FusionInsight

Spark应用开发

www.huawei.com

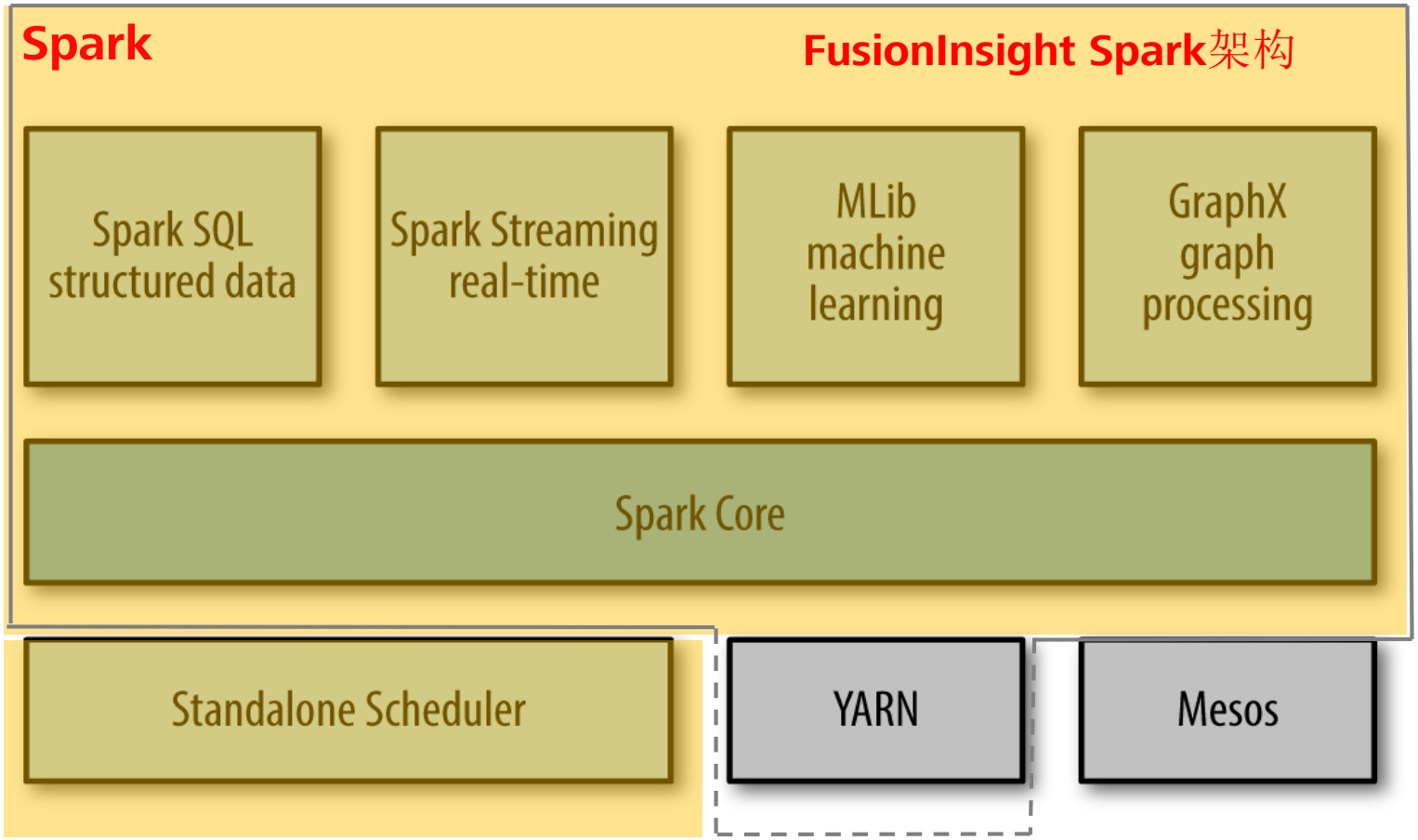
HUAWEI

目标

- 学完本课程后，您将能够:
  - 了解**Spark**基本原理
  - 搭建**Spark**开发环境
  - 开发**Spark**应用程序
  - 调试运行**Spark**应用程序

目录

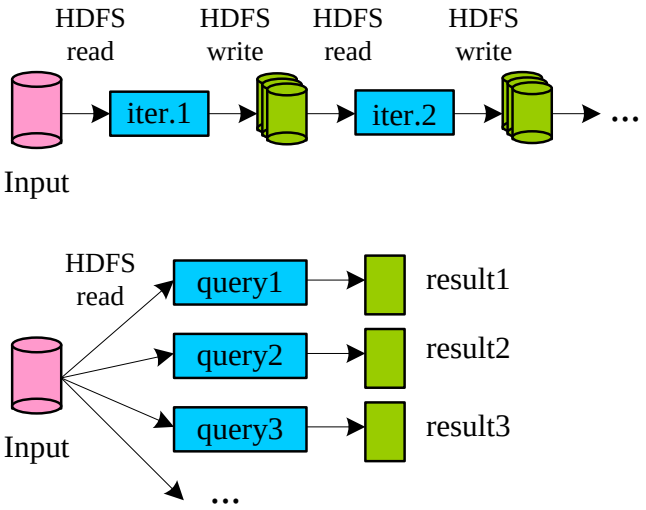
Spark技术架构



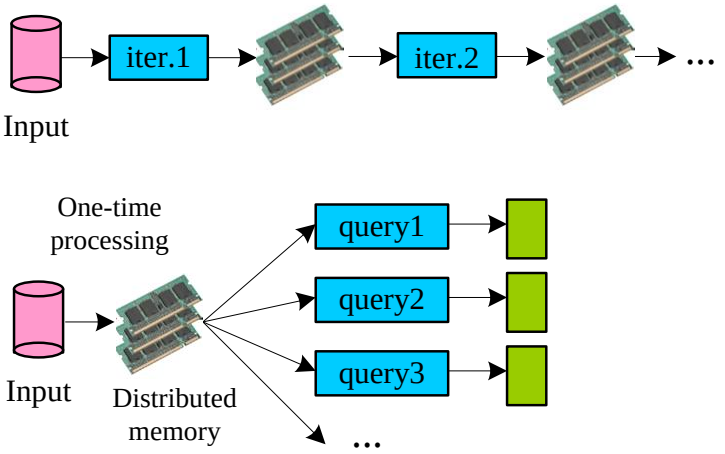
目录

Spark适用场景

大多数现有集群计算框架如**MapReduce**等基于从稳定存储（文件系统）到稳定存储的非循环数据流，数据重用都是基于磁盘的，执行效率比较低。与传统的**MapReduce**任务的频繁读写磁盘数据相比，基于内存计算的**Spark**则更适合应用在**迭代计算，交互式分析**等场景。



Data Sharing in MapReduce



Data Sharing in Spark

目录

开发环境搭建—准备工作

1. FusionInsight环境：

确认**FusionInsight**环境上**Spark**服务已安装，如果开发的应用需要和**HBase**或者**Kafka**交互，请确保所要用到的组件也都安装并正常运行。

2. 开发工具：

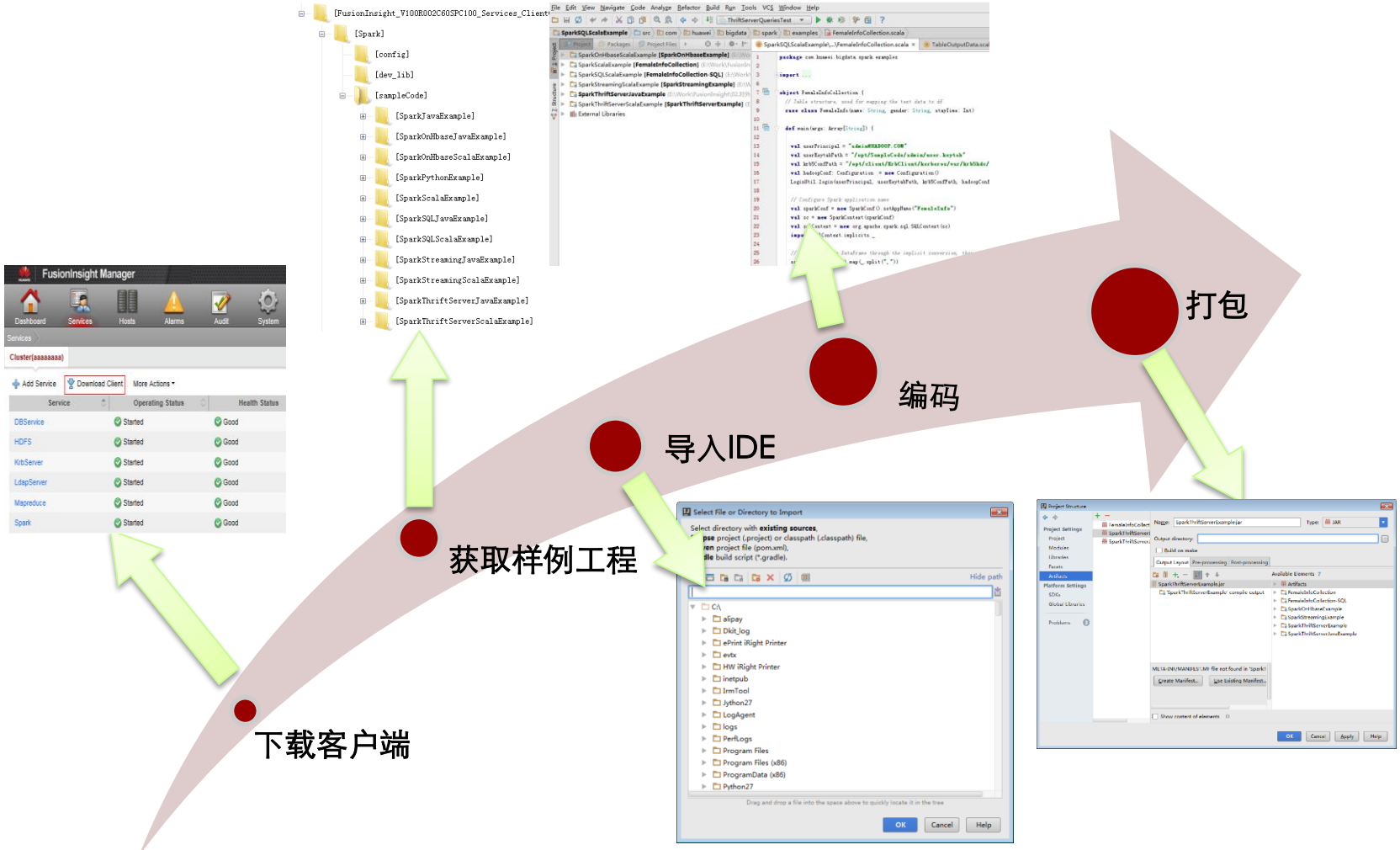
客户端使用**JDK1.7(**或**1.8)**，**intellij idea**使用**13.1.4**版本，**Scala(2.10.4)**版本。

