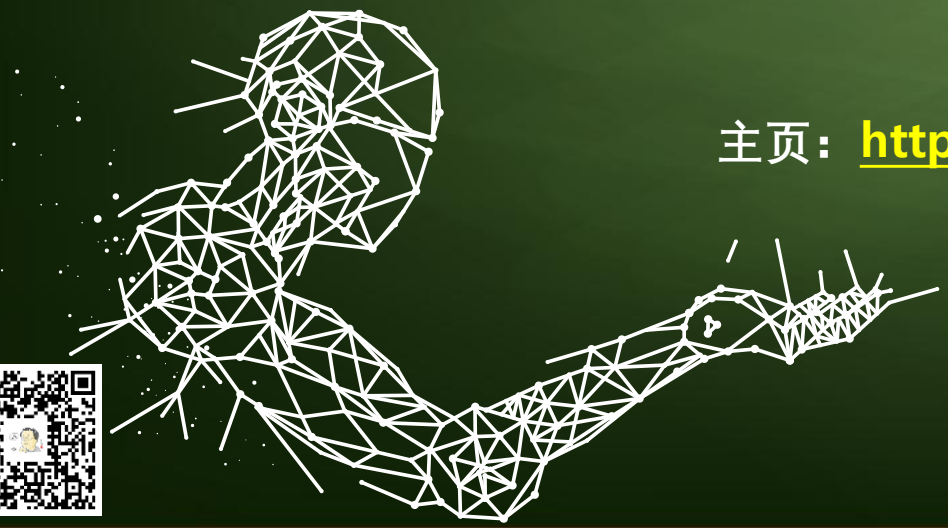
04

第4章 Spark环境搭建和使用方法 (Spark Install)

授课教师：[李阳源]

ywlee1212@gmail.com

主页：https://sites.google.com/view/dalguri



SL-01:提纲

4.1 安装Spark（Local模式）
4.2 在pyspark中运行代码
4.3 使用spark-submit命令提交运行程序
4.4 Spark集群环境搭建（Standalone模式）
4.5 在集群上运行Spark应用程序
4.6 Spark on YARN模式
4.7 安装PySpark类库
4.8 开发Spark独立应用程序
4.9 安装和使用PyCharm

4.1 安装Spark（Local模式）



4.1.1 基础环境

4.1.2 下载安装文件

4.1.3 配置相关文件

4.1.4 验证Spark是否安装成功

4.1.1 基础环境



Spark和Hadoop可以部署在一起，相互协作，
由Hadoop的HDFS、HBase等组件负责数据的存储和管理，
由Spark负责数据的计算。

另外，虽然Spark和Hadoop都可以安装在Windows系统中使用，
但是，我们建议在Linux系统中安装和使用。
本课采用如下环境配置。

- **Linux系统：Ubuntu 16.04。**
- **Hadoop：3.3.5版本。**
- **JDK：1.8版本。**
- **Spark：3.4.0版本。**









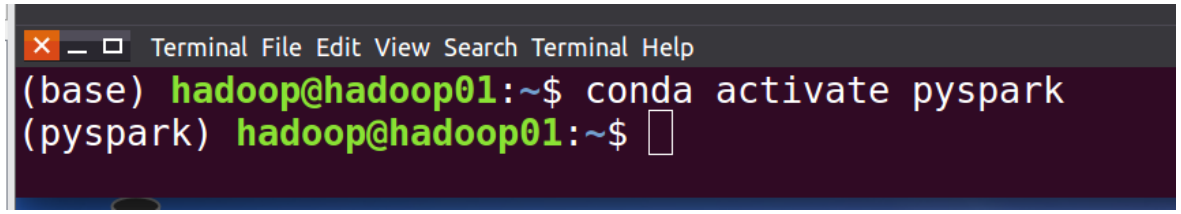
4.1.2 下载安装文件



访问Spark官网（https://archive.apache.org/dist/spark/spark-3.4.0/），
下载3.4.0版本的Spark安装文件spark-3.4.0-bin-without-hadoop.tgz

下载完安装文件以后，需要对文件进行解压。
按照Linux系统使用的默认规范，用户安装的软件一般都存放在“/usr/local/”目录下。
请使用hadoop用户登录Linux系统，
使用组合键【Ctrl+Alt+T】打开一个“终端”
（也就是一个Linux Shell环境，可以在终端窗口里面输入和执行各种Shell命令），
执行如下命令：

pyspark环境下安装Spark




```
$ sudo tar -zxf ~/Downloads/spark-3.4.0-bin-without-hadoop.tgz -C /usr/local/
$ cd /usr/local
$ sudo mv ./spark-3.4.0-bin-without-hadoop ./spark
$ sudo chown -R hadoop:hadoop ./spark # hadoop是当前登录Linux系统的用户名
```

4.1.3 配置相关文件



1．修改spark-env.sh文件

安装文件解压缩以后，还需要修改Spark的配置文件spark-env.sh。首先，可以复制一份由Spark安装文件自带的配置文件模板，命令如下：

```
$ cd   /usr/local/spark
$ cp   ./conf/spark-env.sh.template   ./conf/spark-env.sh
```

然后，使用vim编辑器打开spark-env.sh文件进行编辑，在该文件的第一行添加以下配置信息：

```
export   SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

4.1.3 配置相关文件



2．修改.bashrc文件

需要特别强调的是，.bashrc文件中必须包含以下环境变量的设置。

（1）SPARK_HOME：表示Spark安装路径在哪里。

（2）PYSPARK_PYTHON：表示Spark去哪里寻找Python执行器。

（3）JAVA_HOME：告知Spark Java安装在什么位置。

（4）HADOOP_CONF_DIR：告知Spark Hadoop的配置文件在哪里。

（5）HADOOP_HOME：告知Spark Hadoop安装在哪里。

此外，还可以为Spark配置PATH环境变量。

如果已经设置了上面的5个环境变量，则不需要重复添加设置；如果还没有设置，则需要使用vim编辑器打开.bashrc文件，命令如下：

```
$ vim ~/.bashrc
```

4.1.3 配置相关文件



把该文件修改为如下内容：

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_371
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export HADOOP_HOME=/usr/local/hadoop
export SPARK_HOME=/usr/local/spark
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PYSPARK_PYTHON=/home/hadoop/anaconda3/envs/pyspark/bin/python3.8
export PATH=$PATH:${JAVA_HOME}/bin:$HADOOP_HOME/bin:$SPARK_HOME/bin
```

4.1.3 配置相关文件



最后，需要在Linux终端中执行如下代码让该环境变量生效：

```
$ source ~/.bashrc
```

此外，还可以对Spark运行过程的日志信息进行设置，只显示错误级别的信息。执行如下命令：

```
$ cd /usr/local/spark/conf
$ sudo mv log4j2.properties.template log4j.properties
$ vim log4j.properties
```

打开log4j.properties以后，把rootLogger.level设置为如下形式（文件中的其他内容不要修改）：

```
rootLogger.level = error
```

4.1.4 验证Spark是否安装成功



配置完成后就可以直接使用Spark，不需要像Hadoop那样运行启动命令。通过运行Spark自带的实例SparkPi，可以验证Spark是否安装成功，命令如下：

```
$ cd  /usr/local/spark
$ ./bin/run-example  SparkPi
```

执行时会输出很多屏幕信息，难以找到最终的输出结果。为了从大量的输出信息中快速找到我们想要的执行结果，可以使用grep 命令进行过滤，命令如下：

```
$ ./bin/run-example  SparkPi  2>&1  |  grep  "Pi is roughly"
```

过滤后的运行结果中只会包含π的小数近似值，类似如下信息：

```
(base) hadoop@hadoop01:/usr/local/spark$ ./bin/run-example SparkPi 2>&1 |grep "Pi is roughly"
Pi is roughly 3.1393756968784845
```

4.2 在pyspark中运行代码



4.2.1 pyspark命令

4.2.2 启动pyspark

4.2.1 pyspark命令



pyspark命令及其常用的参数如下：

```
$ pyspark  --master  <master-url>
```

Spark的运行模式取决于传递给SparkContext的<master-url>的值。<master-url>可以是表4-1中的任一种形式。

<master-url>参数	含义
local	使用一个Worker线程本地化运行Spark（完全不并行）
local[*]	使用与逻辑CPU个数相同数量的线程来本地化运行Spark（“逻辑CPU个数”等于“物理CPU个数”乘以“每个物理CPU包含的CPU核数”）
local[K]	使用*K*个Worker线程本地化运行Spark（在理想情况下，*K*应该根据运行机器的CPU核数来确定）
spark://HOST:PORT	Spark采用独立（Standalone）集群模式，连接到指定的Spark集群，默认端口是7077
yarn	Spark采用YARN集群模式，进一步分为两种不同情况：（1）当spark-shell命令中的另一个参数--deploy-mode的值为client时，以客户端模式连接YARN集群，集群的位置可以在HADOOP_CONF_DIR 环境变量中找到；当用户提交了作业之后，不能关掉Client，Driver Program驻留在Client中，负责调度作业的执行；该模式适合运行交互类型的作业，常用于开发测试阶段；（2）当spark-shell命令中的另一个参数--deploy-mode的值为cluster时，以集群模式连接YARN集群，集群的位置可以在HADOOP_CONF_DIR 环境变量中找到；当用户提交了作业之后，就可以关掉Client，作业会继续在YARN上运行；该模式不适合运行交互类型的作业，常用于企业生产环境
mesos://HOST:PORT	Spark采用Mesos集群模式，连接到指定的Mesos集群，默认接口是5050
k8s://HOST:PORT	连接到Kubernetes集群，进一步分为两种不同情况：（1）当spark-shell命令中的另一个参数--deploy-mode的值为client时，以客户端模式连接Kubernetes集群；（2）当spark-shell命令中的另一个参数--deploy-mode的值为cluster时，以集群模式连接Kubernetes集群

4.2.1 pyspark命令



在Spark中采用Local模式启动pyspark的命令主要包含以下参数。

- --master：这个参数表示当前的pyspark要连接到哪个Master，
  如果是local[*]，就是使用Local模式（本地模式）启动pyspark。
  其中，中括号内的星号表示需要使用几个CPU核心（Core），
  也就是启动几个线程模拟Spark集群。
- --jars：这个参数用于把相关的JAR包添加到CLASSPATH中；
  如果有多个jar包，则可以使用逗号分隔符连接它们。

4.2.1 pyspark命令



比如，要采用Local模式，在4个CPU核心上执行pyspark命令，命令如下：

```
$ cd  /usr/local/spark
$ ./bin/pyspark  --master local[4]
```

或者，可以在CLASSPATH中添加code.jar，命令如下：

```
$ cd  /usr/local/spark
$ ./bin/pyspark  --master  local[4]  --jars  code.jar
```

可以执行“pyspark --help”命令，获取完整的选项列表，具体如下：

```
$ cd /usr/local/spark
$ ./bin/pyspark --help
```

4.2.2 启动pyspark



在Linux终端中执行pyspark命令，就可以启动进入Python版本的Spark交互式执行环境。在执行pyspark命令时，可以不带任何参数。如果已经按照上面的“4.1.3 配置相关文件”设置了PYSPARK_PYTHON环境变量，就可以直接使用如下命令启动pyspark：

```
$ cd /usr/local/spark
$ ./bin/pyspark
```

启动以后会进入pyspark交互式执行环境，如图4-1所示。

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.4.0
      /_/

Using Python version 3.8.18 (default, Sep 11 2023 13:40:15)
Spark context Web UI available at http://hadoop01:4040
Spark context available as 'sc' (master = local[*], app id = local-1694568821430).
SparkSession available as 'spark'.
>>>
```

4.2.2 启动pyspark



当使用pyspark命令没有带上任何参数时，默认使用Local[*]模式启动进入pyspark交互式执行环境。
现在，就可以在里面输入Python代码进行调试了。比如，下面在Python命令提示符“>>>”后面输入一个表达式“8 * 2 + 5”，然后按回车键，就会立即得到结果：

```
>>> 8*2+5
21
```

最后，可以使用如下的所示命令“exit()”退出pyspark：

```
>>> exit()
```

4.3 使用spark-submit命令提交运行程序



spark-submit [options] <app-jar> [app-arguments]

可以通过spark-submit命令提交应用程序，该命令的格式如下：

```
spark-submit
  --master <master-url>  #<master-url>的含义与表4-1中的相同
  --deploy-mode <deploy-mode>   #部署模式
  ... #其他参数
  <application-file>  #Python代码文件
  [application-arguments]  #传递给主类的主方法的参数
```

可以执行“spark-submit --help”命令，获取完整的选项列表，具体如下：

```
$ cd /usr/local/spark
$ ./bin/spark-submit --help
```

4.3 使用spark-submit命令提交运行程序



可以运行Spark安装好以后自带的样例程序SparkPi，它的功能是计算得到pi的值（3.1415926）。在Linux Shell中执行如下命令运行SparkPi:

```
$ cd  /usr/local/spark
$ bin/spark-submit  ₩
> --master local[*]  ₩
> /usr/local/spark/examples/src/main/python/pi.py 10 2>&1  |  grep  "
Pi is roughly"
```

(pyspark) hadoop@hadoop01:/usr/local/spark$ ./bin/spark-submit --master local[*] /usr/local/spark/examples/src/main/python/pi.py 10 2>&1 |grep "Pi is roughly"
Pi is roughly 3.143520



skip

4.4 Spark集群环境搭建（Standalone模式）



4.4.1 集群概况
4.4.2 搭建Hadoop集群
4.4.3 安装Anaconda3
4.4.4 在集群中安装Spark
4.4.5 配置环境变量
4.4.6 Spark的配置
4.4.7 启动Spark集群
4.4.8 关闭Spark集群

4.4.1 集群概况



这里采用3台机器（节点）作为实例来演示如何搭建Spark集群。其中，1台机器（节点）作为Master节点和Worker节点（主机名为hadoop01，IP地址是192.168.91.128），另外两台机器（节点）只作为Worker节点，主机名分别为hadoop02和hadoop03，IP地址分别为192.168.91.129和192.168.91.130。三个节点都使用hadoop用户登录Linux系统。

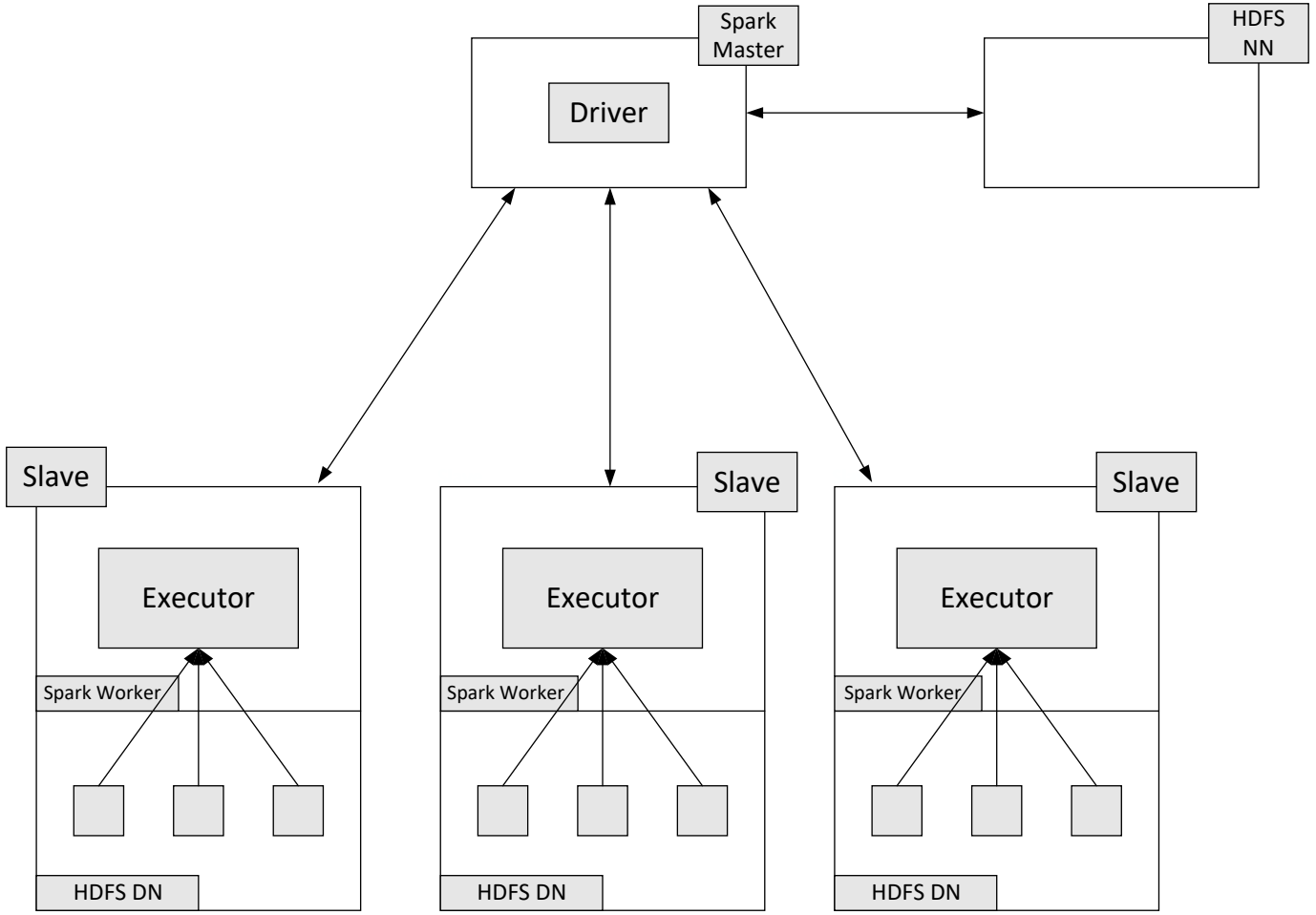
4.4.2 搭建Hadoop集群



Spark作为分布式计算框架，需要与分布式文件系统HDFS组合起来使用，通过HDFS实现数据的分布式存储，使用Spark实现数据的分布式计算。因此，需要在同一个集群中同时部署Hadoop和Spark，这样一来，Spark就可以读写HDFS中的文件。如图4-2所示，在一个集群中同时部署Hadoop和Spark时，HDFS的数据节点（Data Node）和Spark的工作节点（Worker Node）是部署在一起的。这样一来，就可以实现“计算向数据靠拢”，在数据所保存的地方进行计算，减少网络数据的传输。关于Hadoop集群的搭建，可以参考第3.2.7节的内容，按照分布式模式进行搭建。

4.4.2 搭建Hadoop集群





4.4.3安装Anaconda3



参照第3.5节的内容，在所有节点上完成安装Anaconda3的所有步骤，包括安装Anaconda3、设置国内源、创建pyspark环境等。hadoop01节点此前已经安装Anaconda3，这里不用重复安装，只需要为hadoop02和hadoop03安装Anaconda3。

在所有节点上执行如下命令把Python环境设置为pyspark：

```
$ conda activate pyspark
```

然后执行如下命令测试Python环境是否设置成功：

```
$ python
```

最后，执行如下命令退出Python环境：

```
>>> exit()
```

到此为止，所有节点上的Python环境就创建成功了。

4.4.4 在集群中安装Spark



haoop01节点在前面Local模式安装过程中已经安装了Spark，这里不用重复安装。

4.4.5 配置环境变量



在hadoop02和hadoop3节点的终端中执行如下命令：

```
$ vim   ~/.bashrc
```

在.bashrc添加如下配置：

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_371
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export HADOOP_HOME=/usr/local/hadoop
export SPARK_HOME=/usr/local/spark
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PYSPARK_PYTHON=/home/hadoop/anaconda3/envs/pyspark/bin/python3.8
```

运行source命令使配置立即生效：

```
$ source   ~/.bashrc
```

4.4.6 Spark的配置



1．配置workers文件

在hadoop01节点上执行如下命令将 workers.template改名为workers:

```
$ cd   /usr/local/spark/
$ sudo mv   ./conf/workers.template   ./conf/workers
```

在workers文件中设置Spark集群的Worker节点。编辑workers文件的内容，把默认内容localhost替换成如下内容:

```
hadoop01
hadoop02
hadoop03
```

4.4.6 Spark的配置



2．配置spark-env.sh文件

在hadoop01节点上编辑spark-env.sh文件的内容，添加如下内容：

```
#PART1
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_371
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export YARN_CONF_DIR==/usr/local/hadoop/etc/hadoop

#PART2
export SPARK_MASTER_HOST=hadoop01
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8081
export  SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
export SPARK_HISTORY_OPTS="
-Dspark.history.fs.logDirectory=hdfs://hadoop01:9000/sparklog
-Dspark.history.fs.cleaner.enabled=true"

#PART3
export SPARK_WORKER_CORES=1
export SPARK_WORKER_MEMORY=1G
export SPARK_EXECUTOR_CORES=1
export SPARK_EXECUTOR_MEMORY=1G
export SPARK_DRIVER_MEMORY=1G
export SPARK_WORKER_PORT=7078
export SPARK_WORKER_WEBUI_PORT=8082
```

4.4.6 Spark的配置



3.创建历史服务器日志目录

在hadoop01节点的Linux终端中执行如下命令启动HDFS:

```
$ cd /usr/local/hadoop
$ ./sbin/start-dfs.sh
```

执行如下命令在HDFS中创建日志目录sparklog并赋予权限:

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -mkdir /sparklog
$ ./bin/hdfs dfs -chmod 777 /sparklog
```

4.4.6 Spark的配置



4.配置spark-defaults.conf文件

在hadoop01节点上执行如下命令:

```
$ cd /usr/local/hadoop
$ sudo mv spark-defaults.conf.template spark-defaults.conf
$ vim spark-defaults.conf
```

在spark-defaults.conf中写入如下配置信息:

```
spark.eventLog.enabled true  # 开启Spark的日志记录功能
spark.eventLog.dir hdfs://hadoop01:9000/sparklog  #日志记录的保存路径
spark.eventLog.compress true # 是否启动压缩
```

4.4.6 Spark的配置



5．配置Worker节点

在hadoop01节点上执行如下命令，将Master节点（hadoop01节点）上的/usr/local/spark文件夹复制到各个Worker节点上（即hadoop02和hadoop03节点）：

```
$ cd  /usr/local/
$ tar  -zcf  ~/spark.master.tar.gz  ./spark
$ cd  ~
$ scp  ./spark.master.tar.gz  hadoop02:/home/hadoop
$ scp  ./spark.master.tar.gz  hadoop03:/home/hadoop
```

在hadoop02和hadoop03节点上分别执行如下操作：

```
$ cd  ~
$ sudo  rm  -rf  /usr/local/spark/
$ sudo  tar  -zxf  ~/spark.master.tar.gz  -C  /usr/local
$ sudo  chown  -R  hadoop  /usr/local/spark
```

4.4.7 启动Spark集群



1．启动Hadoop集群

在Master节点（hadoop01节点）上执行如下命令：

```
$ cd  /usr/local/hadoop/
$ sbin/start-all.sh
```

2.启动历史服务器

在Master节点（hadoop01节点）上执行如下命令启动历史服务器：

```
$ cd  /usr/local/spark/
$ ./sbin/start-history-server.sh
```

启动成功以后，使用jps命令查看，可以看到一个名称为“HistoryServer”的进程。

4.4.7 启动Spark集群



2．启动Master节点

在Master节点（hadoop01节点）上执行如下命令：

```
$ cd  /usr/local/spark/
$ sbin/start-master.sh
```

3．启动所有Worker节点

在Master节点（hadoop01节点）上执行如下命令：

```
$ cd /usr/local/spark/
$ sbin/start-workers.sh
```

这时，在hadoop01节点上使用jps命令查看进程，可以看到类似如下的进程：

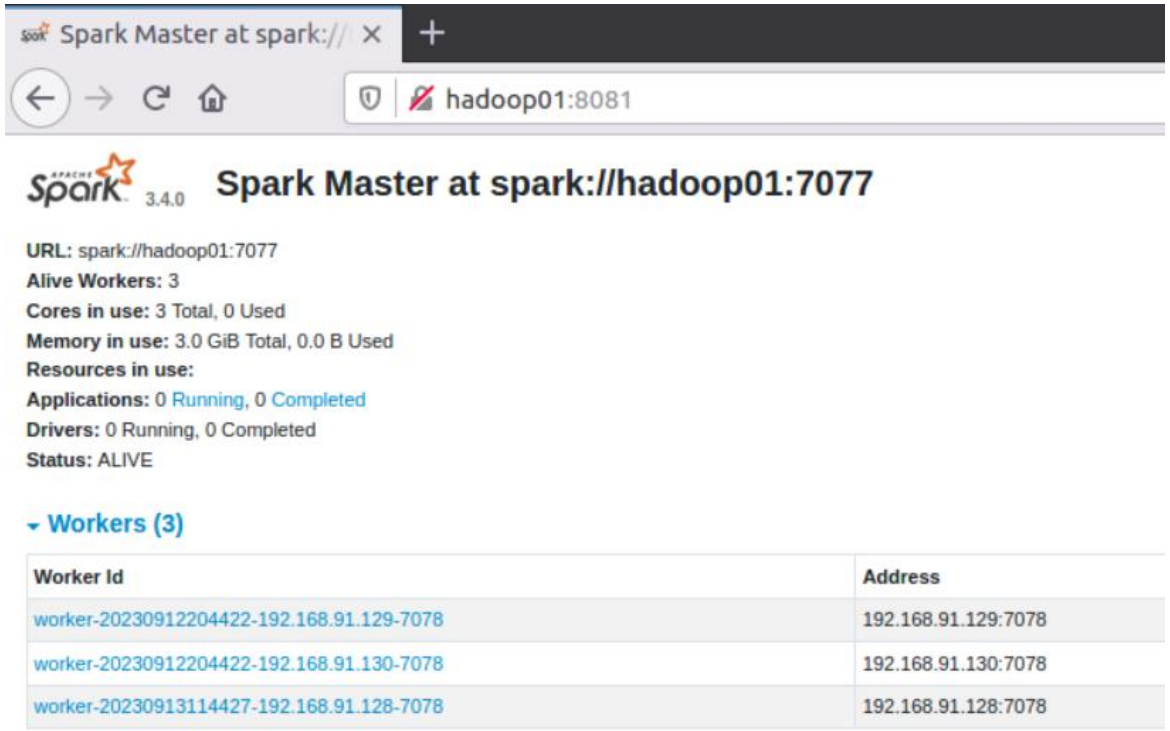
```
5409 DataNode
6434 ResourceManager
7156 Master
7380 Worker
7418 Jps
5228 NameNode
7022 HistoryServer
6607 NodeManager
```

4.4.7 启动Spark集群



4．查看集群信息

在Master主机上打开浏览器，访问http://hadoop01:8081，就可以通过浏览器查看Spark独立集群管理器的集群信息（见图4-3）。



4.4.8 关闭Spark集群



在Master节点（hadoop01节点）上执行下面命令，关闭Spark集群。
首先，关闭Master节点（hadoop01节点），命令如下：

