無料電子ブック

学習

apache-flink

Free unaffiliated eBook created from
Stack Overflow contributors.

#apache-flink

1: apache-flinkをいめる

このセクションでは、apache-flinkのと、なぜがそれをいたいのかをします。

また、apache-flinkのきなテーマについてもし、するトピックにリンクするがあります。 apache-flinkのドキュメンテーションはしいので、これらのトピックのバージョンをするがあるかもしれません。

Examples

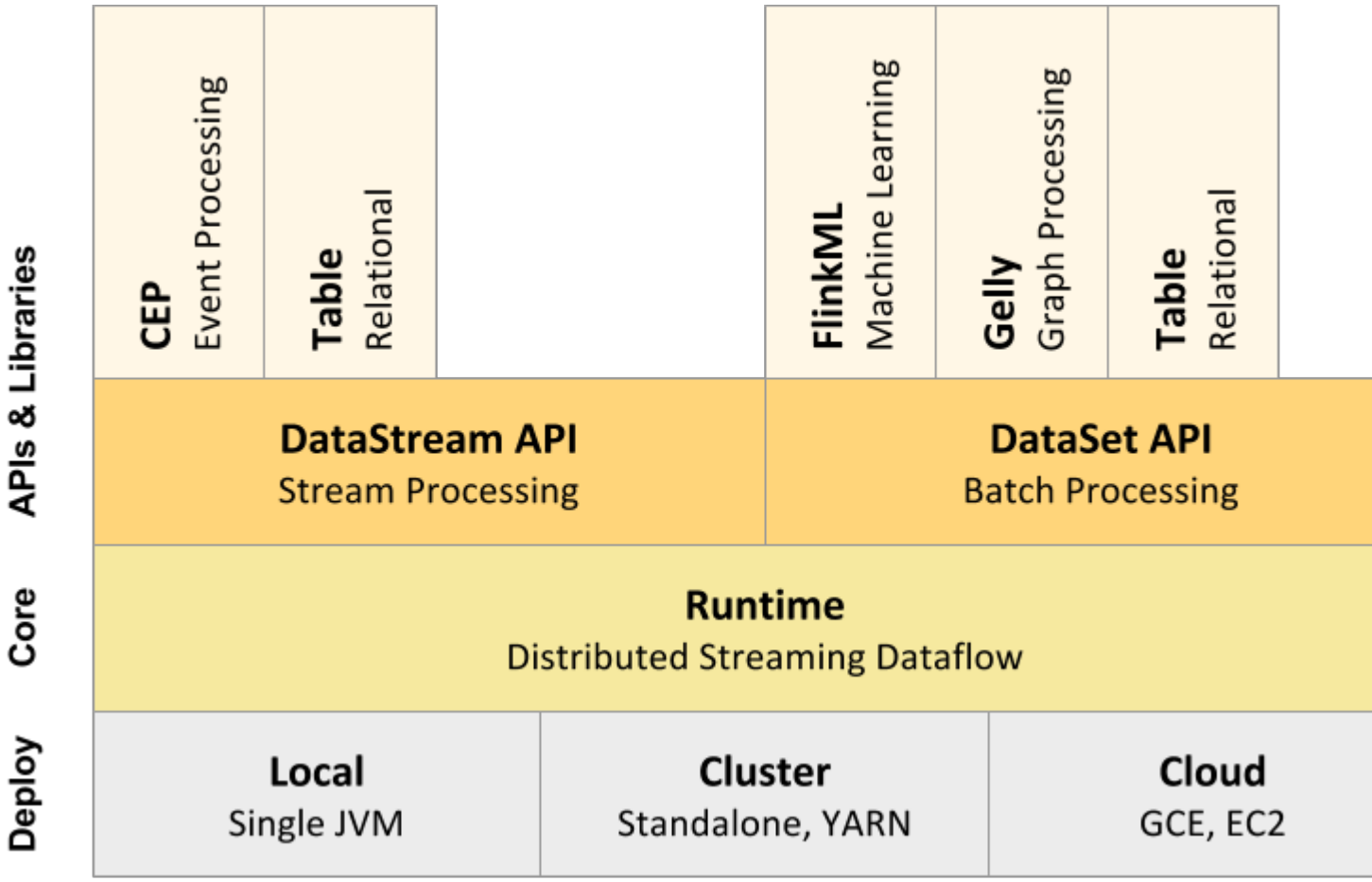
と

Flinkとはですか

Apache HadoopやApache Sparkと、Apache FlinkはBig Data Analyticsのためのコミュニティのオープンソースフレームワークです。 JavaでかれたFlinkにはScala、Java、PythonのAPIがあり、バッチおよびリアルタイムストリーミングがです。

- Linux、Mac OS X、CygwinなどのUNIXの。
- Java 6.X。
- [オプション] Maven 3.0.4。

スタック



Apache Flinkはデータシステムであり、Hadoopの**MapReduce**コンポーネントのです。 MapReduceのにするのではなく、 のランタイムがしています 。このように、Hadoopのエコシステムとはにしてすることができます。

`ExecutionEnvironment`は、プログラムがされるコンテキストです。ニーズにじて、さまざまなをできます。

1. *JVM* FlinkはのJavaマシンでできるため、ユーザーはIDEからFlinkプログラムをテストおよびデバッグできます。このをする、なのはしいmavenだけです。

2. ローカル のFlinkインスタンスIDEからではなくでプログラムをできるようにするには、マシンにFlinkをインストールするがあります。 ローカルセットアップをしてください。

3. クラスタ にしたでFlinkをするには、スタンドアロンまたはクラスターがです。については、クラスタセットアップページまたはこのスライドショーをしてください。 mportant__アーティファクトの`2.11`はスカラのバージョンです。システムのアーティファクトとするようにしてください。