3. 客户端

客户端所在节点的时间与**FusionInsight**集群的时间要保持一致，时差要小于**5**分钟。

客户端和集群节点之间网络互通。

开发环境搭建—样例工程



HUAWEI

目录

Spark应用运行流程—关键角色

Client：需求提出方，负责提交需求（应用）。

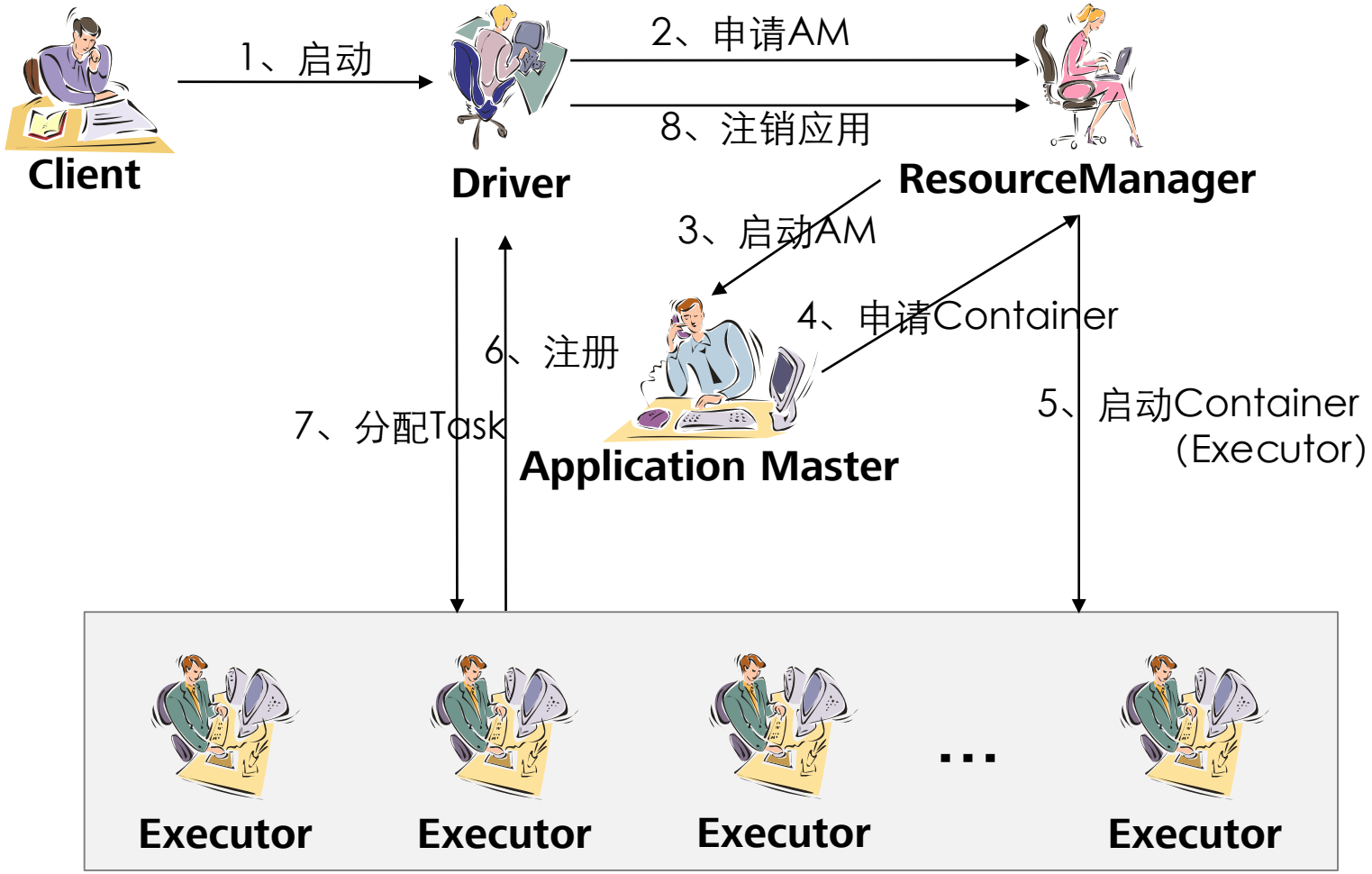
Driver：负责应用的业务逻辑和运行规划（**DAG**）。

ApplicationMaster：负责应用的资源管理，根据应用的需要，向资源管理部门（**ResourceManager**）申请资源.

ResourceManager：资源管理部门，负责整个集群的资源统一调度和分配。

Executor：负责实际计算工作，一个应用会分拆给多个**Executor**来进行计算。

Spark应用运行流程



Spark Application基本概念

Application：

Spark用户程序，提交一次应用为一个**Application**，一个**App**会启动一个**SparkContext**，也就是**Application**的**driver**，驱动整个**Application**的运行。

Job：

一个**Application**可能包含多个**Job**，每个**action**算子对应一个**Job**；**action**算子有**collect**，**count**等。

Stage：

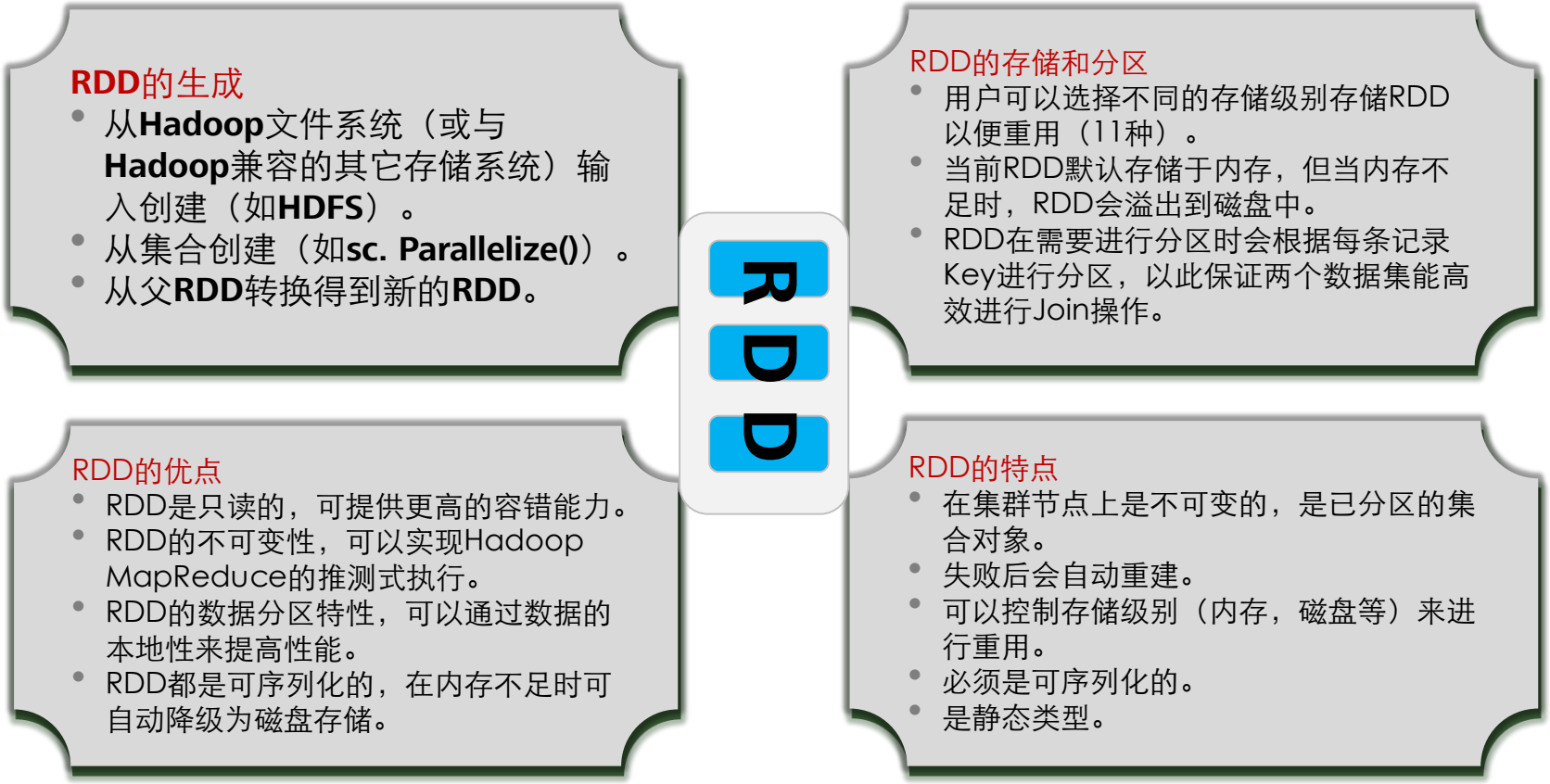
每个**Job**可能包含多层**Stage**，划分标记为**shuffle**过程；**Stage**按照依赖关系依次执行。

Task：

具体执行任务的基本单位，被发到**executor**上执行。

Spark核心概念—RDD

- **RDD**（**Resilient Distributed Datasets**）即弹性分布数据集，指的是一个只读的，可分区的分布式数据集。这个数据集的全部或部分可以缓存在内存中，在多次计算间重用。



HUAWEI

RDD的创建

Spark所有的操作都围绕弹性分布式数据集（**RDD**）进行，这是一个有容错机制并可以被并行操作的元素集合，具有只读、分区、容错、高效、无需物化、可以缓存、**RDD**依赖等特征。

目前有两种类型的基础**RDD**：

- **并行集合：**接收一个已经存在的**Scala**集合，然后进行并行计算。
- **Hadoop数据集**：在一个文件的每条记录上运行函数。只要文件系统是**HDFS**，或者**hadoop**支持的任意存储系统即可。

这两种类型的**RDD**都可以通过相同的方式进行操作，从而获得子**RDD**等一系列拓展，形成血统关系图。

RDD的创建—并行集合

并行集合是通过调用**SparkContext**的**parallelize**方法，在一个已经存在的**Scala**集合（一个**Seq**对象）上创建的。集合的对象将会被拷贝，创建出一个可以被并行操作的分布式数据集。例如，下面的解释器输出，演示了如何从一个数组创建一个并行集合。

例如：

val rdd = sc.parallelize(Array(1 to 10)) 根据能启动的**executor**的数量来进行切分多个**slice**，每一个**slice**启动一个**Task**来进行处理。

val rdd = sc.parallelize(Array(1 to 10), 5) 指定了**partition**的数量**5**。

RDD的创建—Hadoop数据集

Spark可以将任何**Hadoop**所支持的存储资源转化成**RDD**，如本地文件（需要网络文件系统，所有的节点都必须能访问到）、**HDFS**、**Cassandra**、**HBase**等，**Spark**支持文本文件、**SequenceFiles**和任何**Hadoop InputFormat**格式等。

RDD算子：Transformation和Action

Transformation

返回值还是一个**RDD**，如**map**、**filter**、**join**等。 **Transformation**都是**Lazy**的，代码调用到**Transformation**的时候，并不会马上执行，需要等到有**Action**操作的时候才会启动真正的计算过程。

