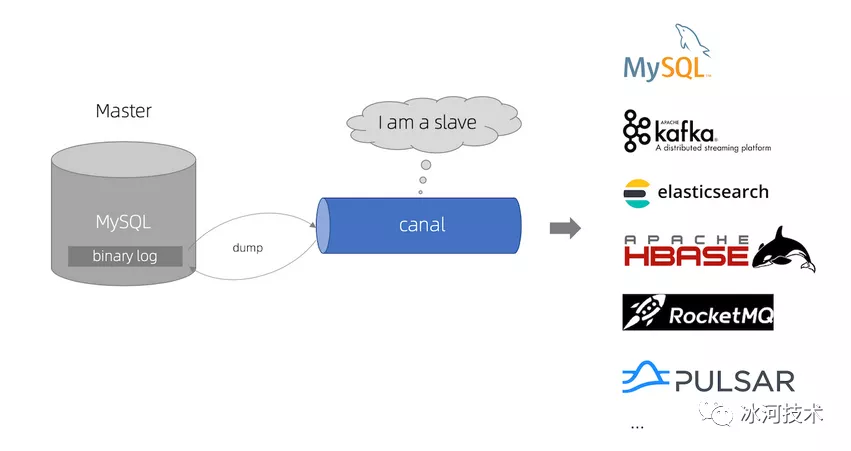
**Canal工作原理**

**MySQL主从复制的实现**

从上图可以看出，主从复制主要分成三步：

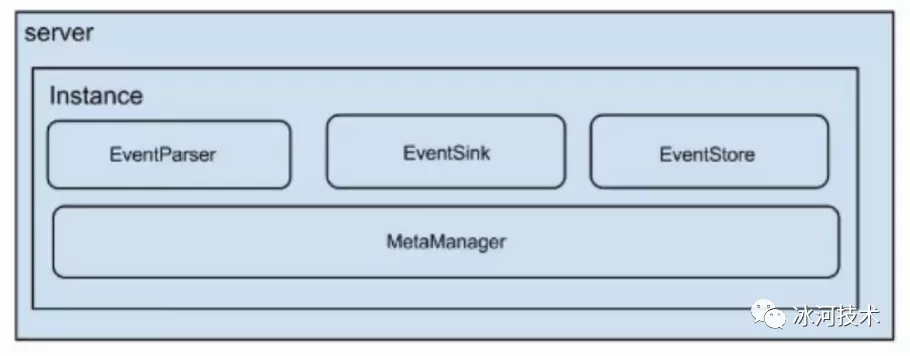
* Master节点将数据的改变记录到二进制日志（binary log）中（这些记录叫做二进制日志事件，binary log events，可以通过show binlog events进行查看）。
* Slave节点将Master节点的二进制日志事件（binary log events）拷贝到它的中继日志（relay log）。
* Slave节点重做中继日志中的事件将改变反映到自己本身的数据库中。

**Canal内部原理**

首先，我们来看下Canal的原理图，如下所示。

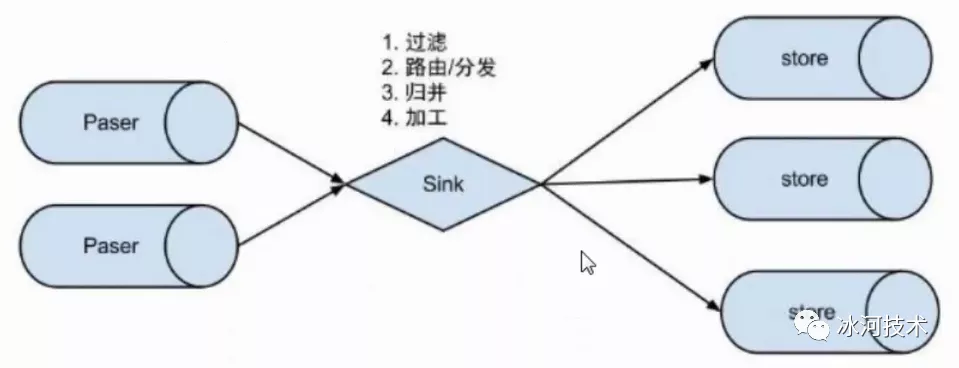
原理大致描述如下：

* Canal 模拟 MySQL slave 的交互协议，伪装自己为 MySQL Slave ，向 MySQL Master 发送dump 协议
* MySQL Master 收到 dump 请求，开始推送 binary log 给 Slave (即 Canal )
* Canal 解析 binary log 对象(原始为 byte 流)

**Canal内部结构**

说明如下：

* Server：代表一个Canal运行实例，对应一个JVM进程。
* Instance：对应一个数据队列（1个Server对应1个或者多个Instance）。

接下来，我们再来看下Instance下的子模块，如下所示。

* EventParser：数据源接入，模拟Slave协议和Master节点进行交互，协议解析。
* EventSink：EventParser和EventStore的连接器，对数据进行过滤、加工、归并和分发等处理。
* EventSore：数据存储。
* MetaManager：增量订阅和消费信息管理。

**Canal环境准备**

**设置MySQL远程访问**

grant all privileges on \*.\* to 'root'@'%' identified by '123456';  
flush privileges;

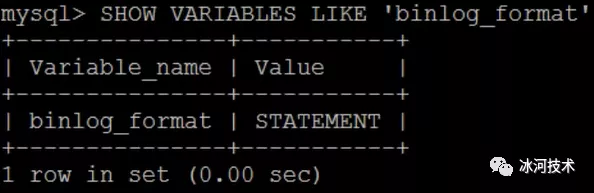
**MySQL配置**

注意：这里的MySQL是基于5.7版本进行说明的。

Canal的原理基于MySQL binlog技术，所以，要想使用Canal就要开启MySQL的binlog写入功能，建议配置binlog的模式为row。

可以在MySQL命令行输入如下命令来查看binlog的模式。

SHOW VARIABLES LIKE 'binlog\_format';