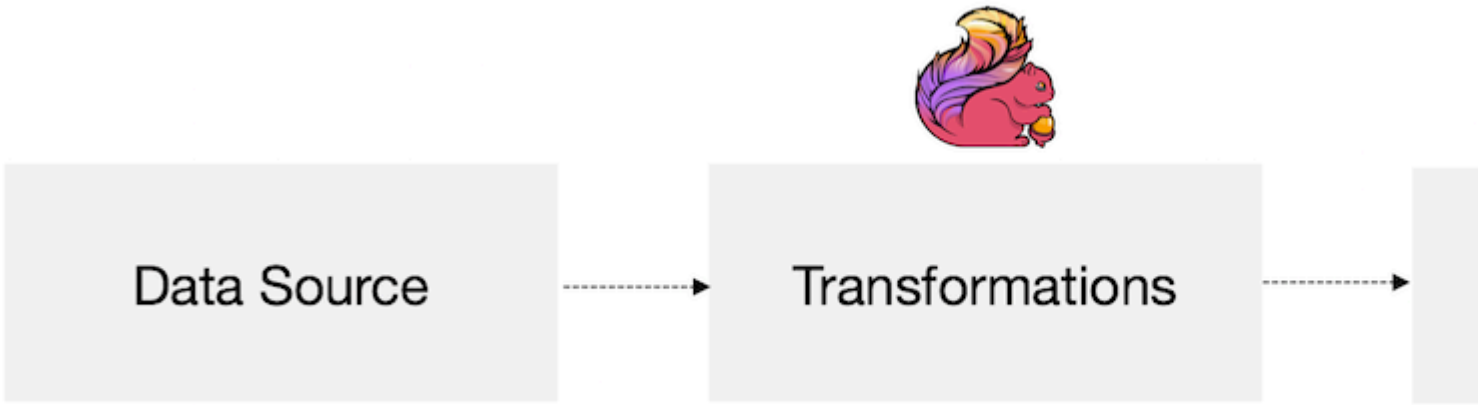
API

Flinkは、ストリームまたはバッチのいずれかにできます。らは3つのAPIをしています

- **DataStream API** ストリーム、つまり、のデータフローでのフィルタ、、。
- **DataSet API** バッチ、つまりデータセットの。
- テーブル**API** バッチアプリケーションとストリーミングアプリケーションのにめむことができる、SQLライクなSparkのデータフレームなど。

ビルディングブロック

もなレベルでは、Flinkはソース、、シンクからされています。



もなレベルでは、Flinkプログラムはのようにされています。

- データソース Flinkがするデータ
- Flinkがデータをするときのステップ
- データシンク Flinkがにデータをする

ソースとシンクには、ローカル/ HDFSファイル、データベース、メッセージキューなどがあります。すでになサードパーティのコネクタがあります。また、のコネクタもにできます。

ローカルランタイムセットアップ

0. Java 6で、 `JAVA_HOME`がされていることをしてください。

1. ここでのflinkバイナリをダウンロードしてください

   ```
   wget flink-XXXX.tar.gz
   ```

   Hadoopでするがないは、hadoop 1バージョンをしてください。また、ダウンロードしたスカラー·バージョンにしてください。そのため、プログラムにしいMavenをすることができます。

2. フリンクをする

   ```
   tar xzvf flink-XXXX.tar.gz
   ```

```
./flink/bin/start-local.sh
```

Flinkはにローカルでするようにされています。 Flinkがされていることをするには、ログを`flink/log/`で`flink/log/`か、 `http://localhost:8081`されているflink jobManagerのインターフェースをきます。

3. フリンク

```
./flink/bin/stop-local.sh
```

Flink

IDEからFlinkプログラムをするにはEclipseまたはIntellij IDEAをすることができます、の2つのがです`flink-java` / `flink-scala`および`flink-clients` 20162。これらのJARSは、MavenとSBTスカラーをしているをしてできます。

- **Maven**

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-java</artifactId>
    <version>1.1.4</version>
</dependency>

<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-clients_2.11</artifactId>
    <version>1.1.4</version>
</dependency>
```

- **SBT**= ""

```
version := "1.0"

scalaVersion := "2.11.8"

libraryDependencies ++= Seq(
  "org.apache.flink" %% "flink-scala" % "1.2.0",
  "org.apache.flink" %% "flink-clients" % "1.2.0"
)
```

アーティファクトの`2.11`はスカラのバージョンです。あなたのシステムにあるものとずさせてください。

WordCount - テーブルAPI

このは*WordCount*とじですが、Table APIをしています。とのについては、 *WordCount*をしてください。

Maven

テーブルAPIをするには、 `flink-table` をmavenとしてします。

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>1.1.4</version>
</dependency>
```

コード

```
public class WordCountTable{

    public static void main( String[] args ) throws Exception{

        // set up the execution environment
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
        final BatchTableEnvironment tableEnv = TableEnvironment.getTableEnvironment( env );

        // get input data
        DataSource<String> source = env.fromElements(
                "To be, or not to be,--that is the question:--",
                "Whether 'tis nobler in the mind to suffer",
                "The slings and arrows of outrageous fortune",
                "Or to take arms against a sea of troubles"
        );

        // split the sentences into words
        FlatMapOperator<String, String> dataset = source
                .flatMap( ( String value, Collector<String> out ) -> {
                    for( String token : value.toLowerCase().split( "\\W+" ) ){
                        if( token.length() > 0 ){
                            out.collect( token );
                        }
                    }
                } )
                // with lambdas, we need to tell flink what type to expect
                .returns( String.class );

        // create a table named "words" from the dataset
        tableEnv.registerDataSet( "words", dataset, "word" );

        // word count using an sql query
        Table results = tableEnv.sql( "select word, count(*) from words group by word" );
        tableEnv.toDataSet( results, Row.class ).print();
    }
}
```

Java <8をするバージョンでは、ラムダをクラスにきえます。

```
FlatMapOperator<String, String> dataset = source.flatMap( new FlatMapFunction<String,
String>(){
        @Override
        public void flatMap( String value, Collector<String> out ) throws Exception{
```

```
            for( String token : value.toLowerCase().split( "\\W+" ) ){
                if( token.length() > 0 ){
                    out.collect( token );
                }
            }
        }
    } );
```

WordCount

Maven

`flink-java`と`flink-client`をし`flink-client` *JVM*のを。

コード

```
public class WordCount{

    public static void main( String[] args ) throws Exception{

        // set up the execution environment
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

        // input data
        // you can also use env.readTextFile(...) to get words
        DataSet<String> text = env.fromElements(
                "To be, or not to be,--that is the question:--",
                "Whether 'tis nobler in the mind to suffer",
                "The slings and arrows of outrageous fortune",
                "Or to take arms against a sea of troubles,"
        );

        DataSet<Tuple2<String, Integer>> counts =
                // split up the lines in pairs (2-tuples) containing: (word,1)
                text.flatMap( new LineSplitter() )
                        // group by the tuple field "0" and sum up tuple field "1"
                        .groupBy( 0 )
                        .aggregate( Aggregations.SUM, 1 );

        // emit result
        counts.print();
    }
}
```

`LineSplitter.java`

```
public class LineSplitter implements FlatMapFunction<String, Tuple2<String, Integer>>{

    public void flatMap( String value, Collector<Tuple2<String, Integer>> out ){
        // normalize and split the line into words
        String[] tokens = value.toLowerCase().split( "\\W+" );

        // emit the pairs
        for( String token : tokens ){
            if( token.length() > 0 ){
```

```
                out.collect( new Tuple2<String, Integer>( token, 1 ) );
            }
        }
    }
}
```