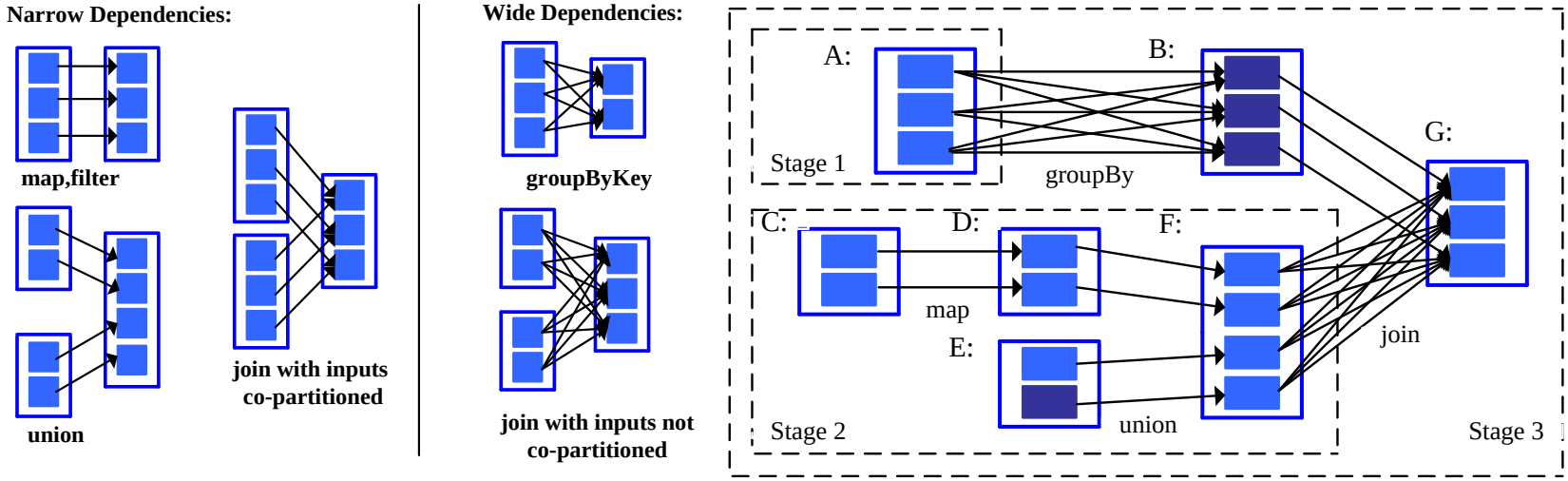
Action

如**count**，**collect**，**save**等，**Action**操作是返回结果或者将结果写入存储的操作。
Action是**Spark**应用真正执行的触发动作。

RDD依赖：宽依赖和窄依赖

RDD父子依赖关系：

- **窄依赖（Narrow）**指父**RDD**的每一个分区最多被一个子**RDD**的分区所用。
- **宽依赖（Wide）**指子**RDD**的分区依赖于父**RDD**的所有分区，是**Stage**划分的依据。



样例程序--WordCount

创建**SparkContext**对象，设置应用名称为**Wordcount**。

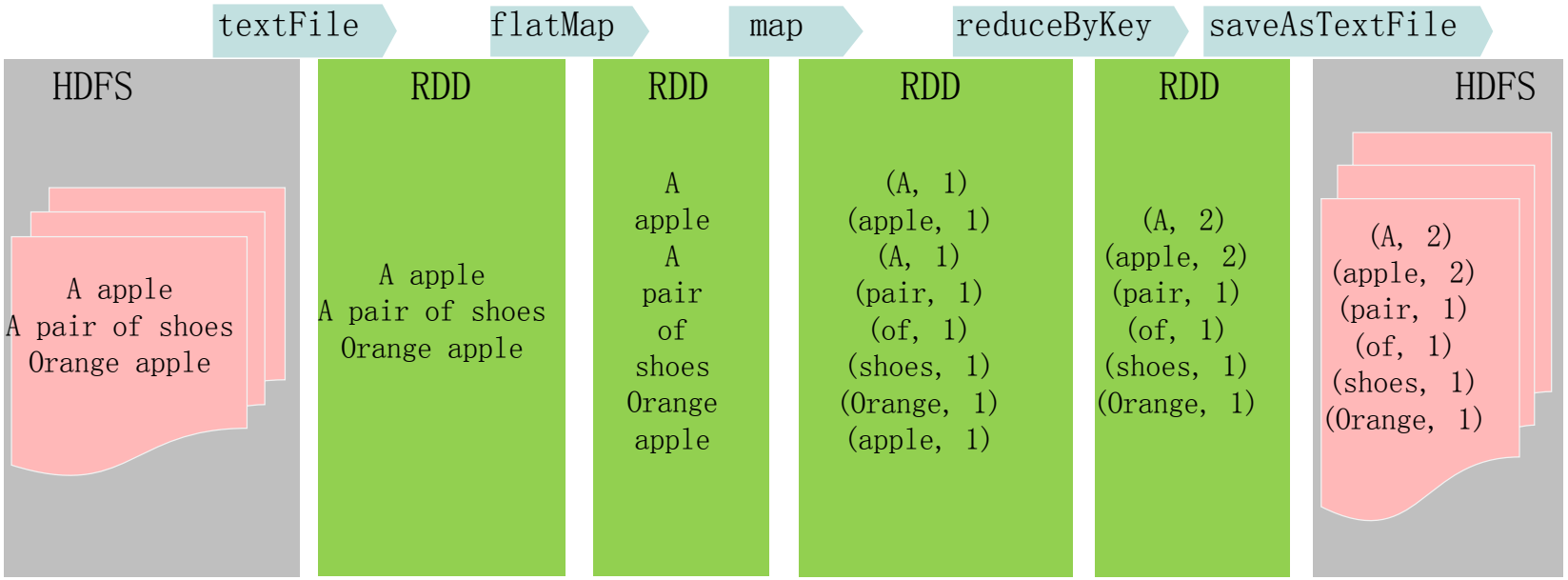
从**HDFS**加载文本文件，得到一个**RDD**。

调用**RDD**的**Transformation**进行计算：
将文本文件按空格分割，然后每个单词计数置为**1**，最后按相同的**Key**将计数求合。
这一步会分发到各个**Executor**上执行。

调用**Action**操作，保存结果。这一行才触发真正的任务执行。

```
object WordCount
{
    def main (args: Array[String]): Unit = {
        //配置Spark应用名称
        val conf = new SparkConf().setAppName("WordCount")
        val sc: SparkContext = new SparkContext(conf)
        val textFile = sc.textFile("hdfs://...")
        val counts = textFile.flatMap(line => line.split(" "))
            .map(word => (word, 1))
            .reduceByKey(_ + _)
        counts.saveAsTextFile("hdfs://...")
    }
}
```

样例程序--WordCount



打包运行程序

1. 用工具将工程生成**Jar**文件。
2. 执行**spark-submit**运行程序提交应用：

#spark-submit --master model --class package.WordCount wordcount.jar

另外，对于简单的语句，可以用**spark-shell**进行交互式执行：

```
189-121-130-115:/opt # spark-shell --master yarn-client
Warning: Ignoring non-spark config property: hadoop_server_path=/opt/huawei/Bigdata/hadoop/
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.5.1
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_72)
Type in expressions to have them evaluated.
Type :help for more information.
spark.driver.cores is set but does not apply in client mode.
Debug is  true storeKey false useTicketCache true useKeyTab false doNotPrompt false ticketCa
s false clearPass is false
Acquire TGT from Cache
Principal is admin@HADOOP.COM
Commit Succeeded

Spark context available as sc.
SQL context available as sqlContext.

scala> val count = sc.makeRDD(1 to 20, 3).count()
count: Long = 20

scala> print(count)
20
```

RDD常用Transformation算子

Transformation	含义
map(func)	对调用**map**的**RDD**数据集中的每个**element**都使用**func**，然后返回一个新的**RDD**。
filter(func)	过滤，对调用**filter**的**RDD**数据集中的每个元素都使用**func**，然后返回一个包含使**func**为**true**的元素构成的**RDD**。
flatMap(func)	和**map**差不多，但是**flatMap**生成的是多个结果。
mapPartitions(func)	和**map**很像，但是**map**是每个**element**，而**mapPartitions**是每个**partition**。
mapPartitionsWithIndex(func)	和**mapPartitions**很像，但是**func**还提供了**paritition**编号的**int**值。

RDD常用Transformation算子

Transformation	含义
sample(withReplacement, fraction, seed)	抽样一部分数据。
union(otherDataset)	返回一个新的**Dataset**，包含当前**dataset**和给定的**dataset**。
intersection(otherDataset)	返回一个新的**Dataset**，包含当前**dataset**和给定的**dataset**的数据交集。
distinct([numTasks]))	去重。
groupByKey([numTasks])	如果**rdd**是键值对形式，返回（键，**Iterable<**值**>**）。
reduceByKey(func, [numTasks])	类似**groupByKey**，但是每一个**key**对应的**value**会根据提供的**func**进行计算以得到一个新的值。
sortByKey([ascending], [numTasks])	根据**key**进行正向或反向排序。

RDD常用Transformation算子

Transformation	含义
join(otherDataset, [numTasks])	如果数据集是（**K**，**V**）关联的数据集是（**K**，**W**），返回（**K**，（**V**，**W**））同时支持**leftOuterJoin**，**rightOutJoin**，和**fullOuterJoin**。
cogroup(otherDataset, [numTasks])	如果数据集是（**K**，**V**）关联的的数据集是（**K**，**W**）返回（**K**，**(Iterable<V>, Iterable<W>)**）。
cartesian(otherDataset)	笛卡尔积。

RDD常用Action算子

Action	含义
reduce(func)	根据函数聚合数据集里的元素。
collect()	一般在**filter**或者足够小的结果的时候，再用**collect**封装返回一个数组。
count()	统计数据集中元素个数。
first()	获取第一个元素。
take(n)	获取数据集最上方的几个元素，返回一个数组。
takeOrdered(n, [ordering])	提供自定义比较器，返回比较后最上方的几个元素。

RDD常用Action算子

Action	含义
saveAsTextFile(path)	把**dataset**写到一个**textfile**中，或者**hdfs**，**Spark**把每条记录都转换为一行记录，然后写到**file**中。
saveAsSequenceFile(path)	只能用在**key-value**对上，然后生成**SequenceFile**写到本地或者**hadoop**文件系统。
saveAsObjectFile(path)	根据**Java**序列化保存**dataset**中的元素，可以用**SparkContext.objectFile()**读取保存的文件。
countByKey()	返回的是**key**对应的个数的一个**map**。
foreach(func)	对数据集内每一个元素作用函数**func**。

Spark任务参数配置—参数配置文件

Spark有**3**种参数配置方式：

1、在spark-default.conf文件配置：

该文件在客户端目**/Spark/spark/conf/spark-defaults.conf**，对于所有应用都适用的配置，可以在此配置中配置。

2、在任务提交的时候，用--conf指定：

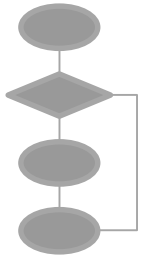
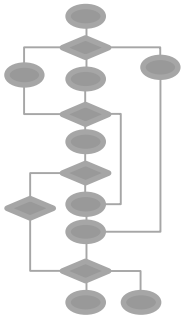
运行过程中需要随时调整的参数，可以通过这种方式在任务提交时动态调整。

