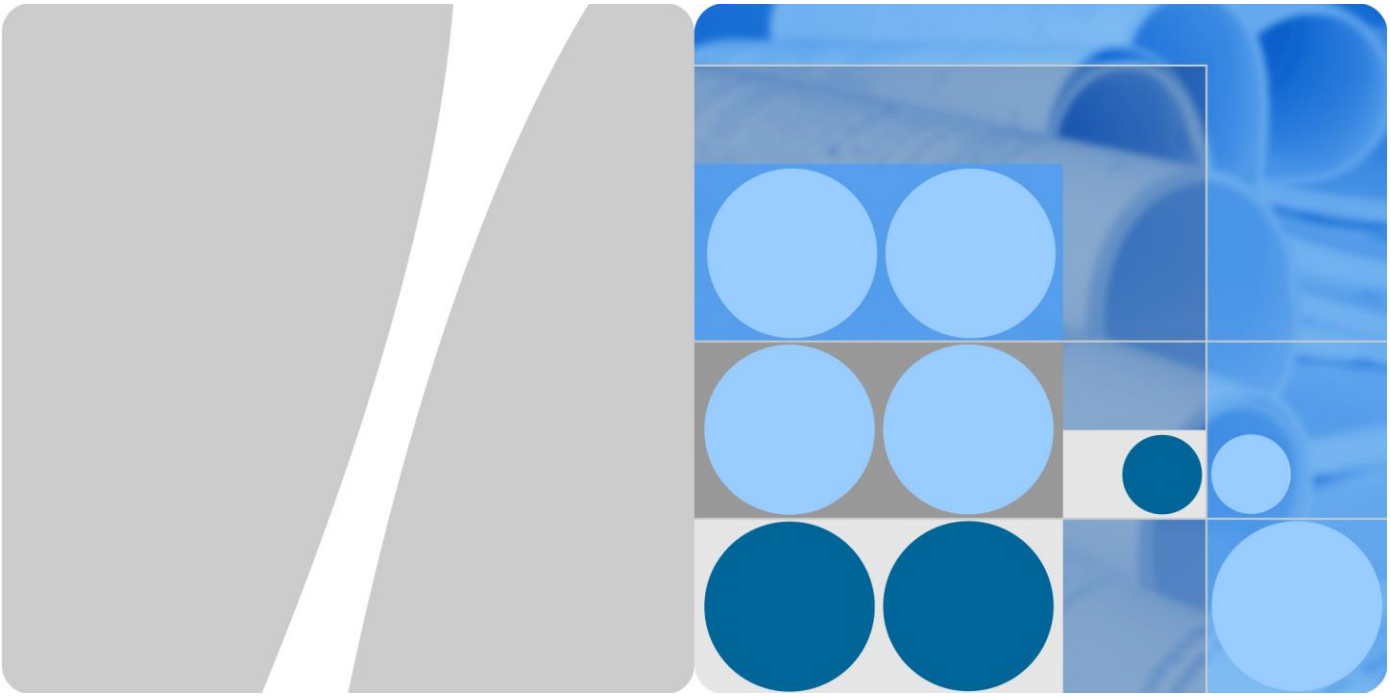


Spark 开发指南

Spark 开发指南

文档版本 01

发布日期 2015-12-25

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 Spark 开发指南

1.1 概述

目标读者

本文档面向 Spark 应用开发人员，并要求用户具备一定的 Java 和 Scala 的开发经验。

简介

Spark 是分布式批处理框架，提供分析挖掘与迭代式内存计算能力，支持多种语言（Scala/Java/Python）的应用开发。 适用以下场景：

- 数据处理（Data Processing）：可以用来快速处理数据，兼具容错性和可扩展性。
- 迭代计算（Iterative Computation）：支持迭代计算，有效应对多步的数据处理逻辑。
- 数据挖掘（Data Mining）：在海量数据基础上进行复杂的挖掘分析，可支持各种数据挖掘和机器学习算法。
- 流式处理（Streaming Processing）：支持秒级延迟的流式处理，可支持多种外部数据源。
- 查询分析（Query Analysis）：支持标准 SQL 查询分析，同时提供 DSL（DataFrame）， 并支持多种外部输入。

基本概念

- **RDD**

  全名：弹性分布数据集（Resilient Distributed Dataset）

  它是 Spark 的核心概念，指的是一个只读，可变分区的分布式数据集，该数据集的内容可以缓存在内存中，并在多次计算中间重用。

  RDD 的生成：

  - 从 Hadoop 文件系统（或与 Hadoop 兼容的其它存储系统）输入（例如 HDFS）创建 。
  - 从父 RDD 转换得到新 RDD。
  - 从集合转换而来。

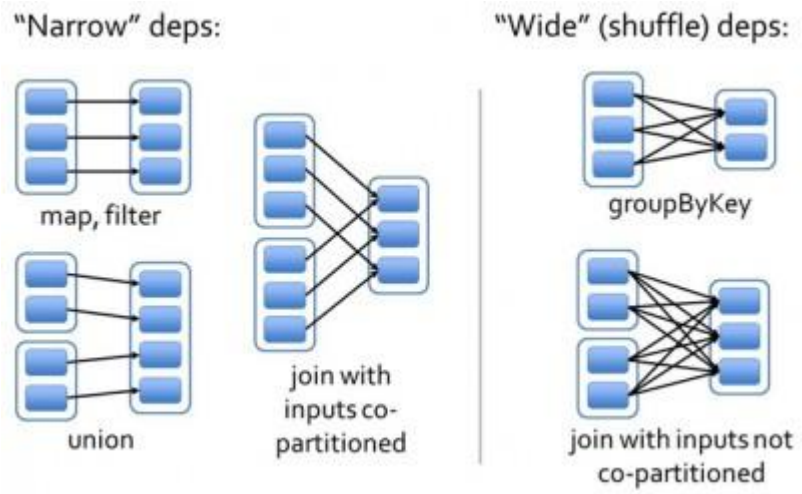
RDD 的存储：

- 用户可以选择不同的存储级别（例如 DISK_ONLY，MEMORY_AND_DISK）存储 RDD 以便重用。
- 当前 RDD 默认只存储于内存，当内存不足时，RDD 也不会溢出到磁盘中。

- **Dependency（RDD 的依赖）**

  父 RDD 与子 RDD 之间的逻辑关系，可分为窄依赖和宽依赖两种。

  图1-1 RDD 的依赖



  - **窄依赖：**指父 RDD 的每一个分区最多被一个子 RDD 的分区所用，表现为一个父 RDD 的分区对应于一个子 RDD 的分区，或者两个父 RDD 的分区对应于一个子 RDD 的分区。上图中，map/filter 和 union 属于第一类前者，对输入进行协同划分（co-partitioned）的 join 属于第二类后者。
  - **宽依赖：**指子 RDD 的分区依赖于父 RDD 的所有分区，子 RDD 必须要通过 shuffle 类操作实现父 RDD 的数据全部重新分配，如上图中的 groupByKey 和未经协同划分的 join。

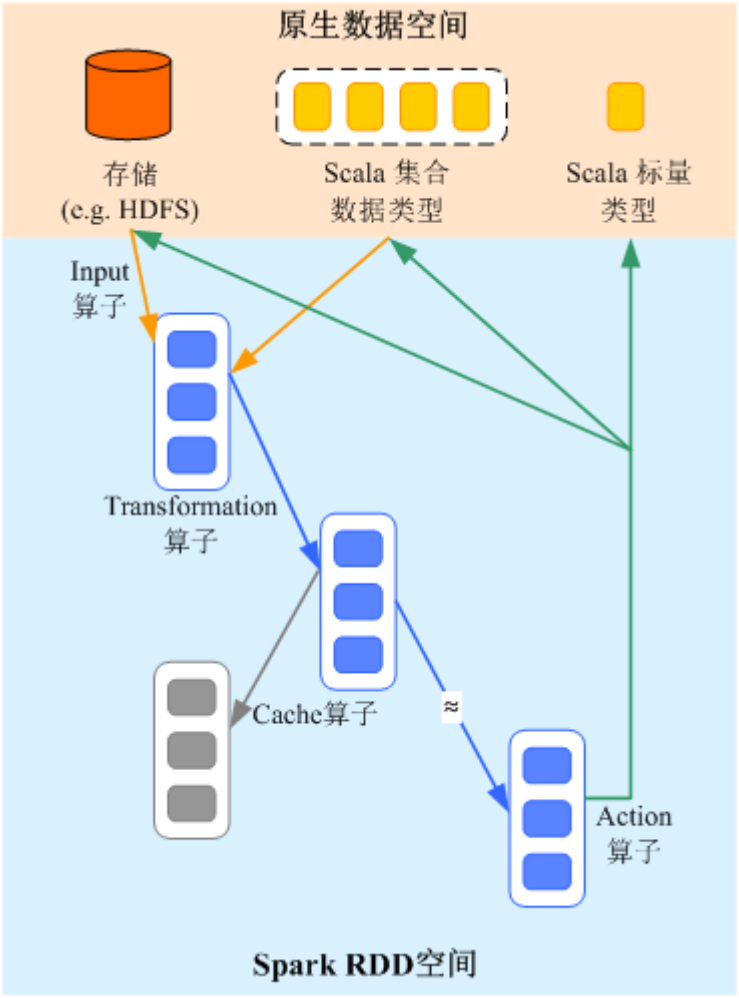
  窄依赖对优化很有利。逻辑上，每个 RDD 的算子都是一个 fork/join（此 join 非上文的 join 算子，而是指同步多个并行任务的 barrier）： 把计算 fork 到每个分区，算完后 join，然后 fork/join 下一个 RDD 的算子。如果直接翻译到物理实现，是很不经济的：一是每一个 RDD（即使是中间结果）都需要物化到内存或存储中，费时费空间；二是 join 作为全局的 barrier，是很昂贵的，会被最慢的那个节点拖死。如果子 RDD 的分区到父 RDD 的分区是窄依赖，就可以实施经典的 fusion 优化，把两个 fork/join 合为一个；如果连续的变换算子序列都是窄依赖，就可以把很多个 fork/join 并为一个，不但减少了大量的全局 barrier，而且无需物化很多中间结果 RDD，这将极大地提升性能。Spark 把这个叫做流水线 （pipeline）优化。

- **Transformation 和 Action（RDD 的操作）**

  对 RDD 的操作包含 Transformation（返回值还是一个 RDD）和 Action（返回值不是一个 RDD）两种。RDD 的操作流程如图 1-2 所示。其中 Transformation 操作是 Lazy 的，也就是说从一个 RDD 转换生成另一个 RDD 的操作不是马上执行，Spark 在遇到 Transformation 操作时只会记录需要这样的操作，并不会去执行，需

要等到有 Action 操作的时候才会真正启动计算过程进行计算。Action 操作会返回结果或把 RDD 数据写到存储系统中。Action 是触发 Spark 启动计算的动因。

图1-2 RDD 操作示例



然后，来看一个简单的例子，如图 1-3 所示。RDD 看起来与 Scala 集合类型没有太大差别，但它们的数据和运行模型大相迥异。

1. **textFile** 算子从 HDFS 读取日志文件，返回 **file**（初始 RDD）。
2. filter 算子筛出带“ERROR”的行，赋给 errors（新 RDD）。filter 算子为一个 Transformation 操作。
3. cache 算子把它缓存下来以备未来使用。
4. count 算子返回 errors 的行数。count 算子为一个 Action 操作。

图1-3 Scala 样例

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

Transformation 操作可以分为如下几种类型：

- 视 RDD 的元素为简单元素。

  输入输出一对一，且结果 RDD 的分区结构不变，主要是 map。

  输入输出一对多，且结果 RDD 的分区结构不变，如 flatMap（map 后由一个元素变为一个包含多个元素的序列，然后展平为一个个的元素）。

  输入输出一对一，但结果 RDD 的分区结构发生了变化，如 union（两个 RDD 合为一个，分区数变为两个 RDD 分区数之和）、coalesce（分区减少）。

  从输入中选择部分元素的算子，如 filter、distinct（去除重复元素）、subtract（本 RDD 有、它 RDD 无的元素留下来）和 sample（采样）。

- 视 RDD 的元素为 Key-Value 对。

  对单个 RDD 做一对一运算，如 mapValues（保持源 RDD 的分区方式，这与 map 不同）；

  对单个 RDD 重排，如 sort、partitionBy（实现一致性的分区划分，这个对数据本地性优化很重要）；

  对单个 RDD 基于 key 进行重组和 reduce，如 groupByKey、reduceByKey；

  对两个 RDD 基于 key 进行 join 和重组，如 join、cogroup。

说明

后三种操作都涉及重排，称为 shuffle 类操作。

Action 操作可以分为如下几种：

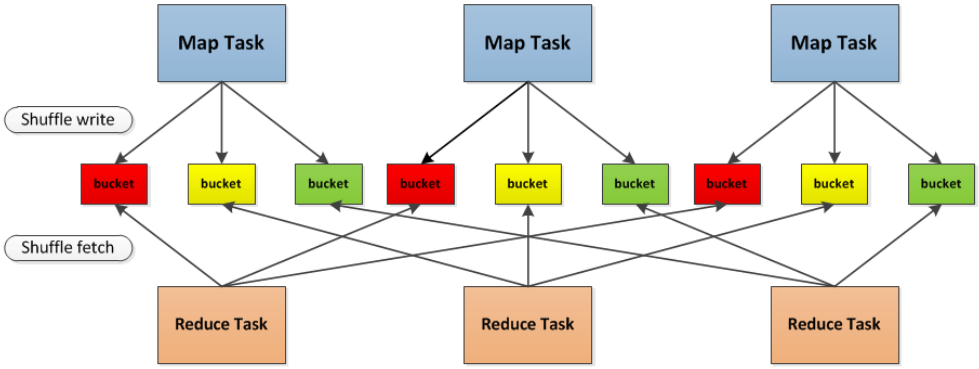
- 生成标量，如 count（返回 RDD 中元素的个数）、reduce、fold/aggregate（返回几个标量）、take（返回前几个元素）。
- 生成 Scala 集合类型，如 collect（把 RDD 中的所有元素倒入 Scala 集合类型）、lookup（查找对应 key 的所有值）。
- 写入存储，如与前文 textFile 对应的 saveAsTextFile。
- 还有一个检查点算子 checkpoint。当 Lineage 特别长时（这在图计算中时常发生），出错时重新执行整个序列要很长时间，可以主动调用 checkpoint 把当前数据写入稳定存储，作为检查点。

- **Shuffle**

  Shuffle 是 Spark 框架中一个特定的 phase，当 Map 的输出结果要被 Reduce 使用时，输出结果需要按 key 哈希，并且分发到每一个 Reducer 上去，这个过程就是 shuffle。由于 shuffle 涉及到了磁盘的读写和网络的传输，因此 shuffle 性能的高低直接影响到了整个程序的运行效率。

