Scala及Spark入门培训

——蔡岩

培训目的及对象

目的：

- 快速掌握基于Scala语言的Spark开发
- Scala不介绍语法细节，Spark不做过多介绍

培训对象基础能力：

- 语言基础：
  - 编译型语言其一（Java、C++、C#、……）
  - 脚本型语言其一（Python、JavaScript、MATLAB、R、……）
- 操作系统：
  - Linux（主要）、Windows

目录

Menu

Scala 简介

Scala

- 主页：http://www.scala-lang.org/
- 历史
  2003-2004年诞生
  2009年Twitter后端从Ruby迁移至Scala
- 多范式
  包含面向对象、函数式特点
- JVM
  编译成Java字节码、可以和Java混编、兼容Java库
- 语言特点

- 高起点

Menu

Spark 简介

Spark

主页：http://spark.apache.org/

诞生于2009 UC Berkeley AMP lab，2010开源

是一个快速、通用的大规模数据处理**引擎**

类似于Hadoop的MapReduce，可伸缩、基于内存、计算中间结果无需写入磁盘。

快速：传统MR的100倍+

易用：多种语言支持：Java、Scala、Python、R等，提供REPL

通用：提供高级工具：SparkSQL、MLib、GraphX、Spark Streaming

集成Hadoop：易于和Hadoop结合，支持YARN、HBase、Hive等组件

Scala
开发环境搭建

安装包准备

Windows平台

1. JDK安装包（exe、zip）
   http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html
2. Scala安装包（msi、zip）
   http://www.scala-lang.org/download/
3. IDE：IDEA（推荐）、Eclipse
   https://www.jetbrains.com/idea/
   http://www.eclipse.org/downloads/
4. Maven、SBT
   https://maven.apache.org/download.cgi
   http://www.scala-sbt.org/download.html
5. Spark
   https://spark.apache.org/downloads.html

JDK安装

1. 安装

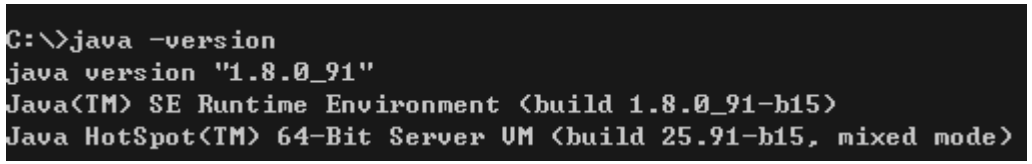
   执行可执行程序：jdk-8u91-windows-x64.exe

   OR 解压压缩文件至指定目录

2. 配置环境变量

   ```
   JAVA_HOME = C:\Program Files\Java\jdk1.8.0_91
   CLASSPATH = .;%Java_Home%\bin;%Java_Home%\lib\dt.jar;%Java_Home%\lib\tools.jar
   PATH += %JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;
   ```

3. 验证



Scala安装

1. 安装

   执行可执行程序：scala-2.11.8.msi

   OR 解压压缩文件至指定目录

2. 配置环境变量

   ```
   SCALA_HOME = C:\Program Files\scala-2.11.8
   PATH += %SCALA_HOME%\bin;
   ```

3. 验证

   ```
   B:\Program Files\scala-2.11.8\bin>scala -version
   Scala code runner version 2.11.8 -- Copyright 2002-2016, LAMP/EPFL
   ```

Maven安装

1. 解压maven压缩包至指定目录

2. 配置换进变量

```
MAVEN_HOME = C:\Program Files\apache-maven-3.1.0
PATH += %MAVEN_HOME%\bin
```

3. 验证

CMD中输入：`mvn -version`

SBT安装

1. 安装

   执行可执行程序

   OR 解压压缩文件至指定目录

2. 配置环境变量

   ```
   SBT_HOME = C:\Program Files\sbt
   PATH += %SBT_HOME%\bin;
   ```

3. 验证

   CMD输入：`sbt -version`

Spark开发环境搭建——Spark安装

1. 解压压缩文件至指定目录

   spark-2.0.0-preview-bin-hadoop2.6.tgz

2. 配置环境变量

   ```
   HADOOP_HOME = D:\spark-2.0.0-preview-bin-hadoop2.6
   PATH += %HADOOP_HOME%
   ```