```
$ cd /usr/local/spark
$ sbin/stop-master.sh
```

其次，关闭Worker节点，命令如下：

```
$ sbin/stop-workers.sh
```

最后，关闭Hadoop集群，命令如下：

```
$ cd  /usr/local/hadoop/
$ sbin/stop-all.sh
```

4.5 在集群上运行Spark应用程序



4.5.1 启动Spark集群
4.5.2 提交运行程序

4.5.1 启动Spark集群



请登录Linux系统，打开一个终端，启动Hadoop集群，命令如下：

```
$ cd  /usr/local/hadoop/
$ sbin/start-all.sh
```

然后启动Spark的Master节点和所有Worker节点，命令如下：

```
$ cd  /usr/local/spark/
$ sbin/start-master.sh
$ sbin/start-workers.sh
```

4.5.2 提交运行程序



1．在集群中运行应用程序

向独立集群管理器提交应用，需要把spark://hadoop01:7077作为主节点参数传递给spark-submit。可以运行Spark安装好以后自带的样例程序SparkPi，它的功能是计算得到pi的值（3.1415926）。在Linux Shell中执行如下命令运行SparkPi:

```
$ cd  /usr/local/spark
$ bin/spark-submit  ₩
> --master spark://hadoop01:7077  ₩
> /usr/local/spark/examples/src/main/python/pi.py 10 2>&1 | grep "Pi is roughly"
```

4.5.2 提交运行程序



2．在集群中运行pyspark
也可以用pyspark连接到独立集群管理器上，在Linux Shell中执行如下命令启动pyspark环境：

```
$ cd  /usr/local/spark/
$ bin/pyspark  --master  spark://hadoop01:7077
```

假设HDFS的根目录下已经存在一个文件README.md，则在pyspark环境中执行相关语句：

```
>>> textFile = sc.textFile("hdfs://hadoop01:9000/README.md")
>>> textFile.count()
105
>>> textFile.first()
'# Apache Spark'
```

4.5.2 提交运行程序



3．查看集群信息

执行完上述操作以后，就可以在独立集群管理Web界面查看应用的运行情况，打开浏览器，访问http://hadoop01:8081/，可以看到如图4-4所示的相关信息。

Spark 3.4.0 **Spark Master at spark://hadoop01:7077**

URL: spark://hadoop01:7077
Alive Workers: 3
Cores in use: 3 Total, 3 Used
Memory in use: 3.0 GiB Total, 3.0 GiB Used
Resources in use:
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores
worker-20230912205223-192.168.91.129-7078	192.168.91.129:7078	ALIVE	1 (1 Used)
worker-20230912205223-192.168.91.130-7078	192.168.91.130:7078	ALIVE	1 (1 Used)
worker-20230913115223-192.168.91.128-7078	192.168.91.128:7078	ALIVE	1 (1 Used)

Running Applications (1)

Application ID		Name	Cores	Memory per Executor	Resources Per Executor
app-20230913115741-0000	(kill)	PySparkShell	3	1024.0 MiB	

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor

4.5.2 提交运行程序



可以在浏览器中输入“http://hadoop01:18080/”查看历史服务器的信息（如图4-5所示）。

Spark 3.4.0 **History Server**

Event log directory: hdfs://hadoop01:9000/sparklog

Last updated: 2023-09-13 12:11:50

Client local time zone: Asia/Shanghai

Version	App ID	App Name	Started	Completed
3.4.0	app-20230913121037-0000	PythonPi	2023-09-13 12:10:35	2023-09-13 12:10:44
3.4.0	app-20230913120739-0003	PythonPi	2023-09-13 12:07:36	2023-09-13 12:08:11
3.4.0	app-20230913115741-0000	PySparkShell	2023-09-13 11:57:39	2023-09-13 12:08:04
3.4.0	app-20230913120453-0002	PythonPi	2023-09-13 12:04:51	2023-09-13 12:07:20
3.4.0	app-20230913120149-0001	PythonPi	2023-09-13 12:01:44	2023-09-13 12:03:57
3.4.0	spark-application-1694577319040	PySparkShell	2023-09-13 11:55:13	2023-09-13 11:56:20

Showing 1 to 6 of 6 entries

Show incomplete applications

4.6 Spark on YARN模式



4.6.1 概述
4.6.2 Spark on YARN模式的部署
4.6.3 采用YARN模式运行pyspark
4.6.4 通过spark-submit命令提交程序到YARN集群
4.6.5 Spark on YARN的两种部署模式

4.6.1 概述



从理论上而言，如果我们想要一个稳定的Spark生产环境，那么，最好的选择是Standalone HA（High Available）模式（Standalone HA模式部署方法可以参考相关网络资料）。但是，在企业中，服务器的资源总是很有限的，许多企业不管做什么业务，基本上会搭建Hadoop集群，因而也就会有YARN集群。对于企业来说，在已有YARN集群的前提下再单独准备Standalone集群，会导致资源的利用率不高，所以，在企业中，在多数场景下，会将Spark运行到YARN集群中。YARN本身是一个资源调度框架，负责对运行在内部的计算框架（比如Spark）进行资源调度管理，作为典型的计算框架，Spark一般也是直接运行在YARN中，并接受YARN的资源调度。所以，对于Spark on YARN模式而言，无需部署Spark集群，只需要找一台服务器，充当Spark客户端，即可提交任务到YARN集群中运行。

4.6.1 概述



在Spark on YARN模式中，Master角色由YARN的ResourceManager担任，Worker角色由YARN的NodeManager担任，Driver角色运行在YARN容器中，或运行在提交任务的客户端进程中，Executor运行在YARN容器中。

要想使用Spark on YARN模式，需要满足3个条件：

（1）已经安装了Hadoop集群（意味着已经同时安装了YARN集群）；

（2）需要具有Spark客户端工具，比如spark-submit，可以把Spark程序提交到YARN中；

（3）需要有被提交的程序。

4.6.2 Spark on YARN模式的部署



Spark on YARN模式的部署非常简单，不需要修改任何配置文件，只需要确保环境变量是否正确配置了。在hadoop01节点上打开“/usr/local/spark/conf/spark-env.sh”文件，确保里面要包含如下两个环境变量的配置：

```
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export YARN_CONF_DIR==/usr/local/hadoop/etc/hadoop
```

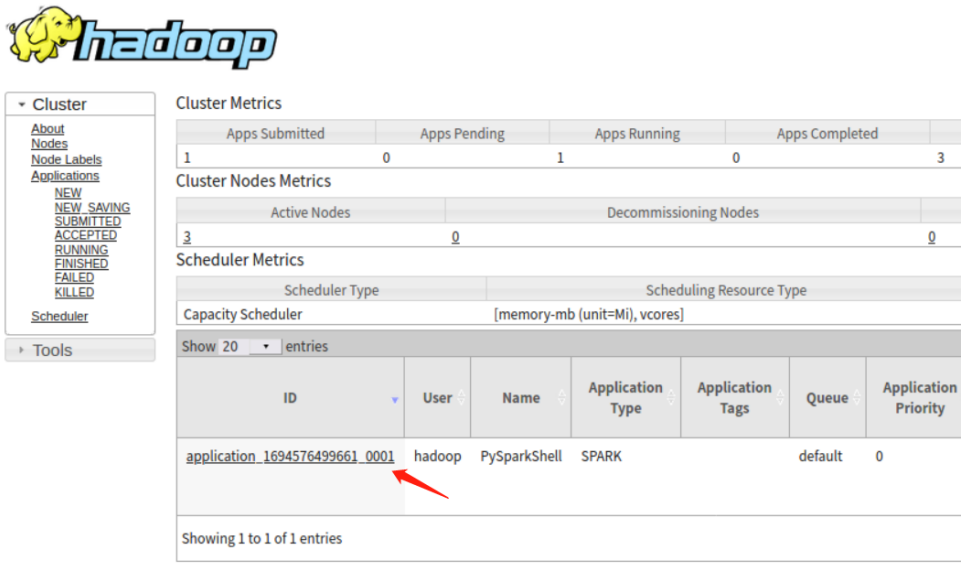
4.6.3 采用YARN模式运行pyspark



要想采用YARN模式运行pyspark，可以hadoop01节点上执行如下命令：

```
$ cd /usr/local/hadoop
$ ./sbin/start-yarn.sh # 启动YARN集群
$ cd /usr/local/spark
$ ./bin/pyspark --master yarn
```

启动成功以后，在浏览器中输入“http://hadoop01:8088”打开YARN集群管理页面（如图4-6所示），在里面就可以查看当前运行的pyspark的信息。



4.6.3 采用YARN模式运行pyspark



假设HDFS的根目录下已经存在一个文件README.md，下面在pyspark环境中执行相关语句：

```
>>> textFile = sc.textFile("hdfs://master:9000/README.md")
>>> textFile.count()
105
>>> textFile.first()
'# Apache Spark'
```

4.6.4 通过spark-submit命令提交程序到YARN集群



要想通过spark-submit命令提交程序到YARN集群中，可以使用如下命令：

```
$ cd  /usr/local/spark
$ bin/spark-submit  ₩
> --master yarn  ₩
> /usr/local/spark/examples/src/main/python/pi.py 10 2>&1 | grep "
Pi is roughly"
```

执行完上述操作以后，就可以在YARN集群管理器Web界面查看应用的运行情况，打开浏览器，访问http://hadoop01:8088，就可以看到相关信息。

4.6.5 Spark on YARN的两种部署模式



Spark on YARN包含两种不同的部署模式，一种是Cluster模式，另一种是Client模式，这两种模式的区别在于Driver的运行位置，具体如下：

（1）Cluster模式：Driver运行在YARN容器内部，和ApplicationMaster在同一个容器内。

（2）Client模式：Driver运行在客户端进程中，比如，Driver运行在spark-submit、pyspark、spark-shell客户端进程中。

4.6.5 Spark on YARN的两种部署模式



表4-2对两种模式进行了比较。总体而言，Client模式一般适用于学习和测试的场景，Cluster模式适用于生产环境。

	Cluster模式	Client模式
Driver运行位置	YARN容器内部	客户端进程内
通讯效率	高	低于Cluster模式
日志查看	日志输出在容器内，不方便查看	日志输出在客户端的标准输出流中，方便查看
生产环境是否使用	推荐使用	不推荐使用
稳定性	稳定	受到客户端进程影响

4.6.5 Spark on YARN的两种部署模式



下面使用Client模式运行程序:

```
$ cd  /usr/local/spark
$ bin/spark-submit  ₩
> --master yarn  ₩
> --deploy-mode client  ₩
> /usr/local/spark/examples/src/main/python/pi.py 10 2>&1  |  grep
"Pi is roughly"
```

如果没有添加“--deploy-mode client”，实际上默认也是采用Client模式。
上面代码执行结束以后，会在屏幕上输出“Pi is roughly 3.141280”。

4.6.5 Spark on YARN的两种部署模式



下面使用Cluster模式运行程序:

```
$ cd  /usr/local/spark
$ bin/spark-submit  ₩
> --master yarn  ₩
> --deploy-mode cluster  ₩
> /usr/local/spark/examples/src/main/python/pi.py 10 2>&1 | grep
"Pi is roughly"
```

上面代码执行结束以后，屏幕上不会输出“Pi is roughly 3.141280”，这个信息需要到日志中查找，可以在浏览器中输入“http://hadoop01:8088”，会出现如图4-7所示界面，点击界面中“ID”下方的链接，会出现如图4-8所示的界面，点击界面中“Logs”下方的链接，会出现如图4-9所示的界面，点击界面中“stdout:.....”链接，会出现如图4-10所示的界面，里面就包含了日志信息“Pi is roughly 3.141280”。

4.6.5 Spark on YARN的两种部署模式



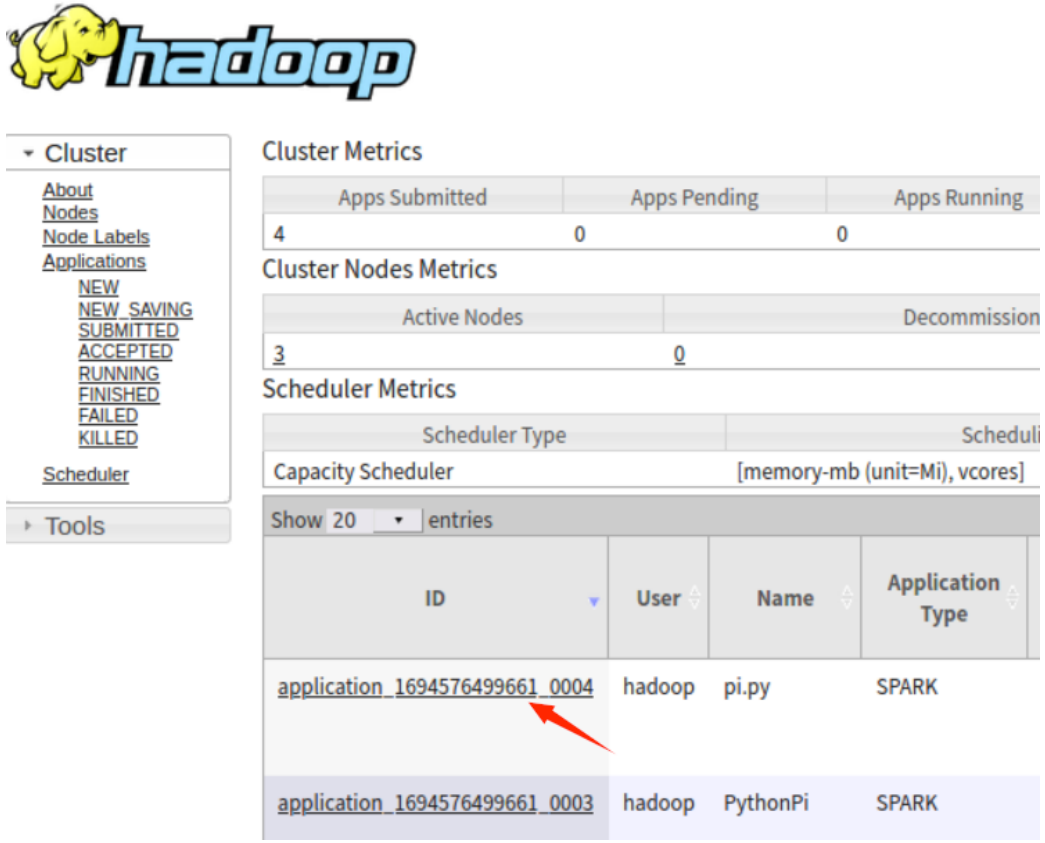


图4-7 YARN集群管理界面（一）

4.6.5 Spark on YARN的两种部署模式



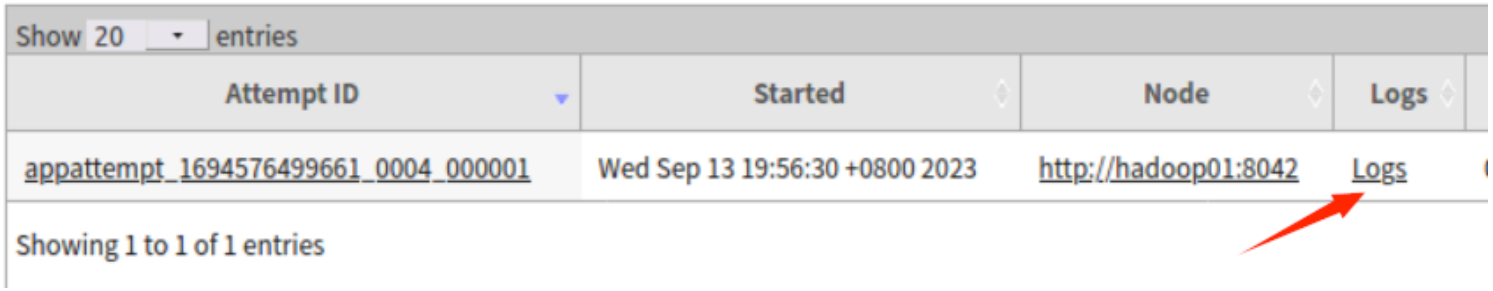


图4-8 YARN集群管理界面（二）

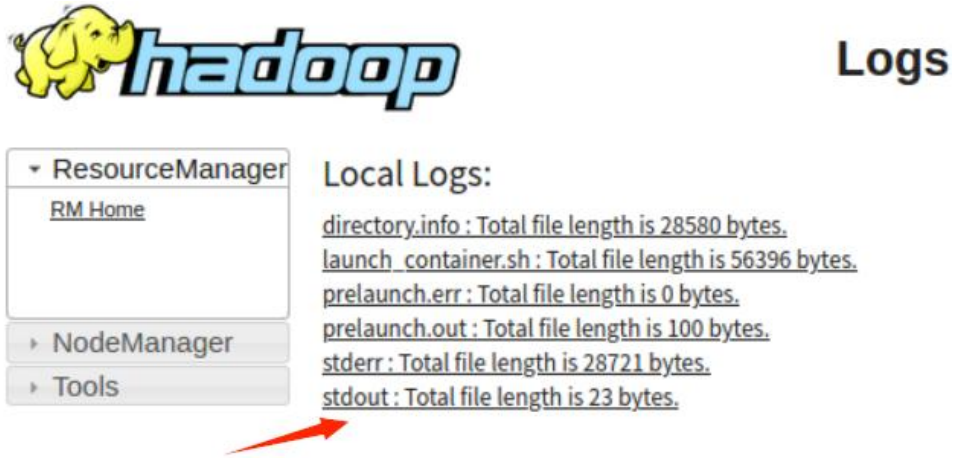


图4-9 YARN集群管理界面（三）

4.6.5 Spark on YARN的两种部署模式





图4-10 YARN集群管理界面（四）

4.7 安装PySpark类库



4.7.1 类库与框架的区别
4.7.2 PySpark类库的安装

4.7.1 类库与框架的区别



类库和框架是不同的两个概念。

- 类库是其他人已经开发好的代码，不能独立运行，可以被导入到其他程序中使用，比如，pandas就是Python的一个第三方类库，我们在编写Python程序时，可以导入pandas类库，编写数据清洗程序代码。
- 而框架是可以独立运行并提供编程结构的一种软件产品，比如，Spark就是一个计算框架，它自己可以独立运行，不需要被导入到其他程序中使用。
- PySpark就是一个Python的类库，内置了完全的Spark API，不能自己独立运行，需要使用“import pyspark”语句被导入到基于Python的Spark应用程序中。
- 这里要特别注意，PySpark类库和“bin/pyspark”是完全不同的两个概念，“bin/pyspark”是一个客户端程序，不是类库，它提供了交互式的Python客户端用于编写基于Spark API的程序。

4.7.1 类库与框架的区别



表4-3给出了PySpark类库和Spark框架的比较。

项目	PySpark	Spark
底层语言	Python	Scala
上层语言支持	仅Python	Python/Java/Scala/R
集群化分布式运行	不支持，仅支持单机	支持
定位	Python类库（客户端）	标准框架（客户端和服务端）
是否可以以守护进程方式运行	不可以	可以
应用场景	本地开发调试Python程序	生产环境集群化运行

4.7.2 PySpark类库的安装



在hadoop01节点中打开一个终端，执行如下命令：

```
$ conda activate pyspark
```

执行如下命令安装PySpark类库：

```
$ pip install pyspark==3.4.0 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

这两个PySpark
完全不同

安装结束后，进入Python环境，执行“import pyspark”命令测试是否成功（如图4-11所示），如果没有报错，就表示安装成功。

```
(pyspark) hadoop@hadoop01:/usr/local/spark$ python
Python 3.8.18 (default, Sep 11 2023, 13:40:15)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyspark
>>>
```

然后，在hadoop02和hadoop03节点上也分别执行上面同样的操作。

4.8 开发Spark独立应用程序



4.8.1 编写程序

4.8.2 通过spark-submit运行程序

4.8.1 编写程序



使用Python进行Spark编程比使用Java和Scala要简单得多。
使用Scala或Java编写Spark程序时，需要使用sbt或者Maven工具进行编译打包。
而使用Python编写的Spark程序，则不需要编译打包，可以直接执行。
下面打开一个Linux终端，
新建一个代码文件“/usr/local/spark/mycode/python/WordCount.py”，
具体内容如下：

```
1  from pyspark import SparkConf, SparkContext
2  conf = SparkConf().setMaster("local[*]").setAppName("My App")
3  sc = SparkContext(conf = conf)
4  logFile = "file:///usr/local/spark/README.md"
5  logData = sc.textFile(logFile, 2).cache()
6  numAs = logData.filter(lambda line: 'a' in line).count()
7  numBs = logData.filter(lambda line: 'b' in line).count()
8  print('Lines with a: %s, Lines with b: %s' % (numAs, numBs))
9  sc.stop()
```

4.8.1 编写程序



对于这段Python代码，可以在hadoop01节点上直接使用如下命令执行：

```
$ conda activate pyspark
$ cd /usr/local/spark/mycode/python
$ python WordCount.py
```

执行该命令以后，可以得到如下结果：

```
Lines with a: 72, Lines with b: 39
```

4.8.2 通过spark-submit运行程序



可以通过spark-submit命令提交应用程序，该命令的格式如下：

```
spark-submit
  --master <master-url>  #<master-url>的含义与表4-1中的相同
  --deploy-mode <deploy-mode>   #部署模式
  ... #其他参数
  <application-file>  #Python代码文件
  [application-arguments]  #传递给主类的主方法的参数
```

可以执行“spark-submit --help”命令，获取完整的选项列表，具体如下：

```
$ cd /usr/local/spark
$ ./bin/spark-submit --help
```

4.8.2 通过spark-submit运行程序



假设采用了Standalone模式的集群，对于前面得到的WordCount.py，可以通过 spark-submit 提交到 Spark 中运行，命令如下：

```
$ /usr/local/spark/bin/spark-submit  ₩
> --master spark://hadoop01:7077  ₩
> /usr/local/spark/mycode/python/WordCount.py
```

执行该命令以后，可以得到如下结果：

```
Lines with a: 72, Lines with b: 39
```

4.9 安装和使用PyCharm



4.9.1 安装PyCharm
4.9.2 使用PyCharm开发Spark程序

4.9.1 安装PyCharm



到PyCharm官网（https://www.jetbrains.com/pycharm/download）下载安装文件pycharm-community-2022.2.3.tar.gz，然后执行如下命令解压缩文件：

```
$ cd ~/Downloads # 假设安装文件在该目录下
$ sudo tar -zxvf pycharm-community-2022.2.3.tar.gz -C /usr/local
$ cd /usr/local
$ sudo mv pycharm-community-2022.2.3 pycharm
$ sudo chown -R hadoop pycharm
```

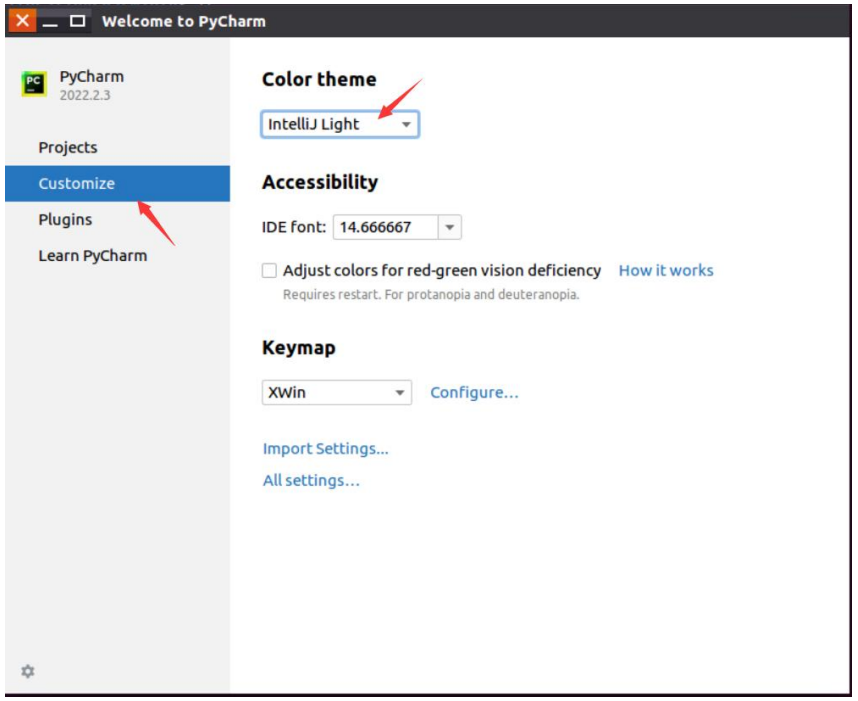
然后，执行如下命令启动PyCharm：

```
$ cd /usr/local/pycharm
$ ./bin/pycharm.sh
```