  下图清晰地描述了 Shuffle 算法的整个流程。

图1-4 算法流程



- **Spark Application 的结构**

  Spark Application 的结构可分为两部分：初始化 SparkContext 和主体程序。

  - 初始化 SparkContext：构建 Spark Application 的运行环境。

    构建 SparkContext 对象，如：

    ```
    new SparkContext(master, appName, [SparkHome], [jars])
    ```

    参数介绍：
    master：连接字符串，连接方式有 local，yarn-cluster，yarn-client 等
    appName：构建的 Application 名称
    SparkHome：集群中安装 Spark 的目录
    jars：应用程序代码和依赖包

  - 主体程序：处理数据

- **Spark shell 命令**

  Spark 基本 shell 命令，支持提交 Spark 应用。命令为：

  ```
  ./bin/spark-submit \
    --class <main-class>
    --master <master-url> \
    ... # other options
    <application-jar> \
    [application-arguments]
  ```

  参数解释：
  --class：Spark 应用的类名
  --master：Spark 用于所连接的 master，如 yarn-client，yarn-cluster 等
  application-jar：Spark 应用的 jar 包的路径
  application-arguments：提交 Spark 应用的所需要的参数（可以为空）。

- **Spark Web UI 界面**

  用于监控正在运行的或者历史的 Spark 作业在 Spark 框架各个阶段的细节以及提供日志显示，帮助用户更细粒度地去开发、配置和调优作业。

1.2 开发环境准备

1.2.1 Java 开发环境准备

操作场景

本开发指南提供了 Spark 组件的样例代码和常用接口，便于开发者快速熟悉 Spark 并行计算框架。为了运行 Spark 组件的样例代码，你需要完成下面的操作。

开发环境可以搭建在 Windows 环境下，而运行环境（即客户端）只能部署在 Linux 环境下。

操作步骤

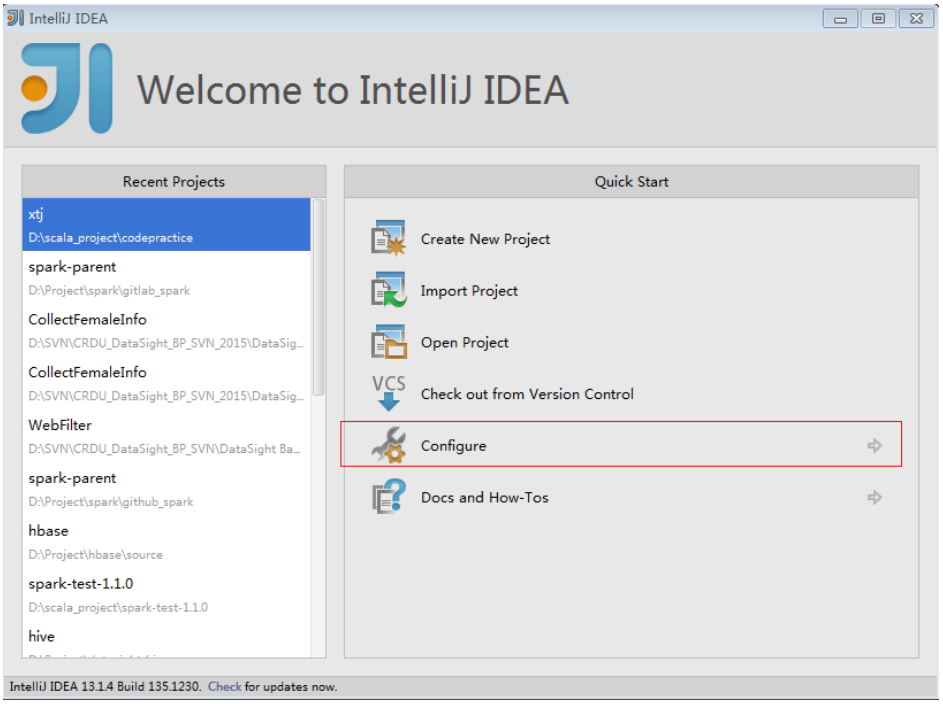
步骤 1 对于 Java 开发环境，推荐使用 IDEA 工具，安装要求如下，具体软件，请到对应的官方网站获取。

- JDK 使用 1.7 版本（或 1.8 版本）
- IntelliJ IDEA（版本：13.1.4，及以上）
- MAVEN（3.3.1 及以上）

步骤 2 安装 IntelliJ IDEA、JDK、MAVEN，并进行相应的配置。本章节以 IntelliJ IDEA 13.1.4 版本为例。

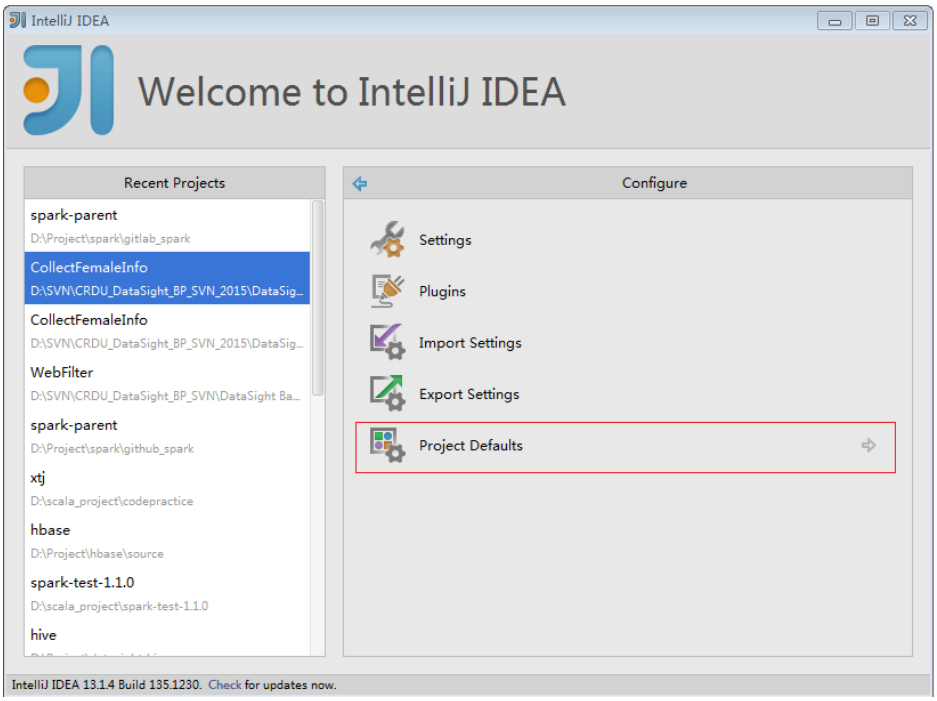
1. 安装 JDK。
2. 安装 IntelliJ IDEA 工具。
3. IntelliJ IDEA 中配置 JDK。
   a. 打开 IntelliJ IDEA，选择首页中的“Configure”，如图 1-5 所示。

图1-5 Quick Start



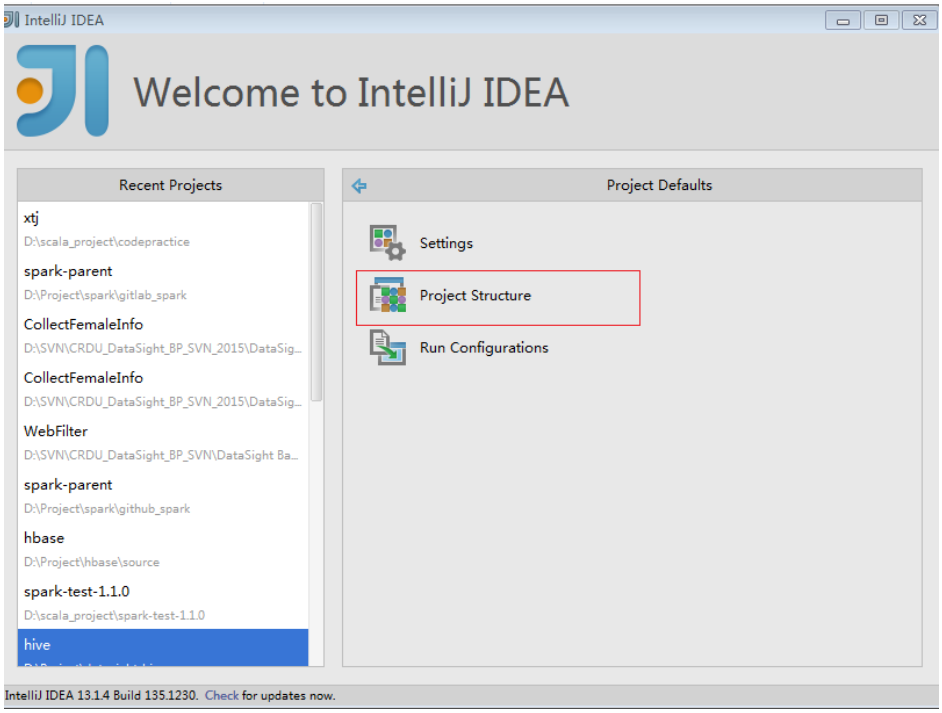
b. 接着选择 Configure 页面中的“Project Defaults”，如图 1-6 所示。

图1-6 Configure



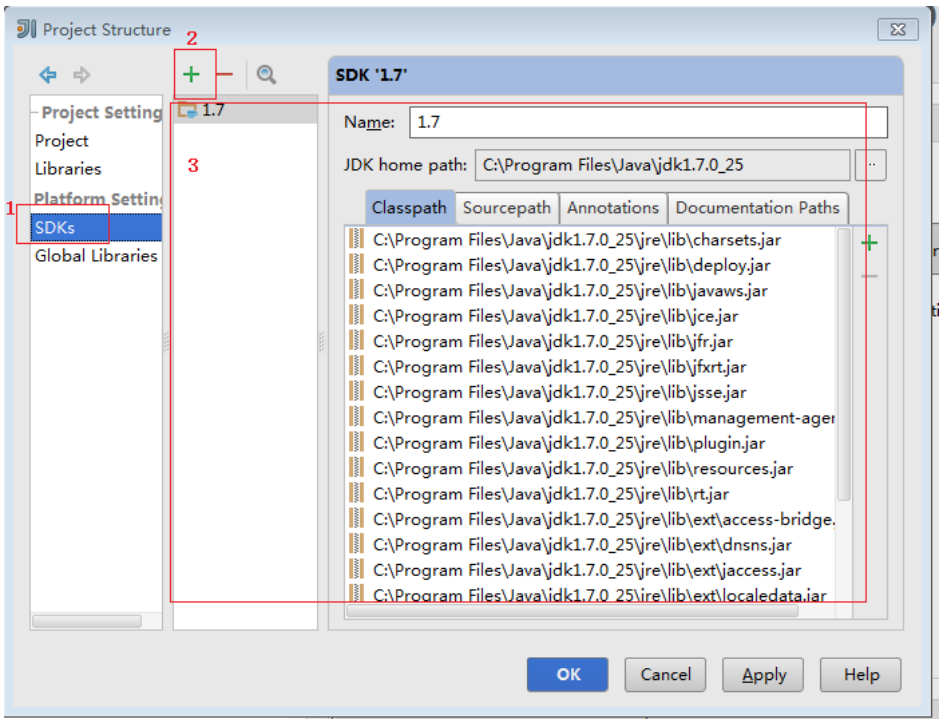
c. 在“Project Defaults”页面中选择“Project Structure”，如图 1-7 所示。

图1-7 Project Defaults



d. 在打开的“Project Structure”页面中，选择“SDKs”，点击“+”并添加JDK，如图 1-8 所示。

图1-8 添加 JDK



步骤 3 下载 Spark 客户端程序到客户端机器中。

1. 登录 FusionInsight Manager 系统。

   在浏览器地址栏中输入访问地址，地址格式为“http://*FusionInsight Manager 系统的 WebService 浮动 IP 地址*:8080/web”。

   例如，在 IE 浏览器地址栏中，输入“http://10.10.10.172:8080/web”。

2. 单击“Services > Spark > Download Client”，“Client type”勾选“All Client Files”，单击“OK”开始下载客户端，等待下载完成。

步骤 4 解压缩客户端压缩文件包“FusionInsight_V100R002C50SPC200_Spark_Client.tar”。文件为 tar 格式，可采用如下方式解压。

说明

客户端程序文件包为 tar 格式，可采用如下方式解压。

- Windows: 下载 7zip 工具，双击压缩包进行解压。
- Linux: 执行 **tar xf** *[客户端文件包]* 命令解压。

步骤 5 导入 Java 代码样例工程到 IDEA 开发环境。

1. 在 windows 上解压后的客户端安装包中存在 Spark 的样例工程“Spark\sampleCode\SparkJavaExample”。

说明

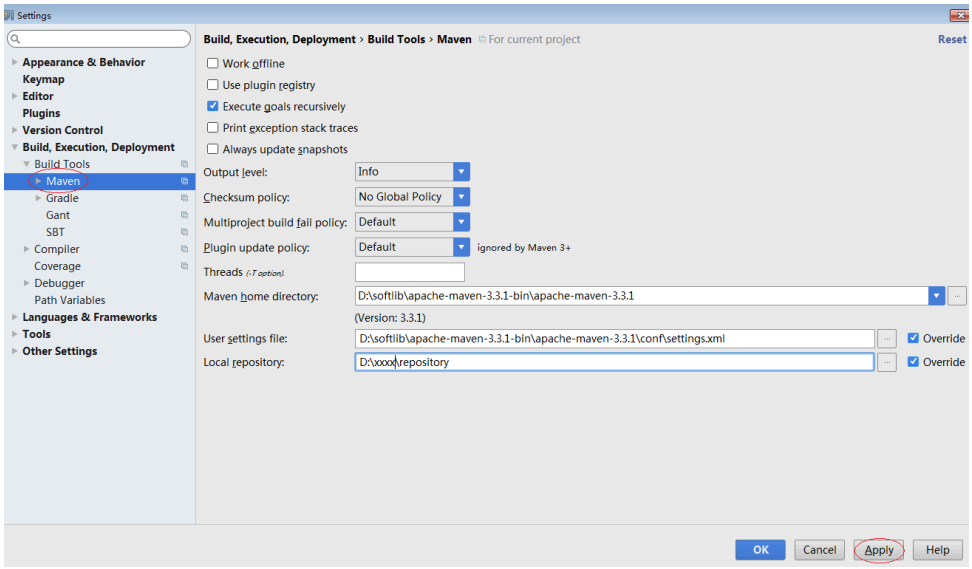
由于样例工程在 windows 中搭建，所以需要把该工程拷贝到 windows 系统中，方便对样例工程进行操作。

2. 在“Spark\sampleCode\SparkJavaExample”目录下新建一个 lib 文件夹，将“Spark\dev_lib”下的所有 jar 包拷贝到新建的 lib 文件夹下，然后删除 log4j.jar。
3. 为工程设置 MAVEN。