3. 添加winutils.exe

   将winutils.exe添加至目录bin中（exe在hadoop-common-2.2.0-bin-master.zip中）

4. 验证

   CMD输入：spark-shell

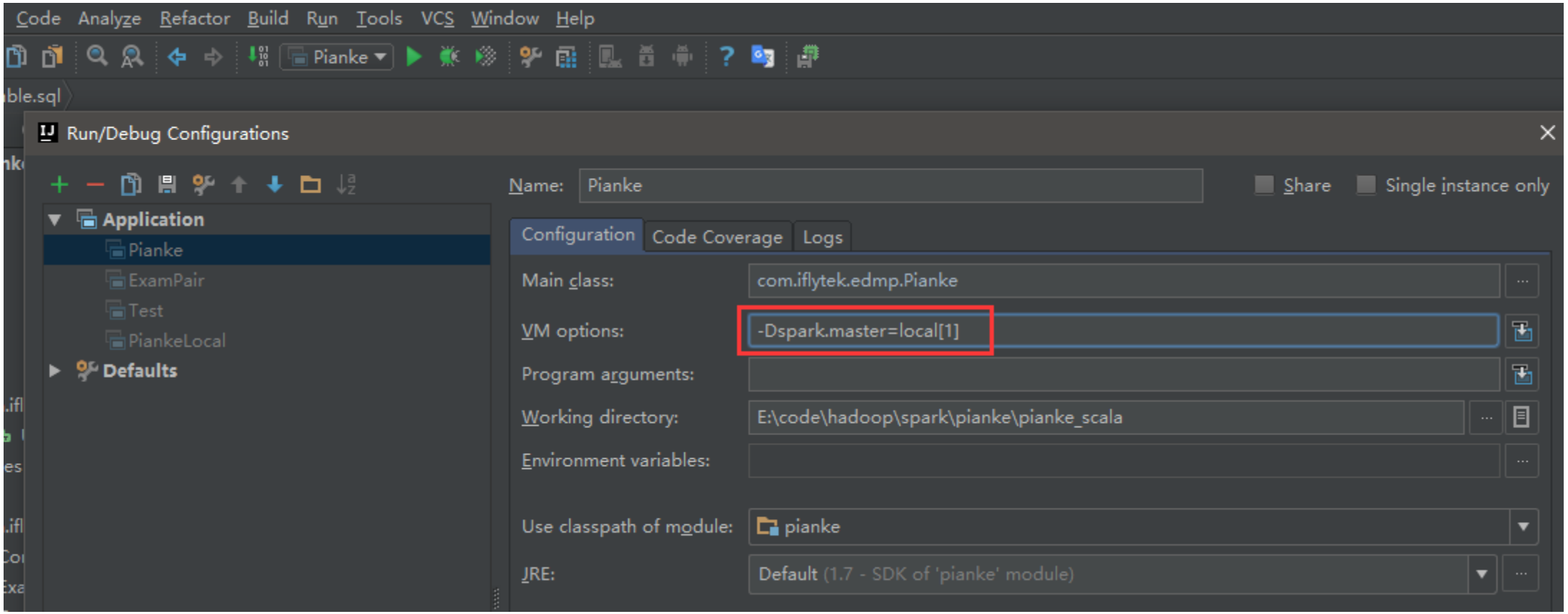
   退出输入":quit"

Spark开发环境搭建——IDEA安装及配置

1. 执行安装程序
2. 安装scala插件
   File -> Settings -> Plugins -> Browse repositories -> 搜索scala 并点击右侧安装

3. 配置Maven
   File -> Settings -> Build, Execution, Deployment -> Build Tools -> Maven
   配置：Maven home directory、User settings file
4. Run & Debug配置
   Run -> Edit Configurations -> Configuration -> VM options: -Dspark.master=local[1]