执行效果如下所示。

可以看到，在MySQL中默认的binlog格式为STATEMENT，这里我们需要将STATEMENT修改为ROW。修改/etc/my.cnf文件。

vim /etc/my.cnf

在[mysqld]下面新增如下三项配置。

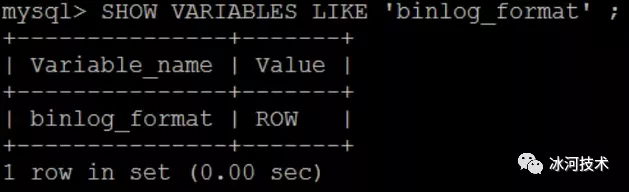
log-bin=mysql-bin  #开启MySQL二进制日志  
binlog\_format=ROW #将二进制日志的格式设置为ROW  
server\_id=1 #server\_id需要唯一，不能与Canal的slaveId重复

修改完my.cnf文件后，需要重启MySQL服务。

service mysqld restart

接下来，我们再次查看binlog模式。

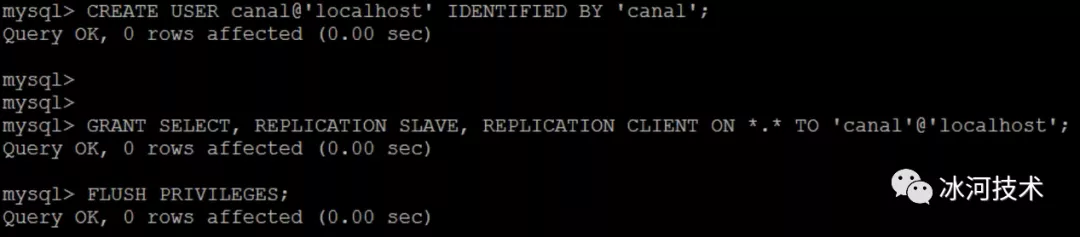
SHOW VARIABLES LIKE 'binlog\_format';



可以看到，此时，MySQL的binlog模式已经被设置为ROW了。

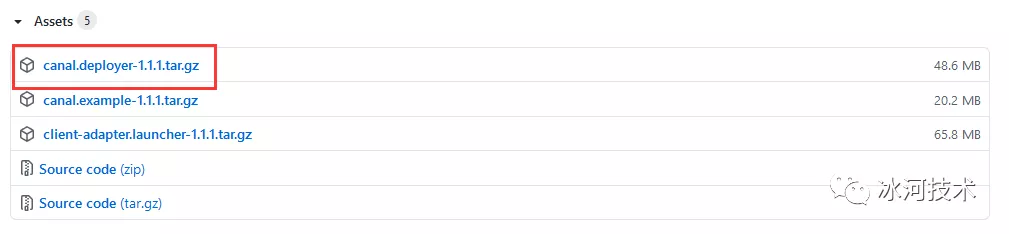
**MySQL创建用户授权**

Canal的原理是模式自己为MySQL Slave，所以一定要设置MySQL Slave的相关权限。这里，需要创建一个主从同步的账户，并且赋予这个账户相关的权限。

CREATE USER canal@'localhost' IDENTIFIED BY 'canal';  
GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'canal'@'localhost';  
FLUSH PRIVILEGES;

**Canal部署安装**

**下载Canal**

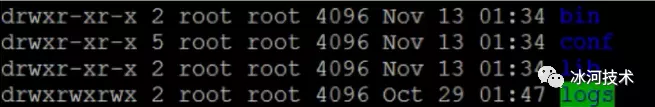
这里，我们以Canal 1.1.1版本进行说明，小伙伴们可以到链接 https://github.com/alibaba/canal/releases/tag/canal-1.1.1 下载Canal 1.1.1版本。

**上传解压**

将下载好的Canal安装包，上传到服务器，并执行如下命令进行解压

mkdir -p /usr/local/canal  
tar -zxvf canal.deployer-1.1.1.tar.gz -C /usr/local/canal/

解压后的目录如下所示。



各目录的说明如下：

* bin：存储可执行脚本。
* conf：存放配置文件。
* lib：存放其他依赖或者第三方库。
* logs：存放的是日志文件。

**修改配置文件**

在Canal的conf目录下有一个canal.properties文件，这个文件中配置的是Canal Server相关的配置，在这个文件中有如下一行配置。

canal.destinations=example

这里的example就相当于Canal的一个Instance，可以在这里配置多个Instance，多个Instance之间以逗号分隔即可。同时，这里的example也对应着Canal的conf目录下的一个文件夹。也就是说，Canal中的每个Instance实例都对应着conf目录下的一个子目录。

接下来，我们需要修改Canal的conf目录下的example目录的一个配置文件instance.properties。

vim instance.properties

修改如下配置项。

#################################################################  
## canal slaveId,注意：不要与MySQL的server\_id重复  
canal.instance.mysql.slaveId = 1234  
  
#position info，需要改成自己的数据库信息  
canal.instance.master.address = 127.0.0.1:3306  
canal.instance.master.journal.name =  
canal.instance.master.position =  
canal.instance.master.timestamp =  
  
#canal.instance.standby.address =  
#canal.instance.standby.journal.name =  
#canal.instance.standby.position =  
#canal.instance.standby.timestamp =  
  
#username/password，需要改成自己的数据库信息  
canal.instance.dbUsername = canal  
canal.instance.dbPassword = canal  
canal.instance.defaultDatabaseName =canaldb  
canal.instance.connectionCharset = UTF-8  
  
#table regex  
canal.instance.filter.regex = canaldb\\..\*  
#################################################################

选项含义：

* canal.instance.mysql.slaveId : mysql集群配置中的serverId概念，需要保证和当前mysql集群中id唯一;
* canal.instance.master.address: mysql主库链接地址;
* canal.instance.dbUsername : mysql数据库帐号;
* canal.instance.dbPassword : mysql数据库密码;
* canal.instance.defaultDatabaseName : mysql链接时默认数据库;
* canal.instance.connectionCharset : mysql 数据解析编码;
* canal.instance.filter.regex : mysql 数据解析关注的表，Perl正则表达式.

