

1 Spark 概述

1.1 Spark 定义

构建与计算集群之上支持大数据集的快速的通用的处理引擎

a) 快速: DAG、Memory

b) 通用: 集成Spark SQL、Streaming、Graphic、R、Batch Process

c) 运行方式:

StandAlone

YARN

Mesos

AWS

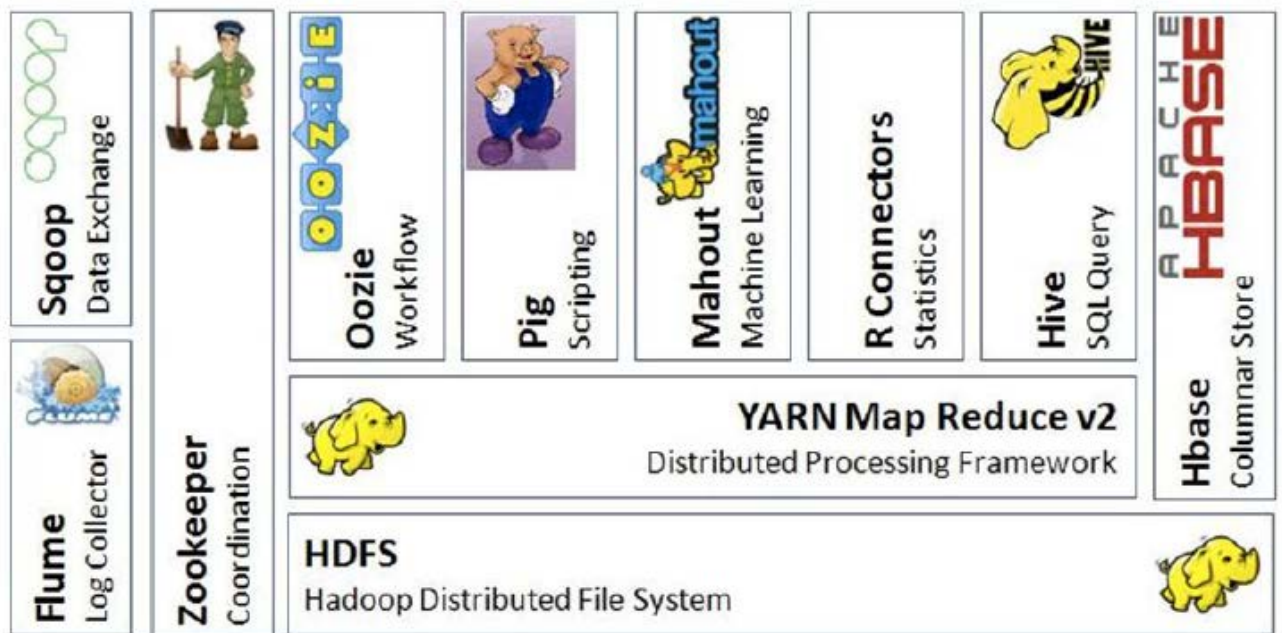
d) 数据来源:

Hdfs Hbase Tachyon Cassandra Hive

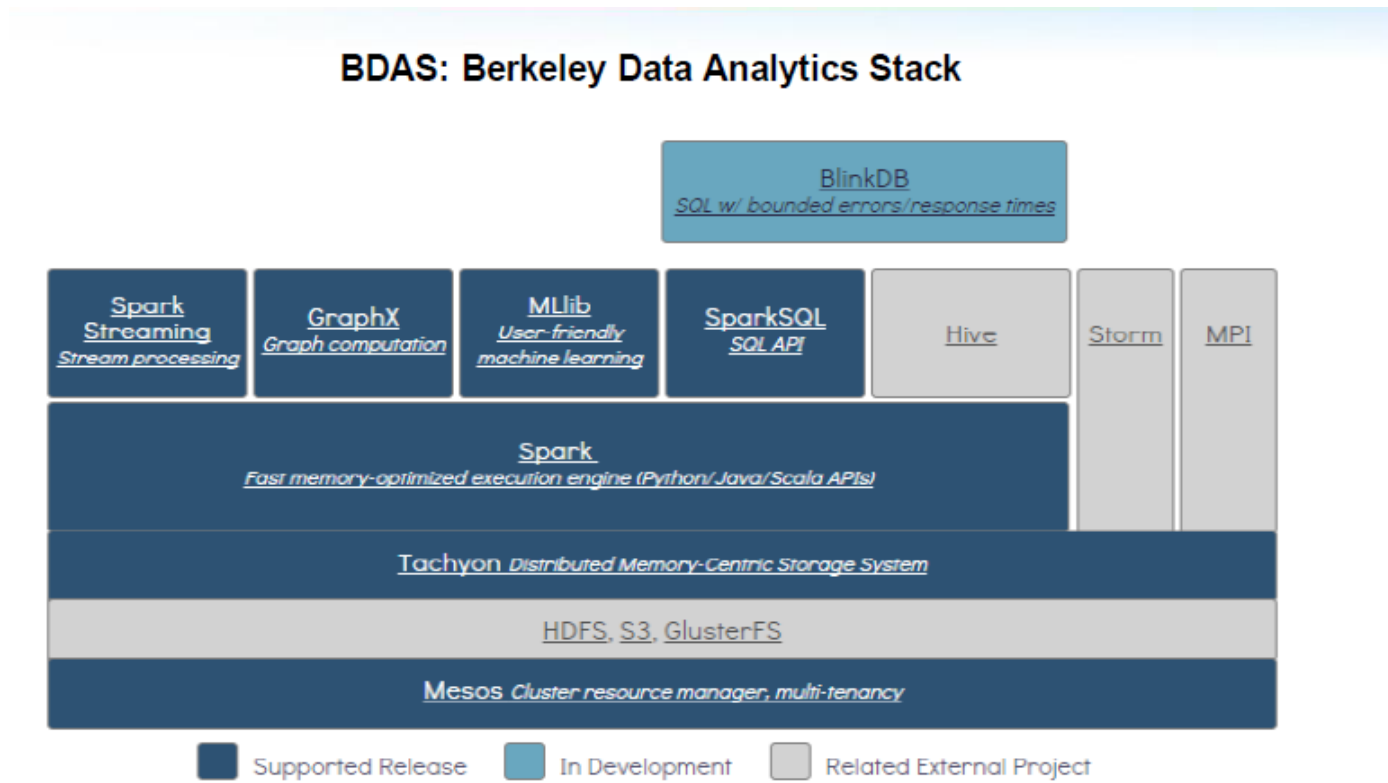
and Any Hadoop Data Source

1.2 Spark 协议栈

1.2.1 Hadoop 生态系统



1.2.2 Spark 协议栈



1.2.3 Spark VS Mapreduce

MapReduce	Spark
数据存储结构：磁盘hdfs文件系统的split	使用内存构建弹性分布式数据集RDD，对数据进行运算和cache
编程范式：Map + Reduce	DAG(有向无环图)：Transformation + action
计算中间数据落磁盘，io及序列化、反序列化代价大	计算中间数据在内存中维护，存取速度是磁盘的多个数量级
Task以进程的方式维护，任务启动就有数秒	Task以线程的方式维护，对小数据集的读取能达到亚秒级的延迟

MapReduce 与Spark比较

1. what? 处理对象

a) MapReduce: 基于磁盘**File**的大数据处理系统

b) Spark: 基于**RDD**(弹性分布式数据集)，可以显示的将RDD数据存储到磁盘和内存中

2. where (软硬件上下文)?

a) MapReduce: Disk

b) Spark: Mem

3. when? (应用场景)

a) MapReduce: 可以处理超大规模数据，适合日志分析挖掘等迭代较少的长任务需求，结合了数据的分布式的计算

b) spark: 适合数据的挖掘，机器学习等多轮迭代式计算任务

容错性:

a) 数据容错性

MapReduce: 容错性基于HDFS 冗余机制 ->安全模式->数据校验->元数据保护

spark: 容错性基于RDD, spark容错性比mapreduce容错性低，但在处理效率上优势比较明显

b) 节点容错性

1.3 Spark 安装配置

===== SetUp HDFS=====

Configuration

```

hadoop-env.sh
    export JAVA_HOME=/opt/modules/jdk1.7.0_67
core-site.xml
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop-spark.dragon.org:8020</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/data02/hadoop-2.6.0-cdh5.4.0/data/tmp</value>
    </property>
hdfs-site.xml
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
slaves
    hadoop-spark.dragon.org
Start HDFS
    NameNode Format
        bin/hdfs namenode -format
    Start NN/DN
        sbin/hadoop-daemon.sh start namenode
        sbin/hadoop-daemon.sh start datanode
    WEB UI
        http://hadoop-spark.dragon.org:50070

===== SetUp Spark=====
Configuration
    spark-env.sh
        HADOOP_CONF_DIR=/opt/data02/hadoop-2.6.0-cdh5.4.0/etc/hadoop
        JAVA_HOME=/opt/modules/jdk1.7.0_67
        SCALA_HOME=/opt/modules/scala-2.10.4
        #####
        SPARK_MASTER_IP=192.168.0.222
        SPARK_MASTER_PORT=7077
        SPARK_MASTER_WEBUI_PORT=8080
        SPARK_WORKER_CORES=1
        SPARK_WORKER_MEMORY=1000m
        SPARK_WORKER_PORT=7078
        SPARK_WORKER_WEBUI_PORT=8081
        SPARK_WORKER_INSTANCES=1
    slaves
        hadoop-spark.dragon.org
    spark-defaults.conf
        spark.master                                spark://hadoop-spark.dragon.org:7077
Start Spark
    Start Master
        sbin/start-master.sh
    Start Slaves

```

```
    sbin/start-slaves.sh
WEB UI
    http://hadoop-spark.dragon.org:8080
    Start History Server:
        sbin/start-history-server.sh
        sc.stop()
```

===== Test Spark=====

```
scala> val rdd=sc.textFile("hdfs://hadoop-spark.dragon.org:8020/user/hadoop/data/wc.input")
```

```
scala> rdd.cache()
```

```
scala> val wordcount=rdd.flatMap(_.split(" ")).map(x=>(x,1)).reduceByKey(_+_)
```

```
scala> wordcount.take(10)
```

```
scala> val wordsort=wordcount.map(x=>(x._2,x._1)).sortByKey(false).map(x=>(x._2,x._1))
```

```
scala> wordsort.take(10)
```

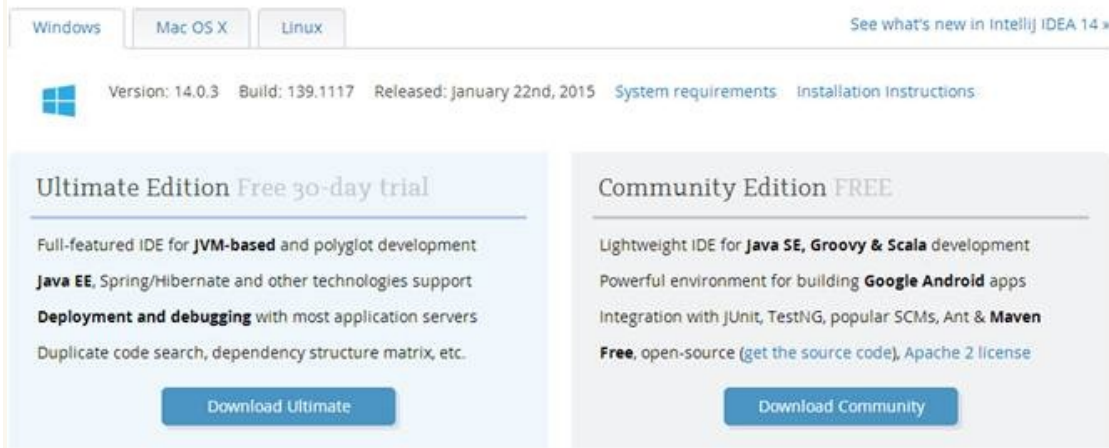
1.4 开发环境搭建

1 安装 IntelliJ IDEA

IDEA 全称 IntelliJ IDEA，是 java 语言开发的集成环境，IntelliJ 在业界被公认为最好的 java 开发工具之一，尤其在智能代码助手、代码自动提示、重构、J2EE 支持、Ant、JUnit、CVS 整合、代码审查、创新的 GUI 设计等方面的功能可以说是超常的。IDEA 是 JetBrains 公司的产品，这家公司总部位于捷克共和国的首都布拉格，开发人员以严谨著称的东欧程序员为主。

IDEA 每个版本提供 Community 和 Ultimate 两个版本，如下图所示，其中 Community 是完全免费的，而 Ultimate 版本可以使用 30 天，过这段时间后需要收费。从安装后使用对比来看，下载一个 Community 版本足够了。

Download IntelliJ IDEA 14



Windows Mac OS X Linux See what's new in IntelliJ IDEA 14 »

Version: 14.0.3 Build: 139.1117 Released: January 22nd, 2015 System requirements Installation Instructions

Ultimate Edition Free 30-day trial

Full-featured IDE for **JVM-based** and polyglot development
Java EE, Spring/Hibernate and other technologies support
Deployment and debugging with most application servers
Duplicate code search, dependency structure matrix, etc.