4.9.1 安装PyCharm



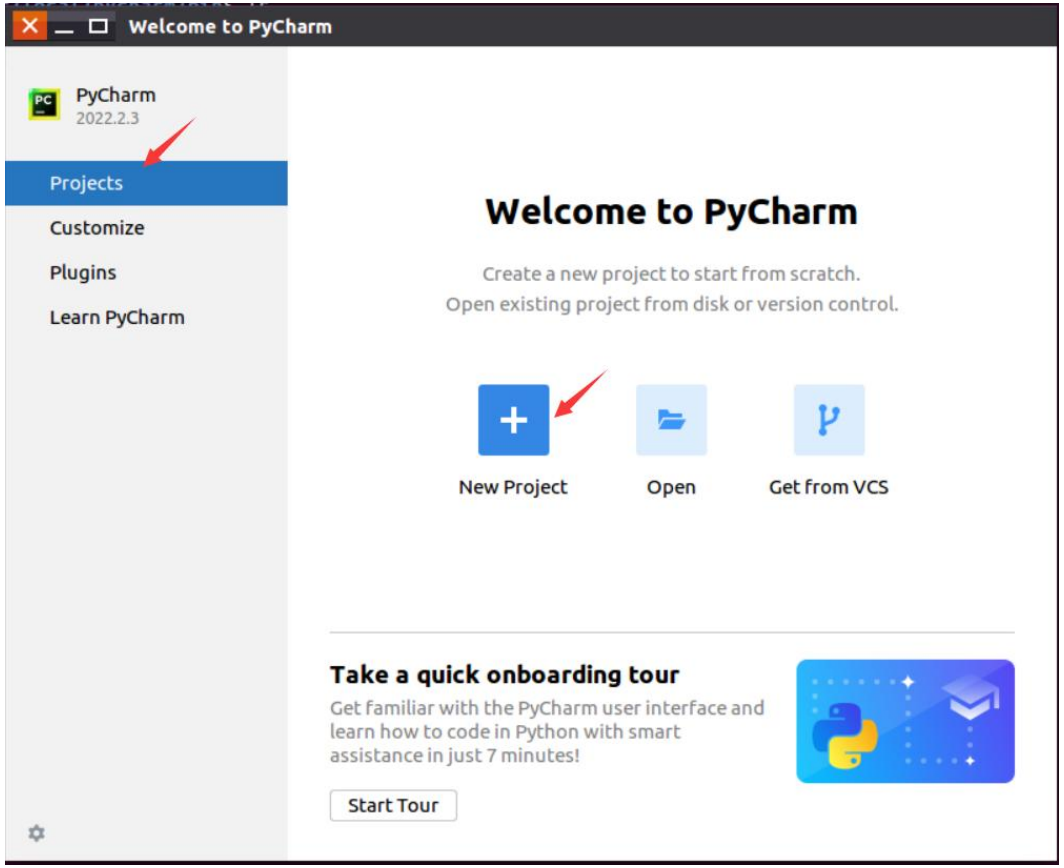
启动以后，需要进行相关配置。如图4-12所示，点击“Customize”，在“Color theme”下拉列表中选择“IntelliJ Light”。



4.9.1 安装PyCharm



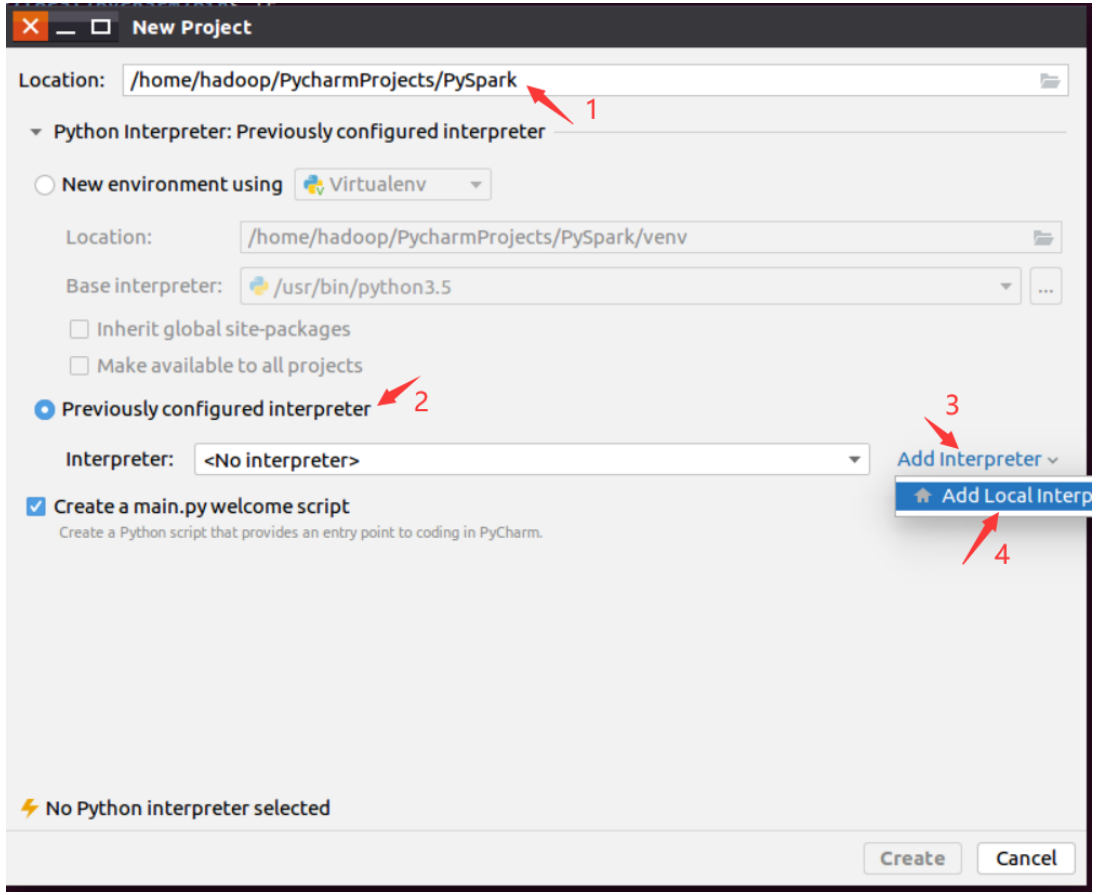
点击“Projects”，再点击“New Project”，新建一个项目，如图4-13所示。



4.9.1 安装PyCharm



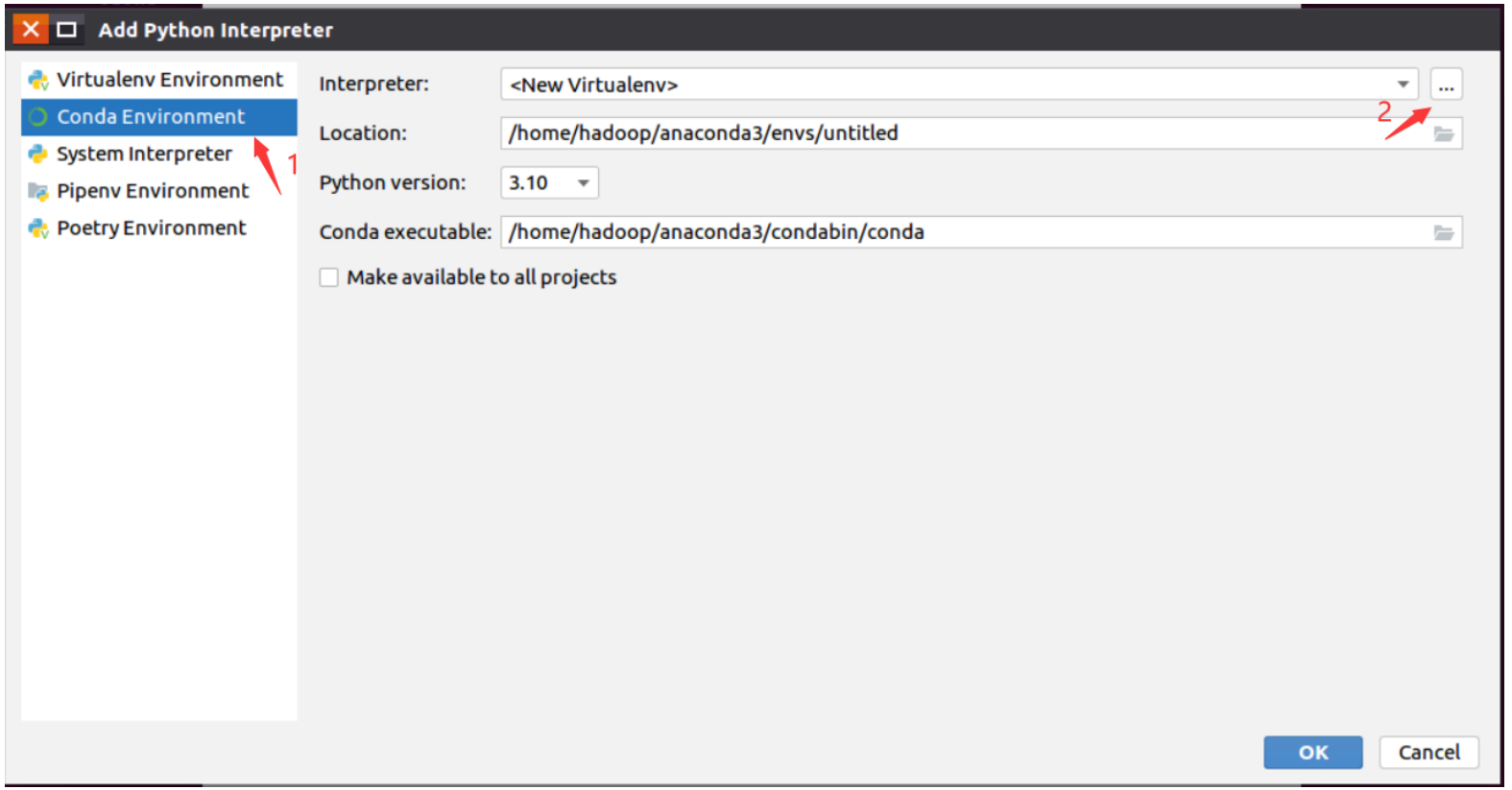
对项目进行配置，如图4-14所示，在“Location”右侧设置项目的保存路径和名称，比如设置为“/home/hadoop/PycharmProjects/PySpark”。然后，点击“Previously configured interpreter”，再点击“Add interpreter”，然后点击“Add Local Interpreter”。



4.9.1 安装PyCharm



在弹出的页面中（如图4-15所示），在界面左侧点击“Conda Environment”，在界面右侧点击“...”。

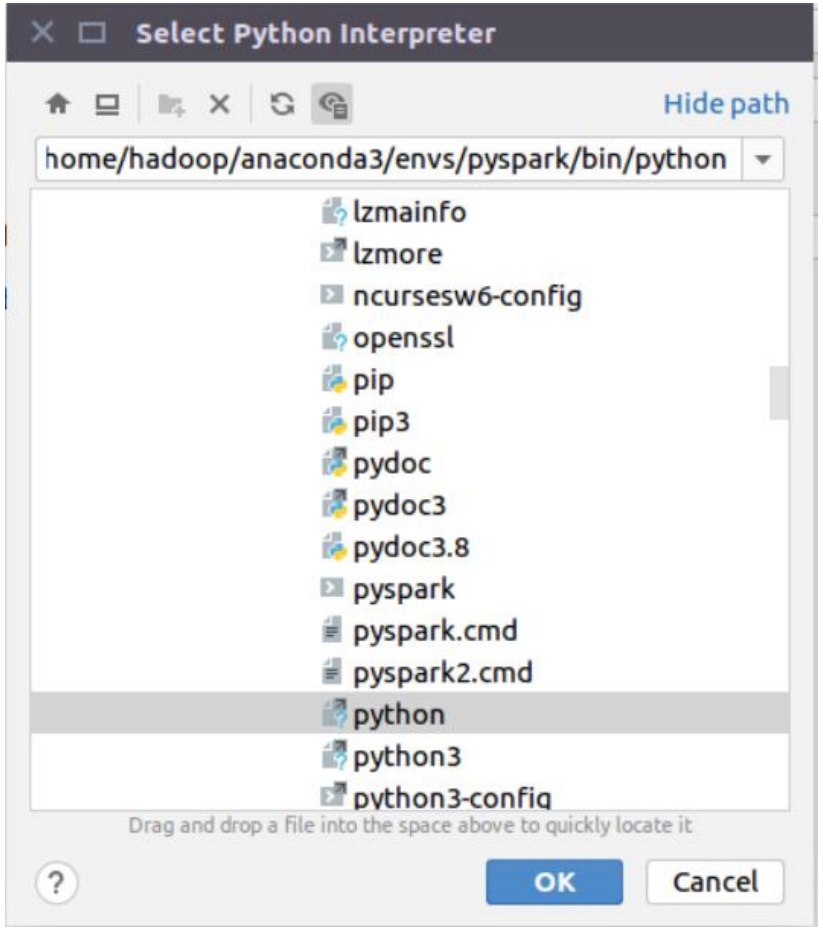


4.9.1 安装PyCharm



在弹出的页面中（如图4-16所示），找到Python解释器的路径“/home/hadoop/anaconda3/envs/pyspark/bin/python”，点击“OK”按钮。