3、在代码中通过SparkConf对象指定：

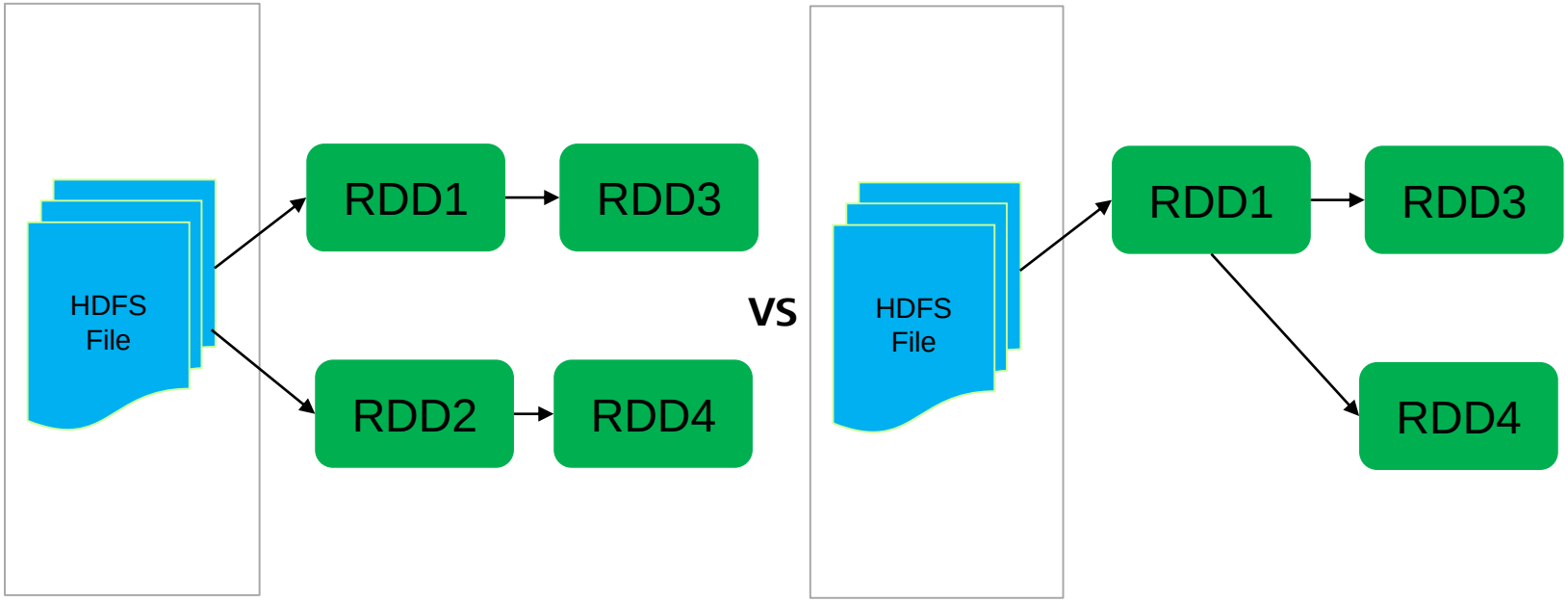
```
val conf = new SparkConf().set(" “,””)
val sc = new SparkContext(conf) 。
```

注：如果同时使用了前面三种方式设置参数。那么**Spark**读取的优先级是：**配置文件 < 动态参数 < 代码配置。**

SparkCore调优

算法		vs	
资源		vs	
硬件		vs	

算法调优—RDD复用



相同的数据，只创建一个RDD

算法调优—RDD缓存

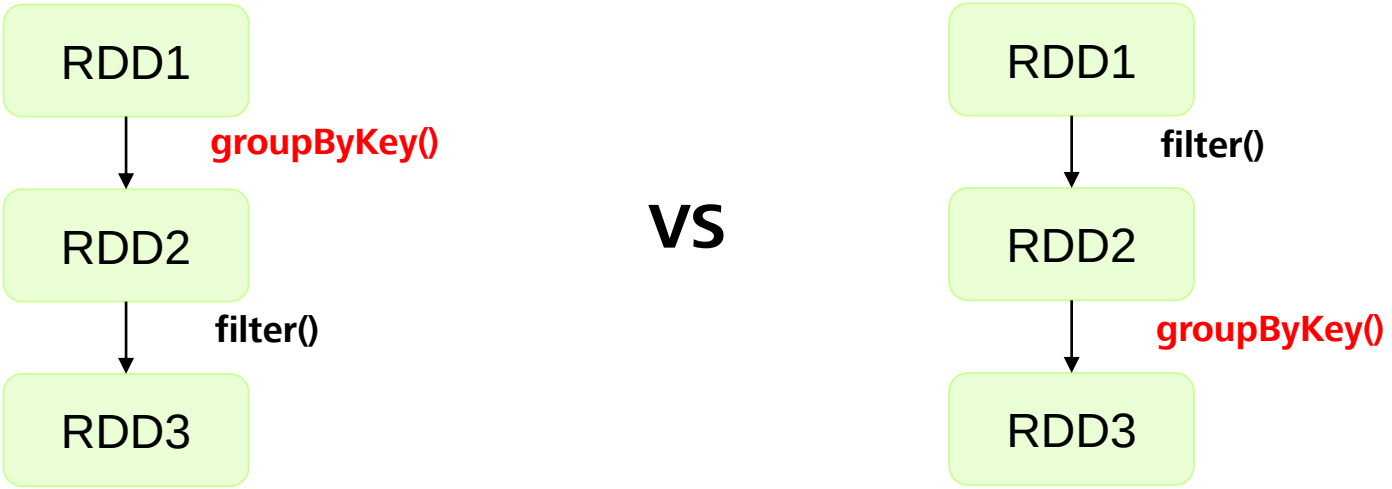
Spark可以使用 **persist** 和 **cache** 方法将任意 **RDD** 缓存到内存、磁盘文件系统中。缓存是容错的，如果一个 **RDD** 分片丢失，可以通过构建它的**transformation**自动重构。被缓存的 **RDD** 被使用的时，存取速度会被大大加速。一般的**executor**内存**60%**做 **cache**， 剩下的**40%**做**task**。

Spark中，**RDD**类可以使用**cache()** 和 **persist()** 方法来缓存。**cache()**是**persist()**的特例，将该**RDD**缓存到内存中。而**persist**可以指定一个**StorageLevel**。

算法调优—避免使用Shuffle

Shuffle过程会有整个**RDD**数据的写和读的操作，成本非常调。应避免命名使用**Shuffle**类的操作算子，如果因业务需要确实无法避免，应注意以下两点：

- 确保**Shuffle**的数据量最小化
- 使用**map-side**预聚合的**shuffle**操作



算法调优—使用高性能的算子

在业务允许的情况下：

- 使用**reduceByKey/aggregateByKey**替代**groupByKey**
- 使用**mapPartitions**替代普通**map**
- 使用**foreachPartitions**替代**foreach**
- 使用**filter**之后进行**coalesce**操作
- 使用**repartitionAndSortWithinPartitions**替代**repartition**与**sort**类操作

算法调优—使用高性能的算子

在业务允许的情况下：

- 使用**reduceByKey/aggregateByKey**替代**groupByKey**
- 使用**mapPartitions**替代普通**map**
- 使用**foreachPartitions**替代**foreach**
- 使用**filter**之后进行**coalesce**操作
- 使用**repartitionAndSortWithinPartitions**替代**repartition**与**sort**类操作

算法调优—RDD共享变量

在应用开发中，一个函数被传递给**Spark**操作（例如**map**和**reduce**），在一个远程集群上运行，它实际上操作的是这个函数用到的所有变量的独立拷贝。这些变量会被拷贝到每一台机器。通常看来，在任务之间中，读写共享变量显然不够高效。然而，**Spark**还是为两种常见的使用模式，提供了两种有限的共享变量：

- 广播变量
- 累加器

算法调优—使用广播变量

外部变量:

默认情况下，**Spark**会将该变量复制多个副本，通过网络传输到**task**中，此时**每个task都有一个变量副本**。

广播变量:

变量广播以后，**每个Executor保留一份**，多个**Task**执行时会共享此变量。对于**Executor**的核数较多时，可以明显降低内存消耗和网络**IO**。

用法:

```
val  aList=bigList
val aListBroadcast = sc.broadcast(aList)
rdd1.map( use aListBroadcast  )
```

算法调优—使用累加器

累加器只支持加法操作，可以高效地并行，用于实现计数器和变量求和。

Spark 原生支持数值类型和标准可变集合的计数器，但用户可以添加新的类型。只有**driver**才能获取累加器的值。

用法：

```
val accum = sc.accumulator(0, "My Accumulator")

sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)
```

算法调优—使用Kryo优化序列化性能

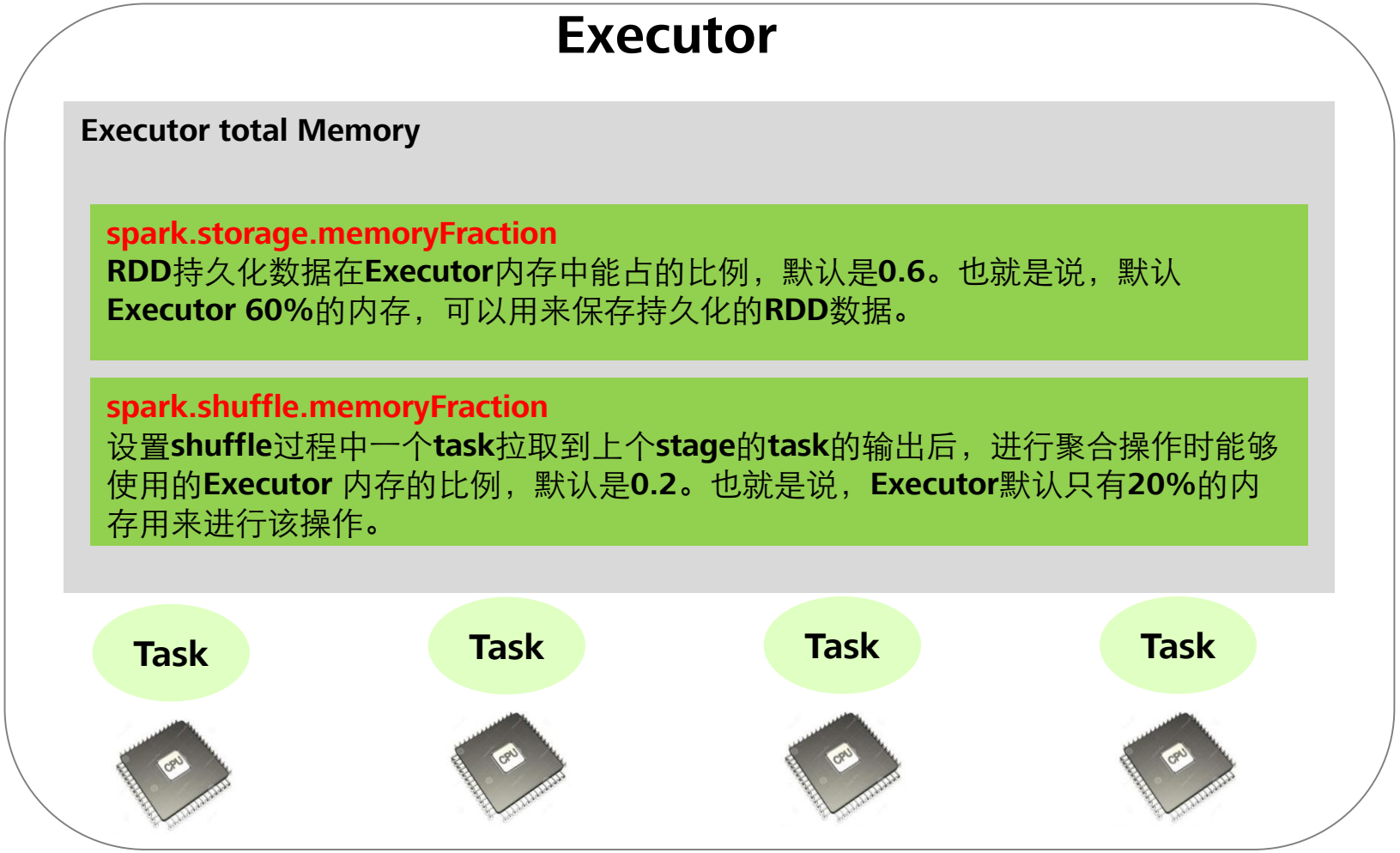
- 在算子函数中使用到外部变量时，该变量会被序列化后进行网络传输。
- 将自定义的类型作为**RDD**的泛型类型时（比如**JavaRDD**，**Student**是自定义类型），所有自定义类型对象，都会进行序列化。因此这种情况下，也要求自定义的类必须实现**Serializable**接口。
- 使用可序列化的持久化策略时（比如**MEMORY_ONLY_SER**），**Spark**会将**RDD**中的每个**partition**都序列化成一个大的字节数组。