[Download Ultimate](#)

Community Edition FREE

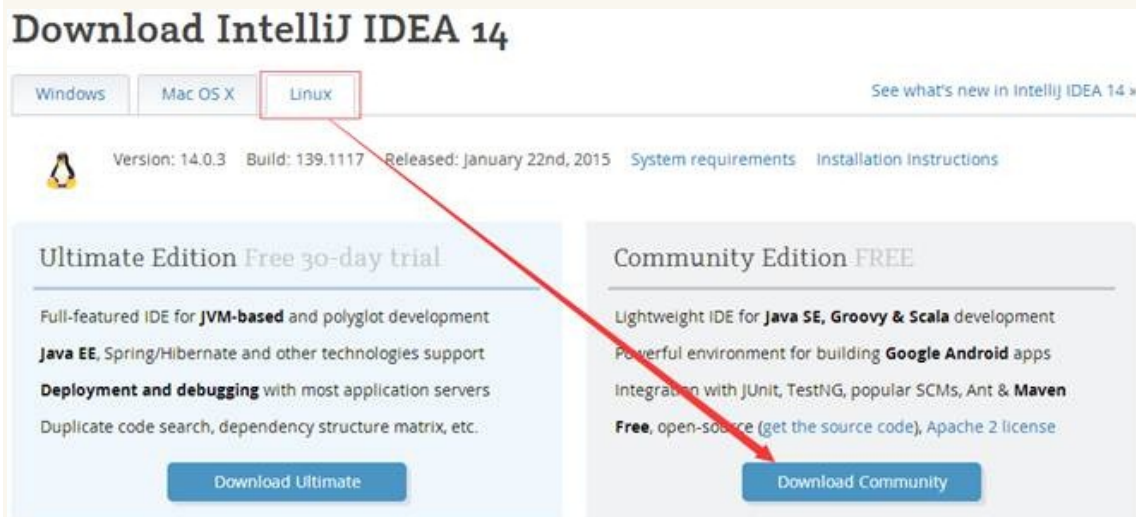
Lightweight IDE for **Java SE, Groovy & Scala** development
Powerful environment for building **Google Android** apps
Integration with JUnit, TestNG, popular SCMs, Ant & **Maven**
Free, open-source ([get the source code](#)), Apache 2 license

[Download Community](#)

1.1 安装软件

1.1.1 下载 IDEA 安装文件

可以到 JetBrains 官网 <http://www.jetbrains.com/idea/download/> ,选择最新的安装文件。由于以后的练习需要在 Linux 开发 Scala 应用程序 ,选择 Linux 系统 IntelliJ IDEA14 , 如下图所示 :



Download IntelliJ IDEA 14

Windows Mac OS X Linux See what's new in IntelliJ IDEA 14 »

Version: 14.0.3 Build: 139.1117 Released: January 22nd, 2015 System requirements Installation Instructions

Ultimate Edition Free 30-day trial

Full-featured IDE for **JVM-based** and polyglot development
Java EE, Spring/Hibernate and other technologies support
Deployment and debugging with most application servers
Duplicate code search, dependency structure matrix, etc.

[Download Ultimate](#)

Community Edition FREE

Lightweight IDE for **Java SE, Groovy & Scala** development
Powerful environment for building **Google Android** apps
Integration with JUnit, TestNG, popular SCMs, Ant & **Maven**
Free, open-source ([get the source code](#)), Apache 2 license

[Download Community](#)

【注】在该系列配套资源的 install 目录下分别提供了 ideaIC-14.0.2.tar.gz(社区版)和 ideaIU-14.0.2.tar.gz(正式版)安装文件 , 对于 Scala 开发来说两个版本区别不大

1.1.2 解压缩并移动目录

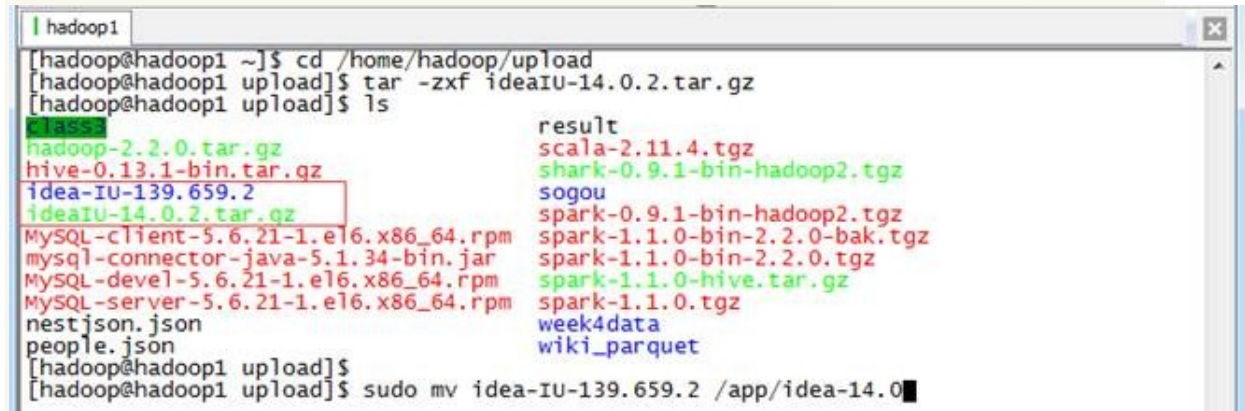
把下载的安装文件上传到目标机器 , 用如下命令解压缩 IntelliJ IDEA 安装文件 , 并迁

移到/app 目录下：

```
cd /home/hadoop/upload
```

```
tar -zxf ideaIU-14.0.2.tar.gz
```

```
sudo mv idea-IU-139.659.2 /app/idea-IU
```



```
hadoop1
[hadoop@hadoop1 ~]$ cd /home/hadoop/upload
[hadoop@hadoop1 upload]$ tar -zxf ideaIU-14.0.2.tar.gz
[hadoop@hadoop1 upload]$ ls
class  result
hadoop-2.2.0.tar.gz  scala-2.11.4.tgz
hive-0.13.1-bin.tar.gz  shark-0.9.1-bin-hadoop2.tgz
idea-IU-139.659.2  sogou
ideaIU-14.0.2.tar.gz  spark-0.9.1-bin-hadoop2.tgz
mysql-client-5.6.21-1.el6.x86_64.rpm  spark-1.1.0-bin-2.2.0-bak.tgz
mysql-connector-java-5.1.34-bin.jar  spark-1.1.0-bin-2.2.0.tgz
mysql-devel-5.6.21-1.el6.x86_64.rpm  spark-1.1.0-hive.tar.gz
mysql-server-5.6.21-1.el6.x86_64.rpm  spark-1.1.0.tgz
nestjson.json  week4data
people.json  wiki_parquet
[hadoop@hadoop1 upload]$
[hadoop@hadoop1 upload]$ sudo mv idea-IU-139.659.2 /app/idea-14.0
```

1.1.3 配置/etc/profile 环境变量

使用如下命令打开/etc/profile 文件：

```
sudo vi /etc/profile
```

确认 JDK 配置变量正确配置 `export JAVA_HOME=/usr/lib/java/jdk1.7.0_55`

```
export PATH=$PATH:$JAVA_HOME
```

```
export JAVA_HOME=/usr/lib/java/jdk1.7.0_55
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar

export PIG_HOME=/usr/local/pig-0.13.0
export PIG_CLASSPATH=/usr/local/hadoop-1.1.2/conf
export PATH=$PATH:/usr/local/hadoop-1.1.2/bin:$PIG_HOME/bin
```

1.2 配置 Scala 环境

1.2.1 启动 IntelliJ IDEA

可以通过两种方式启动 IntelliJ IDEA：

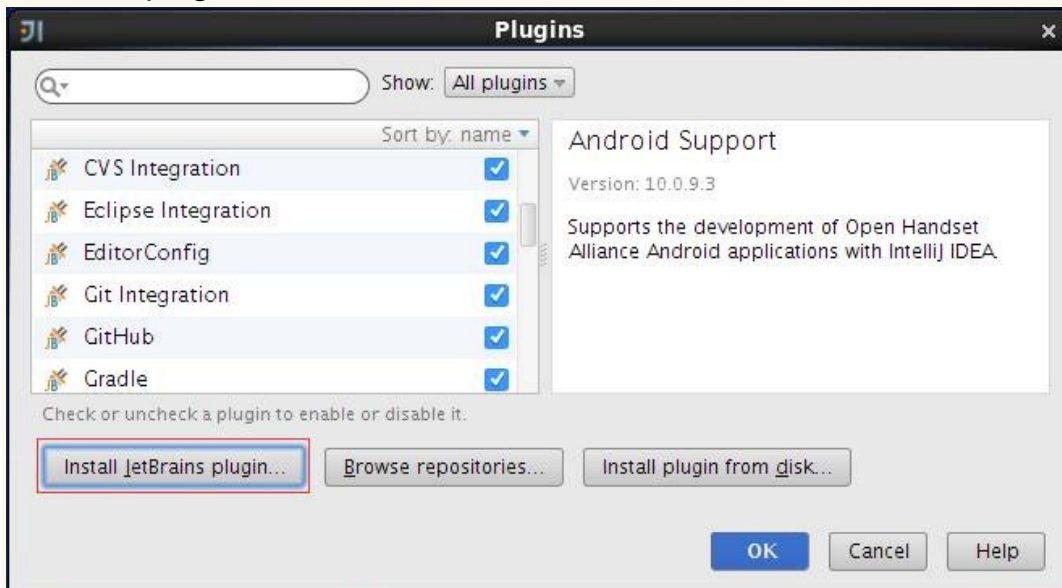
- 到 IntelliJ IDEA 安装所在目录下 进入 bin 目录双击 idea.sh 启动 IntelliJ IDEA；
- 在命令行终端中，进入 \$IDEA_HOME/bin 目录，输入 ./idea.sh 进行启动