**启动Canal**

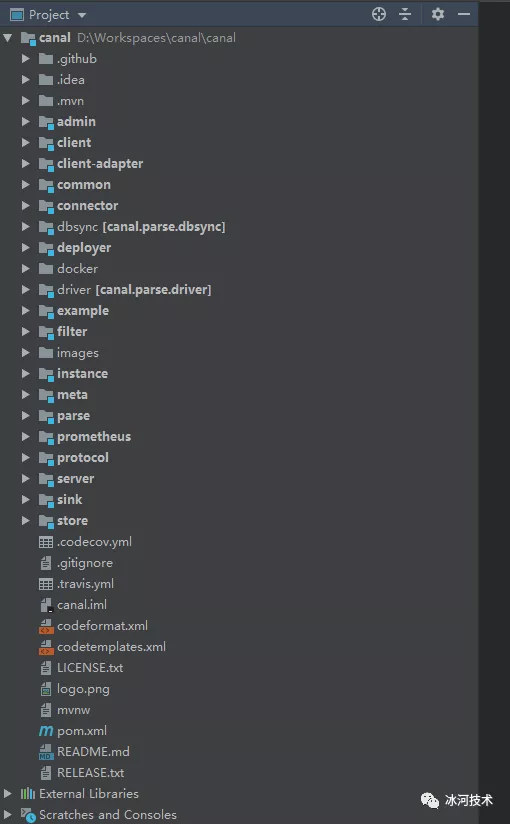
配置完Canal后，就可以启动Canal了。进入到Canal的bin目录下，输入如下命令启动Canal。

./startup.sh

**测试Canal**

**导入并修改源码**

这里，我们使用Canal的源码进行测试，下载Canal的源码后，将其导入到IDEA中。



接下来，我们找到example下的SimpleCanalClientTest类进行测试。这个类的源码如下所示。

package com.alibaba.otter.canal.example;  
  
import java.net.InetSocketAddress;  
  
import com.alibaba.otter.canal.client.CanalConnector;  
import com.alibaba.otter.canal.client.CanalConnectors;  
import com.alibaba.otter.canal.common.utils.AddressUtils;  
  
/\*\*  
 \* 单机模式的测试例子  
 \*   
 \* @author jianghang 2013-4-15 下午04:19:20  
 \* @version 1.0.4  
 \*/  
public class SimpleCanalClientTest extends AbstractCanalClientTest {  
  
    public SimpleCanalClientTest(String destination){  
           super(destination);  
     }  
  
    public static void main(String args[]) {  
        // 根据ip，直接创建链接，无HA的功能  
        String destination = "example";  
        String ip = AddressUtils.getHostIp();  
        CanalConnector connector = CanalConnectors.newSingleConnector(  
            new InetSocketAddress(ip, 11111),  
                destination,  
                "canal",  
                "canal");  
  
        final SimpleCanalClientTest clientTest = new SimpleCanalClientTest(destination);  
        clientTest.setConnector(connector);  
        clientTest.start();  
        Runtime.getRuntime().addShutdownHook(new Thread() {  
  
            public void run() {  
                try {  
                    logger.info("## stop the canal client");  
                    clientTest.stop();  
                } catch (Throwable e) {  
                    logger.warn("##something goes wrong when stopping canal:", e);  
                } finally {  
                    logger.info("## canal client is down.");  
                }  
            }  
  
        });  
    }  
}

可以看到，这个类中，使用的destination为example。在这个类中，我们只需要将IP地址修改为Canal Server的IP即可。

具体为：将如下一行代码。

String ip = AddressUtils.getHostIp();

修改为：

String ip = "192.168.175.100"

由于我们在配置Canal时，没有指定用户名和密码，所以，我们还需要将如下代码。

CanalConnector connector = CanalConnectors.newSingleConnector(  
    new InetSocketAddress(ip, 11111),  
    destination,  
    "canal",  
    "canal");

修改为：

CanalConnector connector = CanalConnectors.newSingleConnector(  
    new InetSocketAddress(ip, 11111),  
    destination,  
    "",  
    "");

修改完成后，运行main方法启动程序。

**测试数据变更**

接下来，在MySQL中创建一个canaldb数据库。

create database canaldb;

此时会在IDEA的命令行输出相关的日志信息。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* Batch Id: [7] ,count : [3] , memsize : [149] , Time : 2020-08-05 23:25:35  
\* Start : [mysql-bin.000007:6180:1540286735000(2020-08-05 23:25:35)]   
\* End : [mysql-bin.000007:6356:1540286735000(2020-08-05 23:25:35)]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