   单击“File > Settings...”，打开设置对话框，如下图所示。

   在框中输入 MAVEN，搜索 MAVEN 设置项（若未安装 MAVEN，需要先安装），然后根据实际情况配置 **MAVEN home directory**，**User settings file**，**Local repository** 等几配置项，完成 MAVEN 绑定，设置后单击“APPLY”保存修改，最后单击“OK”。

图1-9 设置 MAVEN

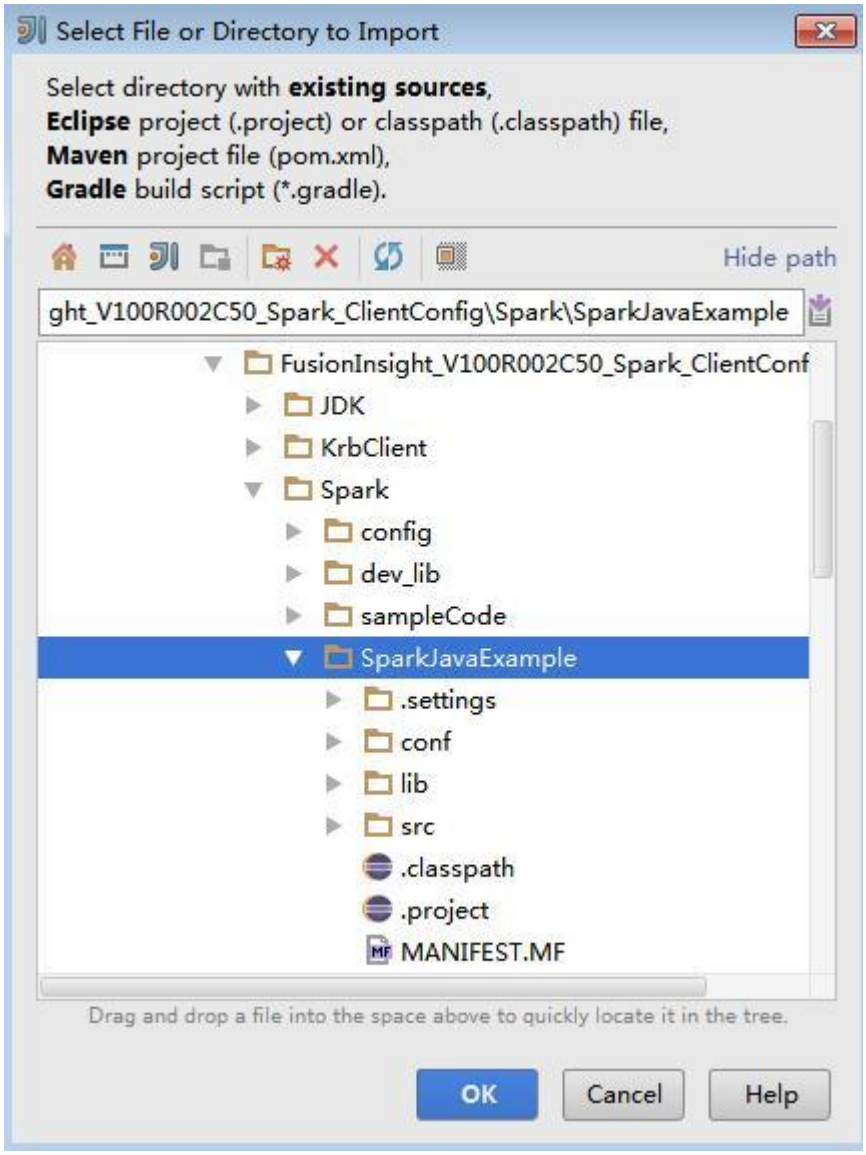


4. 导入工程。

单击“File > New > Project from Existing Sources...”，然后选择下载的样例工程 SparkJavaExample 文件夹，单击“OK”。

在弹出窗口的“Import project from external model”项中选择“Maven”，接下来继续选择“Next”，给工程文件命名，点击“Finsh”完成工程的导入。

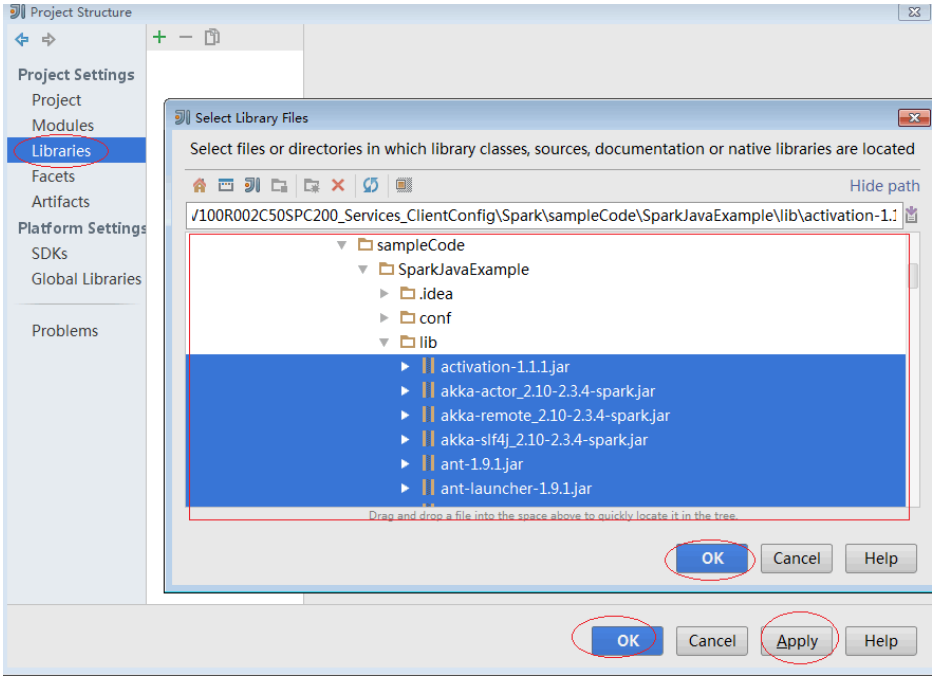
图1-10 导入工程



5. 加入 lib 包。

   导入工程后，如果代码报红，此时还需要加入 lib 包，单击“File > Project Structure > Project Settings > Libraries”，点击“+”，选择“Java”，然后在 SparkJavaExample 文件夹，选中 lib 目录下的所有 jar 包，点击“OK”。

图1-11 加入 lib 包



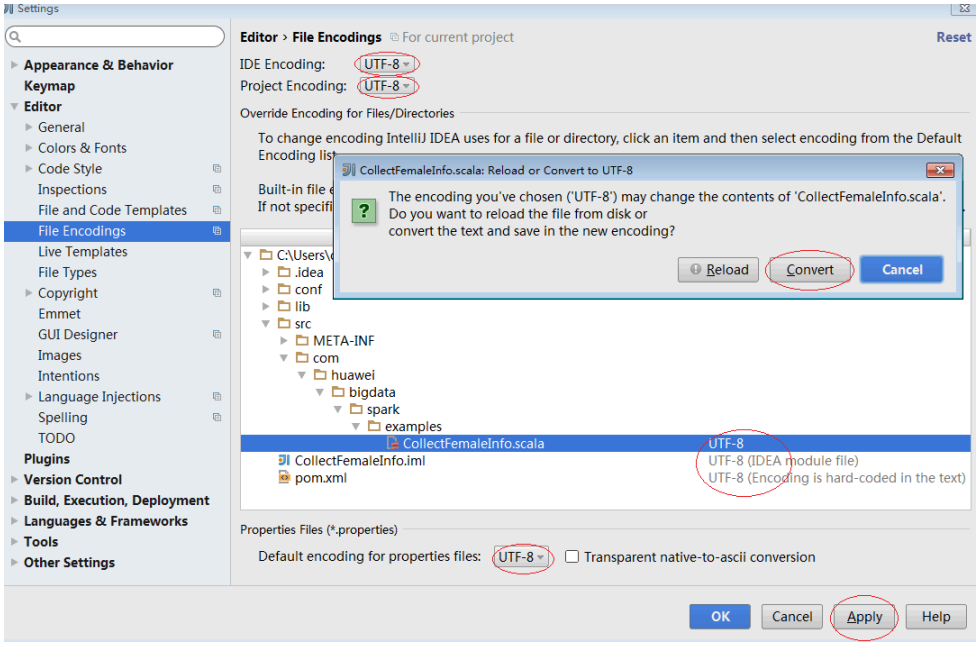
然后继续单击“Project Settings > Modules > Dependecies”，点击右边的“+”选择“Library... > Java”添加前面步骤中新建的 lib 下的 jar 包（如果 jar 包已有，就不需要加）。

6. 设置 IDEA 的文本文件编码格式。

   在 IDEA 的菜单栏中，单击“File > Settings...”弹出设置窗口。

   在左边导航上搜索“File Encodings”，然后将“IDE Encoding”、“Project Encoding”均设置为“UTF-8”，然后单击“Apply”，在弹出的面框中选择“Convert”，单击“OK”。

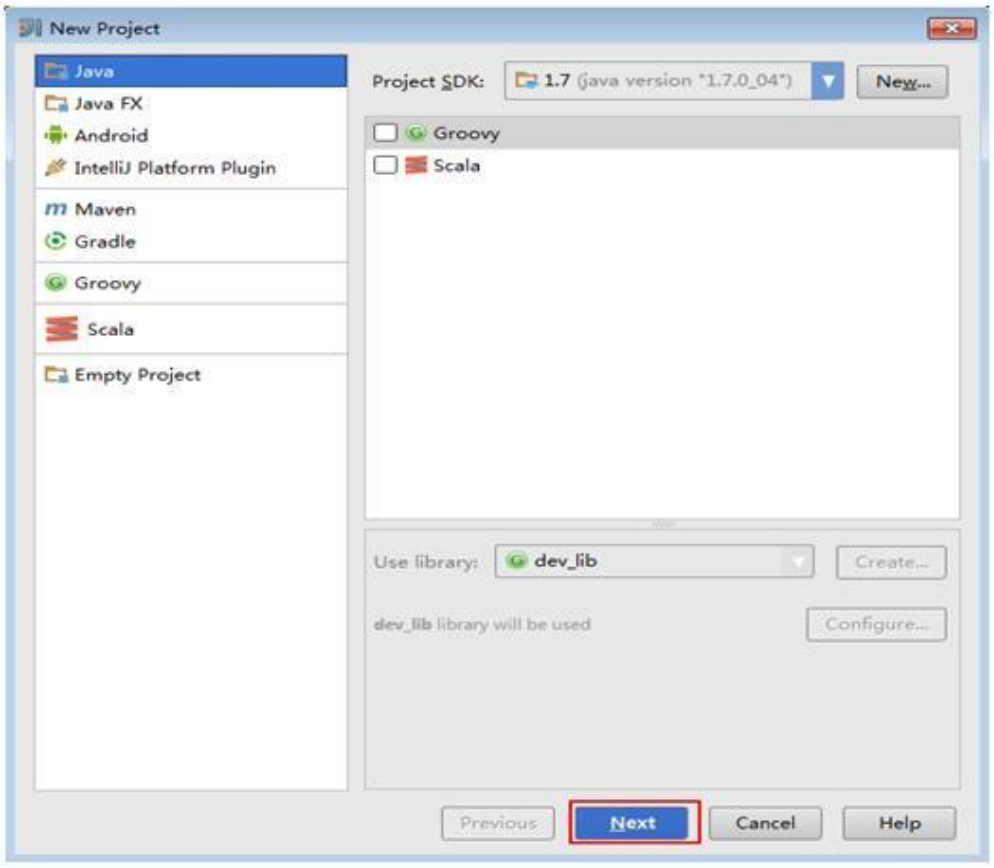
图1-12 设置编码格式



步骤 6 （可选）使用 IDEA 创建一个 Spark 样例工程。

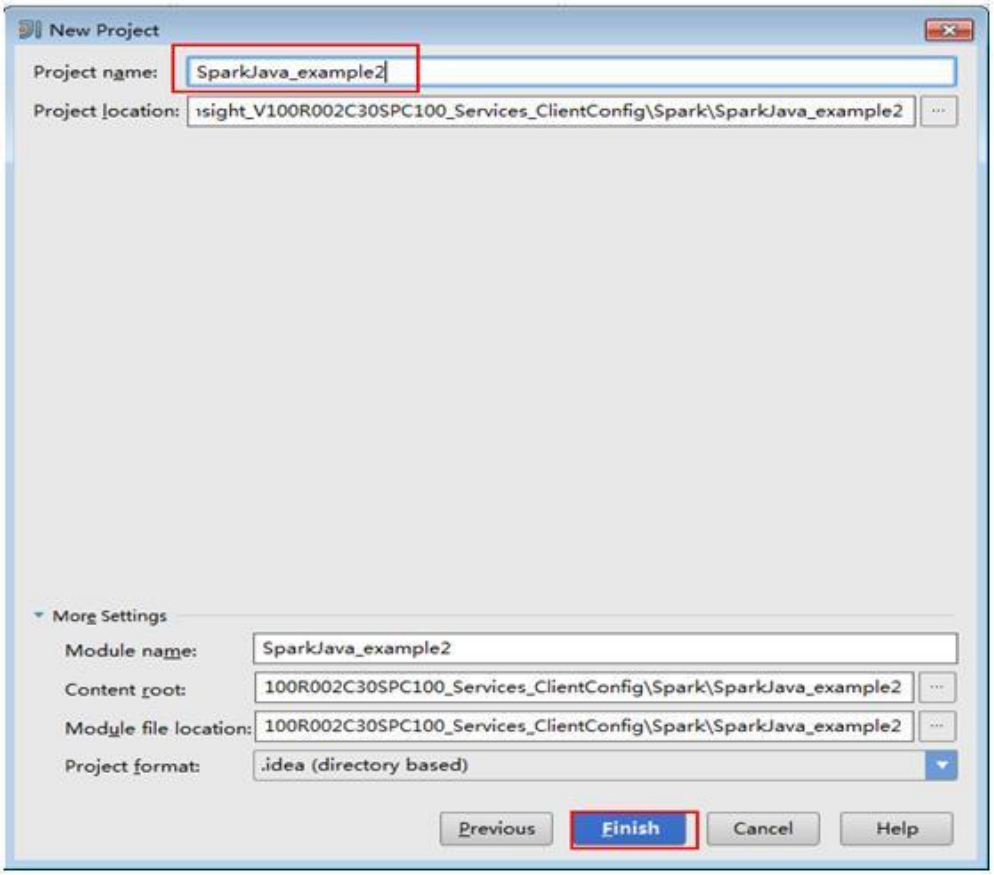
1. 在 IDEA 界面，单击“File > New > Project...”。
2. 如图 1-13 所示，选择“Java”开发环境，并选择“Groovy”，然后单击“Next”。