IDEA 初始启动目录如下，IDEA 默认情况下并没有安装 Scala 插件，需要手动进行安装，安装过程并不复杂，下面将演示如何进行安装。

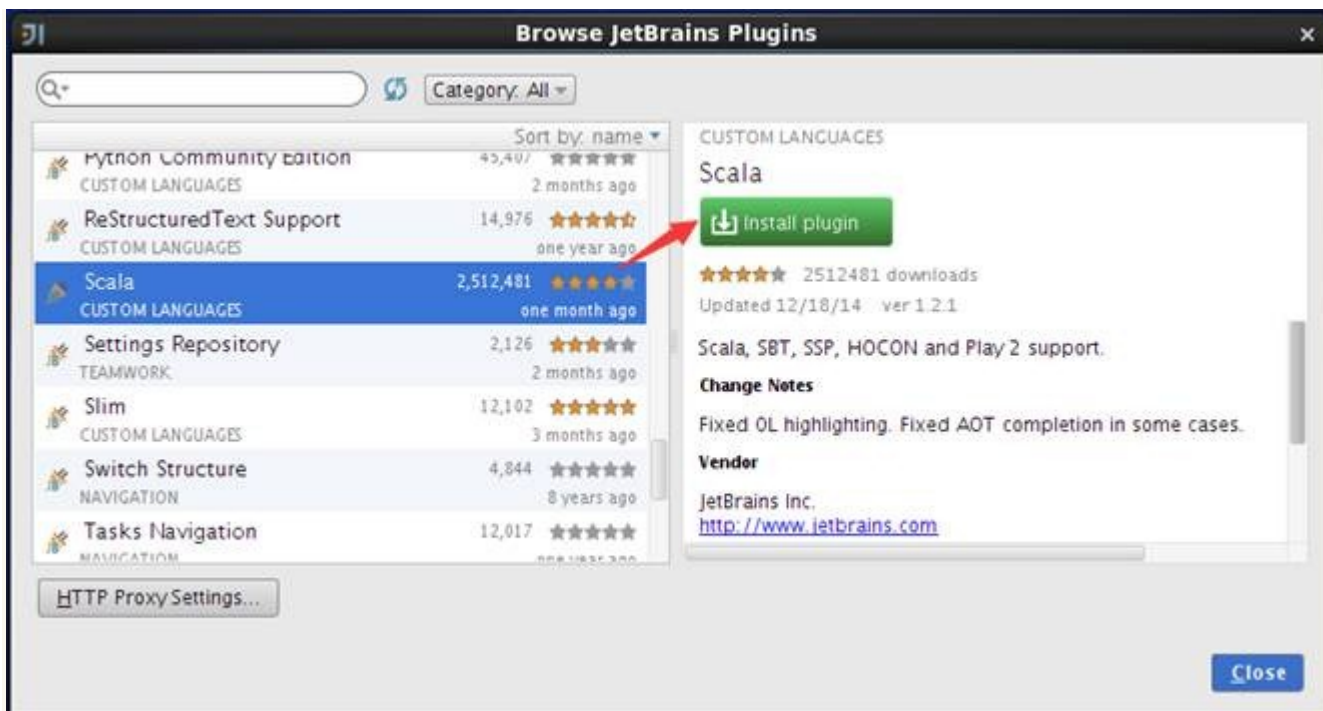


1.2.2 下载 Scala 插件

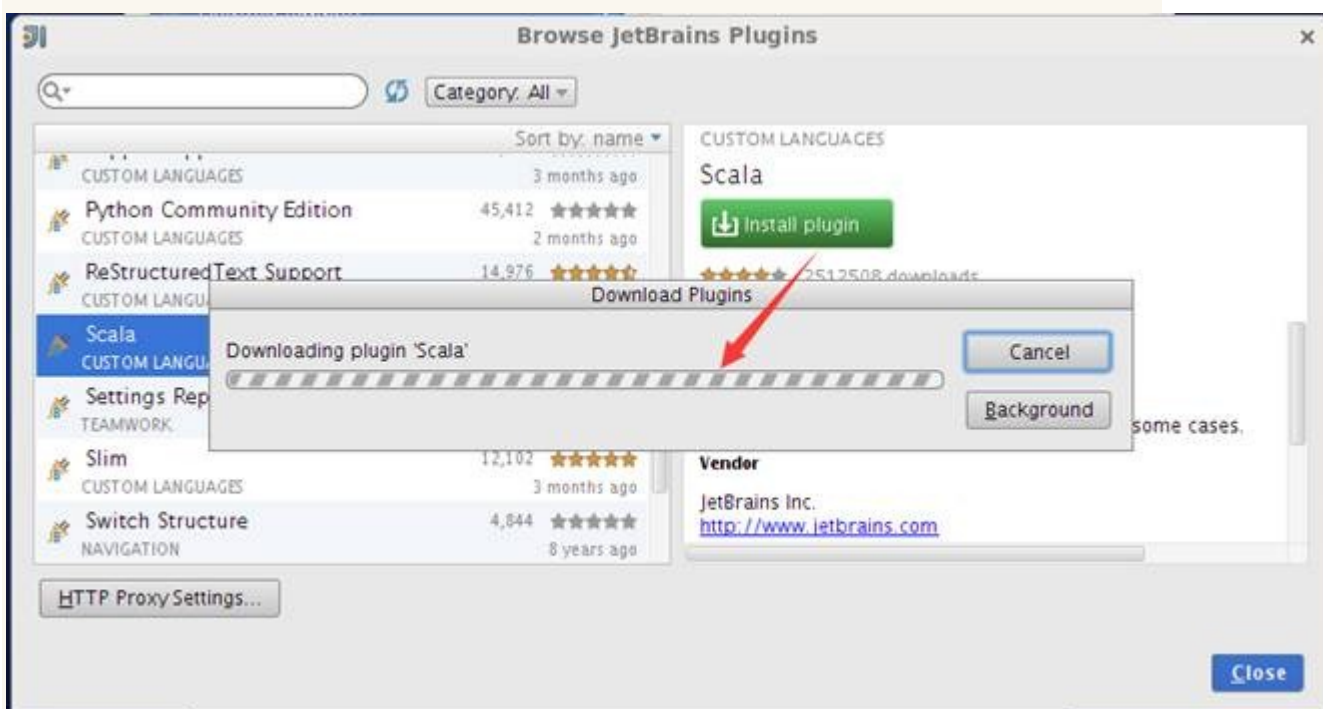
参见上图，在启动界面上选择“Configure-->Plugins”选项，然后弹出插件管理界面，在该界面上列出了所有安装好的插件，由于 Scala 插件没有安装，需要点击“Install JetBrains plugins”进行安装，如下图所示：



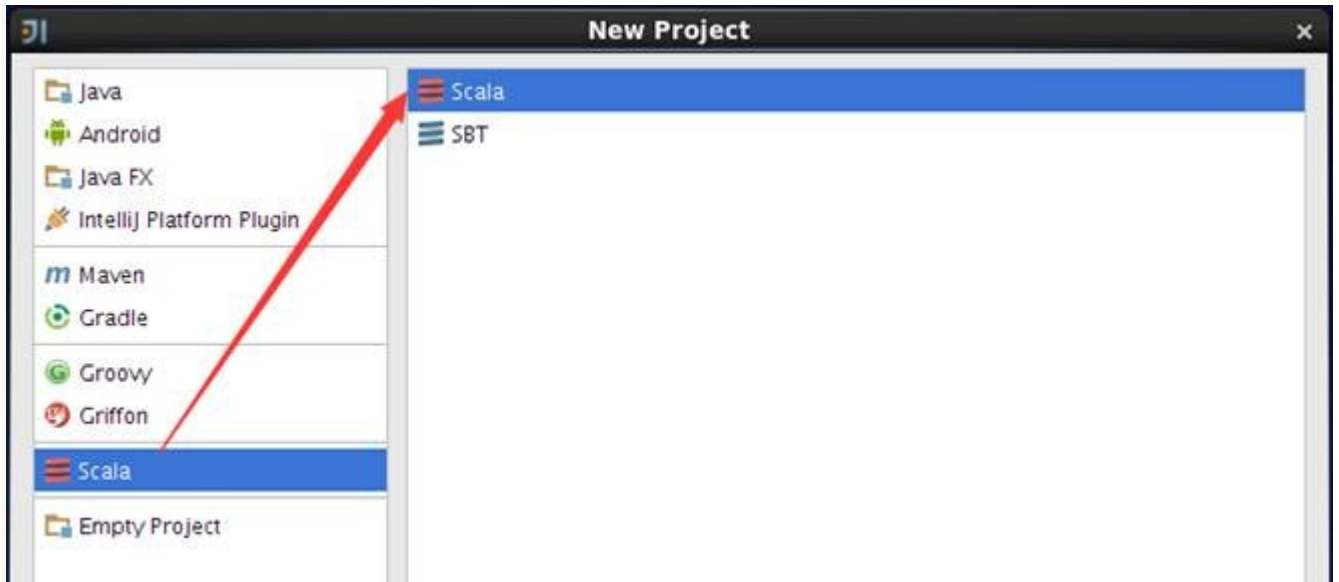
待安装的插件很多，可以通过查询或者字母顺序找到 Scala 插件，选择插件后在界面的右侧出现该插件的详细信息，点击绿色按钮“Install plugin”安装插件，如下图所示：



安装过程将出现安装进度界面，通过该界面了解插件安装进度，如下图所示：

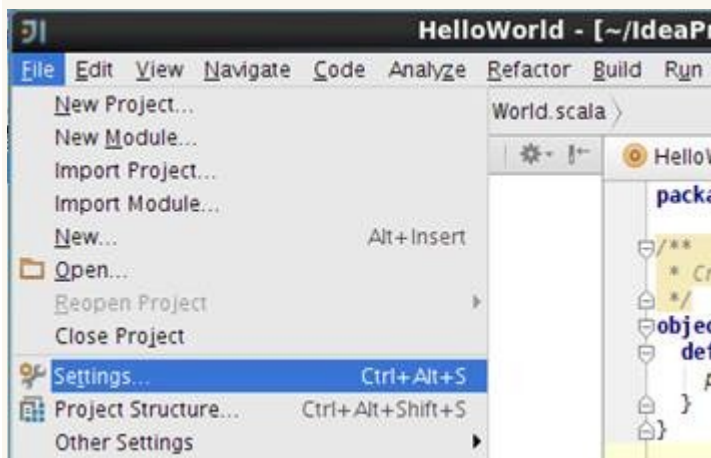


安装插件后，在启动界面中选择创建新项目，弹出的界面中将会出现"Scala"类型项目，选择后将出现提示创建的项目是仅 Scala 代码项目还是 SBT 代码项目，如下图所示：

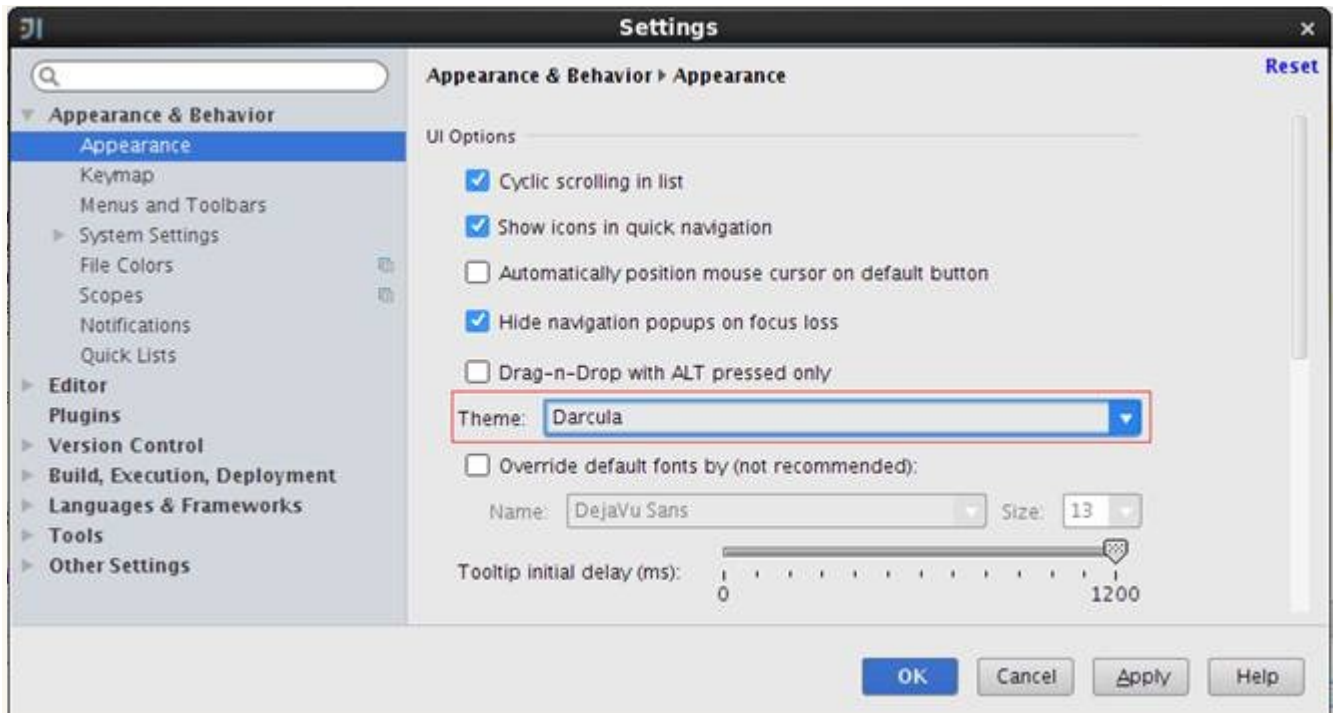


1.2.3 设置界面主题

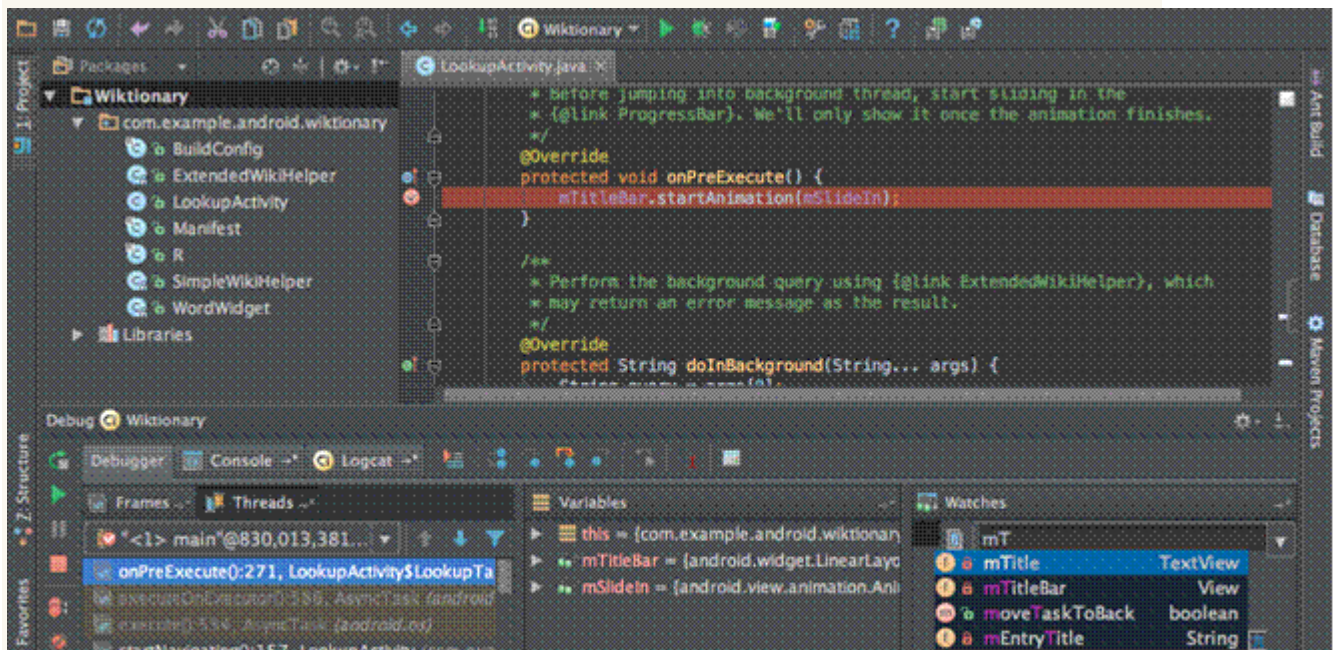
从 IntelliJ IDEA12 开始起推出了 Darcula 主题的全新用户界面，该界面以黑色为主题风格得到很多开发人员的喜爱，下面我们将介绍如何进行配置。在主界面中选择 File 菜单，然后选择 Setting 子菜单，如下图所示：