Spark提供了两种序列化实现：

- **org.apache.spark.serializer.KryoSerializer**：性能好，兼容性差
- **org.apache.spark.serializer.JavaSerializer**：性能一般，兼容性好

使用：**conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")**

资源调优—Executor内存模型



资源调优—资源调优参数

主要参数:

- **Executor个数（ --num-executors ）：** 应用需要多个少**Executor**来进行计算。
- **Executor CPU（ --executor-cores）：** 每个**Executor**的分配的**CPU**核数，此参数决定了一个**Executor**能同时处理的**Task**数。
- **Executor 内存（ --executor-memory ）：** 每个**Executor**分配的总内存，**Executor**内的内存分布参见上一页。
- **Driver CPU（ --driver-cores ）：** 分配给**Driver**的**CPU**核数，只对**yarn-cluster**模式有效。
- **Driver 内存（ --driver-memory ）：** 分配给**Driver**的内存数。
- **并行度（ spark.default.parallelism）：** 最原始的**RDD**的分区取决于数据源上数据的分区，此并行度参数决定**shuffle**之后的分区数。
- **内存比例： （参见上面描述）**
  - **spark.storage.memoryFraction**
  - **spark.shuffle.memoryFraction**

本章小结

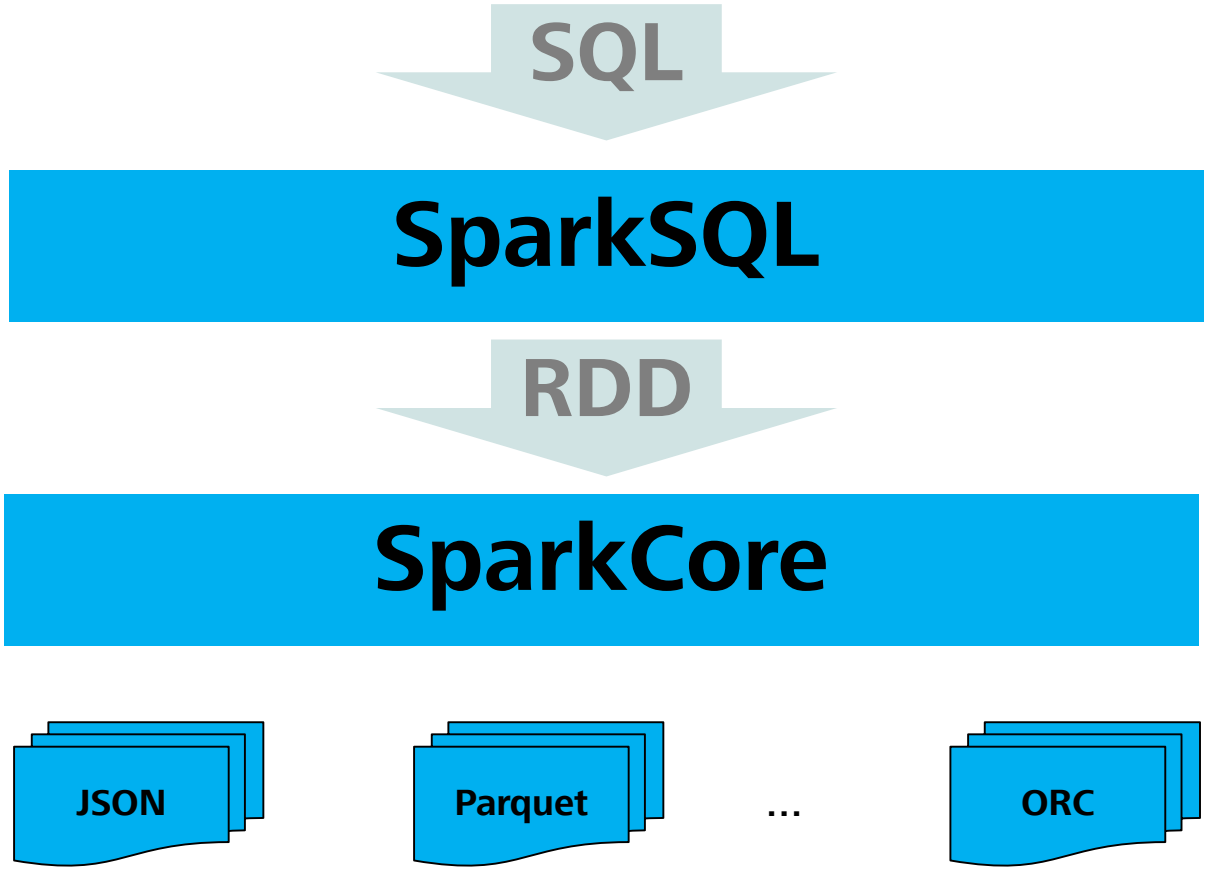
本章主要学习了以下内容

- **Spark**核心概念
- **Spark**应用运行流程
- **Spark**应用开发样例
- **RDD**常用算子
- **Spark**应用参数配置
- **Spark**应用调优方法

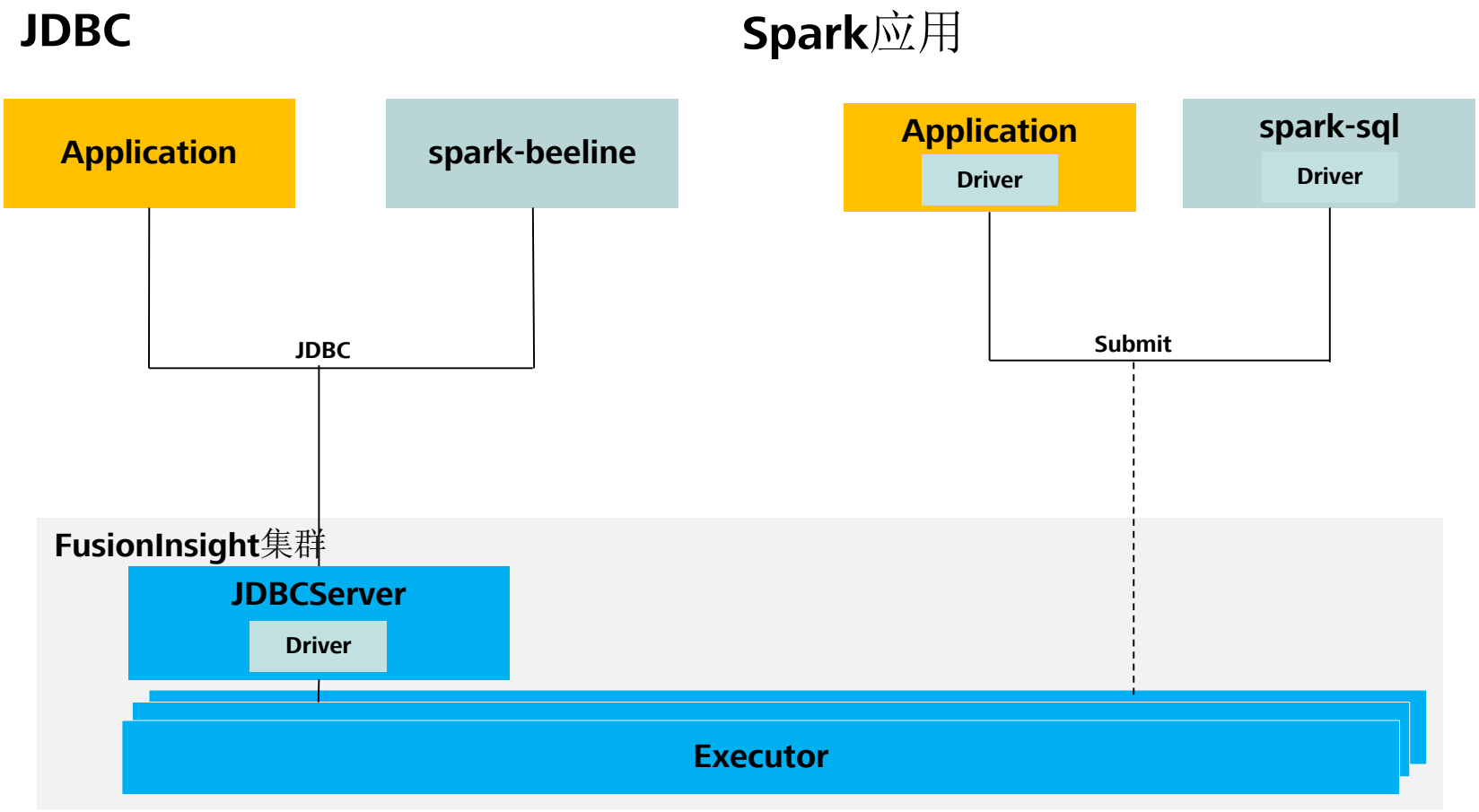
目录

1. 准备工作
2. 第一个Spark应用
   - SparkCore
   - SparkSQL
   - SparkStreaming
3. 调试Spark应用
4. 学习路径/资料（FAQ）