Java 8をするは、 `.flatmap(new LineSplitter())` をラムダできえることができます。

```
DataSet<Tuple2<String, Integer>> counts = text
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap( ( String value, Collector<Tuple2<String, Integer>> out ) -> {
        // normalize and split the line into words
        String[] tokens = value.toLowerCase().split( "\\W+" );

        // emit the pairs
        for( String token : tokens ){
            if( token.length() > 0 ){
                out.collect( new Tuple2<>( token, 1 ) );
            }
        }
    } )
    // group by the tuple field "0" and sum up tuple field "1"
    .groupBy( 0 )
    .aggregate( Aggregations.SUM, 1 );
```

IDEから **IDE**でにします。 FlinkはJVMのにをします。

flinkコマンドラインから スタンドアロンロ―カルをしてプログラムをするには、のようにします。

1. flinkがされていることをします `flink/bin/start-local.sh`。
2. jarファイル `maven package` をします。
3. `flink`コマンドラインツ―ルあなたのFlinkインスト―ルの`bin`フォルダをしてプログラムをします

   ```
   flink run -c your.package.WordCount target/your-jar.jar
   ```

   `-c`オプションをすると、するクラスをできます。 jarがである/メインクラスをするはありません。

```
(a,1)
(against,1)
(and,1)
(arms,1)
(arrows,1)
(be,2)
(fortune,1)
(in,1)
(is,1)
(mind,1)
(nobler,1)
```

```
(not,1)
(of,2)
(or,2)
(outrageous,1)
(question,1)
(sea,1)
(slings,1)
(suffer,1)
(take,1)
(that,1)
(the,3)
(tis,1)
(to,4)
(troubles,1)
(whether,1)
```

WordCount - ストリーミングAPI

このは*WordCount*とじですが、Table APIをしています。とのについては、 *WordCount*をしてください。

Maven

Streaming APIをするには、 `flink-streaming`をmavenとしてします。

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java_2.11</artifactId>
    <version>1.1.4</version>
</dependency>
```

コード

```
public class WordCountStreaming{

    public static void main( String[] args ) throws Exception{

        // set up the execution environment
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        // get input data
        DataStreamSource<String> source = env.fromElements(
                "To be, or not to be,--that is the question:--",
                "Whether 'tis nobler in the mind to suffer",
                "The slings and arrows of outrageous fortune",
                "Or to take arms against a sea of troubles"
        );

        source
                // split up the lines in pairs (2-tuples) containing: (word,1)
                .flatMap( ( String value, Collector<Tuple2<String, Integer>> out ) -> {
                    // emit the pairs
                    for( String token : value.toLowerCase().split( "\\W+" ) ){
                        if( token.length() > 0 ){
```

```
                            out.collect( new Tuple2<>( token, 1 ) );
                        }
                    }
                } )
                // due to type erasure, we need to specify the return type
                .returns( TupleTypeInfo.getBasicTupleTypeInfo( String.class, Integer.class ) )
                // group by the tuple field "0"
                .keyBy( 0 )
                // sum up tuple on field "1"
                .sum( 1 )
                // print the result
                .print();

        // start the job
        env.execute();
    }
}
```

オンラインでapache-flinkをいめるをむ https://riptutorial.com/ja/apache-flink/topic/5798/apache-flinkをいめる

2: カスタムdeシリアライズスキーマをする

き

スキーマはいくつかのコネクタKafka、RabbitMQによってメッセージをJavaオブジェクトにするためにされ、そのもあります。

Examples

カスタムスキーマの

カスタムスキーマをするには、 `SerializationSchema`または`DeserializationSchema`インタフェースのいずれかをするだけです。

```
public class MyMessageSchema implements DeserializationSchema<MyMessage>,
SerializationSchema<MyMessage> {

    @Override
    public MyMessage deserialize(byte[] bytes) throws IOException {
        return MyMessage.fromString(new String(bytes));
    }

    @Override
    public byte[] serialize(MyMessage myMessage) {
        return myMessage.toString().getBytes();
    }

    @Override
    public TypeInformation<MyMessage> getProducedType() {
        return TypeExtractor.getForClass(MyMessage.class);
    }

    // Method to decide whether the element signals the end of the stream.
    // If true is returned the element won't be emitted.
    @Override
    public boolean isEndOfStream(MyMessage myMessage) {
        return false;
    }
}
```

`MyMessage`クラスは、のようにされています。

```
public class MyMessage{

    public int id;
    public String payload;
    public Date timestamp;

    public MyMessage(){}

    public static MyMessage fromString( String s ){
```

```
        String[] tokens = s.split( "," );
        if(tokens.length != 3) throw new RuntimeException( "Invalid record: " + s );

        try{
            MyMessage message = new MyMessage();
            message.id = Integer.parseInt(tokens[0]);
            message.payload = tokens[1];
            message.timestamp = new Date( Long.parseLong(tokens[0]));
            return message;
        }catch(NumberFormatException e){
            throw new RuntimeException("Invalid record: " + s);
        }
    }

    public String toString(){
        return String.format("%d,%s,%dl", id, payload, timestamp.getTime());
    }
}
```

オンラインでカスタムdeシリアライズスキーマをするをむ https://riptutorial.com/ja/apache-flink/topic/9004/カスタム-de-シリアライズスキーマをする

3: カフカのデータをする

Examples

KafkaConsumerの

`FlinkKafkaConsumer`は、1つまたはのカフカトピックのデータをします。

バージョン

するはカフカにします。

- `FlinkKafkaConsumer08` Kafkaのい`SimpleConsumer` APIをします。オフセットはFlinkによってされ、いにねられます。
- `FlinkKafkaConsumer09` オフセットとリバランスをにするKafkaのしいコンシューマーAPIをします。
- `FlinkKafkaProducer010` このコネクタは、とののタイムスタンプをつKafkaメッセージをサポートしていますウィンドウにです。

バイナリはFlinkコアのではないので、それらをインポートするがあります

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-connector-kafka-0.${kafka.version}_2.10</artifactId>
  <version>RELEASE</version>
</dependency>
```

コンストラクタには3つのがあります。

- むべき1つまたはのトピック
- メッセージを/デコードするをFlinkにえるデシリアライゼーションスキーマ
- kafkaコンシューマプロパティそれらは「の」カフカとじです。はのとおりです。
    - `bootstrap.servers` ipportのでカンマりのKafkaブローカのリスト。バージョン8では、わりに`zookeeper.connect` いサーバのリストをしてください
    - `group.id` グループのIDはkafkaのドキュメントを

Javaの

```
Properties properties = new Properties();
properties.put("group.id", "flink-kafka-example");
properties.put("bootstrap.servers", "localhost:9092");

DataStream<String> inputStream = env.addSource(
        new FlinkKafkaConsumer09<>(
            kafkaInputTopic, new SimpleStringSchema(), properties));
```