图1-13 选择开发环境



3. 在如图 1-14 所示的界面中，填写工程名称和存放路径，然后单击“Finish”完成工程创建。

图1-14 填写工程信息



----结束

1.2.2 Scala 开发环境准备

操作场景

本开发指南提供了 Spark 组件的样例代码和常用接口，便于开发者快速熟悉 Spark 并行计算框架。为了运行 Spark 组件的样例代码，你需要完成下面的操作。

开发环境可以搭建在 Windows 环境下，而运行环境（即客户端）只能部署在 Linux 环境下。

操作步骤

步骤 1 对于 scala 开发环境，推荐使用 IDEA 工具，安装要求如下，具体软件，请到对应的官方网站获取。

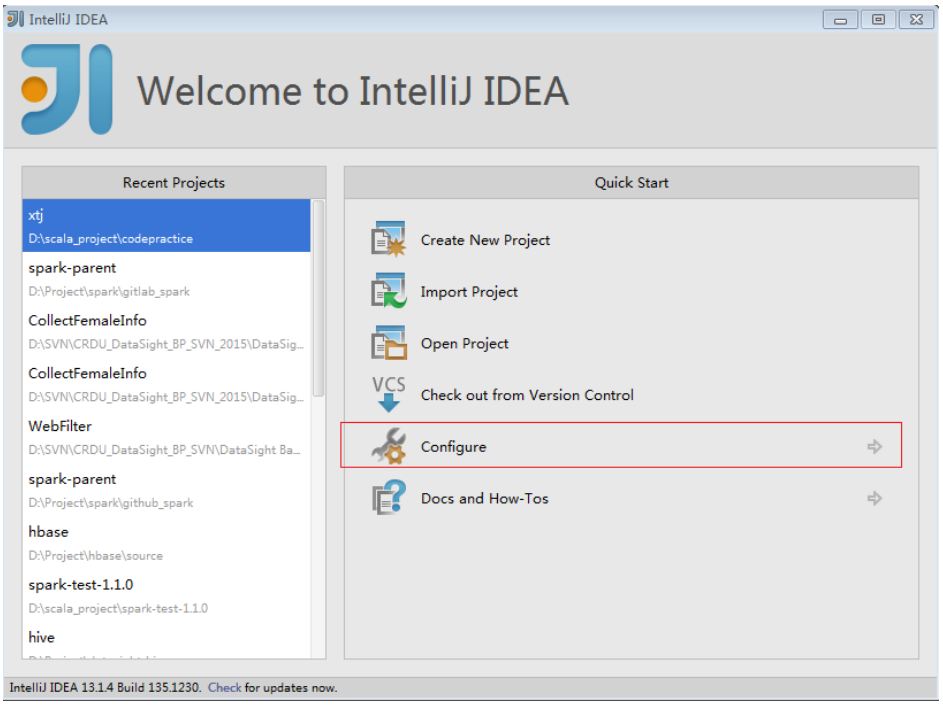
- JDK 使用 1.7 版本（或 1.8 版本）
- IntelliJ IDEA（版本：13.1.4）
- Scala（版本：2.10.4）

- MAVEN （版本 3.3.1）

步骤 2 安装 IntelliJ IDEA、JDK、MAVEN 和 Scala 工具，并进行相应的配置。

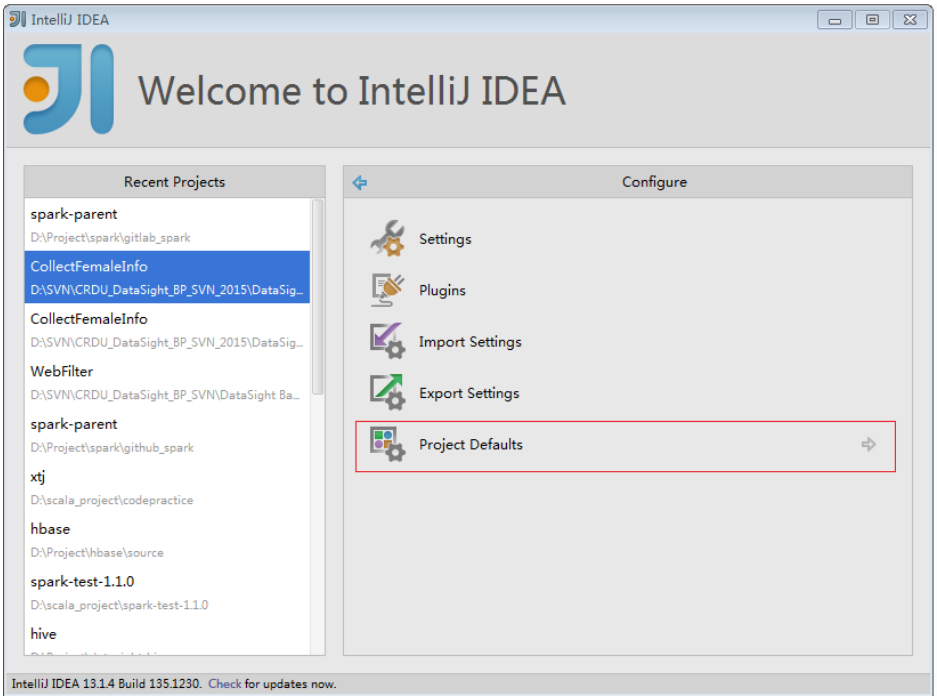
1. 安装 JDK。
2. 安装 IntelliJ IDEA。
3. IntelliJ IDEA 中配置 JDK。
   a. 打开 IntelliJ IDEA，选择首页中的“Configure”，如图 1-15 所示。

图1-15 Quick Start



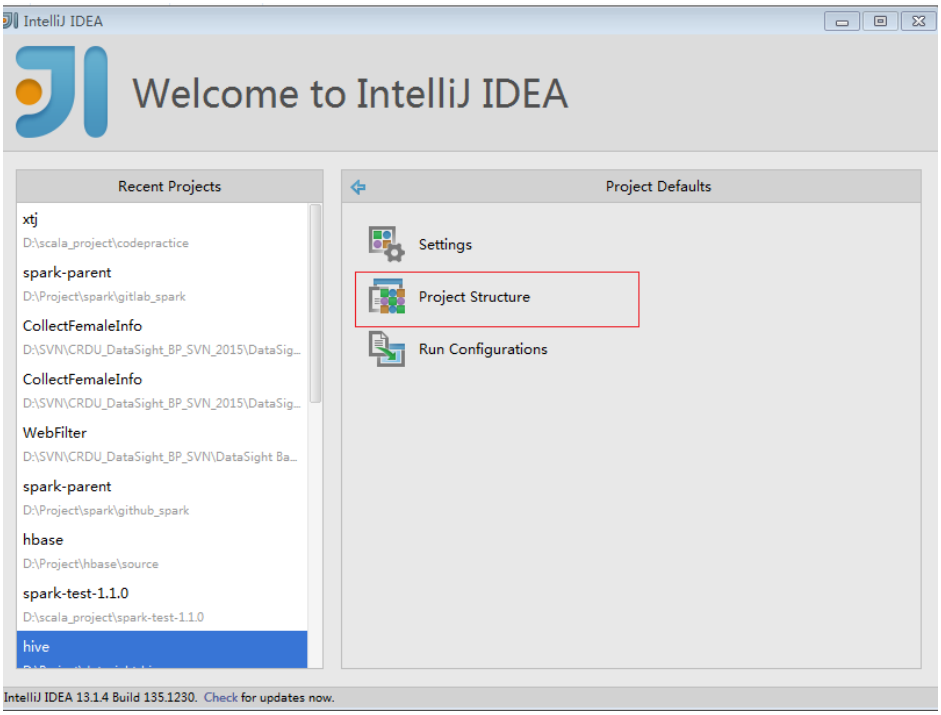
   b. 接着选择“Configure”页面中的“Project Defaults”，如图 1-16 所示。

图1-16 Configure



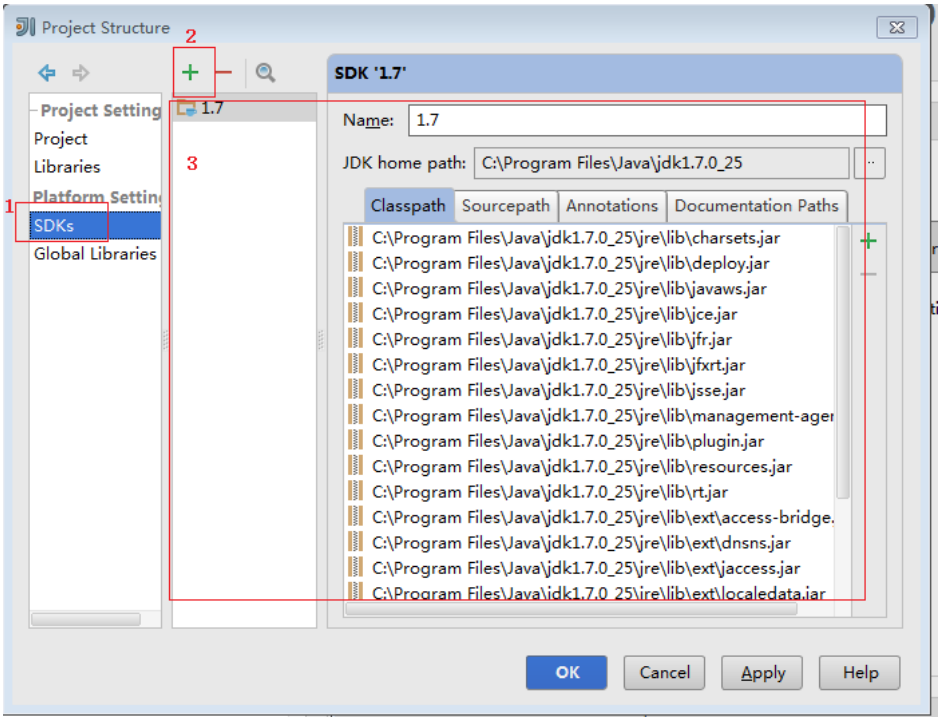
c. 在“Project Defaults”页面中选择“Project Structure”，如图 1-17 所示。

图1-17 Project Defaults



d. 在打开的“Project Structure”页面中，选择“SDKs”，并添加“JDK”，如图 1-18 所示。

图1-18 添加 JDK

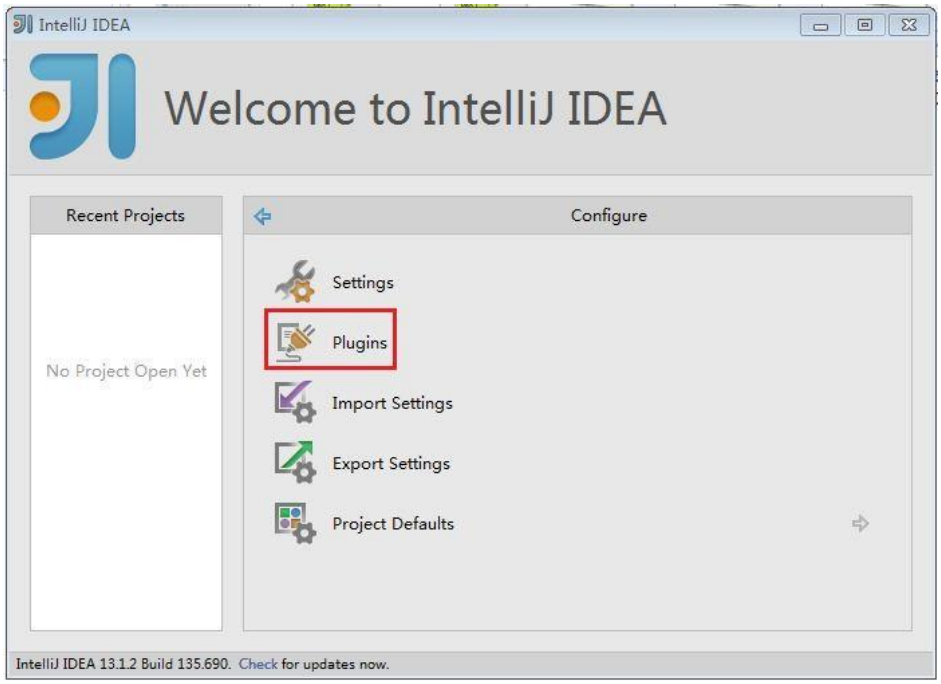


4. 将下载的 scala 安装包点击安装，安装 Scala 工具。
5. 安装 Scala 插件。
   a. 打开 IntelliJ IDEA，如图 1 所示，选择“Configure”。

图1-19 configure

b. 如图 1-20 所示界面中，选择“Plugins”。

图1-20 Plugins

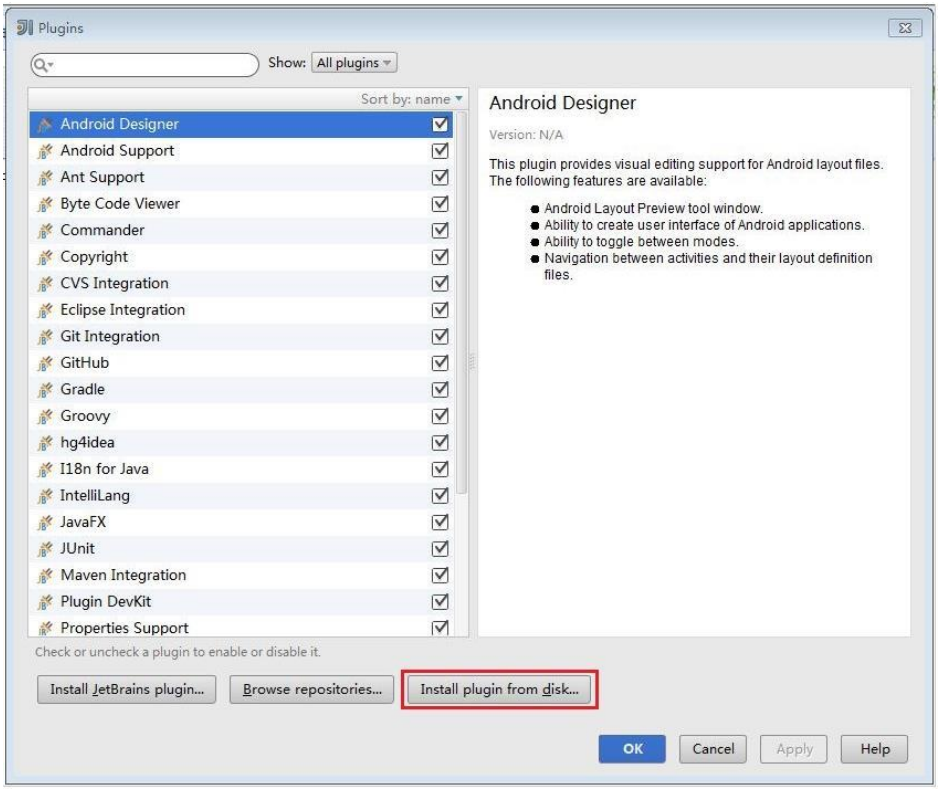


添加 scala 插件的方法，一种是点击“inatall jetBrains plugin”然后在框中搜索 scala，在右边框中点击进行添加。这样可以保证 IDEA 版本与 scala 插件兼容，建议使用这种方法。