在弹出的界面中选择 Appearance & Behavior 中 Appearance，其中 Theme 中选择 Darcula 主题，如下图所示：



保存该主题重新进入，可以看到如下图样式的开发工具，是不是很酷！

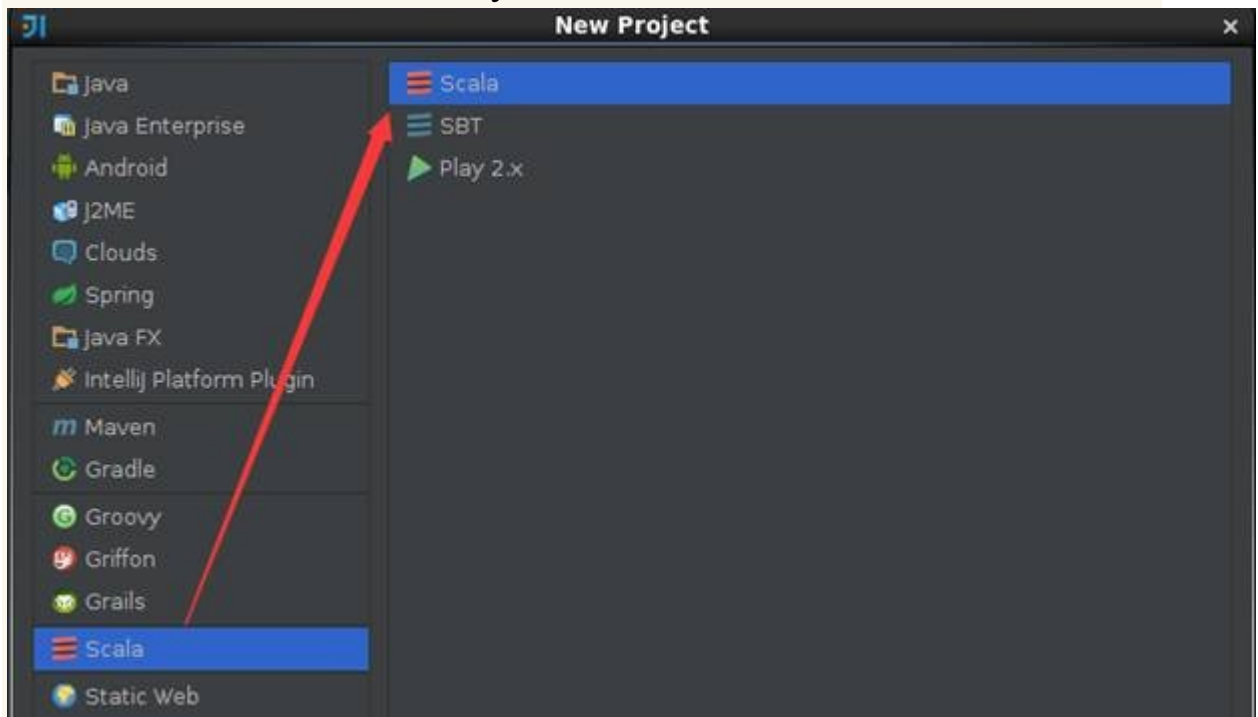


2 使用 IDEA 编写例子

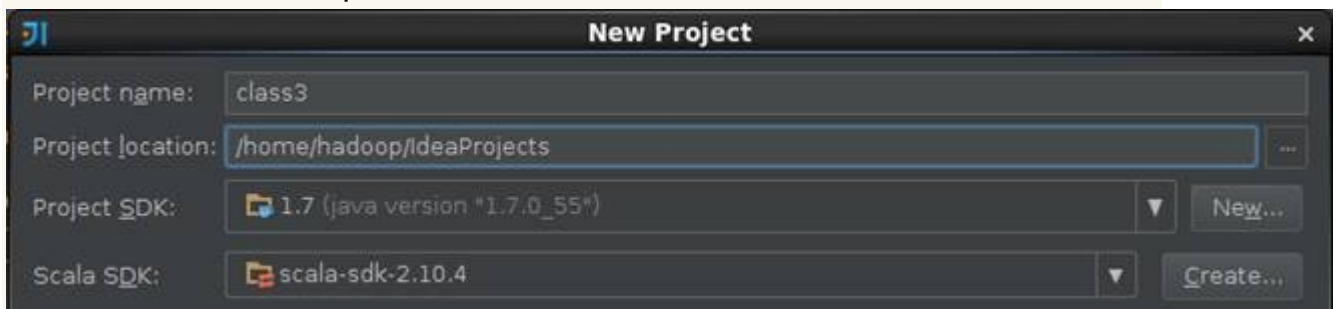
2.1 创建项目

2.1.1 设置项目基本信息

在 IDEA 菜单栏选择 File-> New Project，出现如下界面，选择创建 Scala 项目：

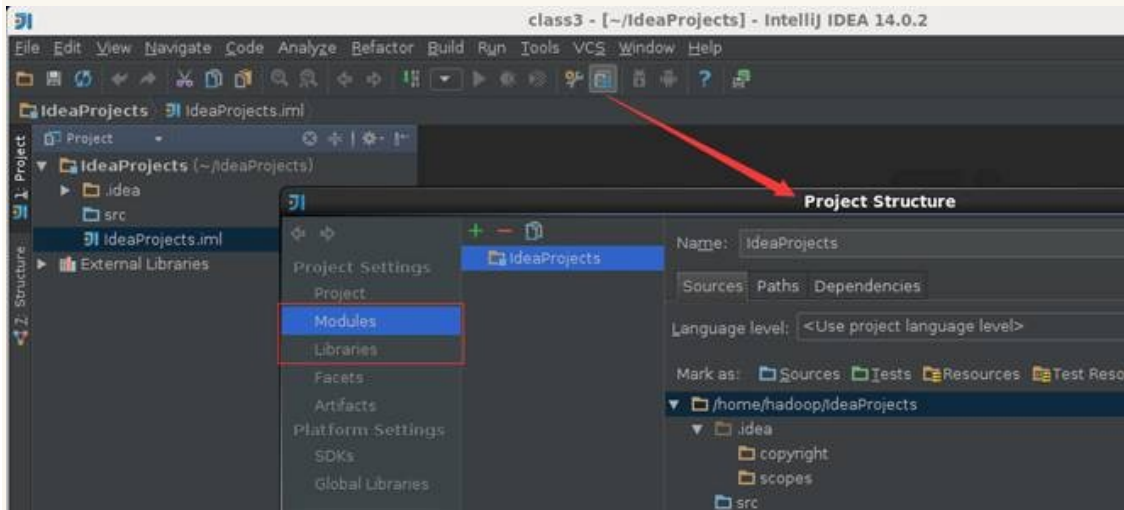


在项目的基本信息填写项目名称、项目所在位置、Project SDK 和 Scala SDK，在这里设置项目名称为 class3 Spark 编译安装介绍：

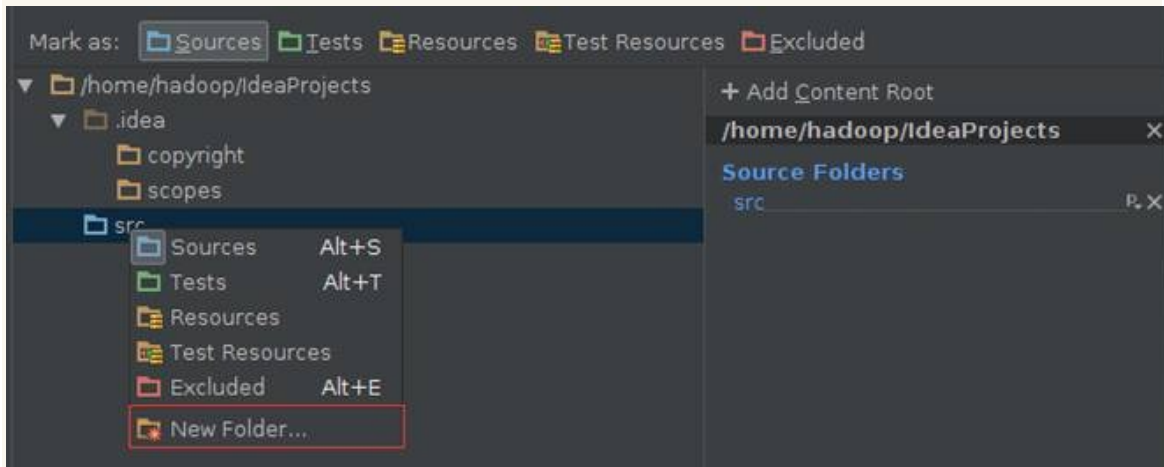


2.1.2 设置 Modules

创建该项目后，可以看到现在还没有源文件，只有一个存放源文件的目录 src 以及存放工程其他信息的杂项。通过双击 src 目录或者点击菜单上的项目结构图标打开项目配置界面，如下图所示：



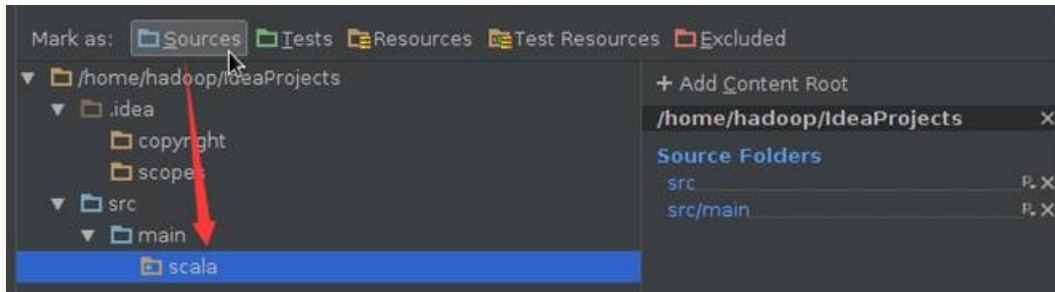
在 Modules 设置界面中，src 点击右键选择“新加文件夹”添加 src->main->scala 目录：



在 Modules 设置界面中，scala 目录为 Sources 类型：

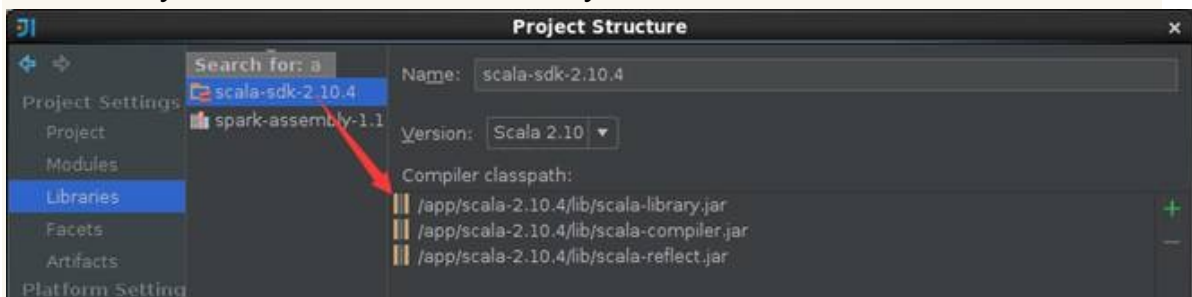
在 src 目录上，右键然后点击 new fold 命名为 main

再右键 main，同样 new fold，命名为 scala，并设置为 sources，如下图，注意，src、main 均不要设置 sources，否则后面编译 helloworld 会报错 XX is already defined as object xx