スカラーで

```
val properties = new Properties();
properties.setProperty("bootstrap.servers", "localhost:9092");
properties.setProperty("group.id", "test");

inputStream = env.addSource(
        new FlinkKafkaConsumer08[String](
            "topic", new SimpleStringSchema(), properties))
```

に、 `enable.auto.commit=false`プロパティの`enable.auto.commit=false`および`auto.offset.reset=earliest`をして、pogramをするたびにじデータをすることができます。

フォールトトレランス

ドキュメントでされているように、

> Flinkのチェックポイントをにすると、Flink Kafka Consumerはトピックからレコードをし、ののとともに、すべてのKafkaオフセットをにチェックポイントします。ジョブがした、Flinkはストリーミングプログラムをのチェックポイントのにし、チェックポイントにされているオフセットからしてカフカのレコードをします。
>
> したがって、チェックポイントのは、がしたに、プログラムがどれだけっていかなければならないかをします。

フォールトトレランスのKafka Consumersをするには、 `enableCheckpointing`メソッドをしてでチェックポイントをにするがあります。

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.enableCheckpointing(5000); // checkpoint every 5 seconds
```

みみデシリアライズスキーマ

SimpleStringSchema `SimpleStringSchema`は、メッセージをとしてシリアルします。メッセージにキーがある、はされます。

```
new FlinkKafkaConsumer09<>(kafkaInputTopic, new SimpleStringSchema(), prop);
```

JSONDeserializationSchema

`JSONDeserializationSchema` *jackson*をしてjsonのメッセージをシリアルし、`com.fasterxml.jackson.databind.node.ObjectNode`オブジェクトのストリームを`com.fasterxml.jackson.databind.node.ObjectNode`ます。フィールドにアクセスするには、`.get("property")`メソッドをします。もう、キーはされます。

```
new FlinkKafkaConsumer09<>(kafkaInputTopic, new JSONDeserializationSchema(), prop);
```

JSONKeyValueDeserializationSchema

`JSONKeyValueDeserializationSchema`はのものとにていますが、jsonエンコードされたキーとをつメッセージをします。

```
boolean fetchMetadata = true;
new FlinkKafkaConsumer09<>(kafkaInputTopic, new
JSONKeyValueDeserializationSchema(fetchMetadata), properties);
```

される`ObjectNode`には、のフィールドがまれます。

- `key` `key`すべてのフィールド
- `value` すべてのメッセージフィールド
- オプション `metadata` メッセージの`offset` 、 `partition` 、 `topic`をしますメタデータをフェッチするためにコンストラクタに`true`をし`true`。

えば

```
kafka-console-producer --broker-list localhost:9092 --topic json-topic \
    --property parse.key=true \
    --property key.separator=|
{"keyField1": 1, "keyField2": 2} | {"valueField1": 1, "valueField2" : {"foo": "bar"}}
^C
```

デコードされる

```
{
    "key":{"keyField1":1,"keyField2":2},
    "value":{"valueField1":1,"valueField2":{"foo":"bar"}},
    "metadata":{
        "offset":43,
        "topic":"json-topic",
        "partition":0
    }
}
```

カフカパーティションと**Flink**の

kafkaでは、じコンシューマグループのコンシューマに1つのパーティションがりてられます。2つのコンシューマがじパーティションからすることはであることにしてください。フリンクのは、フリンクにするデフォルトは1。

のあるケースは3つあります。

1. **kafka partitions == flink parallelism** このケースは、コンシューマが1つのパーティションをするのでです。パーティションでメッセージのバランスがれている、はFlinkオペレータににされます。

2. **kafka**パーティション**<flink parallelism** いくつかのflinkインスタンスはメッセージをけりま

せん。これをけるには、 のにストリームで`rebalance`をびすがあり、これによりデータがパーティションされます。

```
inputStream = env.addSource(new FlinkKafkaConsumer10("topic", new SimpleStringSchema(),
properties));

inputStream
    .rebalance()
    .map(s -> "message" + s)
    .print();
```

3. **kafka partitions> flink parallelism** この、いくつかのインスタンスはのパーティションをします。もう、 `rebalance`をして、ワーカーでメッセージをにさせることができます。

オンラインでカフカのデータをするをむ https://riptutorial.com/ja/apache-flink/topic/9003/カフカのデータをする

4: セーブポイントとされたチェックポイント

き

セーブポイントは、な、キャンセルまたはコードのにステートフルなフリンクプログラムをできるように、にされたチェックポイントです。 Flink 1.2およびされたチェックポイントののに、セーブポイントをにトリガするがありました。

Examples

セーブポイントとメモ

セーブポイントには、aすべてのデータソースの、bオペレータのの2つのものがされます。セーブポイントはくのサーカスでちます

- わずかなアプリケーションコードの
- Flinkの
- の
- ...

バージョン1.3 のバージョンでも

- セーブポイントをにするには、チェックポイントをにするがあります 。にチェックポイントをにすることをれた

  ```
  StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
  env.enableCheckpointing(checkpointInterval);
  ```

  あなたは

  ```
  java.lang.IllegalStateException: Checkpointing disabled. You can enable it via the
  execution environment of your job
  ```

- ウィンドウをするは、なをるためにイベントまたはをすることがです。
- プログラムをアップグレードしてセーブポイントをできるようにするには、 **uid**をするがあります 。これは、デフォルトでFlinkがコードのにのUIDをするためです。
- されたは、のタスクのIDによってされます。のチェーンされたタスクにIDをでりてることはできません。チェーン[a - > b - > c]のaのみがIDをでりてることができますが、bまたはcはりてることはできません。このをするには、タスクチェーンをですることができます。IDりてにると、のによってIDもされますのポイントを。

は、FAQをしてください 。

セーブポイント

はファイルである`flink/conf/flink-conf.yaml` マックOSXので、それはある
`/usr/local/Cellar/apache-flink/1.1.3/libexec/conf/flink-conf.yaml`。

Flink <1.2 はチェックポイントのとによくていますなトピック。のいは、Flinkのシャットダウンにセーブポイントをするがあるため、メモリのセーブポイントバックエンドをすることはがないことです。

```
# Supported backends: filesystem, <class-name-of-factory>
savepoints.state.backend: filesystem
```

```
# Use "hdfs://" for HDFS setups, "file://" for UNIX/POSIX-compliant file systems,
# (or any local file system under Windows), or "S3://" for S3 file system.
# Note: must be accessible from the JobManager and all TaskManagers !
savepoints.state.backend.fs.checkpointdir: file:///tmp/flink-backend/savepoints
```

バックエンドをしない、デフォルトのバックエンドは*jobmanager*です。つまり、セーブポイントはクラスタのシャットダウンにえます。これはデバッグにのみです。

Flink 1.2+ このjiraチケットでされているように、セーブポイントをジョブマネージャのメモリにすることはほとんどがありません。 Flink 1.2、セーブポイントはにファイルにされます。のはのようにきえられました。