另一种是下载可用在 IntelliJ IDEA 上的 scala 插件到本地电脑。

如图 1-21 所示界面中，选择“Install plugin from disk”，然后选择磁盘上的插件包。

图1-21 Install plugin from disk



步骤 3 下载 Spark 客户端程序到客户端节点中。

1. 登录 FusionInsight Manager 系统。

   在浏览器地址栏中输入访问地址，地址格式为“http://*FusionInsight Manager 系统的 WebService 浮动 IP 地址*:8080/web”。

   例如，在 IE 浏览器地址栏中，输入“http://10.10.10.172:8080/web”。

2. 单击“Services > Spark > Download Client”，“Client type”勾选“All Client Files”，单击“OK”开始下载客户端，等待下载完成。

步骤 4 解压缩客户端压缩文件包“FusionInsight_V100R002C50SPC200_Spark_Client.tar”。文件为 tar 格式，可采用如下方式解压。

说明

客户端程序文件包为 tar 格式，可采用如下方式解压。

- Windows：下载 7zip 工具，双击压缩包进行解压。
- Linux：执行 **tar -xf** *[客户端文件包]*命令解压。

步骤 5 导入 Scala 代码样例工程到 IDEA 开发环境。

1. 在 windows 上的客户端安装包中存在 Spark 的样例工程“Spark\sampleCode\SparkScalaExample”。

   说明
   由于样例工程在 windows 中搭建，所以需要把该工程拷贝到 windows 系统中，方便对样例工程进行操作。

2. 在“Spark/sampleCode/SparkScalaExample”目录下新建一个 lib 文件夹，将“Spark\dev_lib”下的所有 jar 包拷贝到新建的 lib 文件夹下。此工程依赖两个 Yarn 组件的 jar 包，需要下载并解压 Yarn 的客户端，继续将 FusionInsight-Hadoop-2.7.1.tar.gz 解压，分别从解压目录获取两个 jar 包：“FusionInsight-Hadoop-2.7.1\hadoop\share\hadoop\mapreduce”下获取“hadoop-mapreduce-client-common-*.jar”和“FusionInsight-Hadoop-2.7.1\adapter\authorization\controller\lib”下获取“hadoop-common-*.jar”。

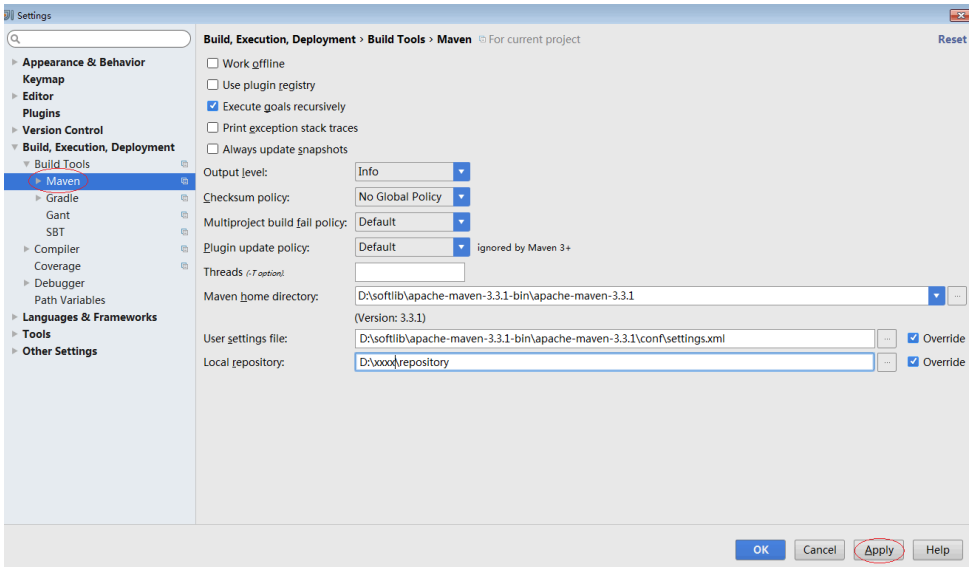
3. 为工程设置 MAVEN。

   单击“File > Settings...”，打开设置窗口，如下图所示。

   在搜索框中输入 MAVEN，搜索 MAVEN 设置项（若未安装 MAVEN，需要先安装），然后根据实际情况配置 **MAVEN home directory**，**User settings file**，**Local repository** 等，完成 MAVEN 绑定。

   设置后单击“APPLY”保存修改，最后单击“OK”。

   图1-22 设置 MAVEN

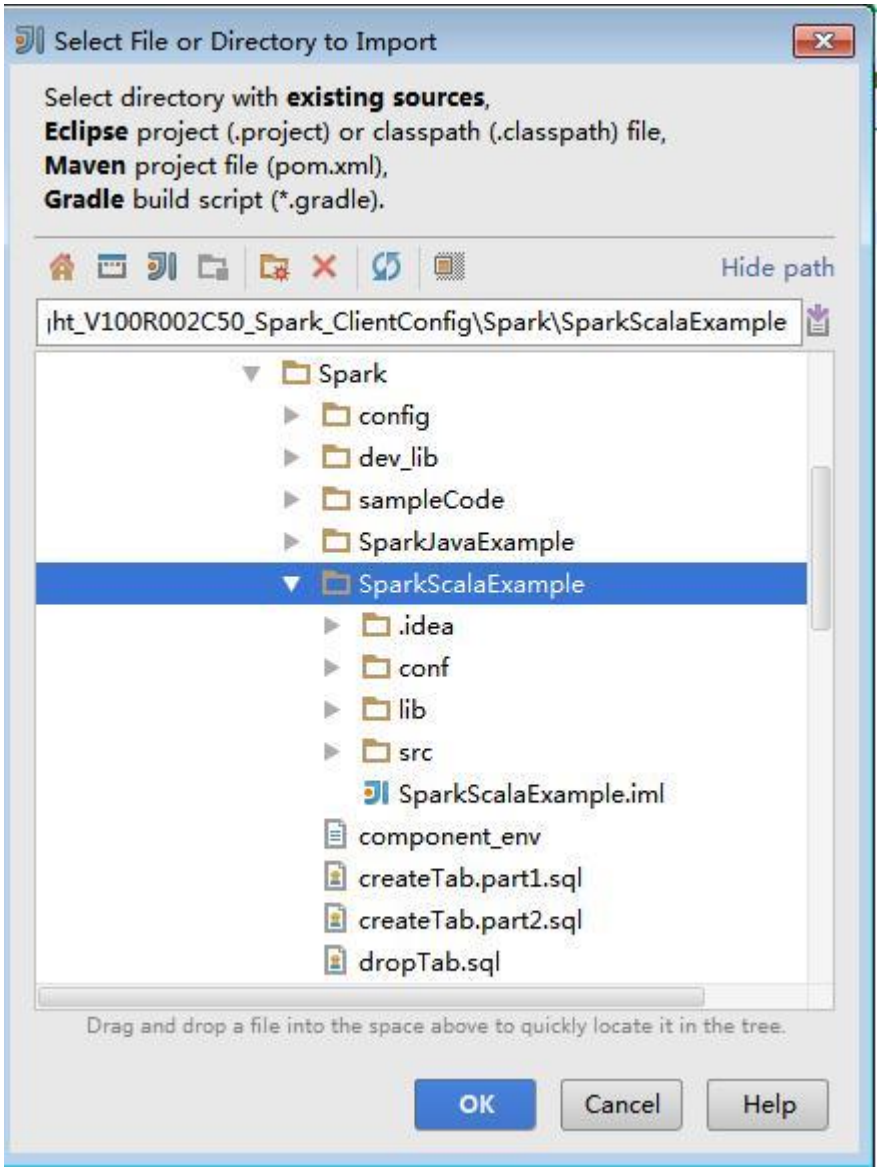
   

4. 导入工程。

   单击“File > New > Project from Existing Sources...”，然后选择下载的样例工程 SparkScalaExample 文件夹，单击“OK”。

   在弹出窗口的“Import project from external model”项中选择“Maven”，接下来继续选择“Next”，给工程文件命名，点击“Finsh”完成工程的导入。

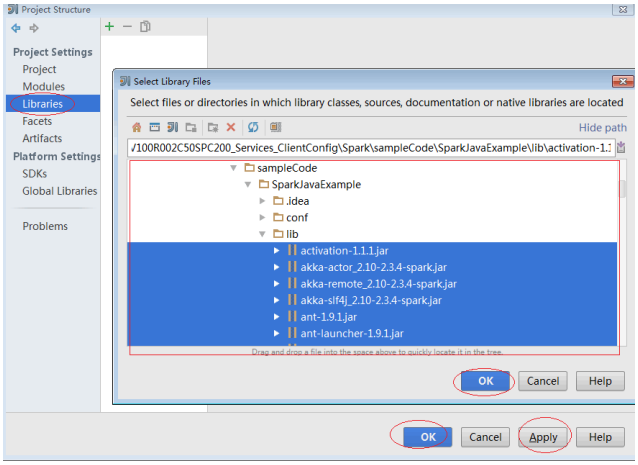
图1-23 Scala 工程导入



5. 加入 lib 包。

   导入工程后，如果代码报红，此时还需要加入 lib 包，单击“File > Project Structure > Project Settings > Libraries”，点击“+”，选择“Java”，然后在 SparkScalaExample 文件夹，选中 lib 目录下的所有 jar 包，点击“OK”。

图1-24 加入 lib 包



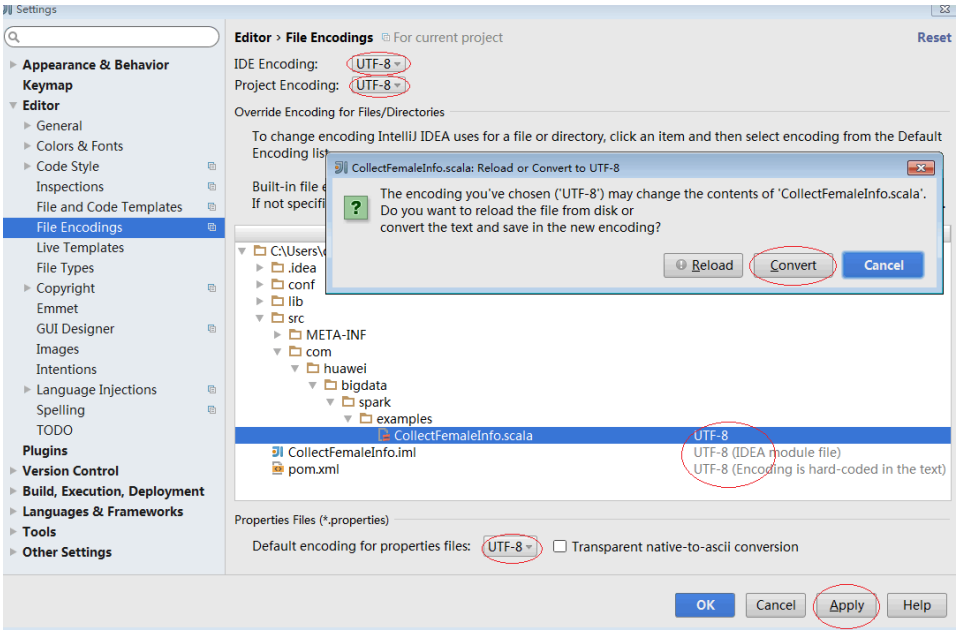
然后继续单击“Project Settings > Modules > Dependecies”，点击右边的“+”选择“Library... > Java”添加前面步骤中新建的 lib 下的 jar 包（如果 jar 包已有，就不需要加）以及 scala 相关的包和 scala 的 sdk 里的包。

6. 设置 IDEA 的文本文件编码格式。

在 IDEA 的菜单栏中，单击“File > Settings...”弹出设置窗口。

在左边导航上搜索“File Encodings”，然后将“IDE Encoding”、“Project Encoding”均设置为“UTF-8”，然后单击“Apply”，在弹出的面框中选择“Convert”，单击“OK”。

图1-25 设置编码方式



----结束

1.2.3 Python 开发环境准备

操作场景

本开发指南提供了 Spark 组件的样例代码和常用接口，便于开发者快速熟悉 Spark 并行计算框架。为了运行 Spark 组件的样例代码，你需要完成下面的操作。

开发环境可以搭建在 Windows 环境下，而运行环境（即客户端）只能部署在 Linux 环境下。

操作步骤

步骤 1 对于 Python 开发环境，直接使用 Notepad++编辑器（或其他编写 Python 应用程序的 IDE）即可。

步骤 2 下载 Spark 客户端程序到客户端机器中。

1. 登录 FusionInsight Manager 系统。

   在浏览器地址栏中输入访问地址，地址格式为“http://*FusionInsight Manager 系统的 WebService 浮动 IP 地址*:8080/web”。

   例如，在 IE 浏览器地址栏中，输入“http://10.10.10.172:8080/web”。

2. 单击“Services > Spark > Download Client”，“Client type”勾选“All Client Files”，单击“OK”开始下载客户端，等待下载完成。

步骤 3 解压缩客户端压缩文件包“FusionInsight_V100R002C50_Spark_Client.tar”。文件为 tar 格式，可采用如下方式解压。

说明

客户端程序文件包为 tar 格式，可采用如下方式解压。

- Windows: 下载 7zip 工具，双击压缩包进行解压。
- Linux: 执行 **tar xf** *[客户端文件包]* 命令解压

步骤 4 Python 样例工程在解压后的客户端文件夹内的“Spark\sampleCode\SparkPythonExample”目录下。打开 Python 代码样例工程中的 Python 文件（*.py），根据业务场景开始编写应用程序。

步骤 5 运行环境安装 Python，要求 2.6 或以上版本。

----结束

1.2.4 运行环境准备

操作场景

Spark 的运行环境（即客户端）只能部署在 Linux 环境下。您可以执行如下操作完成运行环境准备。

操作步骤

步骤 1 安装 Spark 客户端

说明
客户端支持 Linux 安装。

1. 确认服务端 Spark 组件已经安装，并正常运行。
2. 客户端运行环境已安装 1.7 或 1.8 版本的 JDK。
3. 获取并解压缩安装包“FusionInsight_V100R002C50_Spark_Client.tar”。执行如下命令解压：

   tar -zxvf FusionInsight_V100R002C50_Spark_Client.tar

   tar -xvf FusionInsight_V100R002C50_Spark_ClientConfig.tar
4. 进入解压文件夹，执行**./install.sh** *#{client_install_home}* 安装客户端。