Menu


Code Analyze Refactor Build Run Tools VCS Window Help
Pianke
ble.sql
Run/Debug Configurations
Application
Pianke
ExamPair
Test
PiankeLocal
Defaults
Name: Pianke
Share
Single instance only
Configuration
Code Coverage
Logs
Main class: com.iflytek.edmp.Pianke
VM options: -Dspark.master=local[1]
Program arguments:
Working directory: E:\code\hadoop\spark\pianke\pianke_scala
Environment variables:
Use classpath of module: pianke
JRE: Default (1.7 - SDK of 'pianke' module)

Menu

Spark开发环境搭建——IDEA安装及配置

4. 添加Scala工程的maven archetype

```
File -> new -> Project -> Maven -> Create from archetype -> Add Archetype
groupId：net.alchim31.maven
archetypeId: scala-archetype-simple
version: 1.6
```

5. 创建工程

A. 使用IDEA Maven创建scala工程

B. 使用命令行创建scala工程

```
mvn archetype:generate -DarchetypeGroupId=net.alchim31.maven -DarchetypeArtifactId=scala-archetype-simple -DarchetypeVersion=1.6 -DgroupId=com.company.edmp -DartifactId=demo -Dversion=1.0-SNAPSHOT
```

Spark开发环境搭建——QuickStart

1. 编译代码
2. 上传jar
3. 提交任务
4. 查看结果

演示QuickStart

Scala
基本语法介绍

Scala语法直观体验

问题：已知列表[1, 2, 3, 4, 5]，要求获取每个元素乘2后的一个新列表，即[2, 4, 6, 8, 10]

对比：

Java

Scala

Python

Java

```java
// Java7及其以前
List<Integer> ls = Arrays.asList(1, 2, 3, 4 ,5);
List<Integer> ls2 = new ArrayList<Integer>();
for (Integer i : ls) {
    ls2.add(i * 2);
}
System.out.println(ls2);
```

```java
// Java8+
List<Integer> ls = Arrays.asList(1, 2, 3, 4 ,5);
List<Integer> ls2 = ls.stream().map(i -> i * 2).collect(Collectors.toList());
System.out.println(ls2);
```

Scala

```scala
// Scala 两种写法
val ls = List(1, 2, 3, 4, 5)
val ls2 = new scala.collection.mutable.ListBuffer[Int]()
ls.foreach(i => ls2.append(i * 2))

val ls3 = ls.map(_ * 2)

println(ls2)
println(ls3)
```

Python

```python
# Python
ls = [1, 2, 3, 4, 5]
ls2 = [i * 2 for i in ls]
print(ls2)
```

Java：编码刻板，开发效率不高，执行效率高，易于掌握，代码量较大

Scala：编码灵活，开发效率较高，执行效率高，可优化空间大，入门不易

Python：编码灵活，开发效率极高，执行效率低，门槛低

以上纯属个人见解

Scala基本语法——变量声明

- 变量定义：var，val
- var：声明可变变量
- val：声明不可变变量，推荐使用val

```
› var s = 'hello'
› s = 'world'  // OK
```

```
› val a = 1
› a = 2  // ERROR
› val a = 2 // OK
```

Scala基本语法——包

- 包

  package xxx.xxx.xxx

- 导入

  import xxx.xxx.xxx

```
package com.company.edmp

class User {
    var age = 0
    val name = ""

    def introduce() = {
        "My name is " + name + ", I'm " + age
    }

}


import scala.math.pow
val a = pow(2, 3)  // 8.0
```

Scala基本语法——基本数据类型

数据类型	描述
Byte	8位有符号值。范围从-128到127
Short	16位有符号值。范围从-32768至32767
Int	**32 位有符号值。范围从 -2147483648 to 2147483647**
Long	**64位有符号值。 从-9223372036854775808到9223372036854775807**
Float	**32位IEEE754单精度浮点数**
Double	**64位IEEE754双精度浮点数**
Char	16位无符号Unicode字符。范围由U+0000至U+FFFF
String	**字符序列**
Boolean	**true或false**
Unit	对应于没有值
Null	空或空引用
Nothing	每一个其他类型的子类型；包括无值
Any	Any类型的超类型;任何对象是任何类型
AnyRef	任何引用类型的超类型

```
› val a = 123
  a: Int = 123

› val b = 234L
  b: Long = 234

› val c = 12.34F
  c: Float = 12.34

› val d = 3.14
  d: Double = 3.14

› val e = "Hello Scala"
  e: String = Hello Scala

› val f = true
  f: Boolean = true
```

