chuanzhongdu1的专栏

人法地，地法天，天法道，道法自然

个人资料

chuanzhongdu1

访问： 141787次
积分： 2538
等级： BLOG 5
排名： 第9957名

原创： 103篇
转载： 88篇
译文： 2篇
评论： 17条

文章分类

apache (3)
hibernate (0)
jbpm (1)
json (3)
linux (30)
memcache (4)
osgi (0)
restlet (4)
spring (4)
其他 (34)
分布式 (10)
数据库 (37)
云 (9)
soa (4)
uml与建模 (0)
分析模式 (2)
jvm (9)
ajax (4)
架构设计 (2)
struts2 (2)
nosql (6)
jboss (2)
hadoop (6)

kafka connect

2016-05-11 10:56 146人阅读 评论(0) 收藏 举报

分类： kafka（4）

目录(?) [+]

kafka connect是一个kafka与其他系统进行数据流交换的可扩展并且高可用的工具

它可以简单定义connect将大的数据集放入kafka,比如它可以低延迟的将数据库或者应用服务器中的metrics数据放入kafka topic

导出job将kafka topic数据到另外的存储系统或查询系统或者离线系统进行批量处理

kafka connect包括以下特点

1.简单的整合其它系统，简单的发布管理

2.分布式和独立的模式 可扩展到一个大的，集中管理的服务支持整个组织或者规模下降到开发，测试和小型生产部署

3.通过rest服务可以简单管理kafka connect

4.只需要少量的信息提供给connect,kafka 会自动完成offset的管理

5.streaming/batch整合，利用kafka现有的功能，可以同时支持streaming数据与批量数据系统

Running Kafka Connect

kafka connect包括两个运行模式;standalone,distribute

算法 (3)
scala (1)
kafka (5)

文章存档
2016年05月 (1)
2016年04月 (1)
2016年03月 (3)
2015年09月 (1)
2015年06月 (5)
展开

阅读排行
jsonConfig详细使用 (3070)
oracle并行度调整 (2859)
CDH4 impala安装配置 (2353)
ycsb测试mongo笔记 (2093)
hbase数据模型 (1998)
linux下apache启用ssl (1790)
hadoop动态增加删除节 (1565)
JAVA客户端调用memca (1541)
hbase优化 (1485)
树、B-树、B+树、B*树 (1376)

评论排行
ycsb测试mongo笔记 (3)
LINUX目录详解 (2)
mule配置文件元素 (2)
json与java互换 (2)
SSH2中struts注入servic (2)
jboss4.3 需要输入用户名 (2)
restlet session问题 (1)
eclipse stp开发sca (1)
CDH4 impala安装配置 (1)
ubuntu下安装ocfs2 (1)

推荐文章
*Android RocooFix 热修复框架
* android6.0源码分析之Camera API2.0下的初始化流程分析
*Android_GestureDetector手势滑动使用
*Android MaterialList源码解析
*Android开源框架Universal-Image-Loader基本介绍及使用
*Android官方开发文档Training系列课程中文版：创建自定义View之View的创建

最新评论
ubuntu下安装ocfs2
heivy: 楼主，图片被吞了
jboss4.3 需要输入用户名密码的
yanyanwish: 很好用一下试试
mule配置文件元素
chuanzhongdu1: 不好意思，我好长时间不关注这方面了，你再问问别人吧
mule配置文件元素
再见arrogant: spring和mule分开时用spring:import引用，spring的配置文件应该怎么写呢？？
CDH4 impala安装配置

standalone发布配置简单，适合只有一个worker的工作场景，但是它没有完成分布式模式的容错机制

运行standalone使用如下命令

```
> bin/connect-standalone.sh config/connect-standalone.properties connector1.properties [connector2.properties ...]
```

第一个参数是work的配置，包括序列化类型，提交offset的时间等等，剩下的参数是connector配置文件，你可以定义一些connector的特定配置，任务将工作在相同进程的不同线程中

分布式模式将动态扩展，自动负载均衡，容错，offset自动管理，它的执行非常像standalone模式

```
bin/connect-distributed.sh config/connect-distributed.properties
```

不同的是启动类与后面的参数，参数指定的配置文件用来定义怎样分配任务，怎样存储offset,特别是如下的参数至关重要

```
group.id唯一标识，不能冲突
```

```
config.storage.topic 用于存储connect及task的配置，这个topic应该有一个partition多个replica
```

```
offset.storage.topic 用于存储offset信息，这个topic应该有多个partiton和replica
```