步骤 2 配置客户端网络连接。

说明
当客户端所在主机不是集群中的节点时，配置客户端网络连接，可避免执行客户端命令时出现错误。

1. 确认客户端与服务端各个主机网络上互通。
2. 将客户端主机名与 IP 映射关系添加到服务端的 hosts 文件中。

说明
本地 hosts 文件存放路径举例：

- linux：“/etc/hosts”。

步骤 3 确认客户端和集群时间一致性。

客户端机器的时间与 Spark 集群的时间要保持一致，时间差要小于 5 分钟。

----结束

1.3 开发指引

应用程序实例

假定用户有某个周末网民网购停留时间的日志文本，基于某些业务要求，要求开发 Spark 应用程序实现如下功能：

- 统计日志文件中本周末网购停留总时间超过 2 个小时的女性网民信息。
- 周末两天的日志文件第一列为姓名，第二列为性别，第三列为本次停留时间，单位为分钟，分隔符为“,”。

log1.txt：周六网民停留日志

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

```
CaiXuyu,female,50
FangBo,female,60
```

log2.txt：周日网民停留日志

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

数据准备

首先需要把原日志文件放置在 HDFS 系统里。

步骤 1 本地新建两个文本文件，将 log1.txt 中的内容复制保存到 input_data1.txt，将 log2.txt 中的内容复制保存到 input_data2.txt。

在复制内容到 linux 节点上的 input_data1.txt、input_data2.txt 文件时确保文本中没有空行，若有，去掉多余的空行。

步骤 2 在 HDFS 上建立一个文件夹，“/tmp/input”，并上传 input_data1.txt，input_data2.txt 到此目录，命令如下：

1. 在 linux 系统 HDFS 客户端使用命令“**hadoop fs -mkdir /tmp/input**（**hdfs dfs** 命令有同样的作用）”，创建对应目录。
2. 在 linux 系统 HDFS 客户端使用命令“**hadoop fs -put** *本地 input.txt 地址* **/tmp/input**”，将之前两个新建的文本上传。

----结束

代码样例

统计日志文件中本周末网购停留总时间超过 2 个小时的女性网民信息。

Spark 的应用程序可使用 Java 语言或者 Scala 语言来实现：代码样例请参考 1.4.1 Java 代码样例和 1.4.2 Scala 代码样例。

1.4 代码样例

1.4.1 Java 代码样例

功能介绍

统计日志文件中本周末网购停留总时间超过 2 个小时的女性网民信息。

主要分为四个部分：

- 读取原文件数据。
- 筛选女性网民上网时间数据信息。
- 汇总每个女性上网总时间。
- 筛选出停留时间大于两个小时的女性网民信息。

代码样例

下面代码片段仅为演示，具体代码参见 com.huawei.bigdata.spark.examples.CollectFemaleInfo 类：

```
   //创建一个配置类 SparkConf，然后创建一个 SparkContext
   SparkConf conf = new SparkConf().setAppName("CollectFemaleInfo");
   JavaSparkContext jsc = new JavaSparkContext(conf);

   //读取原文件数据,每一行记录转成 RDD 里面的一个元素
   JavaRDD<String> data = jsc.textFile(args[0]);

   //将每条记录的每列切割出来，生成一个 Tuple
   JavaRDD<Tuple3<String,String,Integer>> person = data.map(new
Function<String,Tuple3<String,String,Integer>>()
   {
      private static final long serialVersionUID = -2381522520231963249L;

      @Override
      public Tuple3<String, String, Integer> call(String s) throws Exception
      {
          //按逗号分割一行数据
          String[] tokens = s.split(",");

          //将分割后的三个元素组成一个三元 Tuple
          Tuple3<String, String, Integer> person = new Tuple3<String, String,
Integer>(tokens[0], tokens[1], Integer.parseInt(tokens[2]));
          return person;
      }
   });

   //使用 filter 函数筛选出女性网民上网时间数据信息
   JavaRDD<Tuple3<String,String,Integer>> female = person.filter(new
Function<Tuple3<String,String,Integer>, Boolean>()
   {
      private static final long serialVersionUID = -4210609503909770492L;
```

```
        @Override
        public Boolean call(Tuple3<String, String, Integer> person) throws Exception
        {
            //根据第二列性别，筛选出是 female 的记录
            Boolean isFemale = person._2().equals("female");
            return isFemale;
        }
    });

    //汇总每个女性上网总时间
    JavaPairRDD<String, Integer> females = female.mapToPair(new
PairFunction<Tuple3<String, String, Integer>, String, Integer>()
    {
        private static final long serialVersionUID = 8313245377656164868L;

        @Override
        public Tuple2<String, Integer> call(Tuple3<String, String, Integer> female)
throws Exception
        {
            //取出姓名和停留时间两列，用于后面按名字求逗留时间的总和
            Tuple2<String, Integer> femaleAndTime = new  Tuple2<String,
Integer>(female._1(), female._3());
            return femaleAndTime;
        }
    })
      .reduceByKey(new Function2<Integer, Integer, Integer>()
    {
        private static final long serialVersionUID = -3271456048413349559L;

        @Override
        public Integer call(Integer integer, Integer integer2) throws Exception
        {
            //将同一个女性的两次停留时间相加，求和
            return (integer + integer2);
        }
    });

    //筛选出停留时间大于两个小时的女性网民信息
    JavaPairRDD<String, Integer> rightFemales = females.filter(new
Function<Tuple2<String, Integer>, Boolean>()
    {
        private static final long serialVersionUID = -3178168214712105171L;

        @Override
        public Boolean call(Tuple2<String, Integer> s) throws Exception
        {
            //取出女性用户的总停留时间，并判断是否大于 2 小时
            if(s._2() > (2 * 60))
            {
                return true;
            }
            return false;
        }
    });
```

```
//对符合的 female 信息进行打印显示
for(Tuple2<String, Integer> d: rightFemales.collect())
{
   System.out.println(d._1() + "," + d._2());
}
```

1.4.2 Scala 代码样例

功能介绍

统计日志文件中本周末网购停留总时间超过 2 个小时的女性网民信息。

主要分为三个部分：

- 读取女性网民上网时间数据信息组成 Spark 的 RDD 数据，通过 Spark 中类 SparkContext 中 textFile 方法实现
- 从原文件中筛选女性网民上网时间数据信息，通过 Spark 中 RDD 中 filter 方法实现。
- 汇总每个女性上网时间，并输出时间大于两个小时的女性网民信息，通过 Spark 中 RDD 中 reduceByKey 方法类和 filter 实现。

代码样例

下面代码片段仅为演示，具体代码参见 com.huawei.bigdata.spark.examples.CollectFemaleInfo 类：

样例：类 CollectMapper

```
//配置 Spark 应用名称
val conf = new SparkConf().setAppName("CollectFemaleInfo")

//提交 Spark 作业
val sc = new SparkContext(conf)
//读取数据。其是传入参数 args(0)指定数据路径
val text = sc.textFile(args(0))
//筛选女性网民上网时间数据信息
val data = text.filter(_.contains("female"))
//汇总每个女性上网时间
val femaleData:RDD[(String,Int)] = data.map{line =>
   val t= line.split(',')
   (t(0),t(2).toInt)
}.reduceByKey(_ + _)
//筛选出时间大于两个小时的女性网民信息，并输出
val result = femaleData.filter(line => line._2 > 120)
result.foreach(println)
```

1.4.3 Python 代码样例

功能介绍

统计日志文件中本周末网购停留总时间超过 2 个小时的女性网民信息。

主要分为四个部分：

- 读取女性网民上网时间数据信息组成 Spark 的 RDD 数据，通过 Spark 中类 SparkContext 中 textFile 方法实现
- 从原文件中筛选女性网民上网时间数据信息，通过 Spark 中 RDD 中 filter 方法实现。
- 汇总每个女性上网时间，通过 Spark 中 RDD 中 reduceByKey 方法实现。
- 汇总并输出时间大于两个小时的女性网民信息，通过 Spark 中 filter 方法实现。

代码样例

下面代码片段仅为演示，具体代码参见 collectFemaleInfo.py：

```
def contains(str, substr):
  if substr in str:
    return True
  return False

if __name__ == "__main__":
  if len(sys.argv) < 2:
    print "Usage: CollectFemaleInfo <file>"
    exit(-1)

  # 创建 SparkContext，设置 AppName
  sc = SparkContext(appName = "CollectFemaleInfo")?

  """
  以下程序主要实现以下几步功能:
  1.读取数据。其是传入参数 argv[1]指定数据路径 - textFile
  2.筛选女性网民上网时间数据信息 - filter
  3.汇总每个女性上网时间 - map/map/reduceByKey
  4.筛选出时间大于两个小时的女性网民信息 - filter
  """
  inputPath = sys.argv[1]
  result = sc.textFile(name = inputPath, use_unicode = False) \
    .filter(lambda line: contains(line, "female")) \
    .map(lambda line: line.split(',')) \
    .map(lambda dataArr: (dataArr[0], int(dataArr[2]))) \
    .reduceByKey(lambda v1, v2: v1 + v2) \
    .filter(lambda tupleVal: tupleVal[1] > 120) \
    .collect()
  for (k, v) in result:
    print k + "," + str(v)

  # 停止 SparkContext
  sc.stop()
```

1.4.4 Thrift Server 服务客户端代码样例

功能介绍

用户自定义 Thrift Server 的客户端，使用 JDBC 连接来进行数据表的创建、数据加载、查询和删除。

代码样例

- 数据准备
  - 确保以 HA 模式启动了 Thrift Server 服务，并至少有一个实例对外服务，在主备 Thrift Server 的节点上分别创建“/home/data”文件，内容如下：

    ```
    Miranda,32
    Karlie,23
    Candice,27
    ```

  - 确保其对启动 Thrift Server 的用户有读写权限。
  - 确保 classpath 下有“hive-site.xml”文件，并在其中根据需要配置客户端连接所需要的参数。例如：

    ```
    <?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <configuration>
       <property>
          <name>spark.thriftserver.ha.enabled</name>
          <value>true</value>
       </property>
    </configuration>
    ```

  说明

  安全版本下，需要在启动进程前在 JVM 中添加以下参数：

  ```
  -Djava.security.auth.login.config=#{pathToJaasConf}/jaas.conf
  -Djava.security.krb5.conf=#{pathToKrb5Conf}/krb5.conf
  -Dzookeeper.server.principal=zookeeper.hadoop.hadoop.com
  ```

  其中#{pathToJaasConf}是 jaas.conf 存放的目录，#{pathToKrb5Conf}是 krb5.conf 的存放目录，zookeeper.hadoop.hadoop.com 是 zookeeper 服务端使用的 Kerberos 账号。jaas.conf 的文件内容示例如下：

  ```
  Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="#{pathToKeytab}/spark.keytab"
  storeKey=true
  principal="spark.principal"
  useTicketCache=false
  debug=true;
  };
  ```

  其中：#{pathToKeytab}为存放 keytab 的目录，您可以联系管理员获取相应账号对应权限的 keytab 文件及对应的 principal。

- 数据分析

1. 在 default 数据库下创建 child 表。
2. 把“/home/data”的数据加载进 child 表中。
3. 查询 child 表中的数据。
4. 删除 child 表。

- 使用自定义客户端的 JDBC 接口提交数据分析任务，并返回结果。

1. 定义 Zookeeper 的连接 url。

   ```
   val zkQuorum = "10.96.1.25:2181,10.96.1.26:2181,10.96.1.27:2181"
   ```

   说明

   zkQuorum 的值中 IP 地址和端口号，应根据具体场景写入。

2. 定义 sql 语句。sql 语句必须为单条语句，注意其中不能包含“;”。举例：

```
val sqlList = new ArrayBuffer[String]
sqlList += "CREATE TABLE CHILD (NAME STRING, AGE INT) " +
"ROW FORMAT DELIMITED FIELDS TERMINATED BY ','"
sqlList += "LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD"
sqlList += "SELECT * FROM child"
sqlList += "DROP TABLE child"
```

3. 拼接 JDBC url。

```
val HA_CLUSTER_URL = "ha-cluster"
val sb = new
StringBuilder(s"jdbc:hive2://$HA_CLUSTER_URL/default;user.principal=spark/hadoo
p.hadoop.com@HADOOP.COM;sasl.qop=auth-
conf;auth=KERBEROS;principal=spark/hadoop.hadoop.com@HADOOP.COM;zk.quorum=")
.append(zkQuorum)
val url = sb.toString()
```

HA 模式下 url 的 host 和 port 必须为“ha-cluster”。

4. 加载 Hive JDBC 驱动。

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

5. 获取 JDBC 连接，执行 HQL，输出查询的列名和结果到控制台，关闭 JDBC 连接。

连接字符串中的“zk.quorum”也可以使用配置文件中的配置项“spark.deploy.zookeeper.url”来代替。

还支持设置客户端与 Thrift Server 连接的超时时间，在网络拥塞的情况下，这个特性可以避免客户端由于无限等待服务端的返回而挂起。使用方式如下：

在执行“DriverManager.getConnection”方法获取 JDBC 连接前，添加“DriverManager.setLoginTimeout(n)”方法来设置超时时长，其中 n 表示等待服务返回的超时时长，单位为秒，类型为 Int，默认为“0”（表示永不超时）。

```
var connection: Connection = null
var statement: PreparedStatement = null
try {
  connection = DriverManager.getConnection(url, "root", "")
  for (sql <- sqls) {
    println(s"---- Begin executing sql: $sql ----")
    statement = connection.prepareStatement(sql)

    val result = statement.executeQuery()

    val resultMetaData = result.getMetaData
    val colNum = resultMetaData.getColumnCount
    for (i <- 1 to colNum) {
      print(resultMetaData.getColumnLabel(i) + "\t")
    }
    println()

    while (result.next()) {
      for (i <- 1 to colNum) {
        print(result.getString(i) + "\t")
      }
      println()
    }
    println(s"---- Done executing sql: $sql ----")
  }
```

```
} finally {
  if (null != statement) {
    statement.close()
  }

  if (null != connection) {
    connection.close()
  }
}
```

1.4.5 Spark Streaming 代码样例

功能介绍

实时统计连续网购时间超过半个小时的女性网民信息。

数据介绍

数据来源：Kafka 组件生产和存储数据，Spark Streaming 消费并处理数据

数据格式：文本格式，属性与 1.3 开发指引相同

实现步骤

1. 接收 Kafka 中数据，生成相应 DStream，通过 KafkaUtils.createStream 方法实现。
2. 获取每一个行的字段属性，通过 DStream 中的 map 方法实现。
3. 筛选女性网民上网时间数据信息，通过 DStream 中 filter 方法实现。
4. 汇总在一个时间窗口内每个女性上网时间，通过 DStream 中 reduceByKeyAndWindow 方法实现。
5. 筛选连续上网时间超过阈值的用户，并获取结果，通过 DStream 中的 filter 筛选，通过 DStream 中的 print 打印。