```
# Default savepoint target directory
state.savepoints.dir: hdfs:///flink/savepoints
```

ジョブIDの

セーブポイントをトリガーするには、アプリケーションのジョブIDだけがです。ジョブIDは、ジョブをするときにコマンドラインでされるか、またはで`flink list`をしてできます。

```
flink list
Retrieving JobManager.
Using address localhost/127.0.0.1:6123 to connect to JobManager.
------------------ Running/Restarting Jobs -------------------
17.03.2017 11:44:03 : 196b8ce6788d0554f524ba747c4ea54f : CheckpointExample (RUNNING)
--------------------------------------------------------------
No scheduled jobs.
```

セーブポイントのトリガー

セーブポイントをトリガーするには、 `flink savepoint <jobID>`します。

```
flink savepoint 196b8ce6788d0554f524ba747c4ea54f
Retrieving JobManager.
Using address /127.0.0.1:6123 to connect to JobManager.
Triggering savepoint for job 196b8ce6788d0554f524ba747c4ea54f.
Waiting for response...
```

```
Savepoint completed. Path: file:/tmp/flink-backend/savepoints/savepoint-a40111f915fc
You can resume your program from this savepoint with the run command.
```

2のとしてタ―ゲットディレクトリをすることもできます。これは、 `flink/bin/flink-conf.yaml`されているデフォルトのディレクトリをきします。

Flink 1.2では、 `-s`オプションをしてジョブをキャンセルし、セ―ブポイントをにうこともできます。

```
flink cancel -s 196b8ce6788d0554f524ba747c4ea54f # use default savepoints dir
flink cancel -s hdfs:///savepoints 196b8ce6788d0554f524ba747c4ea54f # specify target dir
```

セ―ブポイントをすることはですが、をしないでください

セ―ブポイントからの

のセ―ブポイントからするには、 `flink run`コマンドの`-s [savepoint-dir]`オプションをします。

```
flink run -s /tmp/flink-backend/savepoints/savepoint-a40111f915fc app.jar
```

UIDの

コ―ドにセ―ブポイントからできるようにするには、しいコ―ドがにじUIDをするようにするがあります。でUIDをりてるには、のに`.uid(<name>)`びします。

```
env
    .addSource(source)
    .uid(className + "-KafkaSource01")
    .rebalance()
    .keyBy((node) -> node.get("key").asInt())
    .flatMap(new StatefulMapper())
    .uid(className + "-StatefulMapper01")
    .print();
```

されたチェックポイントFlink 1.2

1.2よりのバ―ジョンでは、ジョブの/りし/なのにチェックポイントをするためののは、でトリガされるセ―ブポイントによるものでした。バ―ジョン1.2ではなチェックポイントがされました。

なチェックポイントは、のをき、のなチェックポイントとによくたをします。

1. メタデ―タはストレ―ジセ―ブポイントなどにされます。
2. しているジョブがにした、それらはされません。さらに、ジョブがキャンセルされたときにされないようにすることもできます。

したがって、セ―ブポイントとによくています。には、セ―ブポイントはちょうどよりくのをつされたチェックポイントです。

な では、Flinkのチェックポイントコーディネーターはににしたチェックポイントのみをします。つまり、しいチェックポイントがすると、にしたチェックポイントはされます。これはされたチェックポイントにもされます。

[された]チェックポイントにするメタデータがされているは、 `flink-conf.yaml` されますコードできすることはできません。

```
# path to the externalized checkpoints
state.checkpoints.dir: file:///tmp/flink-backend/ext-checkpoints
```

このディレクトリには、チェックポイントのになチェックポイントメタデータのみがまれています。のチェックポイントファイルは、されたディレクトリ `state.bachend.fs.checkpointdir` プロパティにまだされています。

ストリーミングの `getCheckpointConfig()` メソッドをして、コードでチェックポイントをににするがあります。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
// enable regular checkpoints
env.enableCheckpointing(5000); // every 5 sec.
// enable externalized checkpoints
env.getCheckpointConfig()

.enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpointCleanup.RETAIN_ON_CANCELLATION);
```

な `ExternalizedCheckpointCleanup` モードはのとおりです。

- `RETAIN_ON_CANCELLATION` のチェックポイントとそのメタデータはジョブキャンセルにされます。そののはあなたのです。
- `DELETE_ON_CANCELLATION` キャンセルにのチェックポイントがされます。つまり、アプリケーションがしたにのみできます。

されたチェックポイントからするには、セーブポイントをします。えば

```
flink run -s /tmp/flink-backend/ext-checkpoints/savepoint-02d0cf7e02ea app.jar
```

オンラインでセーブポイントとされたチェックポイントをむ https://riptutorial.com/ja/apache-flink/topic/9466/セーブポイントとされたチェックポイント

5: チェックポイント

き

Flink 1.2でテストみ

Flinkのすべての、ソース、はステートフルであるがあります。チェックポイントにより、Flinkはストリームのとをし、アプリケーションにのないとじセマンティクスをえることができます。 フォールトトレランスとにのがされているのは、けです。

をするには、 このをおみください。

チェックポイントは、クラスタでがしたたとえば、タスクマネージャがしたなどにのみです。そのものがしたりキャンセルされたりしても、それはされません。

/りしにステートフルなジョブをできるようにするには、 セーブポイントまたはチェックポイント **flink 1.2**をてください 。

Examples

とセットアップ

チェックポイントは2つのステップでわれます。まず、 バックエンドをするがあります。に、アプリケーションごとにチェックポイントのとモードをできます。

バックエンド

なバックエンド

チェックポイントがされるは、されたバックエンドによってなります。

- `MemoryStateBackend` メモリので、JobManagerの/ ZooKeeperのメモリにバックアップします。のデフォルトで5 MB、カフカオフセットのなどまたはテストとローカルデバッグにのみするがあります。
- `FsStateBackend` はTaskManagersのメモリにされ、スナップショットつまりチェックポイントはファイルシステムHDFS、DS3、ローカルファイルシステムなどにされます。このは、なやいウィンドウ、およびのにちます。
- `RocksDBStateBackend` TaskManagerのデータディレクトリにされているデフォルトで RocksDBデータベースののデータをします。チェックポイントがされると、RocksDBデータベースがファイルにきまれますのように。 FsStateBackendとすると、よりきなディスクスペースとタスクマネージャメモリのサイズのみでされますがになりますが、スループット

はくなりますデータはずしもメモリにされず、ディスクからロードされなければなりません。

バックエンドがであれ、メタデータチェックポイントの、ローカライゼーションなどはにジョブマネージャメモリにされ、チェックポイントはアプリケーションの/キャンセルもされません。