接下来，我在canaldb数据库中创建数据表，并对数据表中的数据进行增删改查，程序输出的日志信息如下所示。

#在mysql进行数据变更后，这里会显示mysql的bin日志。  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* Batch Id: [7] ,count : [3] , memsize : [149] , Time : 2020-08-05 23:25:35  
\* Start : [mysql-bin.000007:6180:1540286735000(2020-08-05 23:25:35)]   
\* End : [mysql-bin.000007:6356:1540286735000(2020-08-05 23:25:35)]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
================> binlog[mysql-bin.000007:6180] , executeTime : 1540286735000(2020-08-05 23:25:35) , gtid : () , delay : 393ms  
 BEGIN ----> Thread id: 43  
----------------> binlog[mysql-bin.000007:6311] , name[canal,canal\_table] , eventType : DELETE , executeTime : 1540286735000(2020-08-05 23:25:35) , gtid : () , delay : 393 ms  
id : 8    type=int(10) unsigned  
name : 512    type=varchar(255)  
----------------  
 END ----> transaction id: 249  
================> binlog[mysql-bin.000007:6356] , executeTime : 1540286735000(2020-08-05 23:25:35) , gtid : () , delay : 394ms  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* Batch Id: [8] ,count : [3] , memsize : [149] , Time : 2020-08-05 23:25:35  
\* Start : [mysql-bin.000007:6387:1540286869000(2020-08-05 23:25:49)]   
\* End : [mysql-bin.000007:6563:1540286869000(2020-08-05 23:25:49)]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
================> binlog[mysql-bin.000007:6387] , executeTime : 1540286869000(2020-08-05 23:25:49) , gtid : () , delay : 976ms  
 BEGIN ----> Thread id: 43  
----------------> binlog[mysql-bin.000007:6518] , name[canal,canal\_table] , eventType : INSERT , executeTime : 1540286869000(2020-08-05 23:25:49) , gtid : () , delay : 976 ms  
id : 21    type=int(10) unsigned    update=true  
name : aaa    type=varchar(255)    update=true  
----------------  
 END ----> transaction id: 250  
================> binlog[mysql-bin.000007:6563] , executeTime : 1540286869000(2020-08-05 23:25:49) , gtid : () , delay : 977ms  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* Batch Id: [9] ,count : [3] , memsize : [161] , Time : 2020-08-05 23:26:22  
\* Start : [mysql-bin.000007:6594:1540286902000(2020-08-05 23:26:22)]   
\* End : [mysql-bin.000007:6782:1540286902000(2020-08-05 23:26:22)]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
================> binlog[mysql-bin.000007:6594] , executeTime : 1540286902000(2020-08-05 23:26:22) , gtid : () , delay : 712ms  
 BEGIN ----> Thread id: 43  
----------------> binlog[mysql-bin.000007:6725] , name[canal,canal\_table] , eventType : UPDATE , executeTime : 1540286902000(2020-08-05 23:26:22) , gtid : () , delay : 712 ms  
id : 21    type=int(10) unsigned  
name : aaac    type=varchar(255)    update=true  
----------------  
 END ----> transaction id: 252  
================> binlog[mysql-bin.000007:6782] , executeTime : 1540286902000(2020-08-05 23:26:22) , gtid : () , delay : 713ms

**数据同步实现**

**需求**

将数据库数据的变化, 通过canal解析binlog日志, 实时更新到solr的索引库中。

**具体实现**

**创建工程**

创建Maven工程mykit-canal-demo，并在pom.xml文件中添加如下配置。

<dependencies>  
    <dependency>  
        <groupId>com.alibaba.otter</groupId>  
        <artifactId>canal.client</artifactId>  
        <version>1.0.24</version>  
    </dependency>  
    <dependency>  
        <groupId>com.alibaba.otter</groupId>  
        <artifactId>canal.protocol</artifactId>  
        <version>1.0.24</version>  
    </dependency>  
    <dependency>  
        <groupId>commons-lang</groupId>  
        <artifactId>commons-lang</artifactId>  
        <version>2.6</version>  
    </dependency>  
    <dependency>  
        <groupId>org.codehaus.jackson</groupId>  
        <artifactId>jackson-mapper-asl</artifactId>  
        <version>1.8.9</version>  
    </dependency>  
  
    <dependency>  
        <groupId>org.apache.solr</groupId>  
        <artifactId>solr-solrj</artifactId>  
        <version>4.10.3</version>  
    </dependency>  
    <dependency>  
        <groupId>junit</groupId>  
        <artifactId>junit</artifactId>  
        <version>4.9</version>  
        <scope>test</scope>  
    </dependency>  
  
</dependencies>

**创建log4j配置文件**

在工程的src/main/resources目录下创建log4j.properties文件，内容如下所示。

log4j.rootCategory=debug, CONSOLE  
  
# CONSOLE is set to be a ConsoleAppender using a PatternLayout.  
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender  
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout  
log4j.appender.CONSOLE.layout.ConversionPattern=%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n  
  
# LOGFILE is set to be a File appender using a PatternLayout.  
# log4j.appender.LOGFILE=org.apache.log4j.FileAppender  
# log4j.appender.LOGFILE.File=d:\axis.log  
# log4j.appender.LOGFILE.Append=true  
# log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout  
# log4j.appender.LOGFILE.layout.ConversionPattern=%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n

**创建实体类**

在io.mykit.canal.demo.bean包下创建一个Book实体类，用于测试Canal的数据传输，如下所示。

package io.mykit.canal.demo.bean;  
import org.apache.solr.client.solrj.beans.Field;  
import java.util.Date;  
public class Book implements Serializable {  
    private static final long serialVersionUID = -6350345408771427834L;{  
  
    @Field("id")  
    private Integer id;  
  
    @Field("book\_name")  
    private String name;  
  
    @Field("book\_author")  
    private String author;  
  
    @Field("book\_publishtime")  
    private Date publishtime;  
  
    @Field("book\_price")  
    private Double price;  
  
    @Field("book\_publishgroup")  
    private String publishgroup;  
  
    public Integer getId() {  
        return id;  
    }  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getAuthor() {  
        return author;  
    }  
  
    public void setAuthor(String author) {  
        this.author = author;  
    }  
  
    public Date getPublishtime() {  
        return publishtime;  
    }  
  
    public void setPublishtime(Date publishtime) {  
        this.publishtime = publishtime;  
    }  
  
    public Double getPrice() {  
        return price;  
    }  
  
    public void setPrice(Double price) {  
        this.price = price;  
    }  
  
    public String getPublishgroup() {  
        return publishgroup;  
    }  
  
    public void setPublishgroup(String publishgroup) {  
        this.publishgroup = publishgroup;  
    }  
  
