Spark环境搭建和使用方法 - 中文讲课脚本

슬라이드 1: 课程开场

大家好！欢迎来到我们今天的课程。今天我们要学习第四章的内容——"Spark环境搭建和使用方法"。

我是李阳源老师，如果大家有任何问题，可以通过我的邮箱ywlee1212@gmail.com联系我，或者访问我的主页。

今天这节课非常重要，因为我们要亲手搭建Spark(스파크)环境。很多同学可能听说过Spark很复杂，安装很困难，但是通过今天的学习，大家会发现其实并不难。

슬라이드 2: 课程大纲介绍

好，我们先来看一下今天的学习内容。今天我们总共要学习九个部分：

首先，4.1部分，我们要学习如何安装Spark的Local模式。什么是Local模式呢？就是在我们自己的电脑上安装，不需要集群，适合我们学习和开发使用。

然后是4.2，学习在pyspark中运行代码。pyspark就是Python版本的Spark，非常适合我们做数据分析。

接下来4.3，我们要学习使用spark-submit命令提交运行程序。这是实际工作中经常用到的。

4.4到4.6是关于集群环境的，包括Standalone模式和YARN模式。这些是企业级应用会用到的。

最后三个部分是关于开发环境的搭建，包括PySpark库的安装、开发独立应用程序，以及PyCharm的使用。

슬라이드 3: 4.1 Spark安装概述

好，现在我们开始第一部分——Spark安装。

我们今天要安装的是Local模式的Spark。Local模式的好处是什么呢？就是简单，不需要复杂的集群配置，在我们自己的电脑上就可以学习Spark的所有功能。

安装过程分为四个步骤：

1. 首先是准备基础环境

2. 然后下载安装文件
3. 接着配置相关文件
4. 最后验证安装是否成功

这四个步骤缺一不可，大家一定要按照顺序来做。

슬라이드 4: 4.1.1 基础环境说明

现在我们来看基础环境的要求。

大家要知道，Spark(스파크)和Hadoop(하둡)是可以很好地配合工作的。Hadoop负责数据存储，比如HDFS(분산파일시스템)和HBase(베이스)；而Spark负责数据计算。它们各司其职，配合得很好。

虽然Spark和Hadoop都可以在Windows上安装使用，但是我强烈建议大家在Linux系统中安装和使用。为什么呢？因为在实际的企业环境中，几乎所有的大数据系统都是部署在Linux上的。

我们今天使用的环境配置是：

- Linux系统：Ubuntu 16.04
- Hadoop：3.3.5版本
- JDK(자바개발키트)：1.8版本
- Spark：3.4.0版本

这些版本都是经过验证的，兼容性很好。

슬라이드 5: 4.1.2 下载安装文件

现在我们来下载Spark安装文件。

大家要访问Spark官网的归档页面。注意，我们下载的是"spark-3.4.0-bin-without-hadoop.tgz"这个文件。为什么选择"without-hadoop"版本呢？因为我们的系统已经安装了Hadoop，所以不需要包含Hadoop的版本。