注意分布式环境下不能通过命令行修改配置，需要用rest服务来修改

Configuring Connectors

连接器的配置是简单的键值映射。对于standalone模式，这些被定义在一个属性文件中，并传递到命令行的连接进程。在分布式模式，他们将包括在JSON中，创建/修改连接

器。大多数配置是连接器相关的，所以它们不能在这里概述。然而，有几个通用的选项：

名称: 连接器的唯一名称。试图注册相同的名称将失败

connector.class 连接器的java类

tasks.max 可创建最大数量的任务。如果无法达到该值，该连接器创建较少的任务。

REST API

kafka connect的目的是作为一个服务运行，它支持rest管理。默认情况下，此服务运行于端口8083。以下是当前支持的：

`GET /connectors` 返回一个活动的connect列表

`POST /connectors` 创建一个新的connect；请求体是一个JSON对象包含一个名称字段和连接器配置参数

`GET /connectors/{name}` 获取有关特定连接器的信息

`GET /connectors/{name}/config` 获得特定连接器的配置参数

`PUT /connectors/{name}/config` 更新特定连接器的配置参数

`GET /connectors/{name}/tasks` 获得正在运行的一个连接器的任务的列表

`DELETE /connectors/{name}` 删除一个连接器，停止所有任务，并删除它的配置

Connector Development Guide

Core Concepts and APIs

Connectors and Tasks

在kafka与其他系统间复制数据需要创建kafka connect,他们将数复制到kafka或者从kafka复制到其他系统
连接器有两种形式：sourceconnectors将另一个系统数据导入kafka，sinkconnectors将数据导出到另一个系统
连接器不执行任何数据复制：它们的描述复制的数据，并且负责将工作分配给多个task
task分为sourcetask与sinktask

每个task从kafka复制数据,connect会保证record与schema的一致性完成任务分配，通常record与schema的映射是明显的，每一个文件对应一个流，流中的每一条记录利用schema解析并且保存对应的offset,另外一种情况是我们需要自己完成这种映射，比如数据库，表的offset不是很明确（没有自增id),一种可能的选择是利用时间（timestamp)来完成增量查询。

Streams and Records

每一个stream是包含key value对的记录的序列，key value可以是原始类型，可以支持复杂结构，除了array,object，嵌套等。数据转换是框架来完成的，record中包含stream id与offset,用于定时offset提交，帮助当处理失败时恢复避免重复处理。

Dynamic Connectors

所有的job不是静态的，它需要监听外部系统的变化，比如数据库表的增加删除，当一个新table创建时，它必须发现并且更新配置由框架来分配给该表一个task去处理，当通知发布后框架会更新对应的task.

Developing a Simple Connector

例子很简单

在standalone模式下实现 SourceConnector/SourceTask 读取文件并且发布record给SinkConnector/SinkTask 由sink写入文件

Connector Example

我们将实现SourceConnector，SinkConnector实现与它非常类似，它包括两个私有字段存放配置信息（读取的文件名与topic名称）

```
public class FileStreamSourceConnector extends SourceConnector {
    private String filename;
    private String topic;
```

getTaskClass()方法定义实现执行处理的task

```
@Override
public Class getTaskClass() {
    return FileStreamSourceTask.class;
}
```

下面定义FileStreamSourceTask，它包括两个生命周期方法start,stop

```
@Override
public void start(Map<String, String> props) {
    // The complete version includes error handling as well.
    filename = props.get(FILE_CONFIG);
    topic = props.get(TOPIC_CONFIG);
}

@Override
public void stop() {
    // Nothing to do since no background monitoring is required.
}
```

最后是真正核心的方法getTaskConfigs（）在这里我们仅处理一个文件，所以我们虽然定义了max task（在配置文件里）但是只会返回一个包含一条entry的list

```
@Override
public List<Map<String, String>> getTaskConfigs(int maxTasks) {
    ArrayList>Map<String, String>> configs = new ArrayList<>();
```

```
    // Only one input stream makes sense.
    Map<String, String> config = new Map<>();
    if (filename != null)
        config.put(FILE_CONFIG, filename);
    config.put(TOPIC_CONFIG, topic);
    configs.add(config);
    return configs;
}
```

即使有多个任务，这种方法的执行通常很简单。它只是要确定输入任务的数量，这可能需要拉取数据从远程服务，然后分摊。请注意，这个简单的例子不包括动态输入。在下一节中看到讨论如何触发任务的配置更新。