Scala基本语法——流程控制——判断

- 判断

```
if (expression) {
}


if (expression) {
} else {
}
```

```
val a = 20

if (a >= 25) {
    println("this is if statement")
} else {
    println("a is larger than 25")
}
```

Scala基本语法——流程控制——循环for

- 循环for

```
for(var x <- range) {
}
```

```
import util.control.Breaks.break

for(i <- 1 to 10) {
    if (i % 2 == 0) {
        println("i: " + i);
    }

    if (i == 6) {
        break();
    }
}
```

Scala基本语法——流程控制——循环while

- 循环while

```
while(condition) {
}
```

```
do {
} while(condition)
```

```
var a = 0;
while (a < 10) {
    println("a: " + a)
    a += 1
}
```

```
var b = 10;
do {
    println("b: " + b)
    b -= 1
} while(b > 0)
```

Scala基本语法——函数（常规命名函数）

- 定义方法：def

```
def funcName(parm: Type, …): ReturnType = {
    // 函数内容
    returnValue
}
```

```
/**
  * 计算a,b之和
  */
def add(a: Int, b: Int): Int = {
    a + b
}
```

```
› println(add(2, 7))
  9
```

Scala基本语法——变参函数

- 变参函数

```
def funcName(parm: Type*): ReturnType = {

    // 函数内容
    returnValue
}
```

```
def show(names: String*) = {
    for (name <- names) {
        println(name)
    }
}
```

```
› show("scala", "spark", "wo~")
  scala
  spark
  wo~
```

Scala基本语法——默认值参数函数

- 函数默认值

```
def funcName(parm: Type,
             param: Type = value,
             …
): ReturnType = {

    // 函数内容
    returnValue
}
```

```
/**
  * 计算a,b之和，b默认为0
  */
def add(a: Int, b: Int = 0): Int = {
    a + b
}

› add(2)
  2
› add(2, 3)
  5
```

Scala基本语法——匿名函数

- lambda表达式

```
› val ls = List(1, 2, 3, 4, 5)
› ls.foreach((a) => println(a*2))
```

- 匿名函数

```
(param: Type, …) => …
```

```
› val add = (a: Int, b: Int) => a + b
› add(2, 4)
  6
```

- 命名

```
val funcName = (param: Type, …) => …
```

Scala基本语法——常用基本集合（collection）

- 数组Array
  长度固定，类型相同，元素可变
- 列表List
  长度固定，类型相同，元素不可变
- 集合Set
  元素不重复
- 元组
  长度固定，类型不同，元素不可变
- 映射
  key-value结构

```
› val a = Array(1,2,3)
› a(0) = 5
› a
  Array[Int] = Array(5, 2, 3)

› val ls = List(1, 2, 3)
› ls(0) = 5  // ERROR

› val s = Set(1, 2, 3, 1)
  s: scala.collection.immutable.Set[Int] = Set(1, 2, 3)

› val t = (1, "hello", 3.14)
  t: (Int, String, Double) = (1,hello,3.14)
› t._2
  String = hello
› t._2 = "world"  // ERROR

› val m = Map("name" -> "tom", "age" -> 21)
› m("name")
  Any = tom
```

Scala基本语法——常用基本集合（collection）

- 可变类型集合

```
import scala.collection.mutable._
```

- 集合类型极为丰富
  - 定长 – 变长
  - 连续 – 链式
  - 同类型 – 不同类型
  - 可修改 – 不可修改
- 可优化空间大

Scala基本语法——Class & Object

- Class普通类

- Object

  所有方法都为static

- **Object中定义main函数，作为程序启动的入口**

```
class User {
    var age = 0
    val name = ""

    def introduce() = {
        "My name is " + name + ", I'm " + age
    }

}


object MainApp {
    def main(args: Array[String]) {
        // 程序入口
    }
}
```