    @Override  
    public String toString() {  
        return "Book{" +  
                "id=" + id +  
                ", name='" + name + '\'' +  
                ", author='" + author + '\'' +  
                ", publishtime=" + publishtime +  
                ", price=" + price +  
                ", publishgroup='" + publishgroup + '\'' +  
                '}';  
    }  
}

其中，我们在Book实体类中，使用Solr的注解@Field定义了实体类字段与Solr域之间的关系。

**各种工具类的实现**

接下来，我们就在io.mykit.canal.demo.utils包下创建各种工具类。

* BinlogValue

用于存储binlog分析的每行每列的value值，代码如下所示。

package io.mykit.canal.demo.utils;  
import java.io.Serializable;  
/\*\*  
 \*   
 \* ClassName: BinlogValue <br/>   
 \*   
 \* binlog分析的每行每列的value值；<br>  
 \* 新增数据：beforeValue 和 value 均为现有值；<br>  
 \* 修改数据：beforeValue是修改前的值；value为修改后的值；<br>  
 \* 删除数据：beforeValue和value均是删除前的值； 这个比较特殊主要是为了删除数据时方便获取删除前的值<br>  
 \*/  
public class BinlogValue implements Serializable {  
  
    private static final long serialVersionUID = -6350345408773943086L;  
  
    private String value;  
    private String beforeValue;  
  
    /\*\*  
     \* binlog分析的每行每列的value值；<br>  
     \* 新增数据：value：为现有值；<br>  
     \* 修改数据：value为修改后的值；<br>  
     \* 删除数据：value是删除前的值； 这个比较特殊主要是为了删除数据时方便获取删除前的值<br>  
     \*/  
    public String getValue() {  
        return value;  
    }  
    public void setValue(String value) {  
        this.value = value;  
    }  
  
    /\*\*  
     \* binlog分析的每行每列的beforeValue值；<br>  
     \* 新增数据：beforeValue为现有值；<br>  
     \* 修改数据：beforeValue是修改前的值；<br>  
     \* 删除数据：beforeValue为删除前的值； <br>  
     \*/  
    public String getBeforeValue() {  
        return beforeValue;  
    }  
    public void setBeforeValue(String beforeValue) {  
        this.beforeValue = beforeValue;  
    }  
}

* CanalDataParser

用于解析数据，代码如下所示。

package io.mykit.canal.demo.utils;  
  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
import org.apache.commons.lang.SystemUtils;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.util.CollectionUtils;  
  
import com.alibaba.otter.canal.protocol.Message;  
import com.alibaba.otter.canal.protocol.CanalEntry.Column;  
import com.alibaba.otter.canal.protocol.CanalEntry.Entry;  
import com.alibaba.otter.canal.protocol.CanalEntry.EntryType;  
import com.alibaba.otter.canal.protocol.CanalEntry.EventType;  
import com.alibaba.otter.canal.protocol.CanalEntry.RowChange;  
import com.alibaba.otter.canal.protocol.CanalEntry.RowData;  
import com.alibaba.otter.canal.protocol.CanalEntry.TransactionBegin;  
import com.alibaba.otter.canal.protocol.CanalEntry.TransactionEnd;  
import com.google.protobuf.InvalidProtocolBufferException;  
  
/\*\*  
 \* 解析数据  
 \*/  
public class CanalDataParser {  
  
    protected static final String DATE\_FORMAT   = "yyyy-MM-dd HH:mm:ss";  
    protected static final String yyyyMMddHHmmss = "yyyyMMddHHmmss";  
    protected static final String yyyyMMdd      = "yyyyMMdd";  
    protected static final String SEP           = SystemUtils.LINE\_SEPARATOR;  
    protected static String  context\_format     = null;  
    protected static String  row\_format         = null;  
    protected static String  transaction\_format = null;  
    protected static String row\_log = null;  
  
    private static Logger logger = LoggerFactory.getLogger(CanalDataParser.class);  
  
    static {  
        context\_format = SEP + "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + SEP;  
        context\_format += "\* Batch Id: [{}] ,count : [{}] , memsize : [{}] , Time : {}" + SEP;  
        context\_format += "\* Start : [{}] " + SEP;  
        context\_format += "\* End : [{}] " + SEP;  
        context\_format += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + SEP;  
  
        row\_format = SEP  
                     + "----------------> binlog[{}:{}] , name[{},{}] , eventType : {} , executeTime : {} , delay : {}ms"  
                     + SEP;  
  
        transaction\_format = SEP + "================> binlog[{}:{}] , executeTime : {} , delay : {}ms" + SEP;  
  
        row\_log = "schema[{}], table[{}]";  
    }  
  
    public static List<InnerBinlogEntry> convertToInnerBinlogEntry(Message message) {  
        List<InnerBinlogEntry> innerBinlogEntryList = new ArrayList<InnerBinlogEntry>();  
  
        if(message == null) {  
            logger.info("接收到空的 message; 忽略");  
            return innerBinlogEntryList;  
        }  
  
        long batchId = message.getId();  
        int size = message.getEntries().size();  
        if (batchId == -1 || size == 0) {  
            logger.info("接收到空的message[size=" + size + "]; 忽略");  
            return innerBinlogEntryList;  
        }  
  
        printLog(message, batchId, size);  
        List<Entry> entrys = message.getEntries();  
  