下载完成后，我们需要解压文件。按照Linux的惯例，用户安装的软件一般都放在"/usr/local/"目录下。

现在请大家跟着我一起执行这些命令：

第一步，解压文件到/usr/local/目录：

sudo tar -zxf ~/Downloads/spark-3.4.0-bin-without-hadoop.tgz -C /usr/local/

第二步，进入目录并重命名：

cd /usr/local

sudo mv ./spark-3.4.0-bin-without-hadoop ./spark

第三步，修改所有权：

sudo chown -R hadoop:hadoop ./spark

这里的hadoop是当前登录的用户名，大家根据自己的情况调整。

슬라이드 6: 4.1.3 配置文件 - spark-env.sh

好，文件解压完成后，我们需要配置一些重要的文件。

首先是spark-env.sh文件。这个文件告诉Spark如何找到Hadoop的相关类库(클래스라이브러리)。

我们先复制模板文件：

cd /usr/local/spark

cp ./conf/spark-env.sh.template ./conf/spark-env.sh

然后用vim编辑器打开文件，在第一行添加这个重要的配置：

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

这行配置的作用是什么呢？它告诉Spark去哪里找Hadoop的类路径(클래스패스)。这样Spark就能正确地与Hadoop协作了。

슬라이드 7: 4.1.3 配置文件 - .bashrc环境变量

接下来我们要配置.bashrc文件。这个文件包含了系统的环境变量(환경변수)。

对于Spark来说，有五个环境变量是必须设置的：

1. SPARK_HOME：告诉系统Spark安装在哪里
2. PYSPARK_PYTHON：告诉Spark用哪个Python解释器(인터프리터)
3. JAVA_HOME：Java安装位置
4. HADOOP_CONF_DIR：Hadoop配置文件位置
5. HADOOP_HOME：Hadoop安装位置

如果这些变量已经设置过了，就不需要重复设置。如果没有设置，我们就需要打开.bashrc文件：

vim ~/.bashrc

슬라이드 8: 4.1.3 .bashrc文件具体配置

现在我们来看具体的配置内容。请大家仔细看这些配置：

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

大家注意几个重点：

- JAVA_HOME要指向你实际的JDK安装目录
- PYSPARK_PYTHON要指向你的Python解释器，这里我们使用的是anaconda环境中的Python
- PATH变量要包含Spark的bin目录，这样我们就可以在任何地方运行Spark命令了

슬라이드 9: 4.1.3 配置文件 - 最终设置

配置完.bashrc文件后，我们需要让这些环境变量生效：

source ~/.bashrc

这个命令很重要，不执行的话，刚才的配置不会生效。

另外，我们还可以配置Spark的日志级别(로그레벨)。默认情况下，Spark会输出很多信息，我们可以设置为只显示错误级别的信息：

cd /usr/local/spark/conf

sudo mv log4j2.properties.template log4j.properties

vim log4j.properties

然后修改这一行：

rootLogger.level = error

这样运行Spark时就不会有太多干扰信息了。

슬라이드 10: 4.1.4 验证Spark安装

好了，配置都完成了。现在我们来验证一下Spark是否安装成功。

Spark安装的一个好处是，配置完成后就可以直接使用，不像Hadoop还需要启动各种服务。

我们可以运行Spark自带的例子程序SparkPi。这个程序用来计算圆周率π的近似值：

cd /usr/local/spark

./bin/run-example SparkPi

运行这个命令后，你会看到很多输出信息。为了快速找到我们要的结果，可以使用grep命令过滤：

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

如果看到类似"Pi is roughly 3.14159570797854"这样的输出，说明Spark安装成功了！

这个结果是不是很接近π的真实值？这就是Spark的强大之处——它可以通过并行计算快速得到结果。

슬라이드 11: 4.2 pyspark使用介绍

现在我们进入第二部分——在pyspark中运行代码。

什么是pyspark呢？pyspark就是Spark的Python接口。通过pyspark，我们可以用Python语言来使用Spark的所有功能。

这部分我们要学习两个内容：

1. pyspark命令的使用方法
2. 如何启动pyspark交互式环境

很多同学可能习惯用Python做数据分析，那么pyspark就是你们的最佳选择。它结合了Python的简洁性和Spark的强大计算能力。

슬라이드 12: 4.2.1 pyspark命令详解

现在我们来详细学习pyspark命令。

pyspark命令的基本格式是：

pyspark --master <master-url>

这里的master-url非常重要，它决定了Spark的运行模式(실행모드)。我给大家详细解释一下这个表格：

- **local**：使用单线程本地运行，不并行
- **local[*]**：使用所有可用的CPU核心
- **local[K]**：使用K个线程，比如local[4]就是使用4个线程
- **spark://HOST:PORT**：连接到Spark集群
- **yarn**：在YARN集群上运行
- **mesos://HOST:PORT**：在Mesos集群上运行
- **k8s://HOST:PORT**：在Kubernetes集群上运行