SparkSQL原理



SparkSQL的使用方式



交互式SQL命令行工具

- **spark-beeline**和**spark-sql**都是**SQL**客户端工具，可以交互式地执行**SQL**语句，使用方法：

```
0: jdbc:hive2://ha-cluster/default> CREATE TABLE CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
+---------+--+
| result  |
+---------+--+
+---------+--+
No rows selected (0.23 seconds)
0: jdbc:hive2://ha-cluster/default> LOAD DATA INPATH '/data.txt' INTO TABLE CHILD;
+---------+--+
| result  |
+---------+--+
+---------+--+
No rows selected (0.528 seconds)
0: jdbc:hive2://ha-cluster/default> SELECT count(*) FROM child;
+------+--+
| _c0  |
+------+--+
| 1    |
+------+--+
```

```
spark-sql> CREATE TABLE CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 4.518 seconds
spark-sql> LOAD DATA INPATH '/data.txt' INTO TABLE CHILD;
Loading data to table default.child
Table default.child stats: [numFiles=1, totalSize=12]
OK
Time taken: 0.804 seconds
spark-sql> SELECT count(*) FROM child;
1
Time taken: 4.259 seconds, Fetched 1 row(s)
```

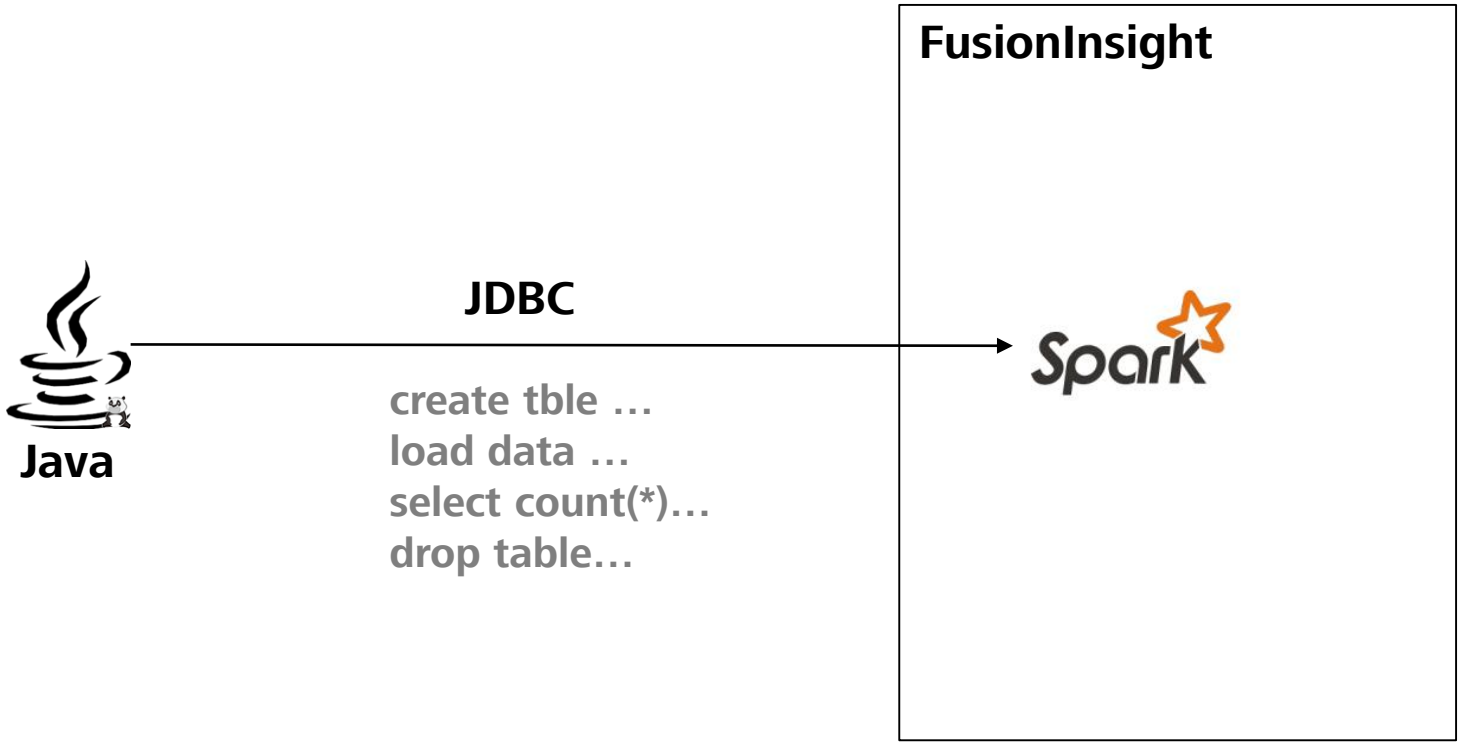
SQL语法

spark.sql.dialect	解释
sql	标准SQL语法子集，推荐使用SQL99Dialect。
hiveql	Hive语法。
SQL99Dialect	标准SQL语法子集，在“sql”方言的基础上，新加了关联子查询、嵌套子查询、Grouping Sets等语法的支持。
BigSQLDialect	“hiveql”方言和“SQL99Dialect”方言的合集。

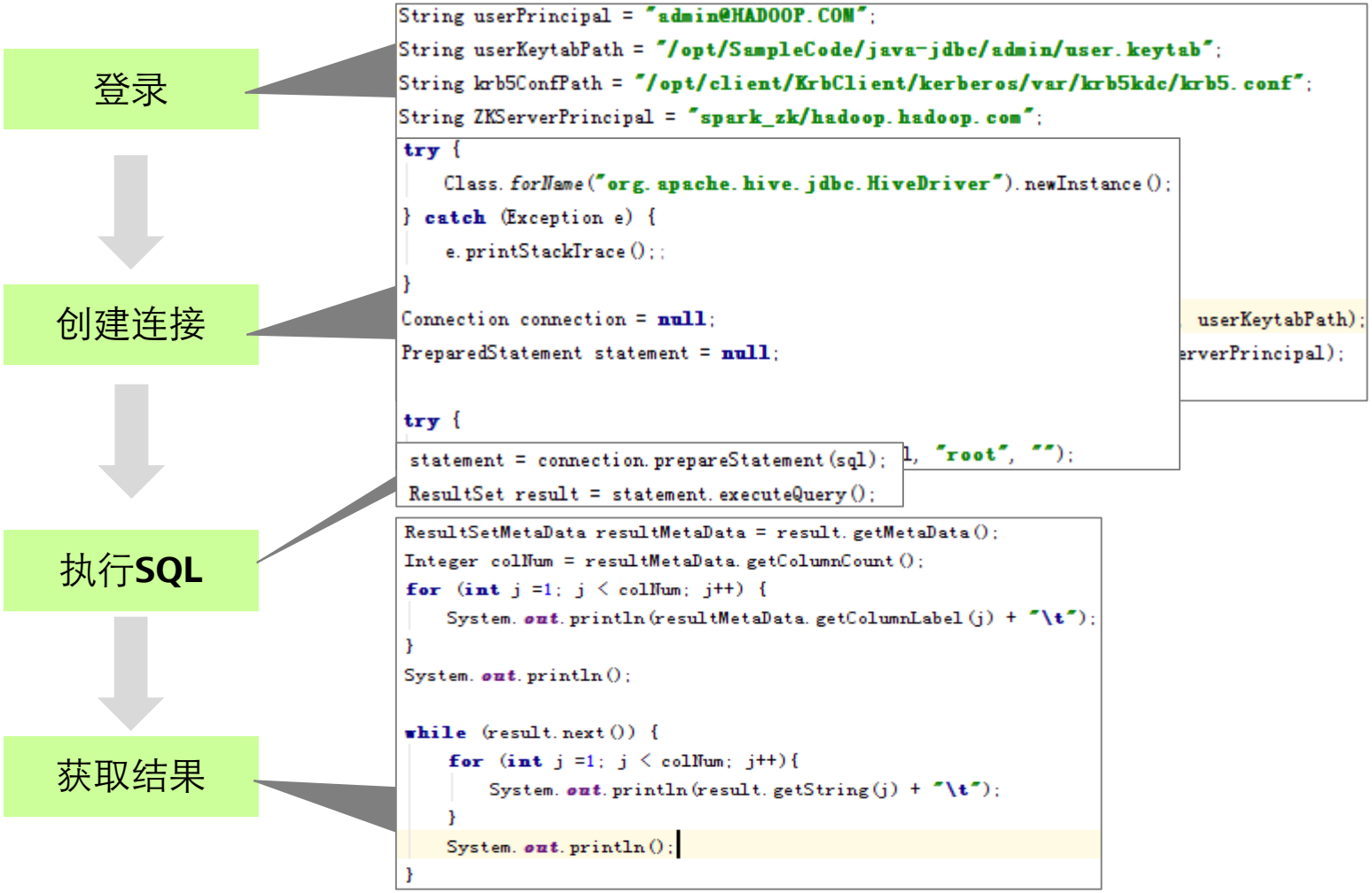
FusionInsight Spark SQL默认采用BigSQLDialect，可同时兼容SQL99和hiveql

开发一个SQL应用—JDBC

- 场景：从**Java**应用中通过**JDBC**向**FusionInsight Spark**提交**SQL**语句，如图：



编写代码



DataFrame介绍

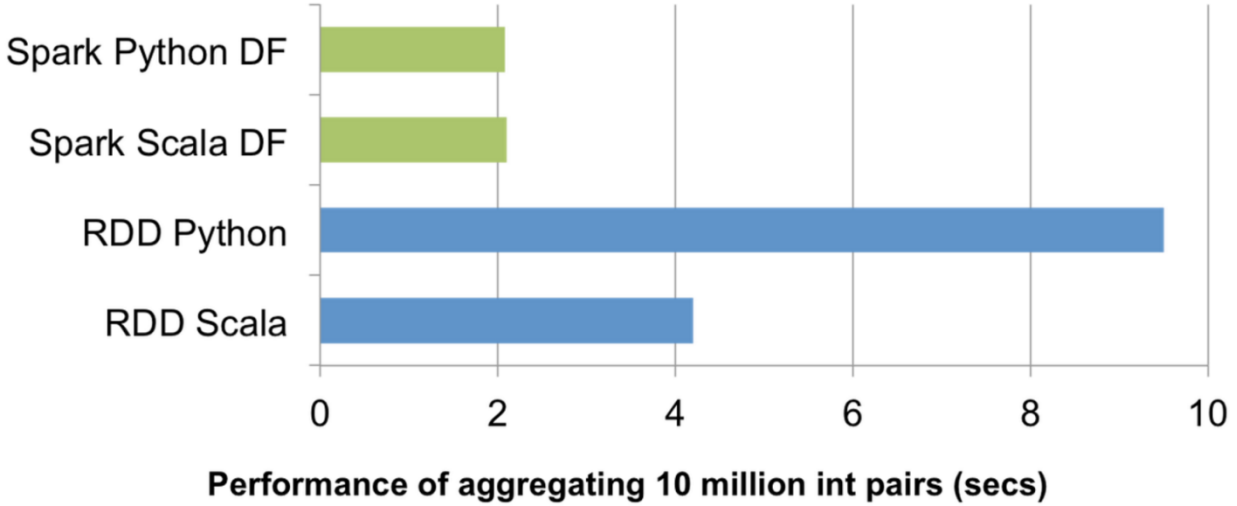
DataFrame：以**RDD**为基础，带有**Schema**信息，类似传统数据库的二维表。注册成表之后可以使用类**SQL**操作（**where**、**join..**）。