        //输出日志  
        for (Entry entry : entrys) {  
            long executeTime = entry.getHeader().getExecuteTime();  
            long delayTime = new Date().getTime() - executeTime;  
  
            if (entry.getEntryType() == EntryType.TRANSACTIONBEGIN || entry.getEntryType() == EntryType.TRANSACTIONEND) {  
                if (entry.getEntryType() == EntryType.TRANSACTIONBEGIN) {  
                    TransactionBegin begin = null;  
                    try {  
                        begin = TransactionBegin.parseFrom(entry.getStoreValue());  
                    } catch (InvalidProtocolBufferException e) {  
                        throw new RuntimeException("parse event has an error , data:" + entry.toString(), e);  
                    }  
                    // 打印事务头信息，执行的线程id，事务耗时  
                    logger.info("BEGIN ----> Thread id: {}",  begin.getThreadId());  
                    logger.info(transaction\_format, new Object[] {entry.getHeader().getLogfileName(),  
                                String.valueOf(entry.getHeader().getLogfileOffset()), String.valueOf(entry.getHeader().getExecuteTime()), String.valueOf(delayTime) });  
  
                } else if (entry.getEntryType() == EntryType.TRANSACTIONEND) {  
                    TransactionEnd end = null;  
                    try {  
                        end = TransactionEnd.parseFrom(entry.getStoreValue());  
                    } catch (InvalidProtocolBufferException e) {  
                        throw new RuntimeException("parse event has an error , data:" + entry.toString(), e);  
                    }  
                    // 打印事务提交信息，事务id  
                    logger.info("END ----> transaction id: {}", end.getTransactionId());  
                    logger.info(transaction\_format,  
                        new Object[] {entry.getHeader().getLogfileName(),  String.valueOf(entry.getHeader().getLogfileOffset()),  
                                String.valueOf(entry.getHeader().getExecuteTime()), String.valueOf(delayTime) });  
                }  
                continue;  
            }  
  
            //解析结果  
            if (entry.getEntryType() == EntryType.ROWDATA) {  
                RowChange rowChage = null;  
                try {  
                    rowChage = RowChange.parseFrom(entry.getStoreValue());  
                } catch (Exception e) {  
                    throw new RuntimeException("parse event has an error , data:" + entry.toString(), e);  
                }  
  
                EventType eventType = rowChage.getEventType();  
  
                logger.info(row\_format, new Object[] { entry.getHeader().getLogfileName(),  
                            String.valueOf(entry.getHeader().getLogfileOffset()), entry.getHeader().getSchemaName(),  
                            entry.getHeader().getTableName(), eventType, String.valueOf(entry.getHeader().getExecuteTime()), String.valueOf(delayTime) });  
  
                //组装数据结果  
                if (eventType == EventType.INSERT || eventType == EventType.DELETE || eventType == EventType.UPDATE) {  
                    String schemaName = entry.getHeader().getSchemaName();  
                    String tableName = entry.getHeader().getTableName();  
                    List<Map<String, BinlogValue>> rows = parseEntry(entry);  
  
                    InnerBinlogEntry innerBinlogEntry = new InnerBinlogEntry();  
                    innerBinlogEntry.setEntry(entry);  
                    innerBinlogEntry.setEventType(eventType);  
                    innerBinlogEntry.setSchemaName(schemaName);  
                    innerBinlogEntry.setTableName(tableName.toLowerCase());  
                    innerBinlogEntry.setRows(rows);  
  
                    innerBinlogEntryList.add(innerBinlogEntry);  
                } else {  
                    logger.info(" 存在 INSERT INSERT UPDATE 操作之外的SQL [" + eventType.toString() + "]");  
                }  
                continue;  
            }  
        }  
        return innerBinlogEntryList;  
    }  
  
    private static List<Map<String, BinlogValue>> parseEntry(Entry entry) {  
        List<Map<String, BinlogValue>> rows = new ArrayList<Map<String, BinlogValue>>();  
        try {  
            String schemaName = entry.getHeader().getSchemaName();  
            String tableName = entry.getHeader().getTableName();  
            RowChange rowChage = RowChange.parseFrom(entry.getStoreValue());  
            EventType eventType = rowChage.getEventType();  
  
            // 处理每个Entry中的每行数据  
            for (RowData rowData : rowChage.getRowDatasList()) {  
                StringBuilder rowlog = new StringBuilder("rowlog schema[" + schemaName + "], table[" + tableName + "], event[" + eventType.toString() + "]");  
  
                Map<String, BinlogValue> row = new HashMap<String, BinlogValue>();  
                List<Column> beforeColumns = rowData.getBeforeColumnsList();  
                List<Column> afterColumns = rowData.getAfterColumnsList();  
                beforeColumns = rowData.getBeforeColumnsList();  
                if (eventType == EventType.DELETE) {//delete  
                    for(Column column : beforeColumns) {  
                        BinlogValue binlogValue = new BinlogValue();  
                        binlogValue.setValue(column.getValue());  
                        binlogValue.setBeforeValue(column.getValue());  
                        row.put(column.getName(), binlogValue);  
                    }  
                } else if(eventType == EventType.UPDATE) {//update  
                    for(Column column : beforeColumns) {  
                        BinlogValue binlogValue = new BinlogValue();  
                        binlogValue.setBeforeValue(column.getValue());  
                        row.put(column.getName(), binlogValue);  
                    }  
                    for(Column column : afterColumns) {  
                        BinlogValue binlogValue = row.get(column.getName());  
                        if(binlogValue == null) {  
                            binlogValue = new BinlogValue();  
                        }  
                        binlogValue.setValue(column.getValue());  
                        row.put(column.getName(), binlogValue);  
                    }  
                } else { // insert  
                    for(Column column : afterColumns) {  
                        BinlogValue binlogValue = new BinlogValue();  
                        binlogValue.setValue(column.getValue());  
                        binlogValue.setBeforeValue(column.getValue());  
                        row.put(column.getName(), binlogValue);  
                    }  
                }   
  
                rows.add(row);  
                String rowjson = JacksonUtil.obj2str(row);  
  
                logger.info("#################################### Data Parse Result ####################################");  
                logger.info(rowlog + " , " + rowjson);  
                logger.info("#################################### Data Parse Result ####################################");  
                logger.info("");  
            }  
        } catch (InvalidProtocolBufferException e) {  
            throw new RuntimeException("parseEntry has an error , data:" + entry.toString(), e);  
        }  
        return rows;  
    }  
  
    private static void printLog(Message message, long batchId, int size) {  
        long memsize = 0;  
        for (Entry entry : message.getEntries()) {  
            memsize += entry.getHeader().getEventLength();  
        }  
  