对于我们学习来说，local模式就足够了。在实际工作中，可能会用到yarn或者spark://模式。

슬라이드 13: 4.2.1 pyspark命令参数

我们重点看一下Local模式的参数：

--master参数：这个参数告诉pyspark要连接到哪个Master。如果是local[*]，就是使用本地模式启动。中括号里的星号表示使用所有可用的CPU核心。你也可以指定具体的数字，比如local[4]表示使用4个核心。

--jars参数：这个参数用来添加JAR包到类路径(글래스패스)中。如果你需要使用额外的Java库，就可以通过这个参数添加。

比如说，如果你的电脑有8个CPU核心，你想用其中的4个来运行Spark，你就可以这样做。这样既能利用多核的优势，又不会占用所有资源影响其他程序。

슬라이드 14: 4.2.1 pyspark命令实例

让我给大家演示几个具体的例子：

如果要在4个CPU核心上运行pyspark：

cd /usr/local/spark

./bin/pyspark --master local[4]

如果要添加外部JAR包：

cd /usr/local/spark

./bin/pyspark --master local[4] --jars code.jar

如果你想了解pyspark的所有参数选项，可以运行：

cd /usr/local/spark

./bin/pyspark --help

这个help命令会显示所有可用的参数，非常有用。建议大家试试看。

슬라이드 15: 4.2.2 启动pyspark

现在我们实际启动pyspark试试看。

在Linux终端中执行pyspark命令，就可以启动Python版本的Spark交互式环境。

如果你已经按照前面的步骤设置了PYSPARK_PYTHON环境变量，就可以直接运行：

cd /usr/local/spark

./bin/pyspark

启动后，你会看到一个漂亮的Spark欢迎界面，显示了版本信息、Python版本等。你还会看到"SparkSession available as 'spark'"这样的提示，这意味着Spark环境已经准备好了。

这个界面显示的信息告诉我们：

- Spark版本是3.4.0
- 使用的Python版本
- Web UI的地址，你可以在浏览器中查看Spark的运行状态
- SparkContext和SparkSession都已经创建好了

슬라이드 16: 4.2.2 pyspark交互式使用

启动pyspark后，默认使用Local[*]模式，也就是使用所有可用的CPU核心。

现在你就可以像使用Python一样输入代码了。我们试试一个简单的计算：

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

看到了吗？立即就得到了结果。这就是交互式环境的好处，可以立即看到执行结果。

在这个环境中，你可以：

- 运行任何Python代码
- 使用Spark的所有功能
- 进行数据分析和处理

当你想退出pyspark时，只需要输入：

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

就可以回到Linux shell了。

슬라이드 17: 4.3 spark-submit命令介绍

现在我们学习第三部分——使用spark-submit命令提交程序。

什么时候用spark-submit呢？当你写好了完整的Spark程序，想要提交到集群运行时，就需要用spark-submit命令。

spark-submit的基本格式是：

```
spark-submit
--master <master-url>  # 运行模式
--deploy-mode <deploy-mode>  # 部署模式
... # 其他参数
<application-file>  # Python代码文件
[application-arguments]  # 传递给程序的参数
```

这个命令功能很强大，可以：

- 提交程序到不同类型的集群
- 设置各种运行参数
- 传递参数给你的程序

同样，你可以用--help查看所有参数：

```
./bin/spark-submit --help
```

슬라이드 18: 4.3 spark-submit实际操作

让我们用一个实际例子来演示spark-submit的使用。

我们运行Spark自带的SparkPi程序。这个程序用蒙特卡洛方法计算π值：

```
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"
```

让我解释一下这个命令：

- --master local[*]：使用本地模式，所有CPU核心
- pi.py：要运行的Python程序
- 10：传递给程序的参数，表示计算精度
- 2>&1 | grep：过滤输出，只显示结果

运行后，你应该看到类似"Pi is roughly 3.14159..."的结果。

这就是spark-submit的威力——可以运行复杂的Spark程序，而不仅仅是简单的交互式命令。

课程总结

好的，今天我们学习了Spark环境搭建的基础部分：

1. **安装Spark Local模式**：包括环境准备、文件下载、配置设置和验证测试
2. **使用pyspark交互式环境**：学会了启动和使用Python版本的Spark
3. **使用spark-submit提交程序**：掌握了如何运行完整的Spark应用程序