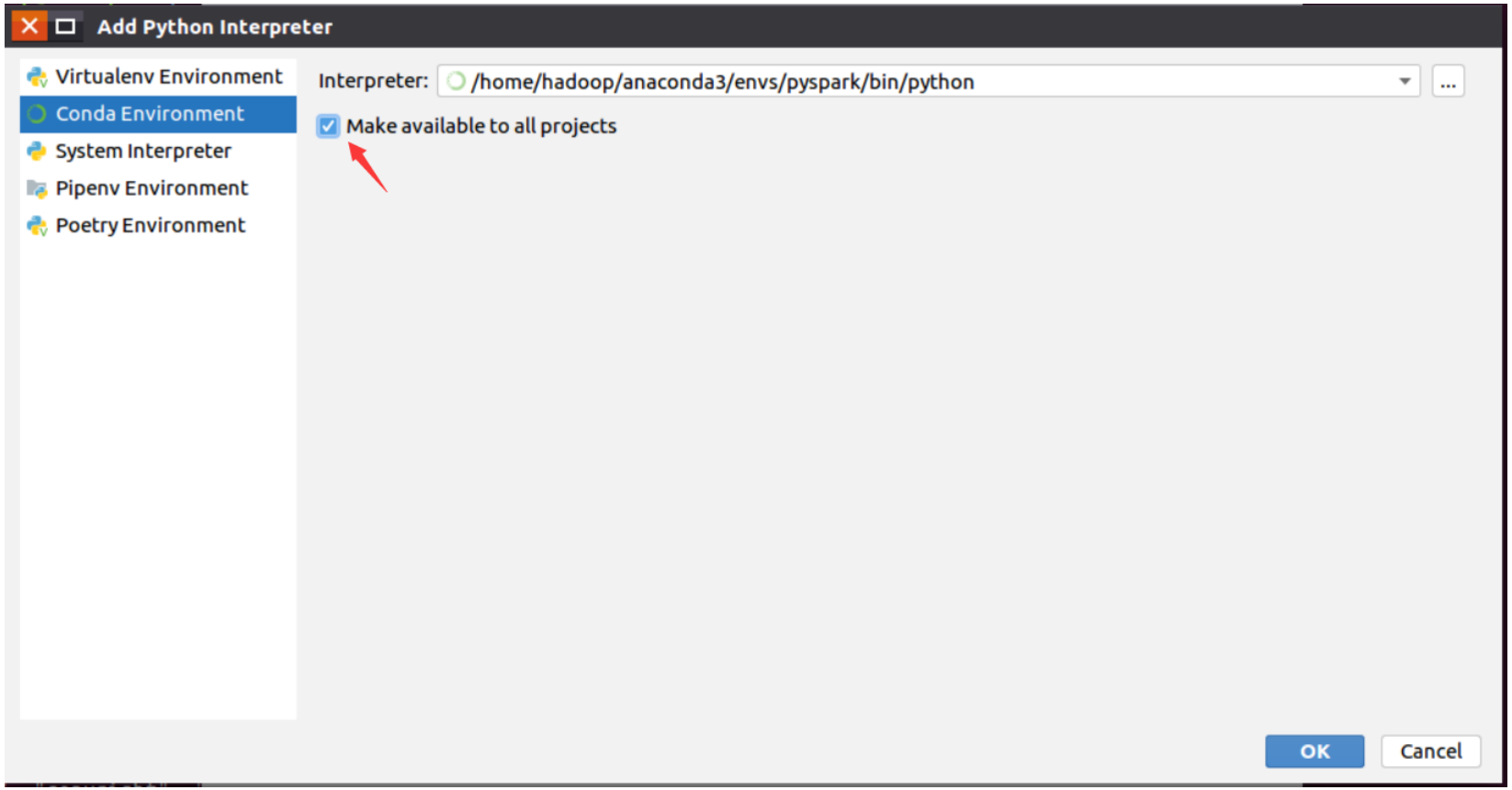
Conda environment



4.9.1 安装PyCharm



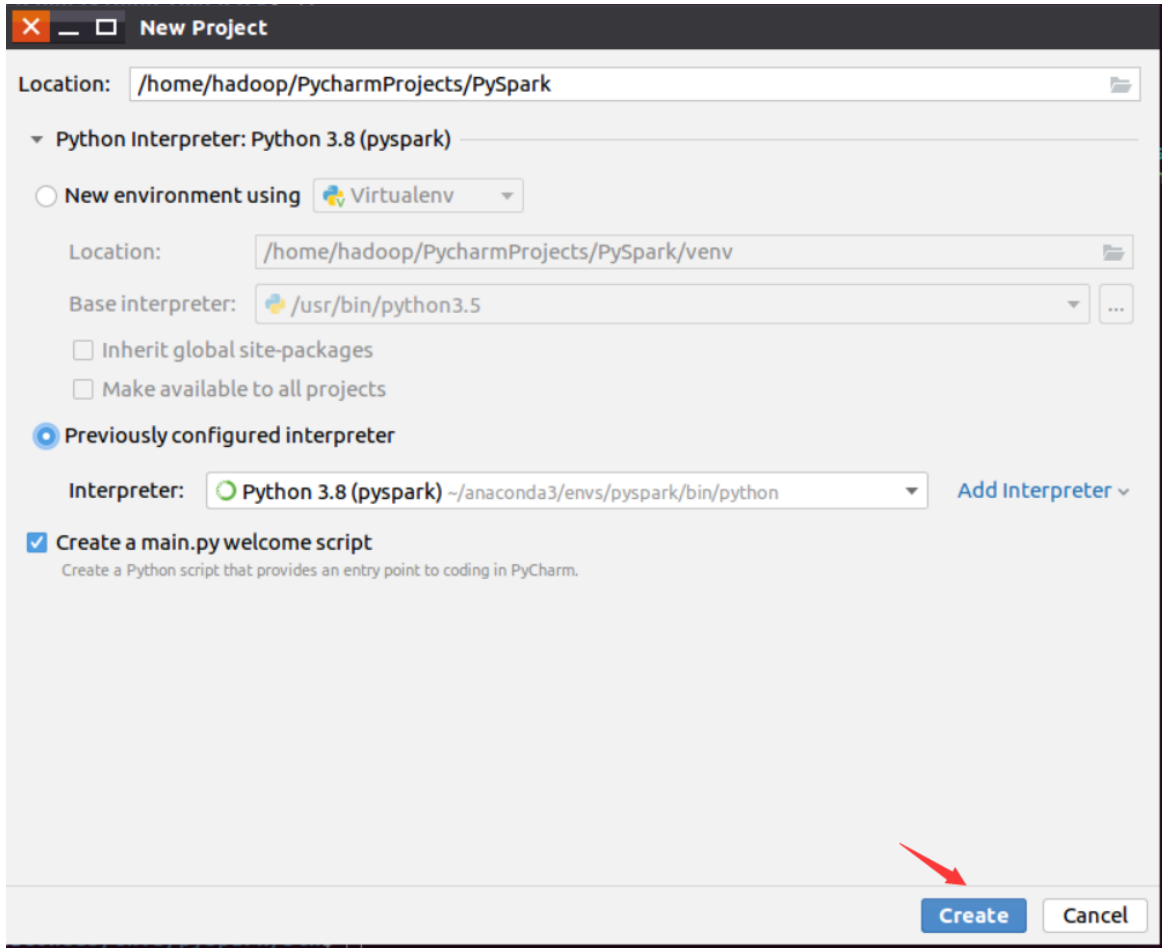
在如图4-17所示界面中，把“Make available to all projects”选中，点击“OK”按钮。



4.9.1 安装PyCharm



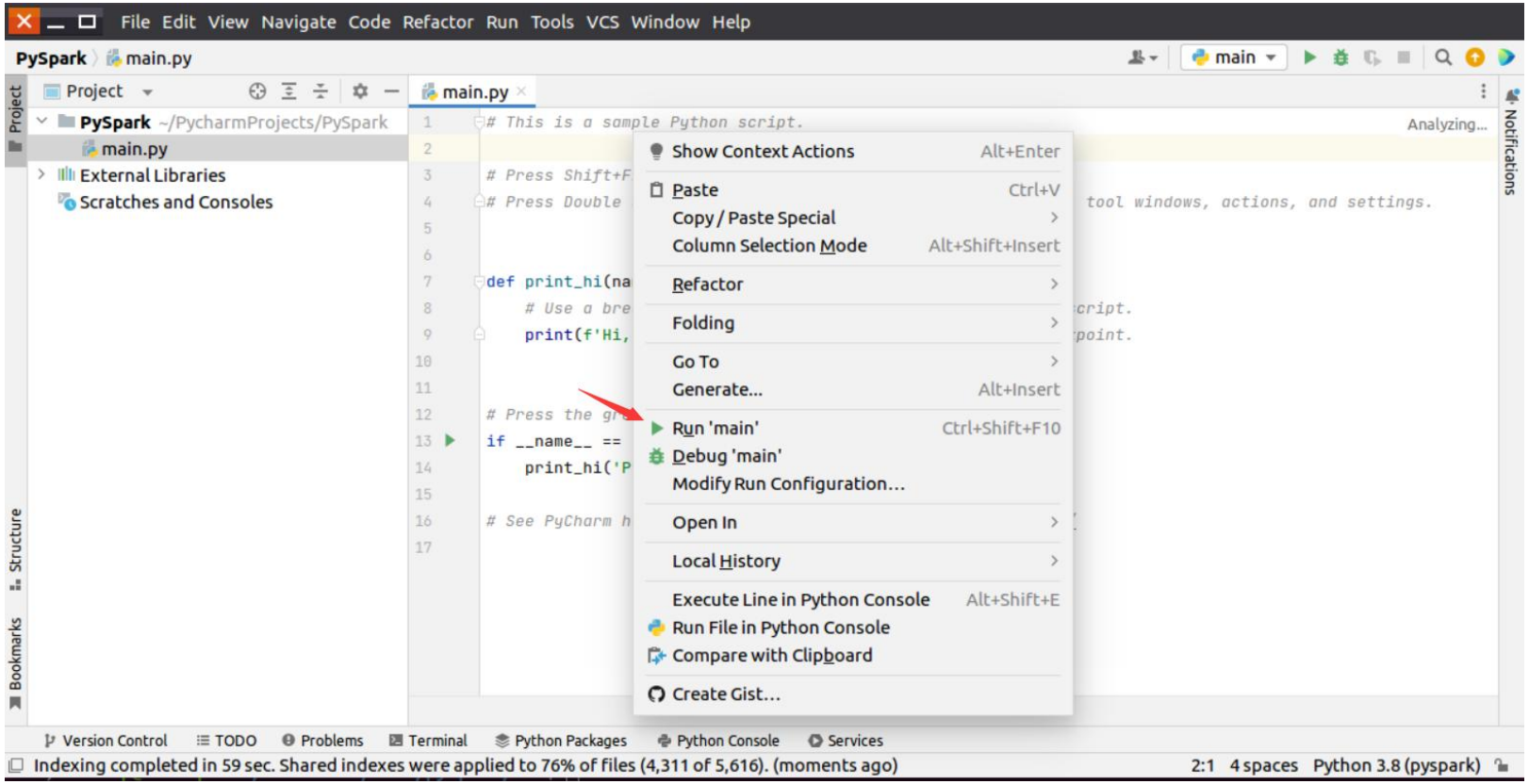
最后，在如图4-18所示的界面中，点击“Create”按钮，就可以完成项目的创建了。