        String startPosition = null;  
        String endPosition = null;  
        if (!CollectionUtils.isEmpty(message.getEntries())) {  
            startPosition = buildPositionForDump(message.getEntries().get(0));  
            endPosition = buildPositionForDump(message.getEntries().get(message.getEntries().size() - 1));  
        }  
  
        SimpleDateFormat format = new SimpleDateFormat(DATE\_FORMAT);  
        logger.info(context\_format, new Object[] {batchId, size, memsize, format.format(new Date()), startPosition, endPosition });  
    }  
  
    private static String buildPositionForDump(Entry entry) {  
        long time = entry.getHeader().getExecuteTime();  
        Date date = new Date(time);  
        SimpleDateFormat format = new SimpleDateFormat(DATE\_FORMAT);  
        return entry.getHeader().getLogfileName() + ":" + entry.getHeader().getLogfileOffset() + ":" + entry.getHeader().getExecuteTime() + "(" + format.format(date) + ")";  
    }  
}

* DateUtils

时间工具类，代码如下所示。

package io.mykit.canal.demo.utils;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class DateUtils {  
  
    private static final String FORMAT\_PATTERN = "yyyy-MM-dd HH:mm:ss";  
  
    private static SimpleDateFormat sdf = new SimpleDateFormat(FORMAT\_PATTERN);  
  
    public static Date parseDate(String datetime) throws ParseException{  
        if(datetime != null && !"".equals(datetime)){  
            return sdf.parse(datetime);  
        }  
        return null;  
    }  
  
  
    public static String formatDate(Date datetime) throws ParseException{  
        if(datetime != null ){  
            return sdf.format(datetime);  
        }  
        return null;  
    }  
  
    public static Long formatStringDateToLong(String datetime) throws ParseException{  
        if(datetime != null && !"".equals(datetime)){  
            Date d =  sdf.parse(datetime);  
            return d.getTime();  
        }  
        return null;  
    }  
  
    public static Long formatDateToLong(Date datetime) throws ParseException{  
        if(datetime != null){  
            return datetime.getTime();  
        }  
        return null;  
    }  
}

* InnerBinlogEntry

Binlog实体类，代码如下所示。

package io.mykit.canal.demo.utils;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Map;  
  
import com.alibaba.otter.canal.protocol.CanalEntry.Entry;  
import com.alibaba.otter.canal.protocol.CanalEntry.EventType;  
  
public class InnerBinlogEntry {  
  
    /\*\*  
     \* canal原生的Entry  
     \*/  
    private Entry entry;  
  
    /\*\*  
     \* 该Entry归属于的表名  
     \*/  
    private String tableName;  
  
    /\*\*  
     \* 该Entry归属数据库名  
     \*/  
    private String schemaName;  
  
    /\*\*  
     \* 该Entry本次的操作类型，对应canal原生的枚举；EventType.INSERT; EventType.UPDATE; EventType.DELETE;  
     \*/  
    private EventType eventType;  
  
    private List<Map<String, BinlogValue>> rows = new ArrayList<Map<String, BinlogValue>>();  
  
  
    public Entry getEntry() {  
        return entry;  
    }  
    public void setEntry(Entry entry) {  
        this.entry = entry;  
    }  
    public String getTableName() {  
        return tableName;  
    }  
    public void setTableName(String tableName) {  
        this.tableName = tableName;  
    }  
    public EventType getEventType() {  
        return eventType;  
    }  
    public void setEventType(EventType eventType) {  
        this.eventType = eventType;  
    }  
    public String getSchemaName() {  
        return schemaName;  
    }  
    public void setSchemaName(String schemaName) {  
        this.schemaName = schemaName;  
    }  
    public List<Map<String, BinlogValue>> getRows() {  
        return rows;  
    }  
    public void setRows(List<Map<String, BinlogValue>> rows) {  
        this.rows = rows;  
    }  
}

* JacksonUtil

Json工具类，代码如下所示。

package io.mykit.canal.demo.utils;  
  
import java.io.IOException;  
import org.codehaus.jackson.JsonGenerationException;  
import org.codehaus.jackson.JsonParseException;  
import org.codehaus.jackson.map.JsonMappingException;  
import org.codehaus.jackson.map.ObjectMapper;  
  
  
public class JacksonUtil {  
    private static ObjectMapper mapper = new ObjectMapper();  
  
    public static String obj2str(Object obj) {  
        String json = null;  
        try {  
            json = mapper.writeValueAsString(obj);  
        } catch (JsonGenerationException e) {  
            e.printStackTrace();  
        } catch (JsonMappingException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        return json;  
    }  
  
    public static <T> T str2obj(String content, Class<T> valueType) {  
        try {  
            return mapper.readValue(content, valueType);  
        } catch (JsonParseException e) {  
            e.printStackTrace();  
        } catch (JsonMappingException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        return null;  
    }  
}