RDD

Personal(Name,Age,Tel)
Personal(Name,Age,Tel)
Personal(Name,Age,Tel)

DataFrame

Name	Age	Tel
String	Int	String
String	Int	String
String	Int	String



开发一个SQL应用—DataFrame

场景：通过**DataFrame**接口开发一个统计文本日志中本周末网购停留总时间超过**2**小时的女性网民信息：

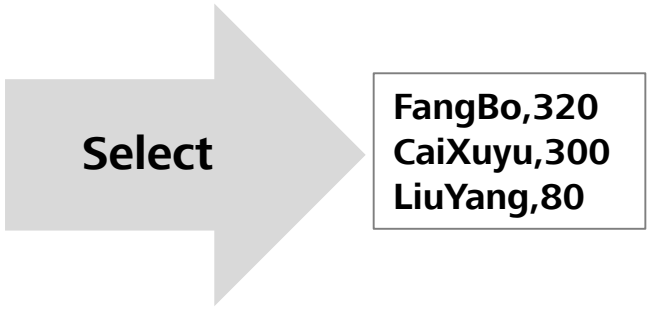
Input/log1.txt

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
....
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

Input/log2.txt

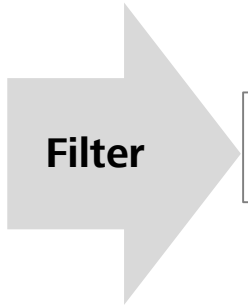
```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
....
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

```
select name,sum(stayTime) as stayTime
from FemaleInfoTable
where gender = 'female'
group by name
```




```
FangBo,320
CaiXuyu,300
LiuYang,80
```

stayTime >= 120




```
FangBo,320
CaiXuyu,300
```

编写代码

登录

```
val userPrincipal = "admin@HADOOP.COM"
val userKeytabPath = "/opt/SampleCode/admin/user.keytab"
val krb5ConfPath = "/opt/client/KrbClient/kerberos/var/krb5kdc/krb5.conf"
val hadoopConf: Configuration  = new Configuration()
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf)
```

注册**Table**

```
sc.textFile(args(0)).map(_.split(","))
  .map(p => FemaleInfo(p(0), p(1), p(2).trim.toInt))
  .toDF.registerTempTable("FemaleInfoTable")
```

执行**SQL**

```
val femaleTimeInfo = sqlContext.sql("select name,sum(stayTime) as " +
  "stayTime from FemaleInfoTable where gender = 'female' group by name")
```

过滤