4.9.1 安装PyCharm



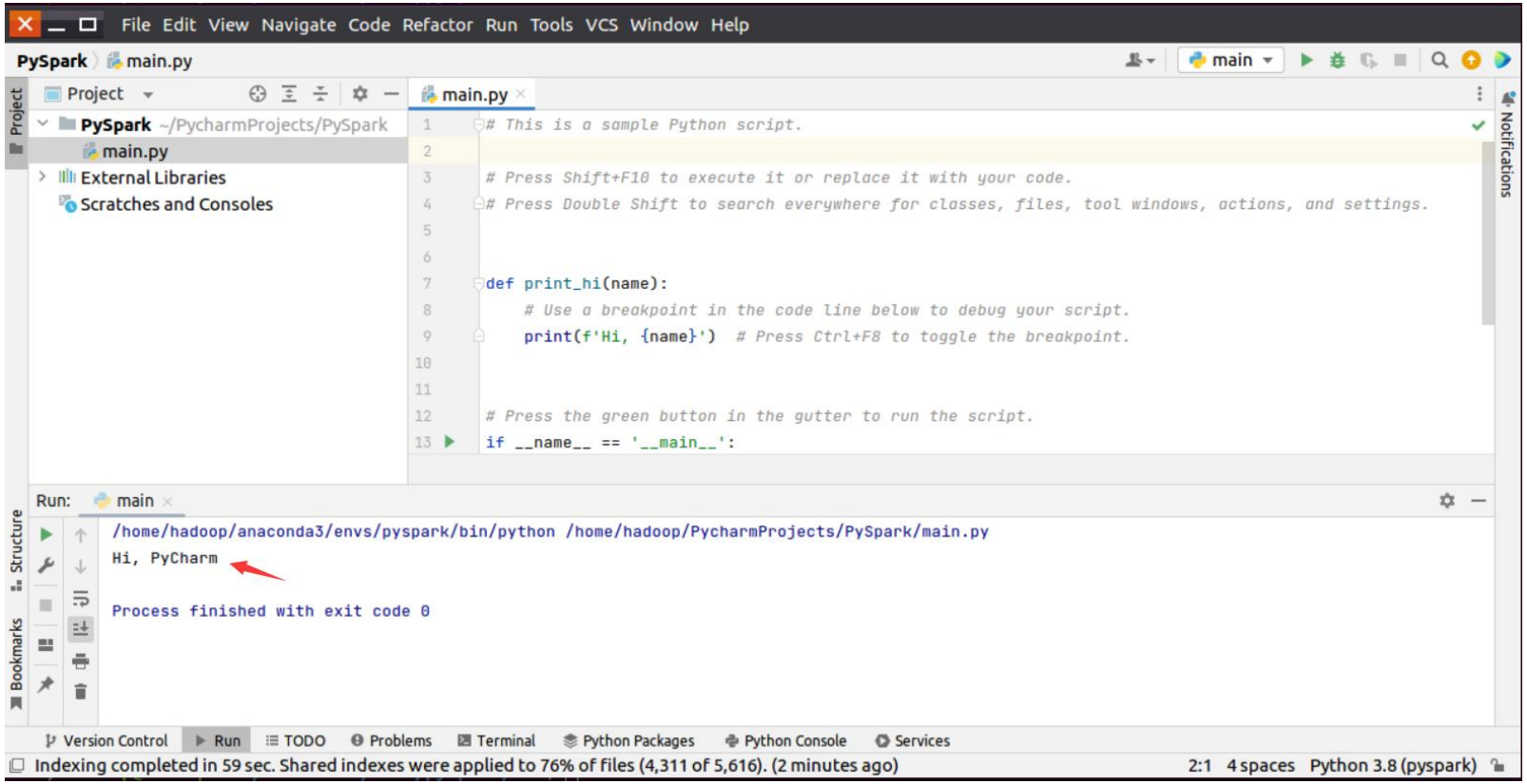
创建好项目以后（如图4-19所示），系统会默认创建一个main.py文件，可以在该文件区域内点击鼠标右键，在弹出的菜单中点击“Run ‘main’”。



4.9.1 安装PyCharm



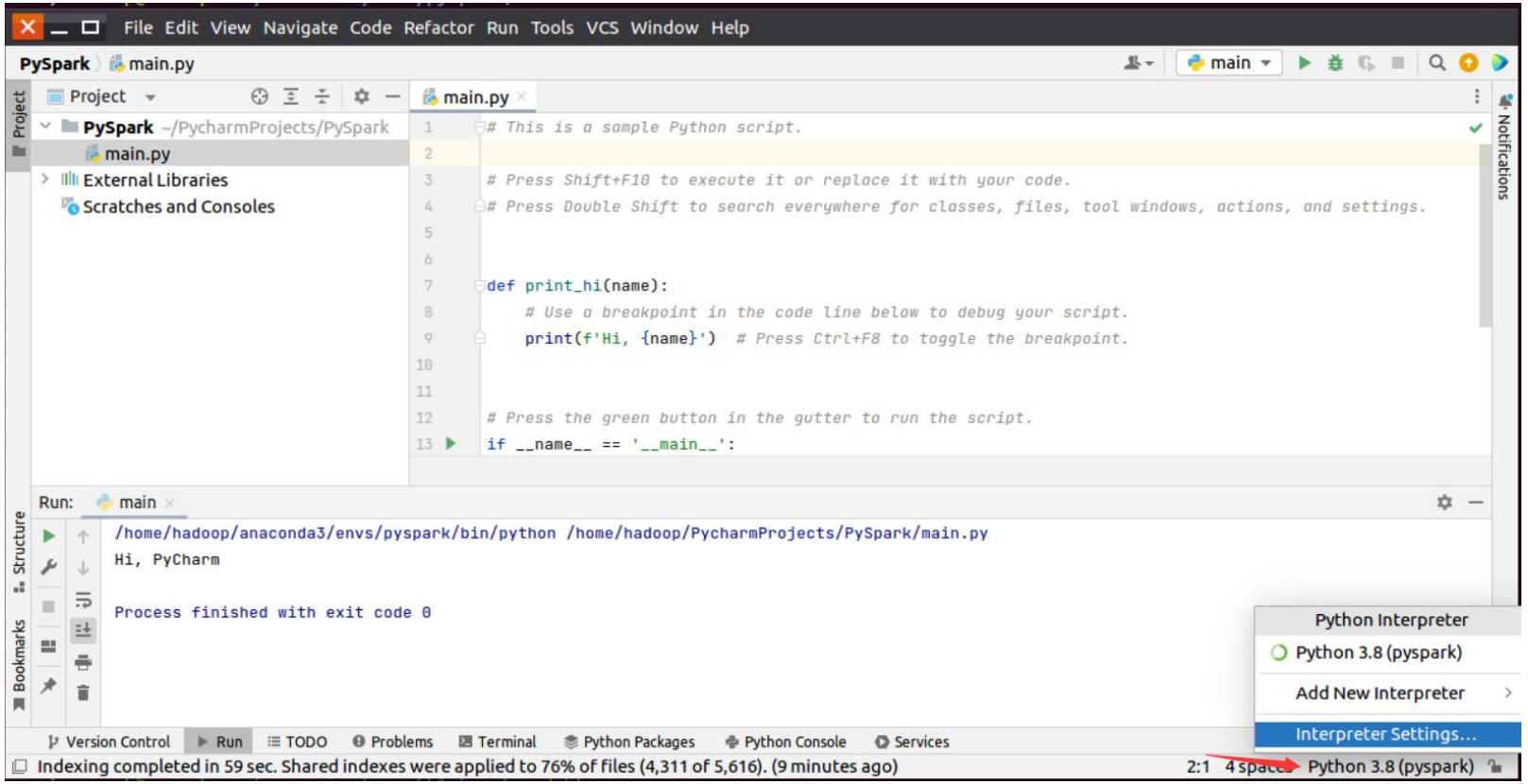
如果代码运行成功，就会在界面下方控制台区域输出“Hi, PyCharm”（如图4-20所示），表明Python解释器工作正常。



4.9.1 安装PyCharm



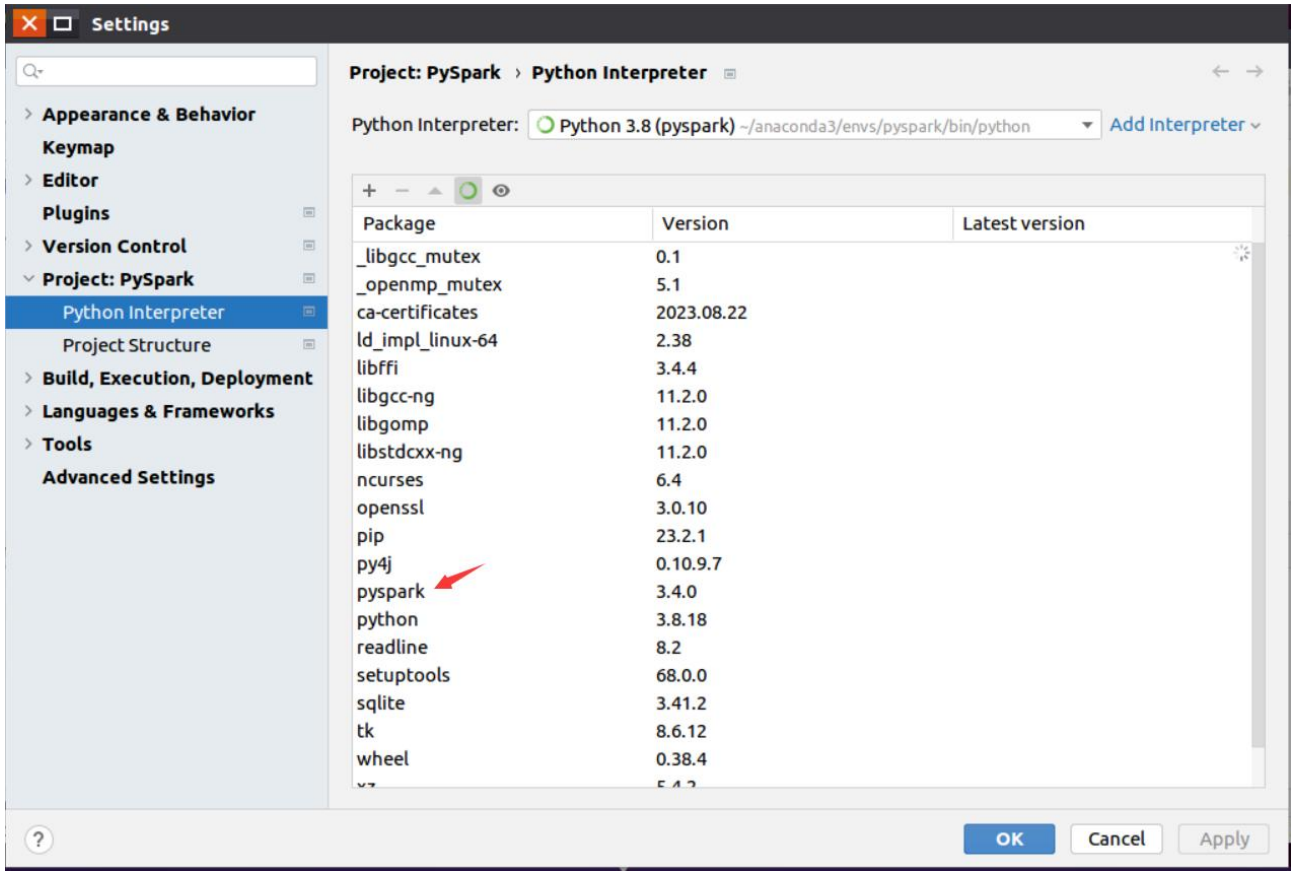
在界面的底部（如图4-21所示），点击“Python3.8(pyspark)”，在弹出的菜单中点击“Interpreter Settings...”，查看Python解释器的相关信息。



4.9.1 安装PyCharm



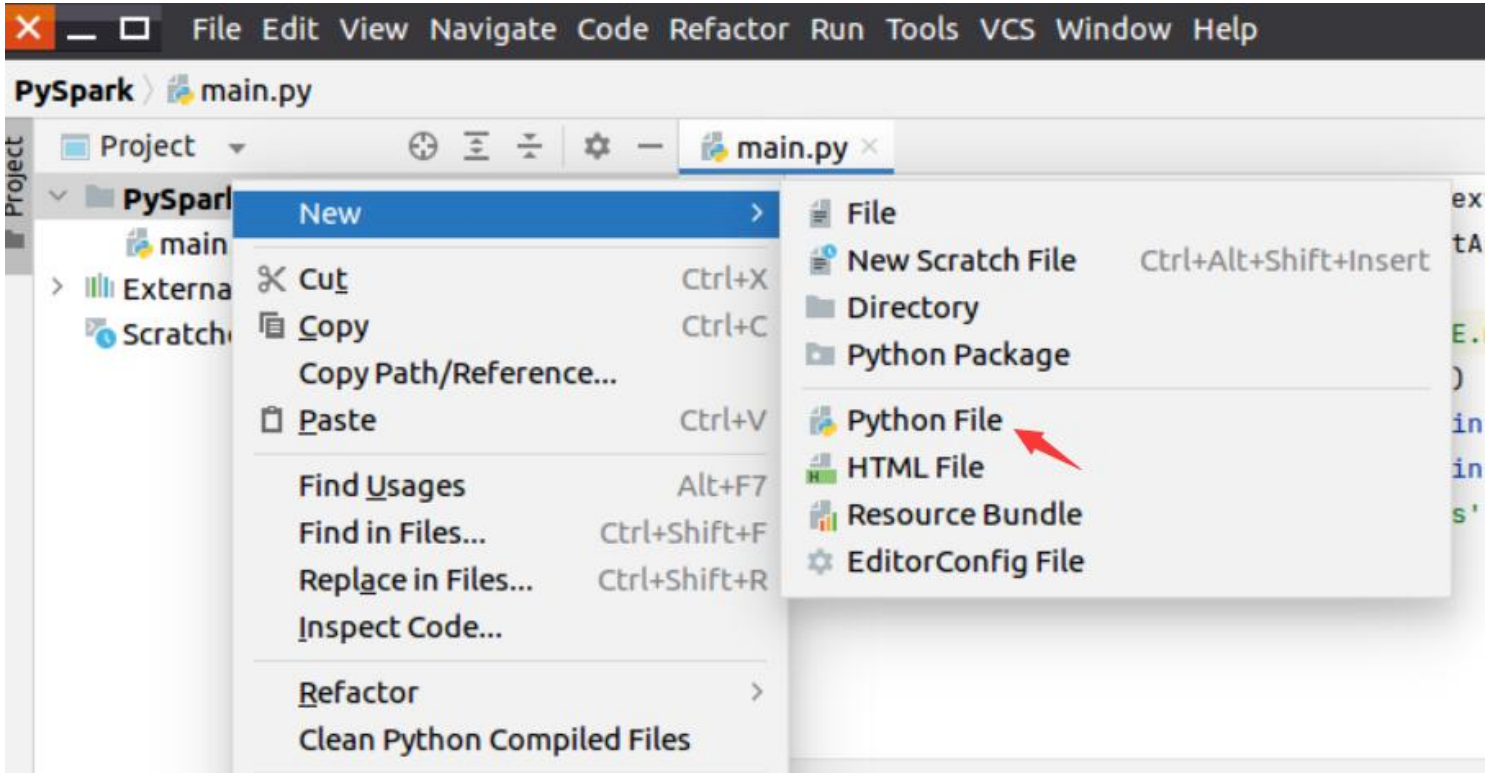
在弹出的界面中（如图4-22所示），在界面左侧选择“Python Interpreter”，可以看到，在右侧界面中存在一个名称为pyspark的包，说明已经成功导入了PySpark类库，可以支持我们进行Spark程序的开发。