バックエンドの

プログラムの`main`メソッドでバックエンドをするには、のようにします。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setStateBackend(new FsStateBackend("hdfs://namenode:40010/flink/checkpoints"));
```

または、デフォルトのバックエンドを`flink/conf/flink-conf.yaml`します。

```
# Supported backends:
#  - jobmanager (MemoryStateBackend),
#  - filesystem (FsStateBackend),
#  - rocksdb (RocksDBStateBackend),
#  - <class-name-of-factory>
state.backend: filesystem

# Directory for storing checkpoints in a Flink-supported filesystem
# Note: State backend must be accessible from the JobManager and all TaskManagers.
# Use "hdfs://" for HDFS setups, "file://" for UNIX/POSIX-compliant file systems,
# "S3://" for S3 file system.
state.backend.fs.checkpointdir: file:///tmp/flink-backend/checkpoints
```

チェックポイントの

すべてのアプリケーションでにチェックポイントをにするがあります。

```
long checkpointInterval = 5000; // every 5 seconds

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.enableCheckpointing(checkpointInterval);
```

オプションでチェックポイントモードをできます。そうでないは、デフォルトはだけです。

```
env.enableCheckpointing(checkpointInterval, CheckpointingMode.AT_LEAST_ONCE);
```

チェックポイントモードは、がしたにシステムがどのようなをするかをします。チェックポイントがになると、のがりされるようにデータストリームがされます。`EXACTLY_ONCE`では、オペレータ/ファンクションがレコードを「に1」と`EXACTLY_ONCE`ようにリカバリがするように、システムはチェックポイントをします。`AT_LEAST_ONCE`と、はにしたチェックポイントがにされます。

チェックポイントのテスト

コード

ここでは、 `Integer`のステートフルマッパーをするなフリンクアプリケーションをします。 `checkpointEnable` 、 `checkpointInterval` 、 および`checkpointMode`をしてそのをできます。

```
public class CheckpointExample {

    private static Logger LOG = LoggerFactory.getLogger(CheckpointExample.class);
    private static final String KAFKA_BROKER = "localhost:9092";
    private static final String KAFKA_INPUT_TOPIC = "input-topic";
    private static final String KAFKA_GROUP_ID = "flink-stackoverflow-checkpointer";
    private static final String CLASS_NAME = CheckpointExample.class.getSimpleName();


    public static void main(String[] args) throws Exception {

        // play with them
        boolean checkpointEnable = false;
        long checkpointInterval = 1000;
        CheckpointingMode checkpointMode = CheckpointingMode.EXACTLY_ONCE;

        // -------------------------------------------------

        LOG.info(CLASS_NAME + ": starting...");
        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

        // kafka source
        // https://ci.apache.org/projects/flink/flink-docs-release-
1.2/dev/connectors/kafka.html#kafka-consumer
        Properties prop = new Properties();
        prop.put("bootstrap.servers", KAFKA_BROKER);
        prop.put("group.id", KAFKA_GROUP_ID);
        prop.put("auto.offset.reset", "latest");
        prop.put("enable.auto.commit", "false");

        FlinkKafkaConsumer09<String> source = new FlinkKafkaConsumer09<>(
                KAFKA_INPUT_TOPIC, new SimpleStringSchema(), prop);

        // checkpoints
        // internals: https://ci.apache.org/projects/flink/flink-docs-
master/internals/stream_checkpointing.html#checkpointing
        // config: https://ci.apache.org/projects/flink/flink-docs-release-
1.3/dev/stream/checkpointing.html
        if (checkpointEnable) env.enableCheckpointing(checkpointInterval, checkpointMode);

        env
                .addSource(source)
                .keyBy((any) -> 1)
                .flatMap(new StatefulMapper())
                .print();

        env.execute(CLASS_NAME);
    }

        /* ************************************************************
         * Stateful mapper
         * (cf. https://ci.apache.org/projects/flink/flink-docs-release-
1.3/dev/stream/state.html)
```

```
         * ************************************************************/

    public static class StatefulMapper extends RichFlatMapFunction<String, String> {
        private transient ValueState<Integer> state;

        @Override
        public void flatMap(String record, Collector<String> collector) throws Exception {
            // access the state value
            Integer currentState = state.value();

            // update the counts
            currentState += 1;
            collector.collect(String.format("%s: (%s,%d)",
                    LocalDateTime.now().format(ISO_LOCAL_DATE_TIME), record, currentState));
            // update the state
            state.update(currentState);
        }

        @Override
        public void open(Configuration parameters) throws Exception {
            ValueStateDescriptor<Integer> descriptor =
                    new ValueStateDescriptor<>("CheckpointExample",
TypeInformation.of(Integer.class), 0);
            state = getRuntimeContext().getState(descriptor);
        }
    }
}
```

サンプルのとのシミュレート

チェックポイントをチェックできるようにするには、 `cluster`をするがあり`cluster`。 よりなは、`flink/bin`ディレクトリで`start-cluster.sh`スクリプトをすること`start-cluster.sh`。

```
start-cluster.sh
Starting cluster.
[INFO] 1 instance(s) of jobmanager are already running on virusnest.
Starting jobmanager daemon on host virusnest.
Password:
Starting taskmanager daemon on host virusnest.
```

さて、あなたのアプリをパッケージし、それをフリンクにしてください

```
mvn clean package
flink run target/flink-checkpoints-test.jar -c CheckpointExample
```

いくつかのデータをする

```
kafka-console-producer --broker-list localhost:9092 --topic input-topic
a
b
c
^D
```

は、 `flink/logs/flink-<user>-jobmanager-0-<host>.out`でになります。えば

```
tail -f flink/logs/flink-Derlin-jobmanager-0-virusnest.out
2017-03-17T08:21:51.249: (a,1)
2017-03-17T08:21:51.545: (b,2)
2017-03-17T08:21:52.363: (c,3)
```

チェックポイントをテストするには、にタスクマネ―ジャをしますこれはエラ―をエミュレートします、デ―タをしてしいものをします。

```
# killing the taskmanager
ps -ef | grep -i taskmanager
kill <taskmanager PID>

# starting a new taskmanager
flink/bin/taskmanager.sh start
```

しい`flink/logs/flink-<user>-jobmanager-1-<host>.out`するときには、のログファイル、つまり`flink/logs/flink-<user>-jobmanager-1-<host>.out` のにづくをします。

をします

- チェックポイントがになっている にデ―タをすると、いなくわれます。しかし、くほどに、カウンタ―はしいでしょう
- チェックポイントをにしました デ―タがわれなくなりましたそしてしいカウンタ―。
- *at-once-once*モ―ドのチェックポイント をすることがあります。に、チェックポイントをくし、タスクマネ―ジャをさせると

オンラインでチェックポイントをむ https://riptutorial.com/ja/apache-flink/topic/9465/チェックポイント

6: テーブルAPI

Examples

Mavenの

テーブルAPIをするには、 `flink-table` をmavenとしてします `flink-clients` および `flink-core` にえて。

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>1.1.4</version>
</dependency>
```