**同步程序的实现**

准备好实体类和工具类后，我们就可以编写同步程序来实现MySQL数据库中的数据实时同步到Solr索引库了，我们在io.mykit.canal.demo.main包中常见MykitCanalDemoSync类，代码如下所示。

package io.mykit.canal.demo.main;  
  
import io.mykit.canal.demo.bean.Book;  
import io.mykit.canal.demo.utils.BinlogValue;  
import io.mykit.canal.demo.utils.CanalDataParser;  
import io.mykit.canal.demo.utils.DateUtils;  
import io.mykit.canal.demo.utils.InnerBinlogEntry;  
import com.alibaba.otter.canal.client.CanalConnector;  
import com.alibaba.otter.canal.client.CanalConnectors;  
import com.alibaba.otter.canal.protocol.CanalEntry;  
import com.alibaba.otter.canal.protocol.Message;  
import org.apache.solr.client.solrj.SolrServer;  
import org.apache.solr.client.solrj.impl.HttpSolrServer;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.net.InetSocketAddress;  
import java.text.ParseException;  
import java.util.List;  
import java.util.Map;  
  
public class SyncDataBootStart {  
  
    private static Logger logger = LoggerFactory.getLogger(SyncDataBootStart.class);  
  
    public static void main(String[] args) throws Exception {  
  
        String hostname = "192.168.175.100";  
        Integer port = 11111;  
        String destination = "example";  
  
        //获取CanalServer 连接  
        CanalConnector canalConnector = CanalConnectors.newSingleConnector(new InetSocketAddress(hostname, port), destination, "", "");  
  
        //连接CanalServer  
        canalConnector.connect();  
  
        //订阅Destination  
        canalConnector.subscribe();  
  
        //轮询拉取数据  
        Integer batchSize = 5\*1024;  
        while (true){  
            Message message = canalConnector.getWithoutAck(batchSize);  
  
            long messageId = message.getId();  
            int size = message.getEntries().size();  
  
            if(messageId == -1 || size == 0){  
                try {  
                    Thread.sleep(1000);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
                }  
            }else{  
                //进行数据同步  
                //1. 解析Message对象  
                List<InnerBinlogEntry> innerBinlogEntries = CanalDataParser.convertToInnerBinlogEntry(message);  
  
                //2. 将解析后的数据信息 同步到Solr的索引库中.  
                syncDataToSolr(innerBinlogEntries);  
            }  
  
            //提交确认  
            canalConnector.ack(messageId);  
  
        }  
  
    }  
    private static void syncDataToSolr(List<InnerBinlogEntry> innerBinlogEntries) throws Exception {  
        //获取solr的连接  
        SolrServer solrServer = new HttpSolrServer("http://192.168.175.101:8080/solr");  
  
        //遍历数据集合 , 根据数据集合中的数据信息, 来决定执行增加, 修改 , 删除操作 .  
        if(innerBinlogEntries != null){  
            for (InnerBinlogEntry innerBinlogEntry : innerBinlogEntries) {  
  
                CanalEntry.EventType eventType = innerBinlogEntry.getEventType();  
  
                //如果是Insert, update , 则需要同步数据到 solr 索引库  
                if(eventType == CanalEntry.EventType.INSERT || eventType == CanalEntry.EventType.UPDATE){  
                    List<Map<String, BinlogValue>> rows = innerBinlogEntry.getRows();  
                    if(rows != null){  
                        for (Map<String, BinlogValue> row : rows) {  
                            BinlogValue id = row.get("id");  
                            BinlogValue name = row.get("name");  
                            BinlogValue author = row.get("author");  
                            BinlogValue publishtime = row.get("publishtime");  
                            BinlogValue price = row.get("price");  
                            BinlogValue publishgroup = row.get("publishgroup");  
  
                            Book book = new Book();  
                            book.setId(Integer.parseInt(id.getValue()));  
                            book.setName(name.getValue());  
                            book.setAuthor(author.getValue());  
                            book.setPrice(Double.parseDouble(price.getValue()));  
                            book.setPublishgroup(publishgroup.getValue());  
                            book.setPublishtime(DateUtils.parseDate(publishtime.getValue()));  
  
  
                            //导入数据到solr索引库  
                            solrServer.addBean(book);  
                            solrServer.commit();  
                        }  
                    }  
  
                }else if(eventType == CanalEntry.EventType.DELETE){  
                    //如果是Delete操作, 则需要删除solr索引库中的数据 .  
                    List<Map<String, BinlogValue>> rows = innerBinlogEntry.getRows();  
                    if(rows != null){  
                        for (Map<String, BinlogValue> row : rows) {  
                            BinlogValue id = row.get("id");  
  
                            //根据ID删除solr的索引库  
                            solrServer.deleteById(id.getValue());  
                            solrServer.commit();  
                        }  
                    }  
  
                }  
            }  
        }  
    }  
}

接下来，启动SyncDataBootStart类的main方法，监听Canal Server，而Canal Server监听MySQL binlog的日志变化，一旦MySQL的binlog日志发生变化，则SyncDataBootStart会立刻收到变更信息，并将变更信息解析成Book对象实时更新到Solr库中。如果在MySQL数据库中删除了数据，则也会实时删除Solr库中的数据。

*部分参考Canal官方文档：https://github.com/alibaba/canal*

<https://mp.weixin.qq.com/s?__biz=MzU0OTE4MzYzMw==&mid=2247490422&idx=4&sn=4f547f4fb4354d3313e3f9a235fbdf62&chksm=fbb29288ccc51b9ec5b724653651f831bff83a0224af239105ead54ddbcf40208337775236bc&mpshare=1&scene=1&srcid=0809mkaP0A7RtYlf64XYifqA&sharer_sharetime=1596950139272&sharer_shareid=6409037e4f38f0936cfb68aeee2b74f0&key=82003248020682fa6e45c336a7661af4ae5bd4ed78bd0f958d4bc20d119ce90b92d0996a9b0d9a346d00ffd72911ebbfe04ceae01ce58bad3d1edc31e3700325d6d378cf6ce7e835823fc6320c522a65&ascene=1&uin=NDM2MDg2NzU1&devicetype=Windows+10+x64&version=62090529&lang=zh_CN&exportkey=AcqohlJr5vGstGmbeKr5tH8%3D&pass_ticket=sjiKuh5Fgot5S95NPHxTTNxBKTX6ks%2FhqHm%2Fb2XQcAMae1fZflBdguxf%2BCnlP8sr>