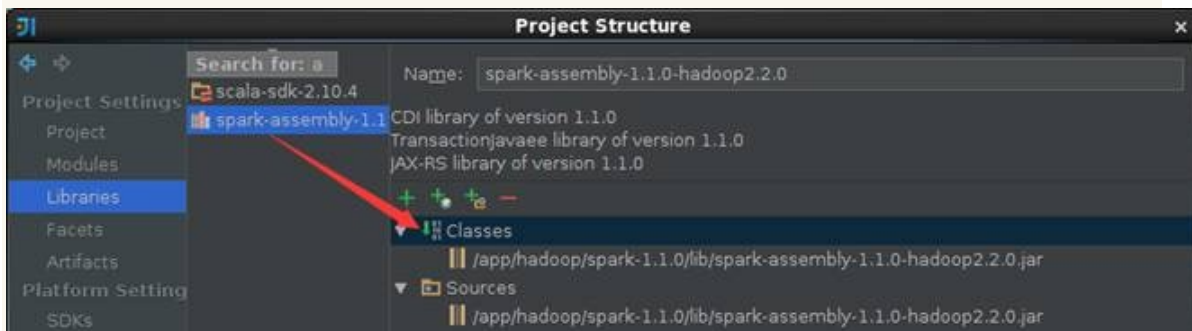


2.1.3 配置 Library

选择 Library 目录，添加 Scala SDK Library，这里选择 scala-2.10.4 版本



添加 Java Library，这里选择的是在 \$SPARK_HOME/lib/spark-assembly-1.1.0-hadoop2.2.0.jar 文件，添加完成的界面如下：



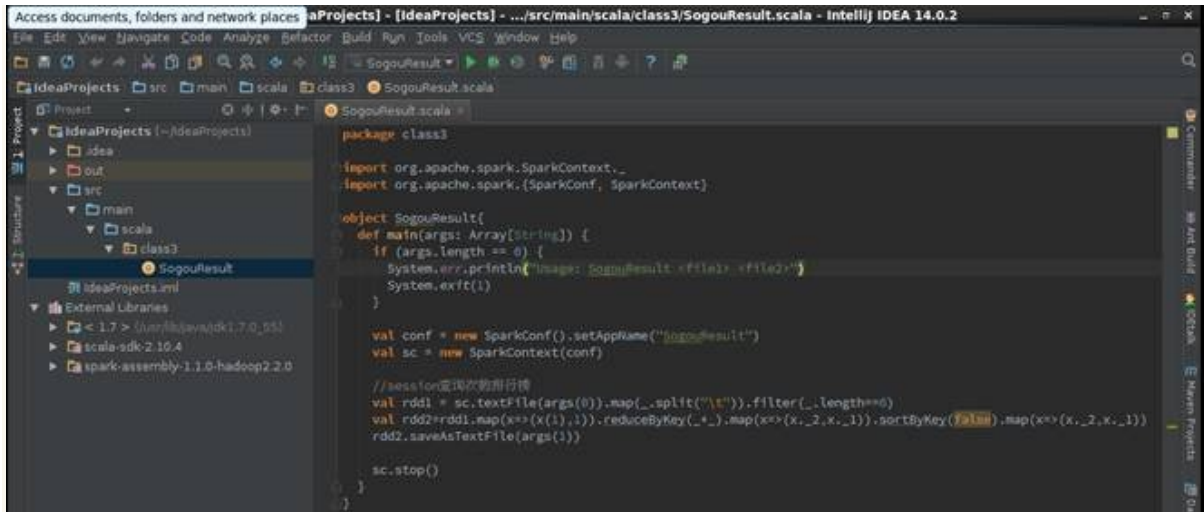
2.2 例子 1：直接运行

《Spark 编程模型（上）--概念及 Shell 试验》中使用 Spark-Shell 进行了搜狗日志的查询，在这里我们使用 IDEA 对 Session 查询次数排行榜进行重新练习，可以发现借助专业的开发工具可以方便快捷许多。

2.2.1 编写代码

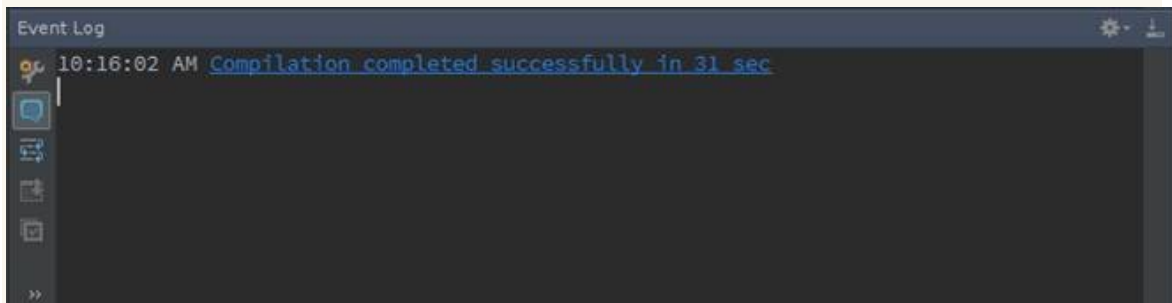
在 src->main->scala 下创建 class3 包，在该包中添加 SogouResult 对象文件，具体代码如下：

```
1 package class3
2
3 import org.apache.spark.SparkContext._
4 import org.apache.spark.{SparkConf, SparkContext}
5
6 object SogouResult{
7   def main(args: Array[String]) {
8     if (args.length == 0) {
9       System.err.println("Usage: SogouResult <file1> <file2>")
10      System.exit(1)
11    }
12
13    val conf = new
SparkConf(). setAppName("SogouResult"). setMaster("local")
14    val sc = new SparkContext(conf)
15
16    //session 查询次数排行榜
17    val rdd1 =
sc. textFile(args(0)). map(_ . split("\t")). filter(_ . length==6)
18 val
rdd2=rdd1. map(x=>(x(1), 1)). reduceByKey(_+_). map(x=>(x. _2, x. _1)). sortByKe
y(false). map(x=>(x. _2, x. _1))
19    rdd2. saveAsTextFile(args(1))
20    sc. stop()
21  }
22 }
```

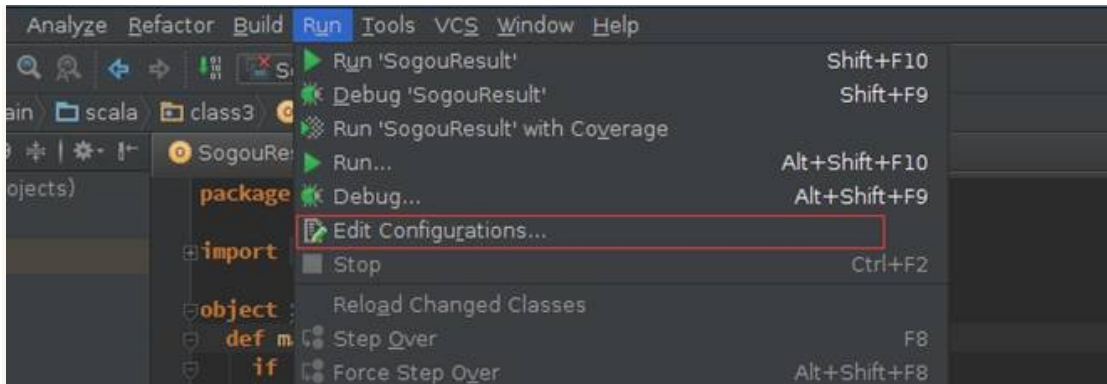
2.2.2 编译代码

代码在运行之前需要进行编译，可以点击菜单 Build->Make Project 或者 Ctrl+F9 对代码进行编译，编译结果会在 Event Log 进行提示，如果出现异常可以根据提示进行修改



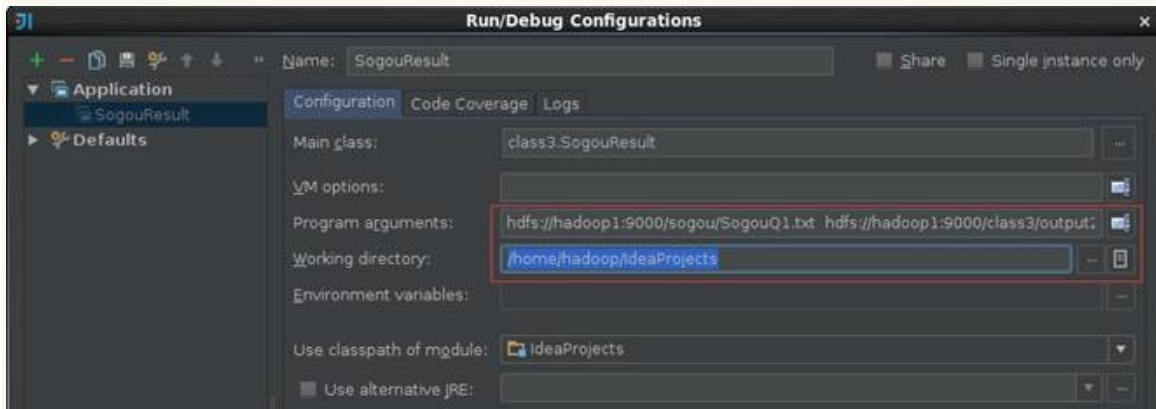
2.2.3 运行环境配置

SogouResult 首次运行或点击菜单 Run->Edit Configurations 打开"运行/调试 配置界面"



运行 SogouResult 时需要输入搜狗日志文件路径和输出结果路径两个参数，需要注意的是 HDFS 的路径参数路径需要全路径，否则运行会报错：

- 搜狗日志文件路径：使用上节上传的搜狗查询日志文件
hdfs://hadoop1:9000/sogou/SogouQ1.txt
- 输出结果路径：hdfs://hadoop1:9000/class3/output2



2.2.4 运行结果查看

启动 Spark 集群，点击菜单 Run->Run 或者 Shift+F10 运行 SogouResult，在运行结果窗口可以运行情况。当然了如果需要观察程序运行的详细过程，可以加入断点，使用调试模式根据程序运行过程。

```
Run SogouResult
15/07/17 10:50:32 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 2) in 7510 ms on localhost (1/1)
15/07/17 10:50:32 INFO DAGScheduler: Stage 0 (saveAsTextFile at SogouResult.scala:19) finished in 7.502 s
15/07/17 10:50:32 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
15/07/17 10:50:32 INFO SparkContext: Job finished: saveAsTextFile at SogouResult.scala:19, took 28.953340209 s
15/07/17 10:50:32 INFO SparkUI: Stopped Spark web UI at http://hadoop1:4040
15/07/17 10:50:32 INFO DAGScheduler: Stopping DAGScheduler
15/07/17 10:50:34 INFO MapOutputTrackerMasterActor: MapOutputTrackerActor stopped!
15/07/17 10:50:34 INFO ConnectionManager: Selector thread was interrupted!
15/07/17 10:50:34 INFO ConnectionManager: ConnectionManager stopped
15/07/17 10:50:34 INFO MemoryStore: MemoryStore cleared
15/07/17 10:50:34 INFO BlockManager: BlockManager stopped
15/07/17 10:50:34 INFO BlockManagerMaster: BlockManagerMaster stopped
15/07/17 10:50:34 INFO SparkContext: Successfully stopped SparkContext
15/07/17 10:50:34 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
```

使用如下命令查看运行结果，该结果和上节运行的结果一致

```
hadoop fs -ls /class3/output2
```

```
hadoop fs -cat /class3/output2/part-00000 | less
```

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop1 sbin]$ hadoop fs -ls /class3/output2
Found 2 items
-rw-r--r-- 3 hadoop supergroup 0 2015-07-17 10:50 /class3/output2/_SUCCESS
-rw-r--r-- 3 hadoop supergroup 10477740 2015-07-17 10:50 /class3/output2/part-00000
[hadoop@hadoop1 sbin]$ hadoop fs -cat /class3/output2/part-00000 | less
(b3c94c37fb154d46c30a360c7941ff7e,676)
(cc7063efc64510c20bcdd604e12a3b26,613)
(955c6390c02797b3558ba223b8201915,391)
(b1e371de5729cdda9270b7ad09484c4f,337)
(6056710d9eafa569ddc800fe24643051,277)
(637b29b47fed3853e117aa7009a4b621,266)
(c9f4ff7790d0615f6f66b410673e3124,231)
(dca9034de17f6c34cfd56db13ce39f1c,226)
```

2.3 例子 2：打包运行

上个例子使用了 IDEA 直接运行结果，在该例子中将使用 IDEA 打包程序进行执行

2.3.1 编写代码

在 class3 包中添加 Join 对象文件，具体代码如下：

```
1 package class3
2
3 import org.apache.spark.SparkContext._
4 import org.apache.spark.{SparkConf, SparkContext}
5
6 object Join{
7   def main(args: Array[String]) {
8     if (args.length == 0) {
9       System.err.println("Usage: Join <file1> <file2>")
```