```
val c = femaleTimeInfo.filter("stayTime >= 120").collect().foreach(println)
```

DataFrame常用Transformation算子

Transformation	含义
filter(conditionExpr: String): DataFrame	根据过滤条件对数据进行过滤。如：**peopleDf.filter(“age > 15”)**。
groupBy(cols: Column*): GroupedData	根据参数列进行分组。如：：**df.groupBy($“department”).avg()**。
join(right: DataFrame, joinExprs: Column): DataFrame	根据指定的表达式进行**Join**。如：**df1.join(df2, $“df1Key” === $“df2Key”)**。
sort(sortCol: String, sortCols: String*): DataFrame	对指定的列排序后，返回一个新的**DataFrame**。如：**df.sort(“sortcol”)**。
select(col: String, cols: String*): DataFrame	查询部分列数据，生成新的**DataFrame**。如：**df.select(“colA”, “colB”)**。
intersect(other: DataFrame): DataFrame	对两个**DataFrame**取交集。
dropDuplicates(): DataFrame	对**DataFrame**中的数据去重。

DataFrame常用Action算子

Action	含义
collect(): Array[Row]	返回所有数据的**Row**数组。
count(): Long	对**DataFrame**中的数据计数。
first(): Row	返回第一条**Row**数据。
show(numRows: Int): Unit	以表格形式显示**DataFrame**中的数据。
take(n: Int): Array[Row]	返回**DataFrame**中的前**n**条数据。

本章小结

本章主要学习了以下内容

- **SparkSQL**架构
- **SparkSQL**使用方式
- **JDBC**连接**SparkSQL**开发样例
- **DataFrame**介绍
- **DataFrame**开发样例

目录

1. 准备工作
2. 第一个Spark应用
   - SparkCore
   - SparkSQL
   - SparkStreaming
3. 调试Spark应用
4. 学习路径/资料（FAQ）

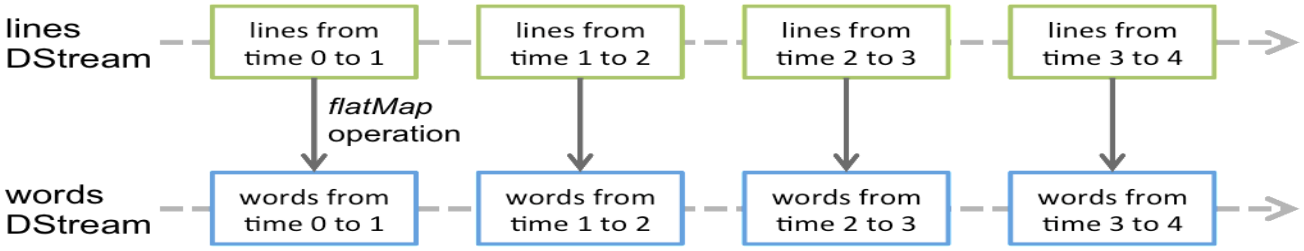
SparkStreaming概述

- **Spark Streaming**是**Spark**核心**API**的一个扩展，它对实时流式数据的处理具有可扩展性、高吞吐量、可容错性等特点。我们可以从**Kafka**、**HDFS**等源获取数据，然后通过由高阶函数**map**、**reduce**、**join**、**window**等组成的复杂算法计算出数据。最后，处理后的数据可以推送到文件系统、数据库、实时仪表盘中。

SparkStreaming原理

- **Spark Streaming**接收实时的输入数据流，然后将这些数据切分为批数据供**Spark**引擎处理，**Spark**引擎将数据生成最终的结果数据。

- 使用**DStream**从**Kafka**和**HDFS**等源获取连续的数据流，**DStreams**由一系列连续的**RDD**组成，每个**RDD**包含确定时间间隔的数据，任何对**DStreams**的操作都转换成对**RDD**的操作。



特性

Spark Streaming的特点

- 高吞吐量，容错能力强
- 数据采集逐条进行，数据处理分批进行

优点

- 粗粒度处理方式可以快速处理小批量数据
- 可以确保“处理且仅处理一次”，更方便实现容错恢复机制
- **DStream**操作基于**RDD**操作，降低学习成本

缺点

- 粗粒度处理引入不可避免的延迟

数据源和可靠性

Spark Streaming数据源

- 基本源：**HDFS**等文件系统、**Socket**连接等
- 高级源：**Kafka**等
- 自定义源：需要实现用户自定义**receiver**

可靠性（二次开发）

- **Reliable Receiver**：能正确应答一个可靠源（如**kafka**等），数据已经被接收并且被正确复制到**Spark**中
- 设置**CheckPoint**
- 确保**Driver**可以自动重启
- 使用**Write Ahead Log**功能

SparkStreaming代码流程

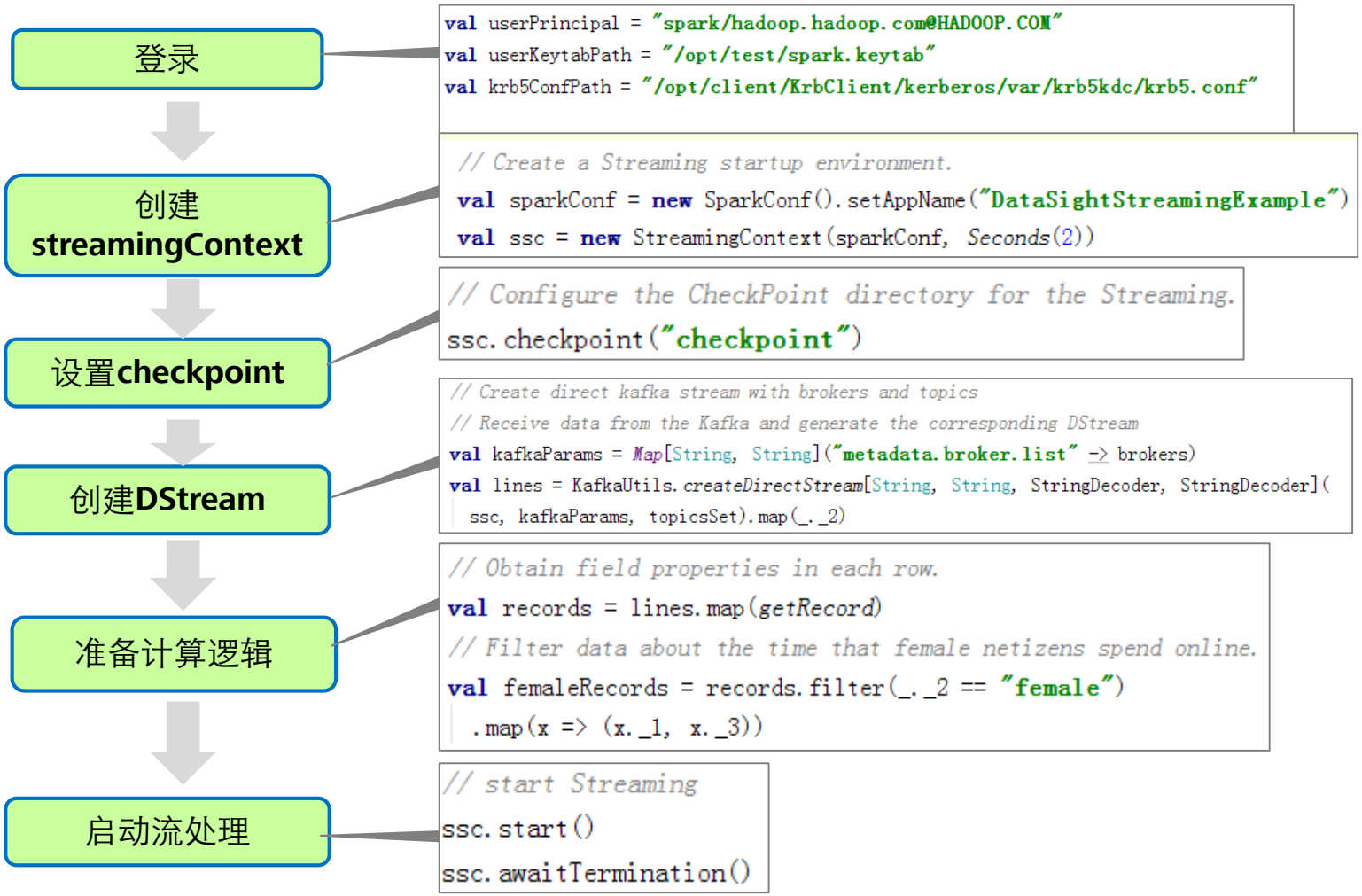
常见业务代码逻辑

1. 创建**StreamingContext**。
2. 定义输入源。
3. 准备应用计算逻辑。
4. 使用**streamingContext.start()**方法接收和处理数据。
5. 使用**streamingContext.stop()**方法停止流计算。

注意事项

- 在**JVM**中，同一时间只能有一个**StreamingContext**处于活跃状态。
- 可以在一个应用中创建多个输入**DStream**来接收多个数据流，每一个输入流**DStream**与一个**Receiver**对象关联，**Receiver**从源中获取数据，并将数据存入内存中用于处理。
- **Receiver**作为常驻进程运行在**executor**中，将占用一个核。

简单样例



常用DStream算子

Transformation	含义
map(func)	利用函数func处理原DStream的每个元素，返回一个新的Dstream。
flatMap(func)	与map相似，但是每个输入项可用被映射为0个或者多个输出项。
filter(func)	返回一个新的DStream，它仅仅包含源DStream中满足函数func的项。
repartition(numPartitions)	通过创建更多或者更少的partition改变这个DStream的并行级别(level of parallelism)。
union(otherStream)	返回一个新的DStream,它包含源DStream和otherStream的联合元素。
count()	通过计算源DStream中每个RDD的元素数量，返回一个包含单元素(single-element)RDDs的新Dstream。
reduce(func)	利用函数func聚集源DStream中每个RDD的元素，返回一个包含单元素(single-element)RDDs的新DStream。函数应该是相关联的，以使计算可以并行化。
countByValue()	这个算子应用于元素类型为K的DStream上，返回一个（K,long）对的新DStream，每个键的值是在原DStream的每个RDD中的频率。

常用DStream算子

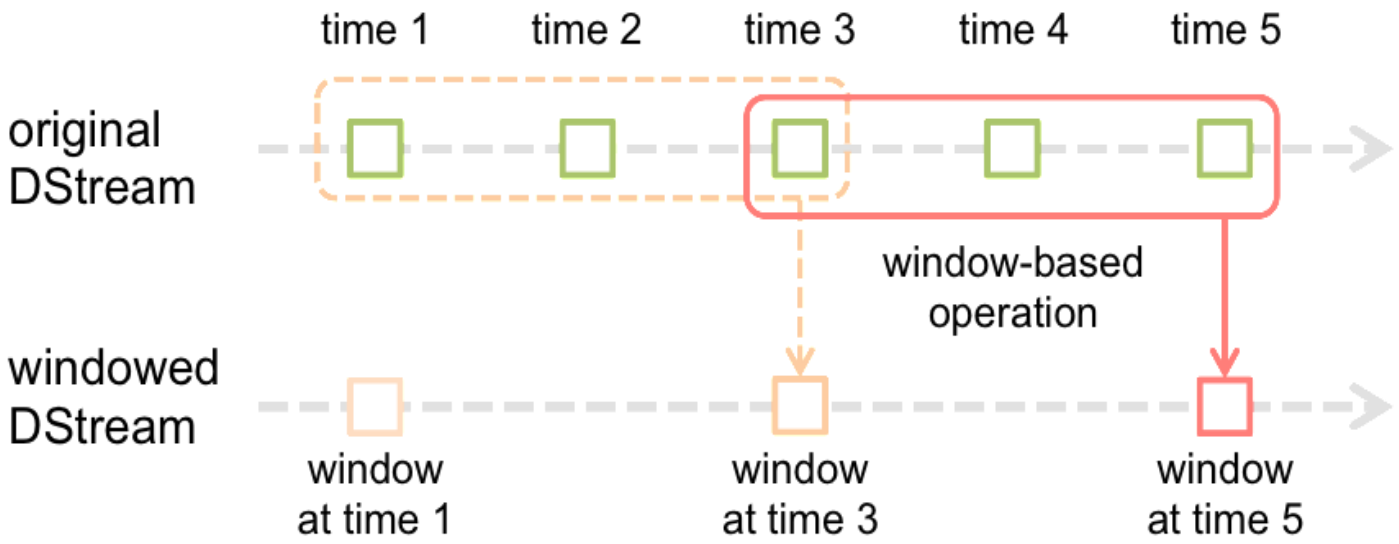
Transformation	含义
reduceByKey(func, [numTasks])	当在一个由**(K,V)**对组成的**DStream**上调用这个算子，返回一个新的由**(K,V)**对组成的**DStream**，每一个**key**的值均由给定的**reduce**函数聚集起来。注意：在默认情况下，这个算子利用了**Spark**默认的并发任务数去分组。你可以用**numTasks**参数设置不同的任务数。
join(otherStream, [numTasks])	当应用于两个**DStream**（一个包含（**K,V**）对,一个包含**(K,W)**对），返回一个包含**(K, (V, W))**对的新**Dstream**。
cogroup(otherStream, [numTasks])	当应用于两个**DStream**（一个包含（**K,V**）对,一个包含**(K,W)**对），返回一个包含**(K, Seq[V], Seq[W])**的元组。
transform(func)	通过对源**DStream**的每个**RDD**应用**RDD-to-RDD**函数，创建一个新的**DStream**。这个可以在**DStream**中的任何**RDD**操作中使用。
updateStateByKey(func)	利用给定的函数更新**DStream**的状态，返回一个新**"state"**的**DStream**。

常用DStream输出操作

Output Operation	Meaning
print()	在DStream的每个批数据中打印前10条元素，这个操作在开发和调试中都非常有用。在Python API中调用pprint()。
saveAsObjectFiles(prefix, [suffix])	保存DStream的内容为一个序列化的文件SequenceFile。每一个批间隔的文件的文件名基于prefix和suffix生成。"prefix-TIME_IN_MS[.suffix]"，在Python API中不可用。
saveAsTextFiles(prefix, [suffix])	保存DStream的内容为一个文本文件。每一个批间隔的文件的文件名基于prefix和suffix生成。“prefix-TIME_IN_MS[.suffix]”。
saveAsHadoopFiles(prefix, [suffix])	保存DStream的内容为一个hadoop文件。每一个批间隔的文件的文件名基于prefix和suffix生成。"prefix-TIME_IN_MS[.suffix]"，在Python API中不可用。
foreachRDD(func)	在从流中生成的每个RDD上应用函数func的最通用的输出操作。这个函数应该推送每个RDD的数据到外部系统，例如保存RDD到文件或者通过网络写到数据库中。需要注意的是，func函数在驱动程序中执行，并且通常都有RDD action在里面推动RDD流的计算。

窗口操作

Spark Streaming支持窗口计算，允许用户在一个滑动窗口数据上应用**transformation**算子。窗口在源**DStream**上滑动，合并和操作落入窗内的源**RDDs**，产生窗口化的**DStream**的**RDDs**。



窗口操作的常用算子

Transformation	含义
window(windowLength, slideInterval)	根据**window**操作的**2**个参数得到新的**Dstream**。
countByWindow(windowLength, slideInterval)	基于**window**操作的**count**操作。
reduceByWindow(func, windowLength, slideInterval)	基于**window**操作的**reduce**操作。
reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])	基于**window**操作的**reduceByKey**操作。
reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])	跟**reduceByKeyAndWindow**方法类似，更有效率，**invFunc**方法跟**func**方法的参数返回值一样，表示从**window**离开的数据。
countByValueAndWindow(windowLength, slideInterval, [numTasks])	基于**window**操作的**countByValue**操作。

缓存和持久化

- **DStreams**默认持久化级别是存储序列化数据到内存中。
- 使用**persist()**方法可以自动地持久化**DStream**中的**RDD**到内存中。
- **reduceByWindow**和**reduceByKeyAndWindow**等窗口操作、**updateStateByKey**操作，持久化是默认的，不需要开发者调用**persist()**方法。
- 通过网络（如**kafka**等）获取的输入数据流，默认的持久化策略是复制数据到两个不同的节点以容错。

Checkpoint

Spark Streaming可以**checkpoint**足够的信息到容错存储系统中， 以使系统崩溃后从故障中恢复。

- **Metadata checkpoint**：保存流计算的定义信息到**HDFS**中，用来恢复应用程序中运行**worker**的节点的故障。元数据包括：
  - **Configuration** ： 创建**Spark Streaming**应用程序的配置信息。
  - **DStream operations** ： 定义**Streaming**应用程序的操作集合。
  - **Incomplete batches**：操作存在队列中的未完成的批。
- **Data checkpoint** ： 保存生成的**RDD**到**HDFS**中，在有状态**transformation**（如结合跨多个批次的数据）中是必须的。

Checkpoint

如何配置Checkpoint

- 创建**CheckPoint**：使用**streamingContext.checkpoint(checkpointDirectory)**在**HDFS**中设置目录保存**checkpoint**信息。
- 从**CheckPoint**恢复：**StreamingContext.getOrCreate(checkpointDirectory, functionToCreateContext _)**。

CheckPoint的性能损耗

- **RDD**的**checkpointing**有存储成本。
- 批数据的处理时间增加。

Spark Steaming性能调优

- 设置合理的批处理时间**(batchDuration)**
- 设置合理的数据接收并行度
  - 设置多个**Receiver**接收数据
  - 设置合理的**Receiver**阻塞时间
- 设置合理的数据处理并行度
- 使用**Kryo**系列化
- 内存调优
  - 设置持久化级别减少**GC**开销
  - 使用并发的标记-清理**GC**算法减少**GC**暂停时间

本章小结

本章主要学习了以下内容

- **SparkStreaming**架构原理
- **SparkStreaming**可靠性特性
- **SparkStreaming**窗口操作
- **SparkStreaming**开发样例
- **SparkStreaming**性能调优

目录

调试Spark应用

- 与普通的**Java**程序一样，分布式的**Spark**应用同样支持远程调试，不同之处在于，**JVM**的远程调试参数需要加到**Driver**或者**Executor**的启动参数中，同时，远程调试前，需先找到相应的进程在集群中的位置。相关参数：
  - **spark.executor.extraJavaOptions**：**Executor**启动参数。
  - **spark.driver.extraJavaOptions**：**yarn-client**模式下，**Driver**的启动参数。
  - **spark.yarn.cluster.driver.extraJavaOptions**：**yarn-cluster**模式下**Driver**的启动参数。

目录

更多资源

- https://spark.apache.org/docs/1.5.1/programming-guide.html
- https://spark.apache.org/docs/1.5.1/sql-programming-guide.html
- https://spark.apache.org/docs/1.5.1/streaming-programming-guide.html
- http://support.huawei.com/learning/Index!toTrainIndex
- http://support.huawei.com/enterprise/servicecenter?lang=zh

思考题

1. 相比**MapReduce**，**Spark**有哪些优势?
2. **Spark**有哪几种参数配置方式?
3. 如何让一个**Spark**程序运行得更快?

习题

- 填空

1. **val rdd = sc.parallelize(Array(1 to 10), 5)** 得到的**RDD**有____个分区。
2. **SparkStreaming**属于____型的流处理引擎，相比**Storm**，有更高的___。
3. 子**RDD**的分区依赖于父**RDD**的所有分区属于__依赖。
4. **FusionInsight Spark SQL**默认采用____，可同时兼容**SQL99**和**hiveql**。

谢谢