Task Example - Source Task

实现task，我们使用伪代码描述核心代码

```
public class FileStreamSourceTask extends SourceTask<Object, Object> {
    String filename;
    InputStream stream;
    String topic;

    public void start(Map<String, String> props) {
        filename = props.get(FileStreamSourceConnector.FILE_CONFIG);
        stream = openOrThrowError(filename);
        topic = props.get(FileStreamSourceConnector.TOPIC_CONFIG);
    }

    @Override
    public synchronized void stop() {
        stream.close()
    }
```

start方法读取之前的offset,并且处理新的数据，stop方法停止stream,下面实现poll方法

```
@Override
public List<SourceRecord> poll() throws InterruptedException {
    try {
        ArrayList<SourceRecord> records = new ArrayList<>();
        while (streamValid(stream) && records.isEmpty()) {
            LineAndOffset line = readToNextLine(stream);
            if (line != null) {
                Map sourcePartition = Collections.singletonMap("filename", filename);
                Map sourceOffset = Collections.singletonMap("position", streamOffset);
                records.add(new SourceRecord(sourcePartition, sourceOffset, topic, Schema.STRING_SCHEMA,
```

```
line));
            } else {
                Thread.sleep(1);
            }
        }
        return records;
    } catch (IOException e) {
        // Underlying stream was killed, probably as a result of calling stop. Allow to return
        // null, and driving thread will handle any shutdown if necessary.
    }
    return null;
}
```

该方法重复执行读取操作，跟踪file offset,并且利用上述信息创建SourceRecord，它需要四个字段：source partition,source offset,topic name,output value(包括value及value的schema)

Sink Tasks

之前描述了sourcetask实现，sinktask与它完全不同，因为前者是拉取数据，后者是推送数据

```
public abstract class SinkTask implements Task {
public void initialize(SinkTaskContext context) { ... }

public abstract void put(Collection<SinkRecord> records);

public abstract void flush(Map<TopicPartition, Long> offsets);
```

put方法是最重要的方法，接收sinkrecords，执行任何需要的转换，并将其存储在目标系统。此方法不需要确保数据已被完全写入目标系统，然后返回。事实上首先放入缓冲，因此，批量数据可以被一次发送，减少对下游存储的压力。sourcerecords中保存的信息与sourcesink中的相同。flush提交offset，它接受任务从故障中恢复，没有数据丢失。该方法将数据推送至目标系统，并且block直到写入已被确认。的offsets参数通常可以忽略不计，但在某些情况保存偏移信息到目标系统确保一次交货。例如，一个HDFS连接器可以确保flush()操作自动提交数据和偏移到HDFS中的位置。

Resuming from Previous Offsets

kafka connect是为了bulk 数据拷贝工作，它拷贝整个db而不是拷贝某个表，这样会使用connnect的input或者output随时改变，source connector需要监听source系统的改变，当改变时通知框架（通过ConnectorContext对象）
举例

```
if (inputsChanged())
    this.context.requestTaskReconfiguration();
```

当接收到通知框架会即时的更新配置，并且在更新前确保优雅完成当前任务

如果一个额外的线程来执行此监控，该线程必须存在于连接器中。该线程不会影响connector。然而，其他变化也会影响task，最常见的是输入流失败在输入系统中，例如如果一个表被从数据库中删除。这时连接器需要进行更改，任务将需要处理这种异常。sinkconnectors只能处理流的加入，可以分配新的数据到task（例如，一个新的数据库表）。框架会处理任何kafka输入的改变，例如当组输入topic的变化因为一个正则表达式的订阅。sinktasks应该期待新的输入流，可能需要在下游系统创造新的资源，如数据库中的一个新的表。在这些情况下，可能会出现输入流之间的冲突（同时创建新资源），其他时候，一般不需要特殊的代码处理一系列动态流

Dynamic Input/Output Streams

FileStream连接器是很好的例子，因为他们很简单的，每一行是一个字符串。实际连接器都需要具有更复杂的数据格式。要创建更复杂的数据，你需要使用kafka connector数据接口:Schema,Struct

```
Schema schema = SchemaBuilder.struct().name(NAME)
                .field("name", Schema.STRING_SCHEMA)
                .field("age", Schema.INT_SCHEMA)
                .field("admin", new SchemaBuilder.boolean().defaultValue(false).build())
                .build();

Struct struct = new Struct(schema)
                    .put("name", "Barbara Liskov")
                    .put("age", 75)
                    .build();
```

如果上游数据与schema数据格式不一致应该在sinktask中抛出异常

上一篇 kafka性能与资源考虑

我的同类文章

kafka（4）

- kafka性能与资源考虑 2016-04-13 阅读 67
- kafka0.9 producer与consum.. 2016-03-11 阅读 130
- kafka0.9 topic level参数 2016-03-10 阅读 473
- kafka 0.9 broker 参数 2016-03-10 阅读 212