Scala基本语法——异常处理

- 异常捕获和处理

```
try {
    // 需要被捕获异常的语句
} catch {
    case ex: ExceptionType1 => {
        // 处理异常
    }
    case ex: ExceptionType2 => {
        // 处理异常
    }
} finally {
    // 最终所要执行的语句
}
```

```
val a = 10
var b = 20

try {
    b = a / 0
} catch {
    case ex: ArithmeticException => {
        println("Error: / by zero")
    }
} finally {
    b = 0
    println("done finally")
}

println(b) // out: 0
```

Spark Scala
常用语法介绍

API doc

Scala

http://spark.apache.org/docs/latest/api/scala/index.html

Python

http://spark.apache.org/docs/latest/api/python/index.html

Java

http://spark.apache.org/docs/latest/api/java/index.html

常用SQL

- load & save
- limit
- select
- where
- order by
- group by
- distinct
- join
- union

Spark基础语法——load & save

- 载入数据和保存数据

```
sqlContext.read.load(path: String)

dataFrame.write.save(path: String)
```

```
› import org.apache.spark.sql.SQLContext
› val sqlContext = new SQLContext(sc)
// 以后将省略以上语句

› val dfUser =
  sqlContext.read.load("/path/user_info.parquet")


› df.save.write("/path/out_dir.parquet")
```

Spark基础语法——limit

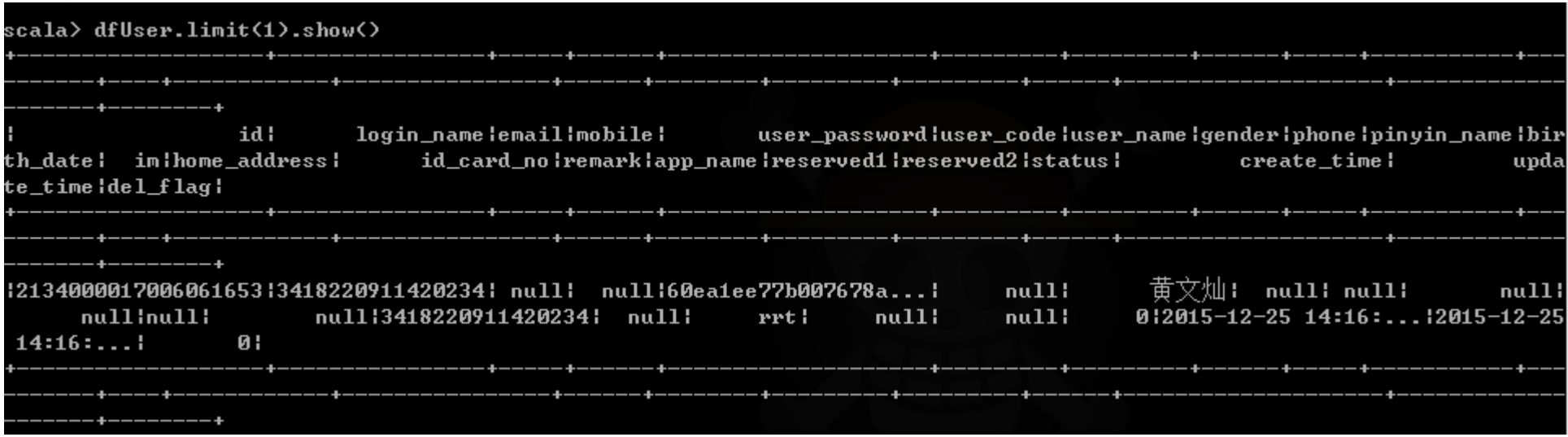
- SQL

  ```
  limit n
  ```

- Spark

  ```
  df.limit(n: Int)
  ```

```
› dfUser.limit(1).show()
```



Spark基础语法——select

- SQL
  SELECT col1, col2, col3
- Spark
  df. select(col: String, cols: String*)
  df.select(col: Column*)