代码样例

```
// 参数解析:
// 第一个为 Streaming 分批的处理间隔，第二个为统计数据的时间跨度,时间单位都是秒
// 第三个为 Kafka 服务器的 Zookeeper 地址，第四个为 Kafka 中的消费组，
// 第五个为 Kafka 中订阅的主题，多个以逗号分隔，第六个为每个主题对应处理线程个数
val Array(batchTime, windowTime,
    zkQuorum, group,
    topics, numThreads) = args
val batchDuration = Seconds(batchTime.toInt)
val windowDuration = Seconds(windowTime.toInt)

// 建立 Streaming 启动环境
val sparkConf = new SparkConf()
sparkConf.setAppName("DataSightStreamingExample")
val ssc = new StreamingContext(sparkConf, batchDuration)

// 设置 Streaming 的 CheckPoint 目录，由于窗口概念存在，该参数必须设置
ssc.checkpoint("checkpoint")
```

```
// 组装 Kafka 的主题列表
val topicMap = topics.split(",").map((_,numThreads.toInt)).toMap

// 1.接收 Kafka 中数据，生成相应 DStream
val lines = KafkaUtils.createStream(ssc, zkQuorum,
  group, topicMap).map(_._2)

// 2.获取每一个行的字段属性
val records = lines.map(getRecord)

// 3.筛选女性网民上网时间数据信息
val femaleRecords = records.filter(_._2 == "female")
  .map(x=>(x._1, x._3))

// 4.汇总在一个时间窗口内每个女性上网时间
val aggregateRecords = femaleRecords
  .reduceByKeyAndWindow(_ + _, _ - _, windowDuration)

// 5.筛选连续上网时间超过阈值的用户，并获取结果
aggregateRecords.filter(_._2 > 0.9 * windowTime.toInt).print()

// Streaming 系统启动
ssc.start()
ssc.awaitTermination()
```

上述代码会引用以下函数

```
// 获取字段函数
def getRecord(line: String): (String, String, Int) = {
  val elems = line.split(",")
  val name = elems(0)
  val sexy = elems(1)
  val time = elems(2).toInt
  (name, sexy, time)
}
```

运行步骤

1. 确保集群安装完成，包括 HDFS、Yarn、Spark 和 Kafka。
2. 启动 Kafka 的 Producer，向 Kafka 发送数据

   ```
   java -cp {ClassPath} com.huawei.bigdata.spark.examples.StreamingExampleProducer
   {BrokerList} {Topic}
   ```

3. 启动 SparkStreaming，处理数据

   ```
   bin/spark-submit --class com.huawei.bigdata.spark.examples.StreamingExample --
   master yarn-client {Path of SparkStreamingExample-1.0.jar} {BatchTime} {Window
   Time} {ZkUrl} {ComsumerGroup} {Topics} {ThreadNum}
   ```

   其中入参介绍，见代码样例中参数解析。

1.4.6 Spark SQL 代码样例

功能介绍

统计日志文件中本周末网购停留总时间超过 2 个小时的女性网民信息。

主要分为四个部分：

- 创建表，将日志文件数据导入到表中。
- 筛选女性网民，提取上网时间数据信息。
- 汇总每个女性上网总时间。
- 筛选出停留时间大于两个小时的女性网民信息。

代码样例

下面代码片段仅为演示，具体代码参见 collectFemaleInfo.sql：

```
#创建空表，定义表数据 schema
create table log_data (name STRING, gender STRING, stayTime INT) row format
delimited fields terminated by ',';

#将日志文件数据导入到表中，实际运行时请替换 path/to/logdata 为数据存放路径
load data local inpath 'path/to/logdata' into table log_data;

#筛选出女性网民数据，提取上网时间信息
select name,stayTime
  from log_data
  where gender = 'female';

#汇总每个女性上网的总时间
select name,sum(stayTime) as totalStayTime
  from
    (select name,stayTime
       from log_data
       where gender = 'female') tmp
  group by name;

#筛选出停留时间大于两个小时的女性网民信息
select *
  from
  (select name,sum(stayTime) as totalStayTime
    from
      (select name,stayTime
        from log_data
        where gender = 'female') tmp
    group by name) tmp2
  where totalStayTime > 120;
```

上面演示的是直接通过 SQL 语句来实现数据处理逻辑，也可以通过 Scala/Java/Python 代码中调用 SparkSQL 的接口来实现，参见如下链接：
http://spark.apache.org/docs/latest/sql-programming-guide.html#running-sql-queries-programmatically

1.5 运行应用

操作场景

在代码完成开发后，您可以上传至 Linux 客户端环境中运行应用。使用 Java、Python 或 Scala 开发的应用程序在 Spark 客户端的运行步骤是一样的。

说明
Spark 组件的应用程序不支持在 Windows 环境下运行。

操作步骤

使用 JAVA 工程运行样例

步骤 1 导出运行 jar 包。

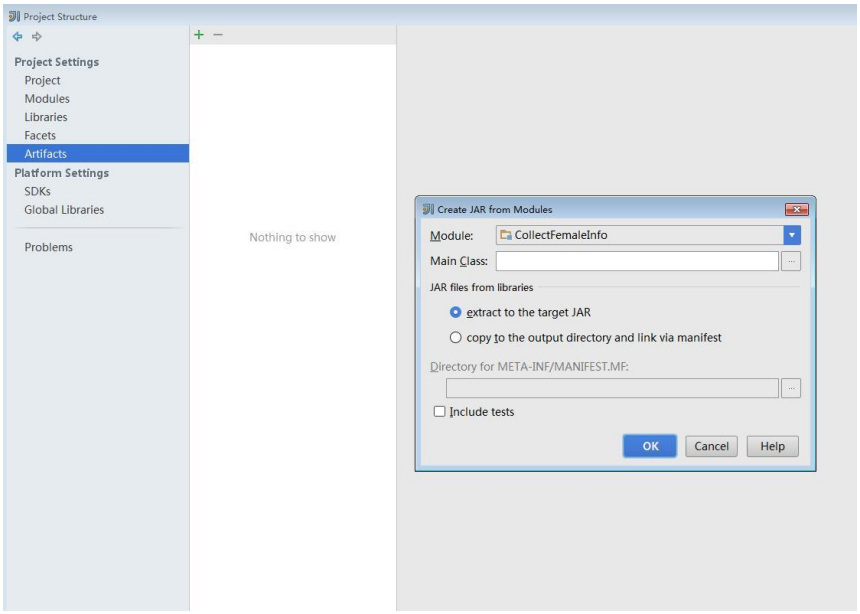
1. （可选）如果使用 keytab 和 principle 进行安全认证，在添加代码之前，需要先导入安全认证要用到的 jar 包。由于 Spark 客户端压缩包中没有安全认证相关的包，所以此处的 jar 包来自 Yarn 客户端安装包中 hadoop-examples 下的 lib 目录，然后在代码中添加用到的类。如下：

```
import org.apache.hadoop.security.UserGroupInformation;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
```

在代码中 main 函数开始处添加如下方法，principlal 名称，keytab 路径和 krb 文件路径等参数通过 main 函数参数传入。

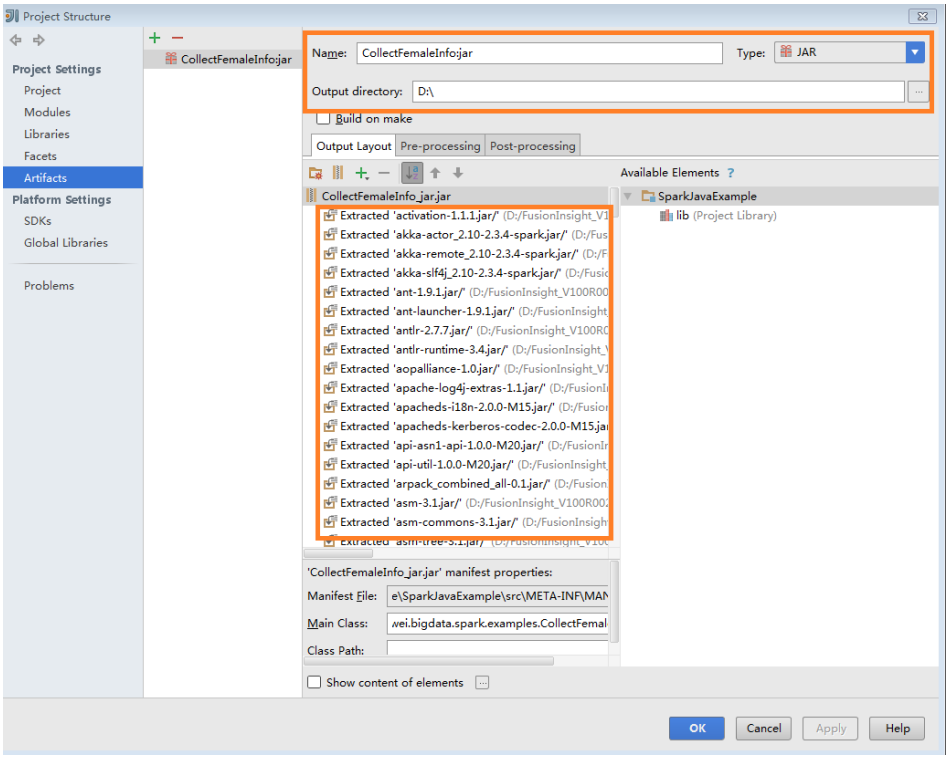
```
String principal ="spark/hadoop.hadoop.com@HADOOP.COM";
String keytab = "/opt/huawei/Bigdata/etc/1_11_SparkResource/spark.keyt
String krbfilepath = "/opt/huawei/Bigdata/etc/1_5_KerberosClient/krb5.conf";
Configuration conf_sec = new Configuration();
conf_sec.set(principal, "spark/hadoop.hadoop.com@HADOOP.COM");
// keytab file
conf_sec.set(keytab, "/opt/huawei/Bigdata/etc/1_11_SparkResource/spark.keytab");
System.setProperty("java.security.krb5.conf", krbfilepath);
UserGroupInformation.setConfiguration(conf_sec);
UserGroupInformation.loginUserFromKeytab(principal, keytab);
```

2. 将实例代码正常导入 Intellij IDEA 后，单击“File > Project Structure > Atifacts”，点击“+”号选择“JAR > From modules with...”，如下图所示，然后点击“OK”。



3. 进入下图所示的界面，先选中 Extracted"XXX.jar，点击“—”号防止导出的包内有多余的 jar 包，留下“<工程名> compile out”目录（此目录一般在列表的最下面）即可。填写包名和导出路径后点击“OK”。

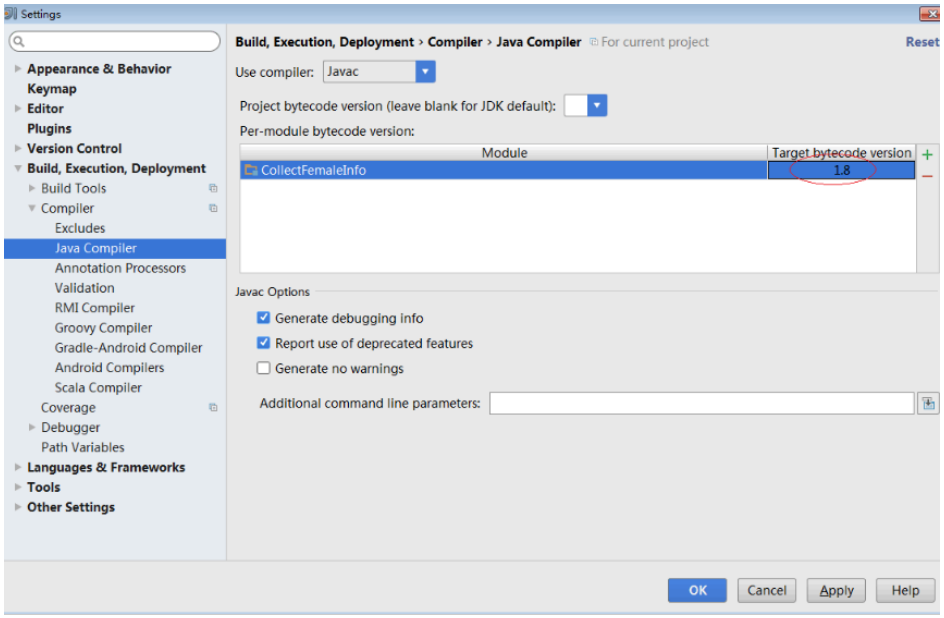
   如本例将 jar 包命名为 CollectFemaleInfo.jar，导出路径为 D 盘根目录。



4. 点击“Build > Build Artifacts”进行导包操作，在 D 盘即可找到导出的 jar 包。

   若报 log4j 的错误，删掉 log4j 的 jar 包即可。

   注：如果在编译打包过程中出现“需要目标发行版 1.6”之类的报错，则可单击“File > Settings...”打开设置窗口，在“Java Compiler”配置页面将“Target bytecode versio”修改为较新版本，然后依次单击“Apply”和“OK”。



步骤 2 运行 jar 包。

说明

在 Spark 任务运行过程中禁止重启 HDFS 服务，否则可能会导致任务失败。

1. 将步骤 1 中生成的 jar 包（如 CollectFemaleInfo.jar）拷贝到客户端节点的某个目录下，如“/opt/CollectFemaleInfo.jar”。

   请确保此 jar 包有可执行权限，即权限为 7XX。

2. 安全模式下，运行应用之前需执行 **source bigdata_env** 导入环境变量，在运行 jar 包之前需要执行 kinit 命令进行安全认证。（如果已执行 1.a，可以跳过此步骤。）

   例如：***kinit*** *spark_user*

   请联系管理员通过 FusionInsight Manager 添加安全认证的账号。

3. 进入客户端节点客户端的“Spark/spark/bin”目录，调用 Spark 的脚本 spark-submit 运行代码，命令如下：

   ./spark-submit --class com.huawei.bigdata.spark.examples.CollectFemaleInfo --master yarn-client 导出 jar 包的路径及名称 <inputpath>

   其中，<inputPath>指 HDFS 文件系统中 input 的路径。样例可参见 1.3 开发指引中的步骤 2。

   例如本样例中，执行：

   ./spark-submit --class com.huawei.bigdata.spark.examples.CollectFemaleInfo --master yarn-client ***/opt/CollectFemaleInfo.jar /tmp/input***