4.9.2 使用PyCharm开发Spark程序



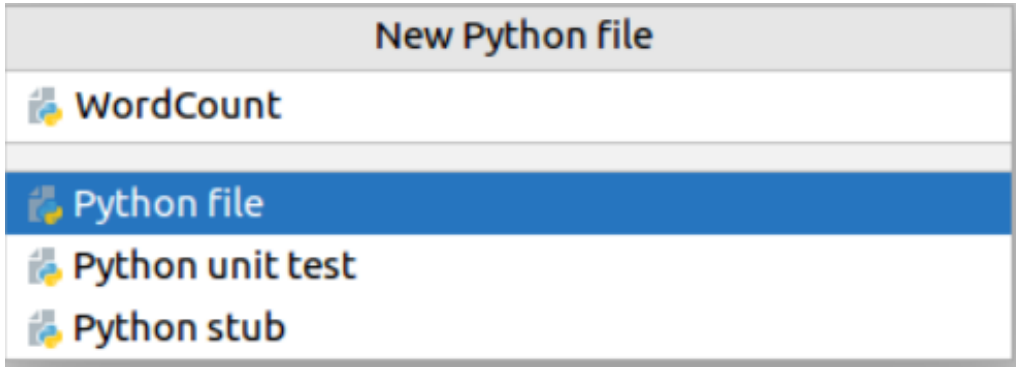
在上面的步骤中，我们在PyCharm中新建了一个项目PySpark，现在在该项目中新建一个代码文件WordCount.py。如图4-23所示，在项目名称“PySpark”上单击鼠标右键，在弹出的菜单中选择“New”，再在弹出的菜单中选择“Python File”。



4.9.2 使用PyCharm开发Spark程序



在弹出的界面中（如图4-24所示），输入代码文件名称“WordCount”，然后按回车键，系统就会创建一个代码文件WordCount.py。



4.9.2 使用PyCharm开发Spark程序



在WordCount.py文件中输入如下内容：

```
# coding:utf8
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    conf = SparkConf().setMaster("local[*]").setAppName("My App")
    sc = SparkContext(conf = conf)
    logFile = "file:///usr/local/spark/README.md"
    logData = sc.textFile(logFile, 2).cache()
    numAs = logData.filter(lambda line: 'a' in line).count()
    numBs = logData.filter(lambda line: 'b' in line).count()
    print('Lines with a: %s, Lines with b: %s' % (numAs, numBs))
    sc.stop()
```

4.9.2 使用PyCharm开发Spark程序



在运行程序之前，需要首先开启Hadoop集群和Spark集群。为了节省系统资源，加快实验运行速度，虽然此前我们安装了3个虚拟机，但是，这里可以只开启hadoop01这一个虚拟机，不用开启hadoop02和hadoop03。在hadoop01上执行如下命令开启Hadoop集群：

```
$ cd /usr/local/hadoop
$ ./sbin/start-dfs.sh  # 这里只启动了HDFS，没有启动YARN
```

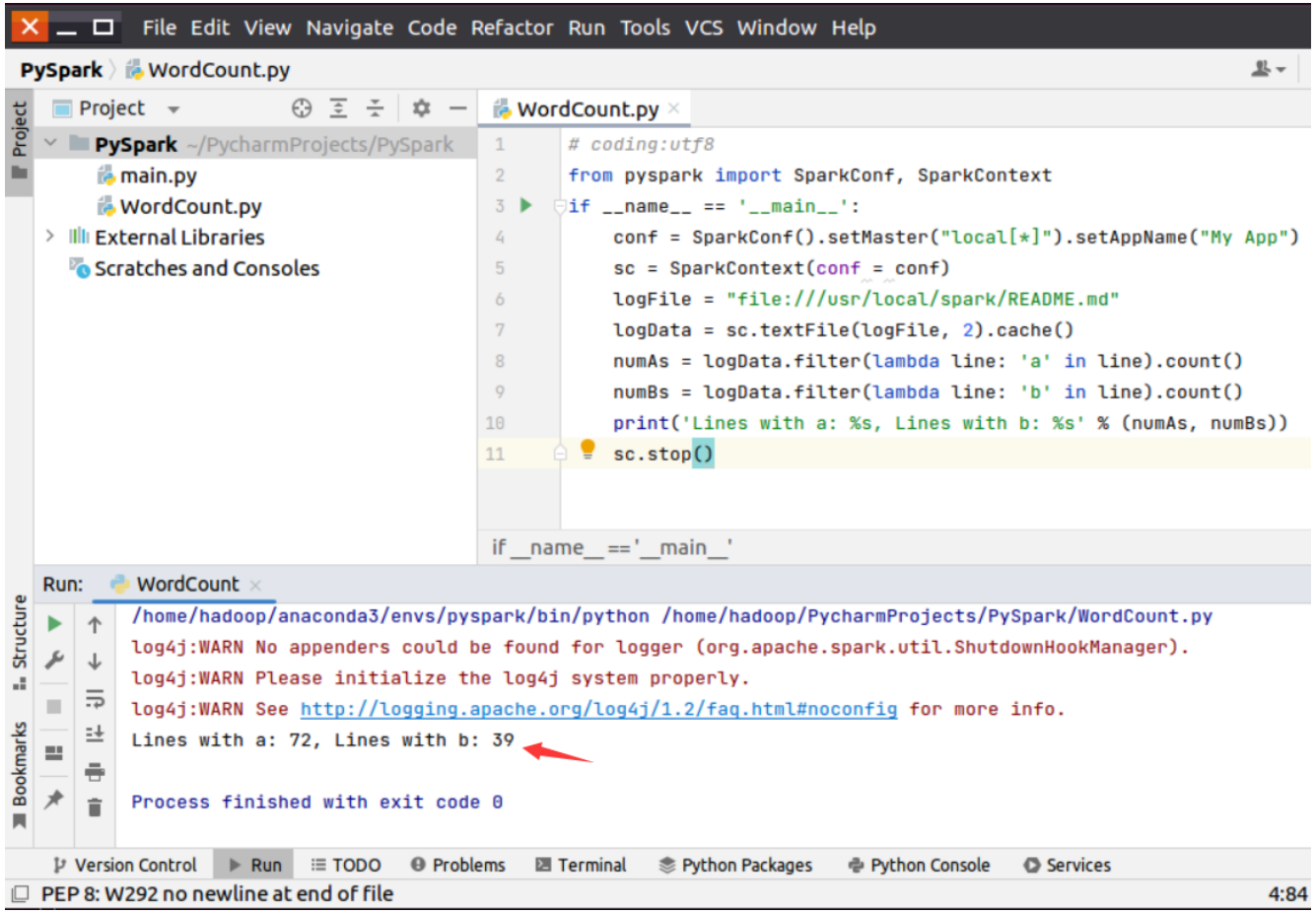
在hadoop01上执行如下命令开启Spark集群：

```
$ cd /usr/local/spark
$ ./sbin/start-all.sh # 同时启动了Spark的Master和Worker
```

4.9.2 使用PyCharm开发Spark程序



然后，在PyCharm中运行WordCount.py，就可以在控制台输出程序执行结果（如图4-25所示）。



4.9.2 使用PyCharm开发Spark程序



如果需要读取HDFS文件，只需要把logFile设置为如下内容：

```
logFile = "hdfs://hadoop01:9000/README.md"
```

需要注意的是，需要提前把本地文件“/usr/local/spark/README.md”上传到HDFS中。

Batch File_spark_setup.sh

1. 실행 권한 부여
chmod +x /home/Hadoop/spark_setup.sh

2. 필요한 파일 다운로드
 - spark_setuo_file_download.sh
3. 최초 1회만 (사전 설정 실행)
 - spark_pre_setup.sh

```
Terminal File Edit View Search Terminal Help
(base) hadoop@hadoop01:/opt/spark-jars$ ll
total 332416
drwxr-xr-x 2 hadoop hadoop      4096 Jul  4 22:31 ./
drwxr-xr-x 4 root   root        4096 Jul  4 22:29 ../
-rw-rw-r-- 1 hadoop hadoop 332655520 Jul  4 22:31 aws-java-sdk-bundle-1.12.488.jar
-rw-rw-r-- 1 hadoop hadoop    145516 Jul  4 22:30 commons-pool2-2.11.1.jar
-rw-rw-r-- 1 hadoop hadoop    760519 Jul  4 22:30 hadoop-aws-3.3.5.jar
-rw-rw-r-- 1 hadoop hadoop   5246932 Jul  4 22:29 kafka-clients-3.5.1.jar
-rw-rw-r-- 1 hadoop hadoop   1081604 Jul  4 22:30 postgresql-42.6.0.jar
-rw-rw-r-- 1 hadoop hadoop    426477 Jul  4 22:29 spark-sql-kafka-0-10_2.12-3.4.0.jar
-rw-rw-r-- 1 hadoop hadoop     56739 Jul  4 22:29 spark-token-provider-kafka-0-10_2.12-3.4.0.jar
(base) hadoop@hadoop01:/opt/spark-jars$ cd ../downloads/
(base) hadoop@hadoop01:/opt/downloads$ ll
total 332416
drwxr-xr-x 2 hadoop hadoop      4096 Jul  4 22:30 ./
drwxr-xr-x 4 root   root        4096 Jul  4 22:29 ../
-rw-rw-r-- 1 hadoop hadoop 332655520 Jun 14  2023 aws-java-sdk-bundle-1.12.488.jar
-rw-rw-r-- 1 hadoop hadoop    145516 Aug 14  2021 commons-pool2-2.11.1.jar
-rw-rw-r-- 1 hadoop hadoop    760519 Mar 16  2023 hadoop-aws-3.3.5.jar
-rw-rw-r-- 1 hadoop hadoop   5246932 Jul 15  2023 kafka-clients-3.5.1.jar
-rw-rw-r-- 1 hadoop hadoop   1081604 Mar 18  2023 postgresql-42.6.0.jar
-rw-rw-r-- 1 hadoop hadoop    426477 Apr  8  2023 spark-sql-kafka-0-10_2.12-3.4.0.jar
-rw-rw-r-- 1 hadoop hadoop     56739 Apr  8  2023 spark-token-provider-kafka-0-10_2.12-3.4.0.jar
```

WinScp에서 수정한 사항을 Linux옮길때 문제



✅ **문제 1: bash: /home/hadoop/.bashrc: line 6: syntax error near unexpected token ₩$'in₩r'`**

🔍 **원인: .bashrc 파일이 Windows 스타일(CRLF) 줄바꿈으로 저장됨**

- 즉, .bashrc 파일이 **Windows에서 작성되었거나**, WinSCP 또는 Notepad++ 등에서 CRLF(Line ending: ₩r₩n) 형식으로 저장되었기 때문입니다.
- Bash는 이 형식을 이해하지 못해 문법 오류가 납니다.

✅ **해결 방법: .bashrc 파일의 줄바꿈 형식을 Unix(LF)로 변환**

① **방법 A: dos2unix 명령어 사용 (권장)**

sudo apt-get install dos2unix # 설치되어 있지 않은 경우

dos2unix ~/.bashrc (콘다인 경우: dos

```
(base) hadoop@hadoop01:~$ dos2unix ~/.condarc
dos2unix: converting file /home/hadoop/.condarc to Unix format ...
```

- 다시 실행: source ~/.bashrc

② **방법 B: sed 명령어 사용 (dos2unix가 없다면)**

sed -i 's/₩r$//' ~/.bashrc

source ~/.bashrc

4.10 本章小结



Spark可以支持多种部署模式，在日常学习和应用开发环节，可以使用单机环境进行部署。本章首先介绍了Spark在单机环境下的安装配置方法。然后，介绍了交互式编程环境pyspark，它是一种交互式开发环境，可以立即解释执行用户输入的语句。

Spark支持Java、Python和Scala等编程语言，使用pyspark命令启动进入的是Python交互式环境。在开发Spark独立应用程序时，采用Python语言编写的Spark程序，可以直接通过spark-submit提交运行。

在实际应用中，企业会搭建Spark集群，而不是单机部署，因此，本章介绍了Spark集群环境的搭建方法以及如何在集群上运行Spark程序。YARN是目前被企业广泛使用的资源调度框架，因此，本章介绍了Spark on YARN模式的集群搭建方法。

在程序开发环节，我们经常会使用PySpark类库去开发Spark应用程序，因此，本章介绍了PySpark类库的安装方法和独立应用程序开发方法。采用开发工具可以大幅提升程序开发效率，因此，本章最后介绍了开发工具PyCharm的安装和使用方法。