```

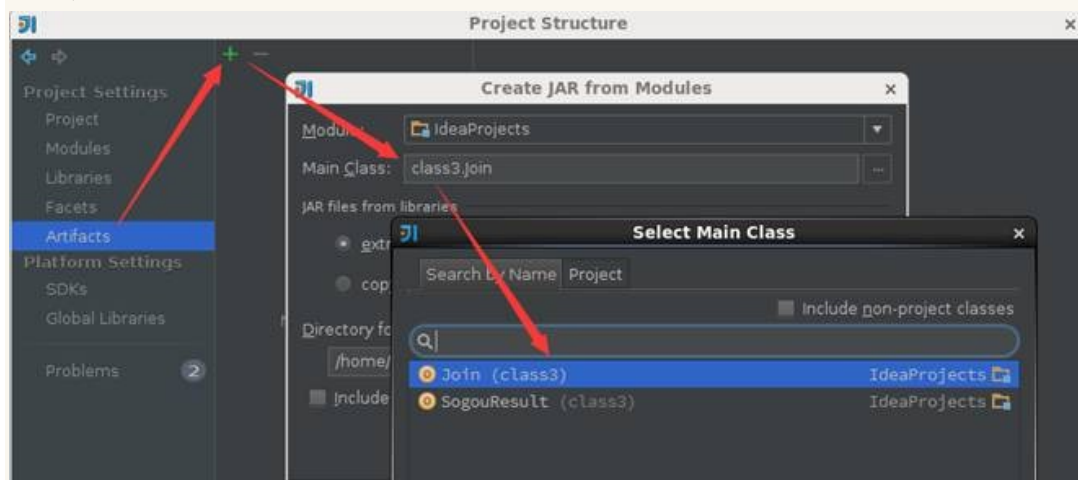
10     System.exit(1)
11 }
12
13 val conf = new SparkConf().setAppName("Join").setMaster("local")
14 val sc = new SparkContext(conf)
15
16 val format = new java.text.SimpleDateFormat("yyyy-MM-dd")
17 case class Register (d: java.util.Date, uuid: String, cust_id: String,
18 lat: Float, lng: Float)
19 case class Click (d: java.util.Date, uuid: String, landing_page: Int)
20 val reg = sc.textFile(args(0)).map(_.split("\t")).map(r => (r(1),
21 Register(format.parse(r(0)), r(1), r(2), r(3).toFloat, r(4).toFloat)))
22 val clk = sc.textFile(args(1)).map(_.split("\t")).map(c => (c(1),
23 Click(format.parse(c(0)), c(1), c(2).trim.toInt)))
24 reg.join(clk).take(2).foreach(println)
25
26 sc.stop()
27 }
28 }

```

2.3.2 生成打包文件

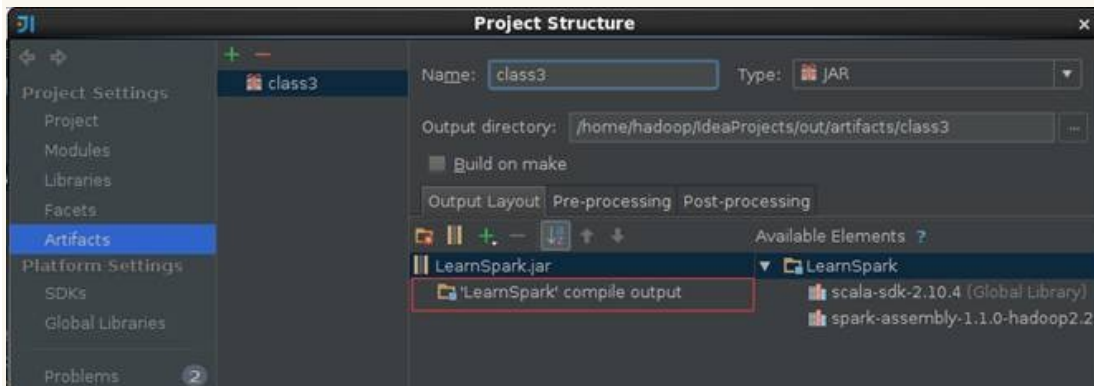
第一步 配置打包信息

在项目结构界面中选择"Artifacts", 在右边操作界面选择绿色"+"号, 选择添加 JAR 包的"From modules with dependencies"方式, 出现如下界面, 在该界面中选择主函数入口为 Join :



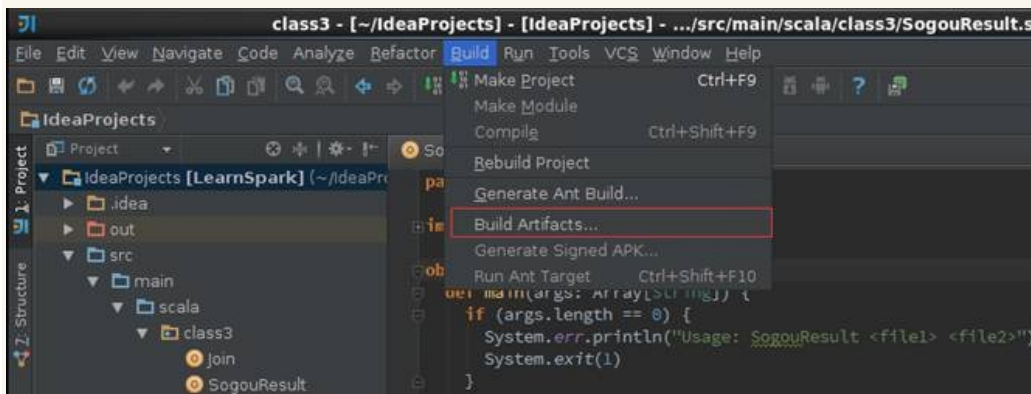
第二步 填写该 JAR 包名称和调整输出内容

【注意】的是默认情况下"Output Layout"会附带 Scala 相关的类包，由于运行环境已经有 Scala 相关类包，所以在这里去除这些包只保留项目的输出内容



第三步 输出打包文件

点击菜单 Build->Build Artifacts，弹出选择动作，选择 Build 或者 Rebuild 动作

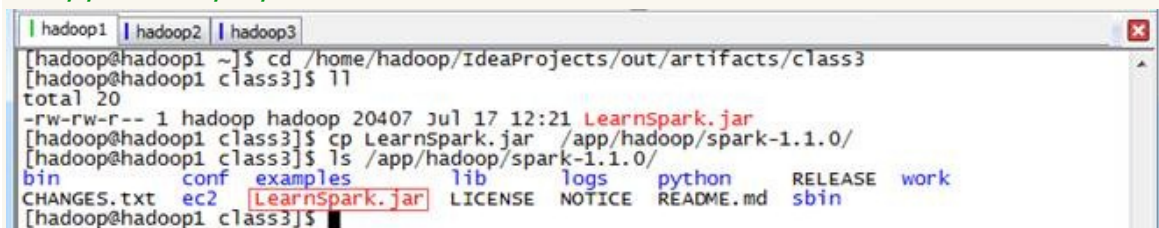


第四步 复制打包文件到 Spark 根目录下

```
cd /home/hadoop/IdeaProjects/out/artifacts/class3
```

```
cp LearnSpark.jar /app/hadoop/spark-1.1.0/
```

```
ls /app/hadoop/spark-1.1.0/
```



2.3.3 运行查看结果

通过如下命令调用打包中的 Join 方法，运行结果如下：

```
cd /app/hadoop/spark-1.1.0
```



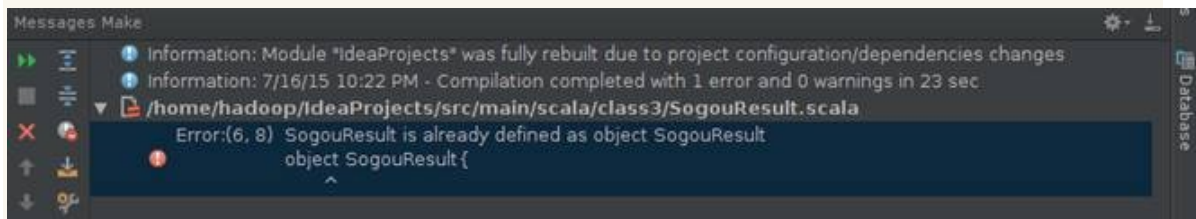
```
bin/spark-submit --master spark://hadoop1:7077 --class class3.Join
--executor-memory 1g LearnSpark.jar hdfs://hadoop1:9000/class3/join/reg.tsv
hdfs://hadoop1:9000/class3/join/clk.tsv
```

```
15/07/17 12:43:48 INFO Executor: Finished task 0.0 in stage 0.0 (TID 2). 1384 bytes result sent to driver
15/07/17 12:43:48 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 2) in 126 ms on localhost (1/1)
15/07/17 12:43:48 INFO DAGScheduler: Stage 0 (take at Join.scala:21) finished in 0.129 s
15/07/17 12:43:48 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
15/07/17 12:43:48 INFO SparkContext: Job finished: take at Join.scala:21. took 1.321228527 s
15/07/17 12:43:48 INFO DAGScheduler: Stopping DAGScheduler
15/07/17 12:43:48 INFO SparkUI: Stopped Spark web UI at http://hadoop1:4040
15/07/17 12:43:48 INFO MapOutputTrackerMasterActor: MapOutputTrackerActor stopped!
15/07/17 12:43:49 INFO ConnectionManager: Selector thread was interrupted!
15/07/17 12:43:49 INFO ConnectionManager: ConnectionManager stopped
15/07/17 12:43:49 INFO MemoryStore: MemoryStore cleared
15/07/17 12:43:49 INFO BlockManager: BlockManager stopped
15/07/17 12:43:49 INFO BlockManagerMaster: BlockManagerMaster stopped
15/07/17 12:43:49 INFO SparkContext: Successfully stopped SparkContext
15/07/17 12:43:49 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
15/07/17 12:43:49 INFO RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote trans
ports.
15/07/17 12:43:49 INFO Remoting: Remoting shut down
15/07/17 12:43:49 INFO RemoteActorRefProvider$RemotingTerminator: Remoting shut down.
[hadoop@hadoop1 spark-1.1.0]$
```

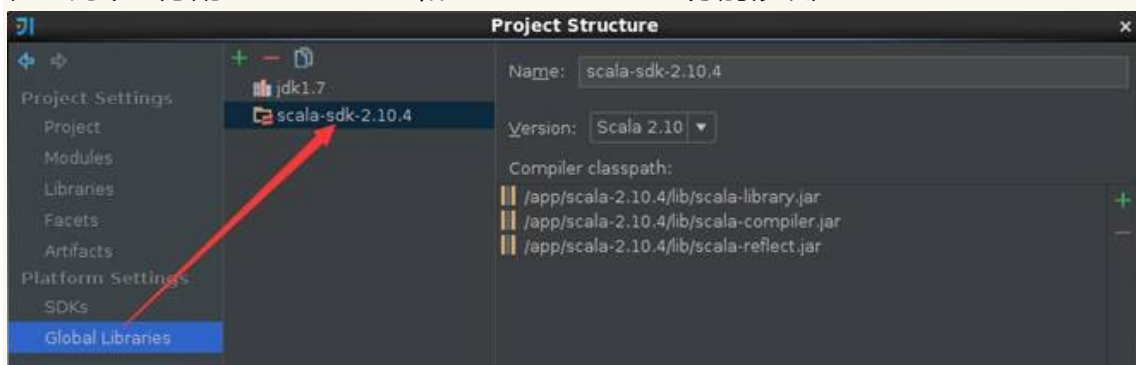
3、问题解决

3.1 出现 "*** is already defined as object ***" 错误

编写好 SogouResult 后进行编译，出现 "Sogou is already as object SogouResult" 的错误，



出现这个错误很可能不是程序代码的问题，很可能是使用 Scala JDK 版本问题，作者在使用 scala-2.11.4 遇到该问题，换成 scala-2.10.4 后重新编译该问题得到解决，需要检查两个地方配置：Libraries 和 Global Libraries 分别修改为 scala-2.10.4



补充：

如何跟 maven 整合

现在主流的开发环境，大多数是采用 maven 来构建项目的，所以建议大家用 maven+plugin 的方式来构建 scala 应用，另外，就象 VB.NET/C#/F#可同时在一个项目中

使用，最大限度发挥各语种特长一样，**java** 与可以与 **scala** 在一个项目中混合使用。见下面的 pom.xml 示例：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>yjmyzz</groupId>
8     <artifactId>MyScala</artifactId>
9     <version>1.0</version>
10
11     <dependencies>
12         <dependency>
13             <groupId>org.scala-lang</groupId>
14             <artifactId>scala-library</artifactId>
15             <version>2.11.7</version>
16         </dependency>
17         <dependency>
18             <groupId>org.scala-lang</groupId>
19             <artifactId>scala-compiler</artifactId>
20             <version>2.11.7</version>
21         </dependency>
22         <dependency>
23             <groupId>org.scala-lang</groupId>
24             <artifactId>scala-reflect</artifactId>
25             <version>2.11.7</version>
26         </dependency>
27         <dependency>
28             <groupId>log4j</groupId>
29             <artifactId>log4j</artifactId>
30             <version>1.2.12</version>
31         </dependency>
32         <dependency>
33             <groupId>com.google.collections</groupId>
34             <artifactId>google-collections</artifactId>
35             <version>1.0</version>
36         </dependency>
```