- SQL
  AS name
- Spark
  alias(name: String)
- df.alias = df.as
- 适用于column和table

```
› dfUser.select(
      dfUser("id"),
      $"login_name",
      $"app_name".alias("app")
  ).limit(1).show()

+-------------------+----------------+---+
|                 id|      login_name|app|
+-------------------+----------------+---+
|2134000017006061653|341822091142023 4|rrt|
+-------------------+----------------+---+
```

Spark基础语法——where

- SQL

  WHERE expresions

- Spark

  df.filter(conditionExpr: String)

  df.filter(condition: Colum)

  df.where = df.filter

```
› dfUser.filter($"gender" === 1)
      .filter("app_name = 'rrt'")
      .select("id", "gender", "app_name")
      .limit(1).show()

+------------------+------+--------+
|                id|gender|app_name|
+------------------+------+--------+
|2134000017006061667|     1|     rrt|
+------------------+------+--------+
```

Spark基础语法——order by

- SQL

```
ORDER BY col1 ASC|DESC, col2 ASC|DESC
```

- Spark

```
df.orderBy(sortExprs: Column*)
df.orderBy(sortCol: String,
    sortCols: String*)

df.sort = df.orderBy
```

```
› dfUser.select("app_name", "id")
      .orderBy($"app_name".desc, $"id")
      .limit(3).show()
```

```
+--------+-------------------+
|app_name|                 id|
+--------+-------------------+
|     rrt|2134000017006061653|
|     rrt|2134000017006061654|
|     rrt|2134000017006061655|
+--------+-------------------+
```

Spark基础语法——group by

- SQL

  ```
  GROUP BY col1, col2
  ```

- Spark

  ```
  df.groupBy(cols: Column*)
  df.groupBy(col1: String, cols: String*)
  ```

```
› dfUser.groupBy($"app_name", $"gender")
      .count().show()

+-----------+------+------+
|   app_name|gender| count|
+-----------+------+------+
|        jsj|     2|     1|
|        jsj|     0|    17|
|        rrt|     2|     1|
|       null|     0|    10|
|        jsj|  null|    78|
|        rrt|      |    18|
|        jsj|     1|     3|
|        rrt|     1|269382|
……
```

Spark基础语法——distinct

- SQL

  DISTINCT col

- Spark

  df.distinct()

```
› dfUser.select($"app_name").distinct().show()

+-----------+
|   app_name|
+-----------+
|        rrt|
|       null|
|qypt_luyang|
|        jsj|
|qyjyzljc_ah|
|    qxpt_hf|
+-----------+
```

Spark基础语法——join

- SQL

```
table1 [LEFT|INNER|…] JOIN table2
    ON table1.col1 = table2.col2
```

- Spark

```
df.join(right: DataFrame,
    joinExprs: Column,
    joinType: String)
```

```
› dfUser.count()
  Long = 740867
› dfLogLogin.count()
  Long = 856551

› dfUser.join(
      dfLogLogin,
      dfUser("id") === dfLogLogin("user_id")
  ).count()
  Long = 30959

› dfUser.join(
      dfLogLogin,
      dfUser("id") === dfLogLogin("user_id"),
      "left"
  ).count()
  Long = 761220
```

Spark基础语法——union

- SQL

  table1 UNION ALL table2

- Spark

  df. unionAll(other: DataFrame)

注：UNION消除重复行，UNION ALL不消除重复行

```
› val dfUserMale = dfUser.filter($"gender"
  === "1").limit(10)
› val dfUserFamale = dfUser.filter($"gender"
  === "0").limit(10)
› val dfUnion =
  dfUserMale.unionAll(dfUserFamale)
› dfUnion.groupBy("gender").count().show()
  +------+-----+
  |gender|count|
  +------+-----+
  |     0|   10|
  |     1|   10|
  +------+-----+
```

```
› dfUnion.unionAll(dfUserMale).count()
  Long = 30
```

Q&A

附：
pyspark
开发及调试

pyspark开发工具简介

命令行工具——pyspark（推荐：Python2.7+，IPython）

开发IDE——PyCharm

直接使用spark-submit提交py脚本，无需编译