スカラーバージョンここでは2.11がシステムとがあることをしてください。

CSVからのな

えられたCSVファイル `peoples.csv`

```
1,Reed,United States,Female
2,Bradley,United States,Female
3,Adams,United States,Male
4,Lane,United States,Male
5,Marshall,United States,Female
6,Garza,United States,Male
7,Gutierrez,United States,Male
8,Fox,Germany,Female
9,Medina,United States,Male
10,Nichols,United States,Male
11,Woods,United States,Male
12,Welch,United States,Female
13,Burke,United States,Female
14,Russell,United States,Female
15,Burton,United States,Male
16,Johnson,United States,Female
17,Flores,United States,Male
18,Boyd,United States,Male
19,Evans,Germany,Male
20,Stephens,United States,Male
```

、、にをえます

```
public class TableExample{
    public static void main( String[] args ) throws Exception{
        // create the environments
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
        final BatchTableEnvironment tableEnv = TableEnvironment.getTableEnvironment( env );

        // get the path to the file in resources folder
```

```
        String peoplesPath = TableExample.class.getClassLoader().getResource( "peoples.csv"
).getPath();
        // load the csv into a table
        CsvTableSource tableSource = new CsvTableSource(
                peoplesPath,
                "id,last_name,country,gender".split( "," ),
                new TypeInformation[]{ Types.INT(), Types.STRING(), Types.STRING(),
Types.STRING() } );
        // register the table and scan it
        tableEnv.registerTableSource( "peoples", tableSource );
        Table peoples = tableEnv.scan( "peoples" );

        // aggregation using chain of methods
        Table countriesCount = peoples.groupBy( "country" ).select( "country, id.count" );
        DataSet<Row> result1 = tableEnv.toDataSet( countriesCount, Row.class );
        result1.print();

        // aggregation using SQL syntax
        Table countriesAndGenderCount = tableEnv.sql(
                "select country, gender, count(id) from peoples group by country, gender" );

        DataSet<Row> result2 = tableEnv.toDataSet( countriesAndGenderCount, Row.class );
        result2.print();
    }
}
```

はのとおりです。

```
Germany,2
United States,18

Germany,Male,1
United States,Male,11
Germany,Female,1
United States,Female,7
```

テーブルの

`peoples.csv`えて *CSV*からのなを 、とをす2つの*CSV*があります。

`sales.csv` people_id、product_id

```
19,5
6,4
10,4
2,4
8,1
19,2
8,4
5,5
13,5
4,4
6,1
3,3
8,3
17,2
6,2
```

```
1,2
3,5
15,5
3,3
6,3
13,2
20,4
20,2
```

products.csv ID、、

```
1,Loperamide,47.29
2,pain relief pm,61.01
3,Citalopram,48.13
4,CTx4 Gel 5000,12.65
5,Namenda,27.67
```

たちは40ドルをえるりげごとにとをたいとっています

```
public class SimpleJoinExample{
    public static void main( String[] args ) throws Exception{

        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
        final BatchTableEnvironment tableEnv = TableEnvironment.getTableEnvironment( env );

        String peoplesPath = TableExample.class.getClassLoader().getResource( "peoples.csv"
).getPath();
        String productsPath = TableExample.class.getClassLoader().getResource( "products.csv"
).getPath();
        String salesPath = TableExample.class.getClassLoader().getResource( "sales.csv"
).getPath();

        Table peoples = csvTable(
                tableEnv,
                "peoples",
                peoplesPath,
                "pe_id,last_name,country,gender",
                new TypeInformation[]{ Types.INT(), Types.STRING(), Types.STRING(),
Types.STRING() } );

        Table products = csvTable(
                tableEnv,
                "products",
                productsPath,
                "prod_id,product_name,price",
                new TypeInformation[]{ Types.INT(), Types.STRING(), Types.FLOAT() } );

        Table sales = csvTable(
                tableEnv,
                "sales",
                salesPath,
                "people_id,product_id",
                new TypeInformation[]{ Types.INT(), Types.INT() } );

        // here is the interesting part:
        Table join = peoples
                .join( sales ).where( "pe_id = people_id" )
                .join( products ).where( "product_id = prod_id" )
```

```
                .select( "last_name, product_name, price" )
                .where( "price < 40" );

        DataSet<Row> result = tableEnv.toDataSet( join, Row.class );
        result.print();

    }//end main


    public static Table csvTable( BatchTableEnvironment tableEnv, String name, String path,
String header,
                                  TypeInformation[]
                                          typeInfo ){
        CsvTableSource tableSource = new CsvTableSource( path, header.split( "," ), typeInfo);
        tableEnv.registerTableSource( name, tableSource );
        return tableEnv.scan( name );
    }

}//end class
```

になるをすることがです。そうでない、flinkは「のあいまいな」についてをちます。

```
Burton,Namenda,27.67
Marshall,Namenda,27.67
Burke,Namenda,27.67
Adams,Namenda,27.67
Evans,Namenda,27.67
Garza,CTx4 Gel 5000,12.65
Fox,CTx4 Gel 5000,12.65
Nichols,CTx4 Gel 5000,12.65
Stephens,CTx4 Gel 5000,12.65
Bradley,CTx4 Gel 5000,12.65
Lane,CTx4 Gel 5000,12.65
```

シンクの

テーブルはTableSinkにきむことができます.TableSinkは、さまざまなフォーマットとファイルシステムをサポートするインターフェイスです。バッチは`BatchTableSink`のみきむことができますが、ストリーミングには`StreamTableSink`が`StreamTableSink`。

、flinkは`CsvTableSink`インターフェイスのみをしています。

のでは、をきえます。

```
DataSet<Row> result = tableEnv.toDataSet( table, Row.class );
result.print();
```

with

```
TableSink sink = new CsvTableSink("/tmp/results", ",");
// write the result Table to the TableSink
table.writeToSink(sink);
// start the job
env.execute();
```

`/tmp/results`はフォルダで、flinkはをうためです。したがって、4つのプロセッサをしているは、フォルダに4つのファイルがまれているがあります。

また、に`env.execute()`びすことにしてください。これはflinkジョブをするためにですが、のでは`print()`いました。

オンラインでテーブルAPIをむ https://riptutorial.com/ja/apache-flink/topic/8966/テーブルapi

7: ロギング

き

このトピックでは、Flinkアプリケーションでログlog4jをおよびするをします。

Examples

あなたのコードでロガーをう

`pom.xml` `slf4j`をしてください

```
<properties>
    <slf4j.version>1.7.21</slf4j.version>
</properties>

<!-- ... -->

<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>${slf4j.version}</version>
</dependency>
```