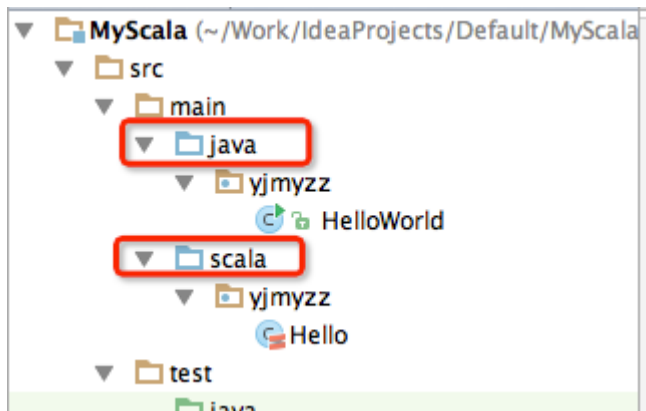
```

37     </dependencies>
38
39     <build>
40         <plugins>
41             <plugin>
42                 <groupId>org.scala-tools</groupId>
43                 <artifactId>maven-scala-plugin</artifactId>
44                 <version>2.15.2</version>
45                 <executions>
46                     <execution>
47                         <goals>
48                             <goal>compile</goal>
49                             <goal>testCompile</goal>
50                         </goals>
51                     </execution>
52                 </executions>
53             </plugin>
54         </plugins>
55     </build>
56 </project>

```

最下面的 plugin 是用来编译 scala 源代码的，毕竟 java 与 scala 是二种不同的语言，有各自的 sdk 和编译器，所以需要专门的 maven 插件来处理 scala 的编译。

项目的目录结构，大体跟 maven 的默认约定一样，只是 src 下多了一个 scala 目录，如下图：



这样，java 源代码放在/src/java 下，scala 源代码放在/src/scala 下，管理起来也比较清爽，上图中 scala 下的 Hello.scala 源代码如下：

1	package yjmyzz
2	

```
3 class Hello {
4     def sayHello(x: String): Unit = {
5         println("hello," + x);
6     }
7 }
```

然后 java 下的 HelloWorld.java 里就可以调用 scala 的 Hello 类:

```
1 package yjmyzz;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         Hello h = new Hello();
7         h.sayHello("scala");
8     }
9 }
```

4、scala 项目 maven 的编译打包

如果直接运行 `mvn clean package`，会杯具的发现

[ERROR]

`/Users/jimmy/Work/IdeaProjects/Default/MyScala/src/main/java/yjmyzz/HelloWorld.java:[7,9] cannot find symbol`

[ERROR] symbol: `class Hello`

[ERROR] location: `class yjmyzz.HelloWorld`

原因是 `mvn clean package` 默认只处理 java 源代码的编译、打包，而不管 scala，所以编译时遇到 Hello 这个由 scala 语言编写的 class，此时 scala 还没编译生成 class，所以找不到相应的调用入口。

解决办法:

mvn clean scala:compile compile package

如上，在 `compile` 前加入 `scala:compile`，这是 `maven-scala-plugin` 插件提供的选项，表示编译 scala，这样一来，先编译 scala，再编译 java，最后打包，妥妥滴！

2 spark 编程模型

2.1 Spark Application

Spark应用程序由两部分组成

- 1) Driver
- 2) Executor

spark应用程序基本概念

Application: 基于Spark的用户程序，包含了一个Driver Program和集群中多个的executor

Driver Program: :运行Application的main() 函数并且创建**SparkContext**，通常用SparkContext代表Driver Program.

Executor: 是为某个Application运行在worker node上的一个进程，该进程负责运行Task，并且负责将数据持久化到内存或者磁盘上。每个Application都有各自独立的executors.

Cluster Manager: 集群的资源调度服务

Worker Node: 集群中任何可以运行Application代码的节点

Task: executor的工作单元

Job: 包含多个Task组成的并行计算任务，由Spark Action触发任务的执行

Stage: Job的调度单位，每个job被拆分成多组task, 每组任务被称为stage，可也称为taskSet

RDD: Spark的基本计算单元，可以通过一系列算子进行操作(Transformation Action)

2.2 Spark RDD

RDD三大特性:

弹性分布式数据集

1. 分区
2. 计算
3. 依赖
 - a) 宽依赖
 - b) 窄依赖

1、RDD 创建

- a) 外部数据源

HDFS

Spark支持textFile sequenceFile

- b) 集合Parallelized Collections

RDD 分区partition

sc.textFile("",4): RDD有四个分区，手动的指定分区个数

2、RDD Transformation

支持 map flatMap filter union distinct join groupByKey reduceByKey sortByKey 从一个RDD变为另外一个RDD,

WordCount:map reduceByKey text -> RDD[String] -> RDD[(String,Int)] ->

RDD[(String,Int)] -> collect wordRDD kvRDD resultRDD 一行一行的数据

lazy, 懒执行

lineage, 生命周期, 转换

3、RDD action 触发计算，进行实际的数据处理

reduce collect count countByKey first take foreach saveAsTextFile

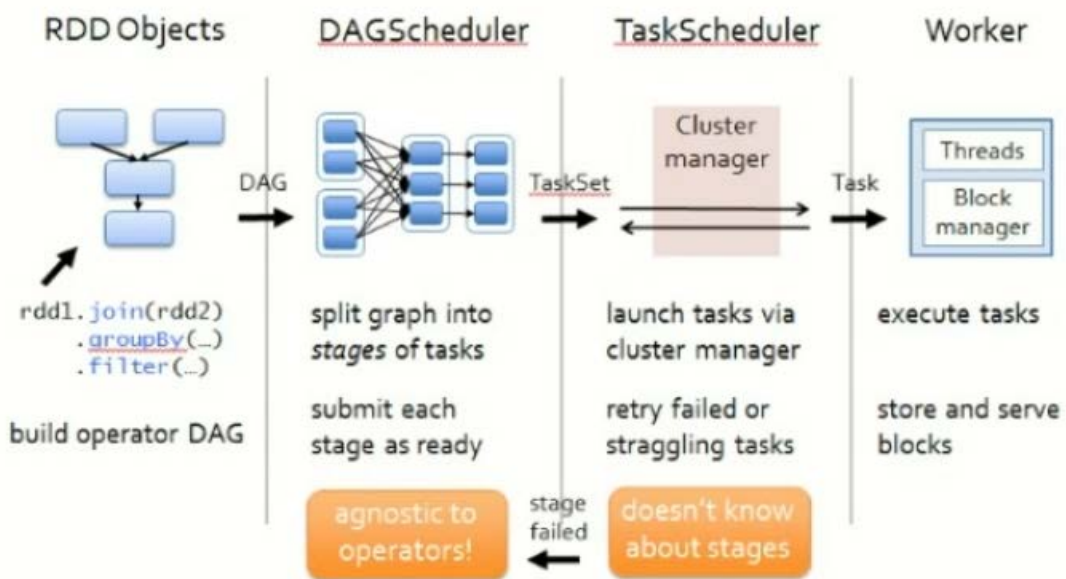
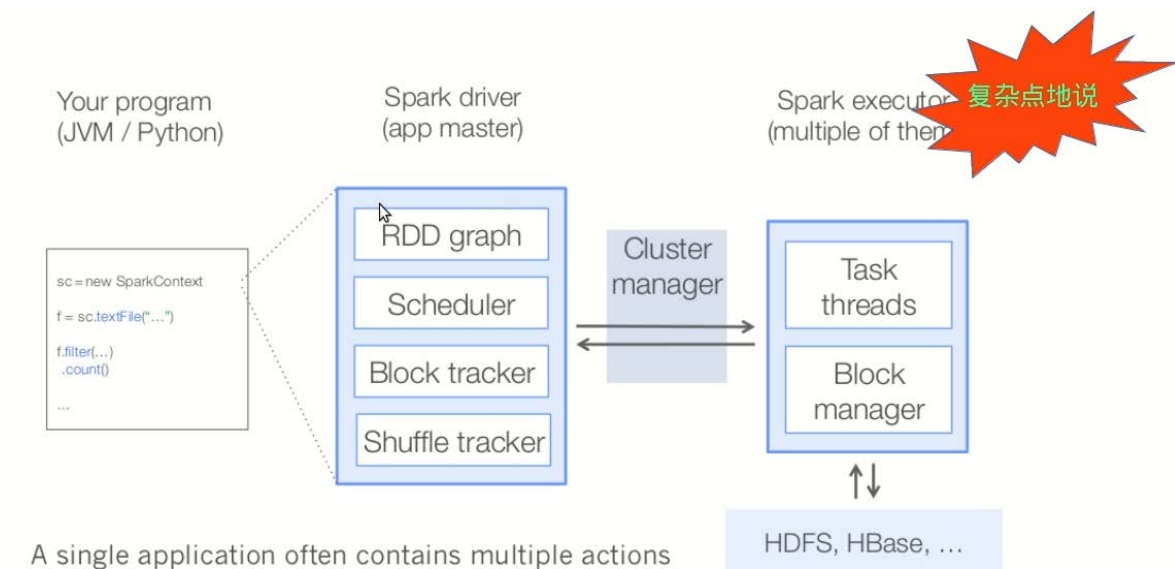
saveAsSequenceFile

4、RDD cache

cache方法，是延迟执行，需要在一个action执行以后，进行缓存RDD。

cache是persistent特殊缓存方式，将RDD放到内存中

2.3 Spark 运行架构



2.3.1 DAGScheduler

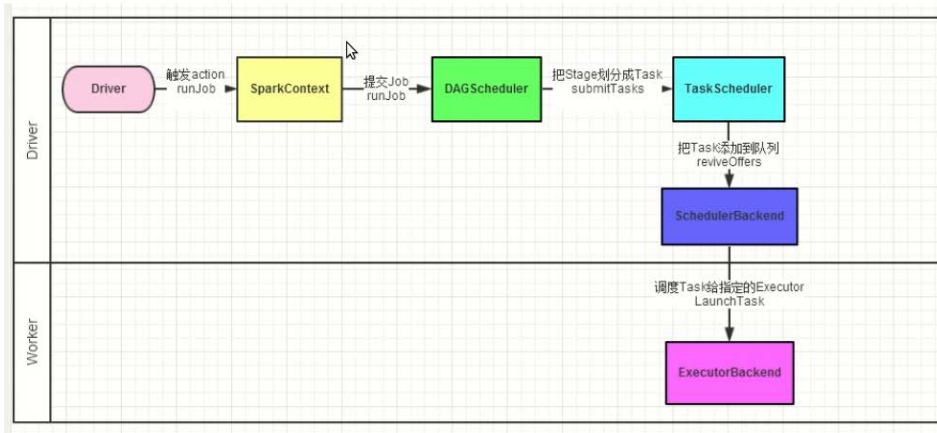
1. 接收用户提交的job
2. 构建Stage，记录哪个RDD或者Stage输出被物化
3. 重新提交shuffle输出丢失的stage
4. 将Taskset传输给底层调度器