   该命令行包含了设置 job 参数以及提交 job 的操作。

说明

1. Spark Streaming 代码样例，具体运行方法见运行步骤。

2. Thrift Server 服务客户端代码样例，运行方法可用 java 命令，如：

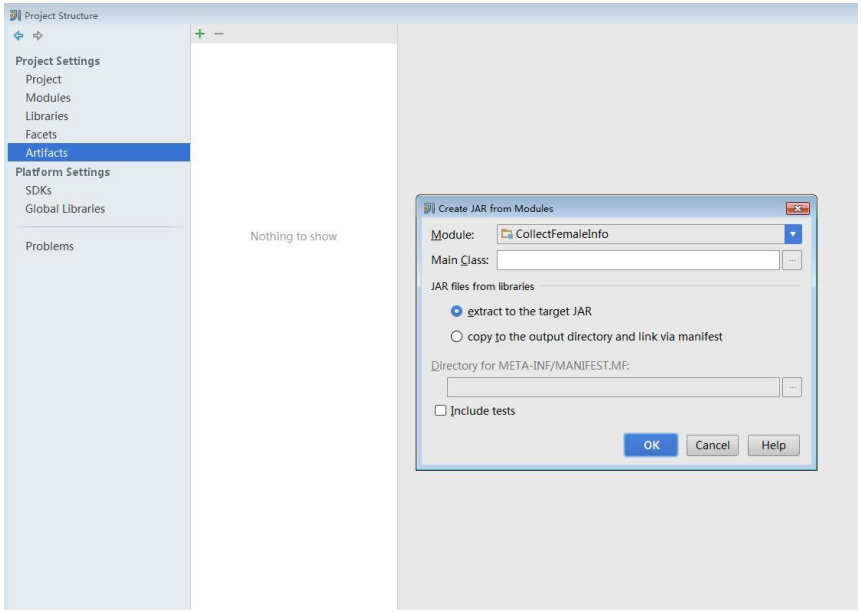
```
java -cp <SPARK_HOME>/lib/:/opt/female/SparkThriftServerExample-1.0.jar
com.huawei.bigdata.spark.examples.SparkThriftServerExample
```

----结束

使用 SCALA 工程运行应用

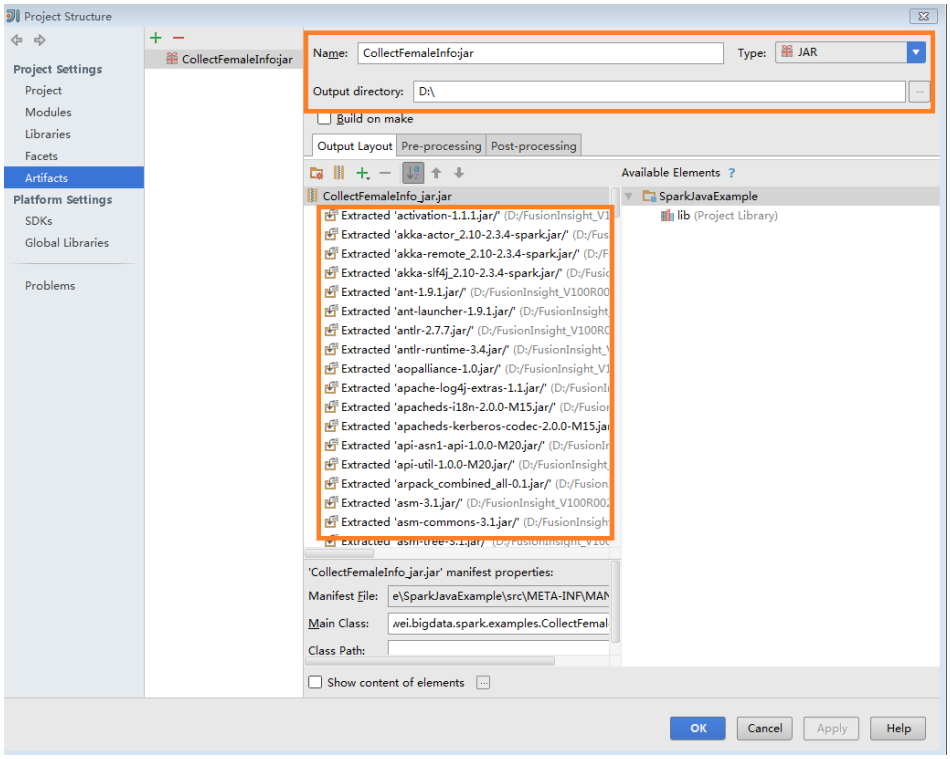
步骤 1 导出运行 jar 包。

1. 将实例代码正常导入 Intellij IDEA 后，单击“File > Project Structure > Atifacts”，点击“+”号选择“JAR > From modules with...”，如下图所示，然后单击“OK”。



2. 进入下图所示的界面，先选中 Extracted"XXX.jar"点击“—”号防止导出包时导出的包内有多余的 jar 包，只保留“<工程名> compile out”目录（一般在列表的最下面）即可。填写包名和导出路径后点击“OK”。

   如本例将 jar 包命名为 CollectFemaleInfo.jar，导出路径为 D 盘根目录。



3. 点击“Build > Build Artifacts”进行导包操作。在 D 盘即可找到导出的 jar 包。

   若报 log4j 的错误，删掉 log4j 的 jar 包即可。

步骤 2 运行 jar 包。

1. 将步骤 1 中生成的 jar 包（如 CollectFemaleInfo.jar）拷贝到客户端节点的某个目录下，如“/opt/CollectFemaleInfo.jar”。

   请确保此 jar 包有可执行权限，即权限为 7XX。

2. 安全模式下，运行应用之前需执行 **source bigdata_env** 导入环境变量，在运行 jar 包之前需要执行 kinit 命令进行安全认证。

   例如：***kinit*** *spark_user*

   请联系管理员通过 FusionInsight Manager 添加安全认证的账号。

3. 进入客户端节点客户端的 Spark/spark/bin 目录，调用 Spark 的脚本 spark-submit 运行代码，命令如下：

   ./spark-submit --class com.huawei.bigdata.spark.examples.CollectFemaleInfo --master yarn-client 导出 jar 包的路径及名称 <inputpath>

   其中，<inputPath>指 HDFS 文件系统中 input 的路径。样例可参见 1.3 开发指引中的步骤 2。

   例如本样例中，执行：

   ./spark-submit --class com.huawei.bigdata.spark.examples.CollectFemaleInfo --master yarn-client *./opt/CollectFemaleInfo.jar* *./tmp/input*

----结束

Python 执行应用

在样例的“Spark\sampleCode\SparkPythonExample”路径下将 collectFemaleInfo.py 文件拷贝到 linux 节点中。

比如目录为“/opt/collectFemaleInfo.py”，进入客户端节点客户端的“Spark/spark/bin”目录，执行如下命令：

./spark-submit --master yarn-client *./opt/collectFemaleInfo.py* *<inputpath>*

此处的 inputpath 为“/tmp/input”。

1.6 对外接口

1.6.1 Java API

Spark 完整的类及方法参考官方网站的描述：
https://spark.apache.org/docs/latest/api/java/index.html

常用接口

Spark 主要使用到如下这几个类：

- JavaSparkContext： 是 Spark 的对外接口，负责向调用该类的 java 应用提供 Spark 的各种功能，如连接 Spark 集群，创建 RDD，累积量和广播量等。它的作用是一个容器。
- SparkConf：Spark 应用配置类，如设置应用名称，执行模式，executor 内存等。

  JavaRDD ：用于在 java 应用中定义 JavaRDD 的类，功能类似于 scala 中的 RDD(Resilient Distributed Dataset)类。
- JavaPairRDD：表示 key-value 形式的 JavaRDD。该类提供的方法有 groupByKey，reduceByKey 等。
- Broadcast： 广播变量类。广播变量允许保留一个只读的变量，缓存在每一台机器上，而非每个任务保存一份拷贝。
- StorageLevel： 数据存储级别，有内存（MEMORY_ONLY），磁盘（DISK_ONLY），内存+磁盘（MEMORY_AND_DISK）等。

JavaRDD 支持两种类型的操作： transformation 和 action，这两种类型的常用方法如表 1-1 和表 1-2。

表1-1 Transformation

方法	说明
<R> JavaRDD<R> map(Function<T,R> f)	对 RDD 数据集中的每个 element 使用 Function。
JavaRDD<T> filter(Function<T,Boolean> f)	对 RDD 中所有元素调用 Function，返回为 true 的元素。
<U> JavaRDD<U> flatMap(FlatMapFunction<T,U> f)	先对 RDD 所有元素调用 Function，然后将结果扁平化。
JavaRDD<T> sample(boolean withReplacement, double fraction, long seed)	抽样。
JavaRDD<T> distinct(int numPartitions)	去除重复元素。
JavaPairRDD<K,Iterable<V>> groupByKey(int numPartitions)	返回(K,Seq[V])，将 key 相同的 value 组成一个集合。
JavaPairRDD<K,V> reduceByKey(Function2<V,V,V> func, int numPartitions	对 key 相同的 value 调用 func。
JavaPairRDD<K,V> sortByKey(boolean ascending, int numPartitions)	按照 key 来进行排序，是升序还是降序，ascending 是 boolean 类型。
JavaPairRDD<K,scala.T	当有两个 KV 的 dataset(K,V)和(K,W)，返回的是(K,(V,W))

方法	说明
uple2<V,W>> join(JavaPairRDD<K,W > other)	的 dataset,numTasks 为并发的任务数。
JavaPairRDD<K,scala.T uple2<Iterable<V>,Itera ble<W>>> cogroup(JavaPairRDD< K,W> other, int numPartitions)	当有两个 KV 的 dataset(K,V)和(K,W)，返回的是 <K,scala.Tuple2<Iterable<V>,Iterable<W>>>的 dataset,numTasks 为并发的任务数。
JavaPairRDD<T,U> cartesian(JavaRDDLike< U,?> other)	返回该 RDD 与其它 RDD 的笛卡尔积。

表1-2 Action

方法	说明
T reduce(Function2<T,T,T > f)	对 RDD 中的元素调用 Function2。
java.util.List<T> collect()	返回包含 RDD 中所有元素的一个数组。
long count()	返回的是 dataset 中的 element 的个数。
T first()	返回的是 dataset 中的第一个元素。
java.util.List<T> take(int num)	返回前 n 个 elements。
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	对 dataset 随机抽样，返回有 num 个元素组成的数组。 withReplacement 表示是否使用 replacement。
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io.co mpress.CompressionCod ec> codec)	把 dataset 写到一个 text file 中，或者 hdfs，或者 hdfs 支持的文件系统中，spark 把每条记录都转换为一行记录，然后写到 file 中。
java.util.Map<K,Object> countByKey()	对 RDD 中每个元素出现的次数进行统计。
void foreach(VoidFunction<T > f)	在数据集的每一个元素上，运行函数 func。
java.util.Map<T,Long> countByValue()	对 RDD 中每个元素出现的次数进行统计。

1.6.2 Scala API

Spark 完整的类及方法参考官方网站的描述：
https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package。

常用接口

Spark 主要使用到如下这几个类：

- SparkContext：是 Spark 的对外接口，负责向调用该类的 scala 应用提供 Spark 的各种功能，如连接 Spark 集群，创建 RDD 等。
- SparkConf：Spark 应用配置类，如设置应用名称，执行模式，executor 内存等。
- RDD（Resilient Distributed Dataset）：用于在 Spark 应用程序中定义 RDD 的类，该类提供数据集的操作方法，如 map，filter。
- PairRDDFunctions：为 key-value 对的 RDD 数据提供运算操作，如 groupByKey。
- Broadcast： 广播变量类。广播变量允许保留一个只读的变量，缓存在每一台机器上，而非每个任务保存一份拷贝。
- StorageLevel： 数据存储级别，有内存（MEMORY_ONLY），磁盘（DISK_ONLY），内存+磁盘（MEMORY_AND_DISK）等。

RDD 上支持两种类型的操作: ：transformation 和 action，这两种类型的常用方法如表 1-3 和表 1-4。

表1-3 Transformation

方法	说明
map[U](f: (T) => U): RDD[U]	对调用 map 的 RDD 数据集中的每个 element 都使用 f 方法，生成新的 RDD。
filter(f: (T) => Boolean): RDD[T]	对 RDD 中所有元素调用 f 方法，生成将满足条件数据集以 RDD 形式返回。
flatMap[U](f: (T) => TraversableOnce[U])(implicit arg0: ClassTag[U]): RDD[U]	先对 RDD 所有元素调用 f 方法，然后将结果扁平化，生成新的 RDD。
sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong): RDD[T]	抽样，返回 RDD 一个子集。
union(other: RDD[T]): RDD[T]	返回一个新的 RDD，包含源 RDD 和给定 RDD 的元素的集合。
distinct([numPartitions: Int]): RDD[T]	去除重复元素，生成新的 RDD。

方法	说明
groupByKey(): RDD[(K, Iterable[V])]	返回(K,Iterable[V])，将 key 相同的 value 组成一个集合。
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	对 key 相同的 value 调用 func。
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	按照 key 来进行排序，是升序还是降序，ascending 是 boolean 类型。
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	当有两个 KV 的 dataset(K,V)和(K,W)，返回的是(K,(V,W)) 的 dataset,numPartitions 为并发的任务数。
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	将当有两个 key-value 对的 dataset(K,V)和(K,W)，返回的是 (K, (Iterable[V], Iterable[W]))的 dataset,numPartitions 为并发的任务数。
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	返回该 RDD 与其它 RDD 的笛卡尔积。

表1-4 Action

API	说明
reduce(f: (T, T) => T): T	对 RDD 中的元素调用 f。
collect(): Array[T]	返回包含 RDD 中所有元素的一个数组。
count(): Long	返回的是 dataset 中的 element 的个数。
first(): T	返回的是 dataset 中的第一个元素。
take(num: Int): Array[T]	返回前 n 个 elements。
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]	takeSample(withReplacement，num，seed)对 dataset 随机抽样，返回有 num 个元素组成的数组。withReplacement 表示是否使用 replacement。
saveAsTextFile(path: String): Unit	把 dataset 写到一个 text file 中，或者 hdfs，或者 hdfs 支持的文件系统中，spark 把每条记录都转换为一行记录，然后写到 file 中。
saveAsSequenceFile(pat	只能用在 key-value 对上，然后生成 SequenceFile 写到本地

API	说明
h: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	或者 hadoop 文件系统。
countByKey(): Map[K, Long]	对每个 key 出现的次数做统计。
foreach(func: (T) => Unit): Unit	在数据集的每一个元素上，运行函数 func。
countByValue()(implicit ord: Ordering[T] = null): Map[T, Long]	对 RDD 中每个元素出现的次数进行统计。

1.6.3 Python API

如果您需要使用 Python 语言的客户端运行 Spark 实例，您可以使用 Spark 提供的 Python API。请直接参考官方网站上的详细描述了解其使用：https://spark.apache.org/docs/latest/api/python/index.html。