クラスでするためのロガーオブジェクトをします。

```
private Logger LOGGER = LoggerFactory.getLogger(FlinkApp.class);
```

`RichMapFunction`サブクラスなど、シリアライズするのあるクラスでは、 `LOGGER`を`transient`としてすることをれないでください。

```
private transient Logger LOG = LoggerFactory.getLogger(MyRichMapper.class);
```

あなたのコードでは、いつものように`LOGGER`をいます。オブジェクトなどのにプレースホルダ `{}` をします。

```
LOGGER.info("my app is starting");
LOGGER.warn("an exception occurred processing {}", record, exception);
```

ロギング

ローカルモード

たとえば、IDEからアプリケーションをするローカルモードでは、りに`log4j`をできます。つまり、 `log4j.properties`クラスパスでできるようにすることができます。 mavenのなは、`src/main/resources`フォルダに`log4j.properties`をすること`log4j.properties`。 にをします。

```
log4j.rootLogger=INFO, console

# patterns:
#  d = date
#  c = class
#  F = file
#  p = priority (INFO, WARN, etc)
#  x = NDC (nested diagnostic context) associated with the thread that generated the logging
event
#  m = message


# Log all infos in the console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{dd/MM/yyyy HH:mm:ss.SSS} %5p [%-10c] %m%n

# Log all infos in flink-app.log
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.file=flink-app.log
log4j.appender.file.append=false
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{dd/MM/yyyy HH:mm:ss.SSS} %5p [%-10c] %m%n

# suppress info messages from flink
log4j.logger.org.apache.flink=WARN
```

スタンドアロンモード

> スタンドアロンモードでは、されるのは`jar`ファイルのものではありません。これは、Flinkにのファイルがあるためです。

のファイル Flinkには、ののプロパティファイルがしています。

- `log4j-cli.properties`コマンドラインクライアントでされます `flink run` クラスタでされるコードではありません
- `log4j-yarn-session.properties`コマンドラインクライアントがYARNセッションをするときにします `yarn-session.sh`
- `log4j.properties` / Taskmanagerログスタンドアロンと YARNの

`${log.file}`デフォルトで`flink/log`。 それはでオーバーライドすることができ`flink-conf.yaml`することで、 `env.log.dir`、

> `env.log.dir`は、Flinkログがされるディレクトリをします。それはパスでなければなりません。

ログの ログはロ―カルなものです 。 つまり、JobManagers/ Taskmanagersをしているマシンでされます。

Yarn FlinkをYarnでするときは、Hadoop YARNのログにするがあります。そのためのもなは、YARNログです。これをにするには、 `yarn-site.xml file` `yarn.log-aggregation-enable`プロパティを`true`にし`yarn-site.xml file`。 これをにすると、をしてしたYARNセッションのすべてのログファイルをできます。

```
yarn logs -applicationId <application ID>
```

ながら、ログは、セッションのがした、たとえばにのみできます。

アプリケ―ションごとになるをする

さまざまなアプリケ―ションになるがなは、Flink 1.2のようになはありません。

もしあなたがflinkのジョブごとに1つのクラスのモ―ドをしているつまり、 `flink run -m yarn-cluster` ...でスクリプトをした、があります

1. あなたのプロジェクトのどこかに`conf`ディレクトリをする
2. `flink/conf`のすべてのファイルのシンボリックリンクをします。

   ```
   mkdir conf
   cd conf
   ln -s flink/conf/* .
   ```

3. symlinkの`log4j.properties` またはしたいのファイルをのできえます
4. あなたのをめるに、

   ```
   export FLINK_CONF_DIR=/path/to/my/conf
   ```

flinkのバ―ジョンにじて、 `flink/bin/config.sh`ファイルをするがあります。このラインをえてれば

```
FLINK_CONF_DIR=$FLINK_ROOT_DIR_MANGLED/conf
```

それをする

```
if [ -z "$FLINK_CONF_DIR" ]; then
    FLINK_CONF_DIR=$FLINK_ROOT_DIR_MANGLED/conf;
fi
```

Flink-on-Yarnのrsyslogをしてリアルタイムでログをする

Yarnは、アプリケ―ションがするにデフォルトでログをするわけではなく、さえしないストリ―

ミングジョブでになることがあります。

は、ほとんどのLinuxマシンでな`rsyslog`をすることです。

まず、 `/etc/rsyslog.conf`ののののコメントをして、incoming udpをします。

```
$ModLoad imudp
$UDPServerRun 514
```

`log4j.properties` このページののををして、 `log4j.properties`をし`SyslogAppender`。

```
log4j.rootLogger=INFO, file

# TODO: change package logtest to your package
log4j.logger.logtest=INFO, SYSLOG

# Log all infos in the given file
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.file=${log.file}
log4j.appender.file.append=false
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=bbdata: %d{yyyy-MM-dd HH:mm:ss,SSS} %-5p %-60c %x
- %m%n

# suppress the irrelevant (wrong) warnings from the netty channel handler
log4j.logger.org.jboss.netty.channel.DefaultChannelPipeline=ERROR, file

# rsyslog
# configure Syslog facility SYSLOG appender
# TODO: replace host and myTag by your own
log4j.appender.SYSLOG=org.apache.log4j.net.SyslogAppender
log4j.appender.SYSLOG.syslogHost=10.10.10.102
log4j.appender.SYSLOG.port=514
#log4j.appender.SYSLOG.appName=bbdata
log4j.appender.SYSLOG.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appender.SYSLOG.layout.conversionPattern=myTag: [%p] %c:%L - %m %throwable %n
```

rsyslogはをしいログエントリとしてうため、レイアウトがです。のでは、スタックトレースなどはスキップされます。/タブきログを「に」させたいは、 `rsyslog.conf`をして`rsyslog.conf`をします。

```
$EscapeControlCharactersOnReceive off
```

すべてのログをのファイルにリダイレクトするは、 `conversionPattern`のに`myTag:`をするとです。これをうには、 `rsyslog.conf`をしてのルールをします。

```
if $programname == 'myTag' then /var/log/my-app.log
& stop
```

オンラインでロギングをむ https://riptutorial.com/ja/apache-flink/topic/9713/ロギング

クレジット

S. No		Contributors
1	apache-flinkをいめる	Community, Derlin, vdep
2	カスタムdeシリアライズスキーマをする	Derlin
3	カフカのデータをする	alpinegizmo, Derlin
4	セーブポイントとされたチェックポイント	Derlin
5	チェックポイント	Derlin
6	テーブルAPI	Derlin
7	ロギング	Derlin