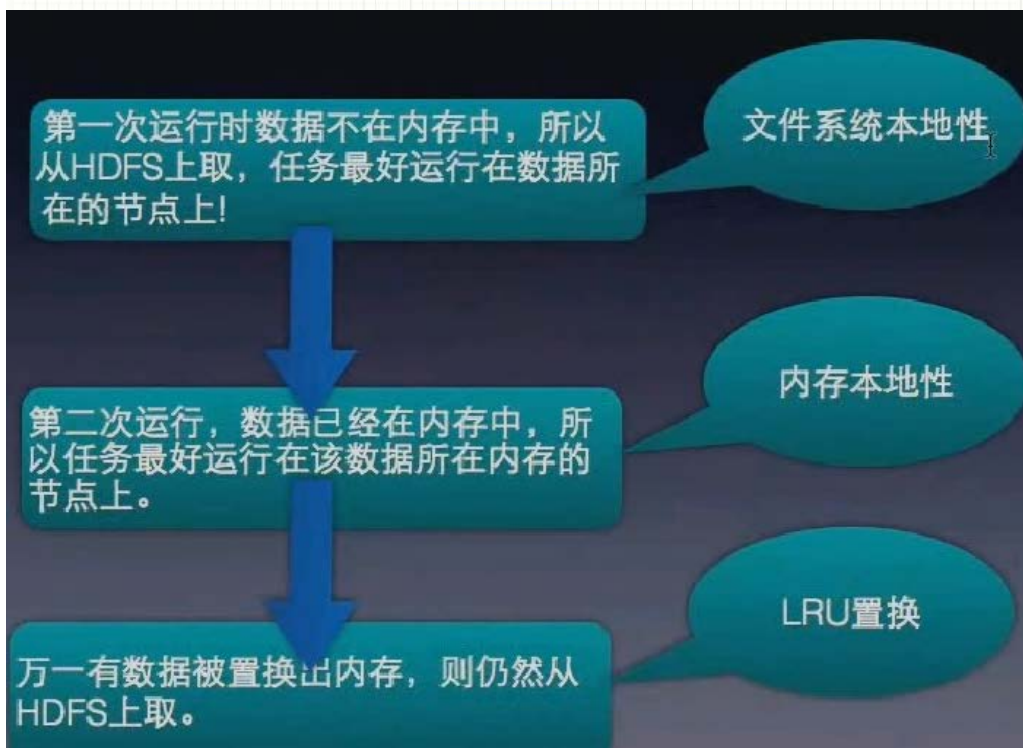
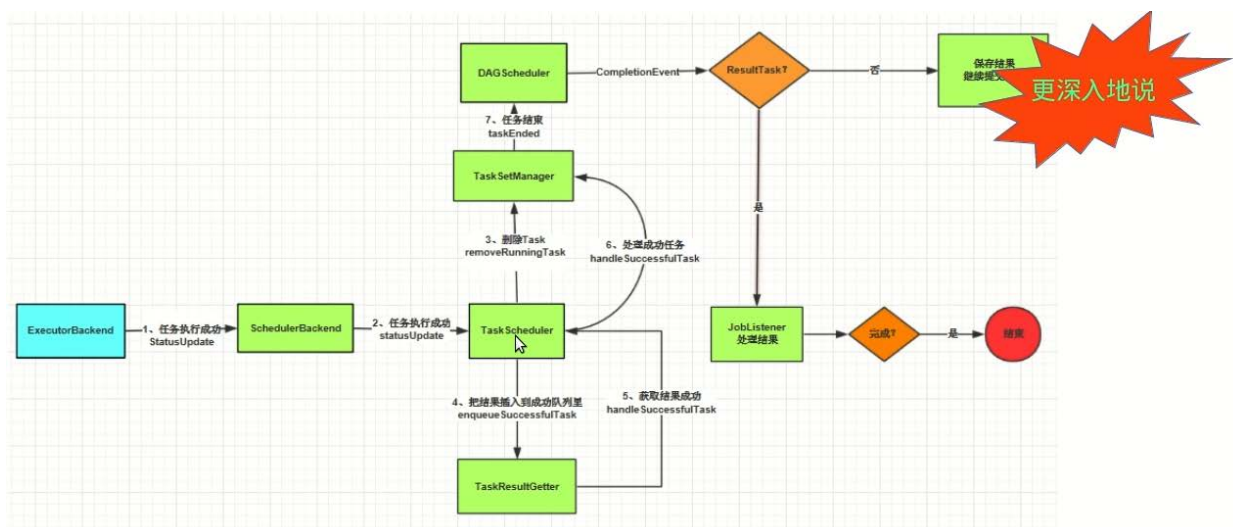
2.3.2 TaskScheduler

1. 提交taskset到集群运行并监控
2. 为每一个taskset构建一个TaskSetManager实例管理这个TaskSet的生命周期
3. 数据本地性决定每个task最佳位置 (process-local node-local rack-local and then any)
4. 推测执行，碰到straggle任务需要放到其他的节点重试出现shuffle输出lost要报告fetch failed 错误

2.3.3 Task

1. Task是Executor中的执行单元
2. Task处理数据的两个来源：外部存储以及shuffle数据
3. Task可以运行在集群中的任意一个节点上
4. 为了容错，会将shuffle输出写到磁盘或者内存中





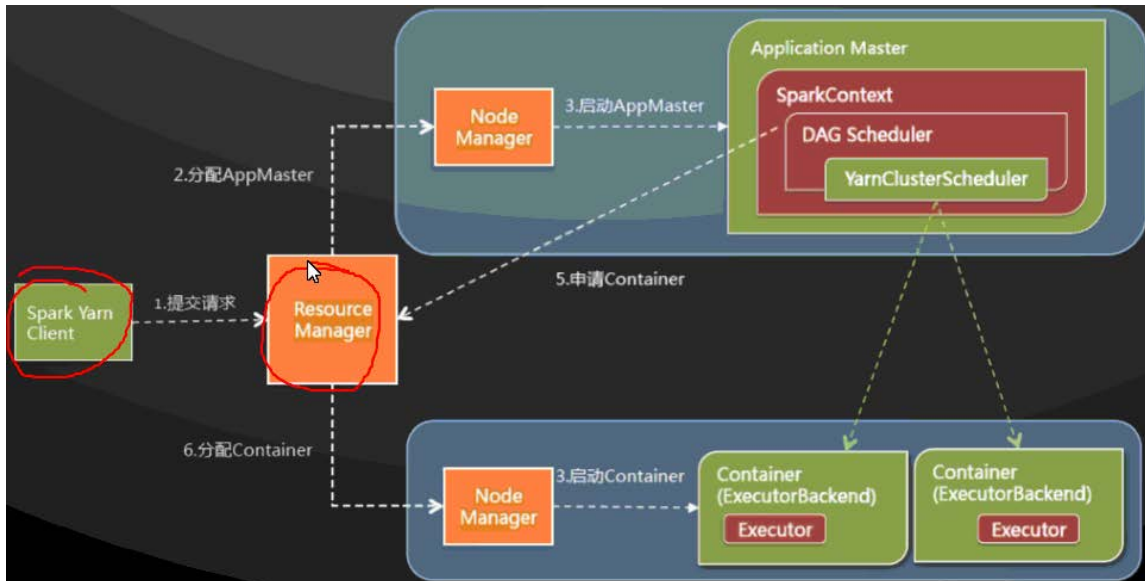
2.3.4 SparkContext 初始化过程

1. 依据SparkContext的构造方法中的参数SparkConf创建一个SparkEnv
2. 初始化，SparkUI，以便Spark Application在运行时，方便用户监控，默认端口为4040
3. 创建和启动Scheduler
 1. 创建TaskScheduler、SchedulerBackend
 2. 创建DAGScheduler
 3. 启动TaskScheduler、DAGScheduler
4. 启动Executors

3 Spark On Yarn

Spark JobHistory与Spark on YARN

- 1、Spark App监控
- 2、Spark JobHistory
- 3、Hadoop YARN部署与工作原理
- 4、Spark on YARN



```
bin/spark-submit \  
    --class org.apache.spark.examples.SparkPi \  
    lib/spark-examples-1.3.0-hadoop2.6.0-cdh5.4.0.jar \  
    10
```

```
bin/spark-submit \  
    --deploy-mode cluster \  
    --class org.apache.spark.examples.SparkPi \  
    lib/spark-examples-1.3.0-hadoop2.6.0-cdh5.4.0.jar \  
    10
```

```
=====
```

```
bin/spark-submit \  
    --master yarn \  
    --class org.apache.spark.examples.SparkPi \  
    lib/spark-examples-1.3.0-hadoop2.6.0-cdh5.4.0.jar \  
    10
```

```
[cyhp@hadoop-yarn spark-1.3.0-bin-2.6.0-cdh5.4.0]$ bin/spark-submit --help  
Spark assembly has been built with Hive, including Datanucleus jars on classpath  
Usage: spark-submit [options] <app jar | python file> [app arguments]  
Usage: spark-submit --kill [submission ID] --master [spark://...]  
Usage: spark-submit --status [submission ID] --master [spark://...]
```


Options:

<code>--master MASTER_URL</code>	spark://host:port, mesos://host:port, yarn, or local.
<code>--deploy-mode DEPLOY_MODE</code>	Whether to launch the driver program locally ("client") or on one of the worker machines inside the cluster ("cluster") (Default: client).
<code>--class CLASS_NAME</code>	Your application's main class (for Java / Scala apps).
<code>--name NAME</code>	A name of your application.
<code>--jars JARS</code>	Comma-separated list of local jars to include on the driver and executor classpaths.
<code>--packages</code>	Comma-separated list of maven coordinates of jars to include on the driver and executor classpaths. Will search the local maven repo, then maven central and any additional remote repositories given by <code>--repositories</code> . The format for the coordinates should be groupId:artifactId:version.
<code>--repositories</code>	Comma-separated list of additional remote repositories to search for the maven coordinates given with <code>--packages</code> .
<code>--py-files PY_FILES</code>	Comma-separated list of .zip, .egg, or .py files to place on the PYTHONPATH for Python apps.
<code>--files FILES</code>	Comma-separated list of files to be placed in the working directory of each executor.
<code>--conf PROP=VALUE</code>	Arbitrary Spark configuration property.
<code>--properties-file FILE</code>	Path to a file from which to load extra properties. If not specified, this will look for conf/spark-defaults.conf.
<code>--driver-memory MEM</code>	Memory for driver (e.g. 1000M, 2G) (Default: 512M).
<code>--driver-java-options</code>	Extra Java options to pass to the driver.
<code>--driver-library-path</code>	Extra library path entries to pass to the driver.
<code>--driver-class-path</code>	Extra class path entries to pass to the driver. Note that jars added with <code>--jars</code> are automatically included in the classpath.
<code>--executor-memory MEM</code>	Memory per executor (e.g. 1000M, 2G) (Default: 1G).
<code>--proxy-user NAME</code>	User to impersonate when submitting the application.
<code>--help, -h</code>	Show this help message and exit
<code>--verbose, -v</code>	Print additional debug output
<code>--version,</code>	Print the version of current Spark

Spark standalone with cluster deploy mode only:

<code>--driver-cores NUM</code>	Cores for driver (Default: 1).
<code>--supervise</code>	If given, restarts the driver on failure.
<code>--kill SUBMISSION_ID</code>	If given, kills the driver specified.
<code>--status SUBMISSION_ID</code>	If given, requests the status of the driver specified.

Spark standalone and Mesos only:

<code>--total-executor-cores NUM</code>	Total cores for all executors.
---	--------------------------------

YARN-only:

--driver-cores NUM	Number of cores used by the driver, only in cluster mode (Default: 1).
--executor-cores NUM	Number of cores per executor (Default: 1).
--queue QUEUE_NAME	The YARN queue to submit to (Default: "default").
--num-executors NUM	Number of executors to launch (Default: 2).
--archives ARCHIVES	Comma separated list of archives to be extracted into the working directory of each executor.

```
sc.textFile("hdfs://hadoop-spark.dragon.org:8020/user/cyhp/spark/wc.input").flatMap(_.  
split(" ")).map((_, 1)).reduceByKey(_ + _).collect
```

```
=====Hadoop YARN=====
yarn-site.xml
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hadoop-yarn.cloudyhadoop.com</value>
    </property>

    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
slaves
    hadoop-yarn.cloudyhadoop.com
```

4 Spark as a Service

4.1 概述

spark-jobserver 提供了一个用于提交和管理 Apache Spark 作业(job)、jar 文件和作业上下文 (SparkContext) 的 RESTful 接口。该项目位于 git (<https://github.com/ooyala/spark-jobserver>)

特性

“Spark as a Service”: 简单的面向 job 和 context 管理的 REST 接口

通过长期运行的 job context 支持亚秒级低延时作业(job)

可以通过结束 context 来停止运行的作业(job)

分割 jar 上传步骤以提高 job 的启动

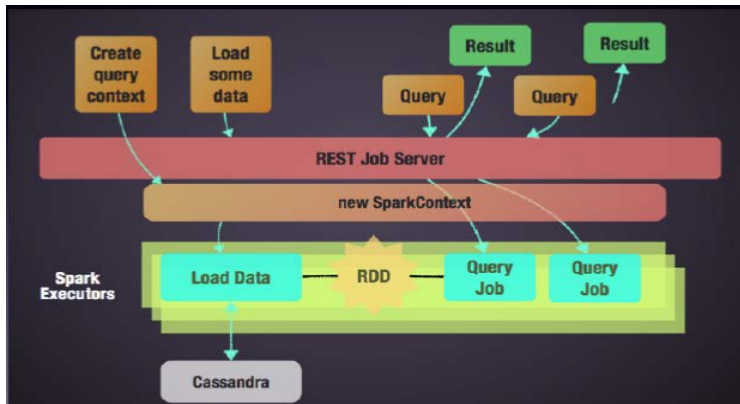
异步和同步的 job API，其中同步 API 对低延时作业非常有效

支持 Standalone Spark 和 Mesos

Job 和 jar 信息通过一个可插拔的 DAO 接口来持久化

命名 RDD 以缓存，并可以通过该名称获取 RDD。这样可以提高作业间 RDD 的共享和重用

4.2 架构



5 Spark Streaming

DStream

将流式计算分解成一些列确定并且较小的批处理作业
将失败或者执行较慢的任务在其他节点并行执行
较强的容错能力

数据源:

Kafka flume twitter MQ TCP socket Akka actor HDFS
Hbase 自定义

Transformation:

- 1.map flatMap reduce filter reduceByKey
- 2.带状态:
UpdateStateByKey
- 3.window 操作
Window countByWindow reduceByWindow
RecudeByKeyAndWindow

DStream 输出

Print
ForeachRDD
SaveAsObjectFiles

SaveAsTextFiles
SaveAsHadoopFiles

持久化

默认级别<内存+序列化>

Checkpoint

对于 window 和 stateful 操作必须 checkpointing
通过 StreamingContext 的 checkpoint 来制定目录
通过 DStream 的 checkpoint 指定时间间隔
间隔必须是 slide interval 的倍数