这些是使用Spark的基础技能。掌握了这些，你就可以开始进行Spark开发和数据分析了。

下节课我们将学习更高级的内容，包括集群模式的搭建和使用。

有什么问题现在可以提出来，我们一起讨论。

Spark集群环境搭建课堂讲义脚本

课程概览

本节课我们将学习如何搭建Spark集群环境，采用Standalone模式。这是一个非常重要的实践课程，大家要跟着我一步一步操作。

슬라이드1：课程目录介绍

大家好！今天我们要学习的是第4.4节的内容——Spark集群环境搭建(Standalone模式)。

大家先看一下我们今天的学习内容，一共有八个部分：

- 首先，我们会了解集群的整体概况
- 然后搭建Hadoop集群(Hadoop Cluster)，因为Spark需要依赖HDFS
- 接着在所有节点安装Anaconda3
- 再在集群中安装Spark
- 配置相关的环境变量
- 进行Spark的详细配置
- 启动Spark集群
- 最后学会如何关闭集群

这八个步骤环环相扣，缺一不可。大家要认真听讲，有问题随时举手提问。

슬라이드2：集群概况

好，我们先来看看今天要搭建的集群是什么样的。

为了让大家更好地理解，我用三台机器来做演示。实际工作中，你们可能会用到更多的机器。

这三台机器的角色是这样分配的：

- 第一台机器叫hadoop01，IP地址是192.168.91.128，它既是Master节点(Master Node)又是Worker节点(Worker Node)
- 第二台机器叫hadoop02，IP地址是192.168.91.129，只做Worker节点
- 第三台机器叫hadoop03，IP地址是192.168.91.130，也只做Worker节点

所有机器都用hadoop用户登录Linux系统。这个用户名很重要，大家记住了。

这种架构的好处是什么呢？主节点既可以管理集群，又可以参与计算，充分利用了资源。

슬라이드3：为什么要搭建Hadoop集群

有同学可能会问，我们不是要学Spark吗，为什么要先搭建Hadoop集群(Hadoop Cluster)呢？

这是一个很好的问题。让我来解释一下：

Spark作为分布式计算框架(Distributed Computing Framework)，它需要一个分布式文件系统来存储数据。HDFS就是这个文件系统。

你们想想看，如果数据分散在不同的机器上，Spark怎么处理呢？通过HDFS，我们可以：

- 实现数据的分布式存储
- 让Spark读写HDFS中的文件
- 实现"计算向数据靠拢"的原则

什么叫"计算向数据靠拢"呢？就是在数据所在的地方进行计算，这样可以减少网络传输，提高效率。这是分布式计算的一个重要原则。

关于Hadoop集群的详细搭建，大家可以参考第3.2.7节的内容，按照分布式模式来搭建。

슬라이드4：集群架构图

现在大家看这个架构图，这很重要!

在这个图中，你们可以看到:

- 上面是Spark Master(Spark主节点)和HDFS NameNode(HDFS名称节点)
- 下面是三个Slave节点，每个节点都包含:
    - Spark Worker(Spark工作节点)
    - Executor(执行器)
    - HDFS DataNode(HDFS数据节点)

这种设计的精妙之处在于，HDFS的数据节点和Spark的工作节点部署在一起。这样当Spark需要处理数据时，就可以直接在本地读取，不需要通过网络传输大量数据。

这就是我刚才说的"计算向数据靠拢"原则的具体体现。

슬라이드5：安装Anaconda3

好，现在我们开始实际操作。第一步是在所有节点安装Anaconda3。

为什么要安装Anaconda3呢？因为我们要用Python来编写Spark程序，Anaconda3提供了很好的Python环境管理。

具体的安装步骤，大家参考第3.5节的内容。需要完成这些操作：

- 安装Anaconda3
- 设置国内源（这样下载包会快一些）
- 创建pyspark环境

注意！hadoop01节点之前已经安装过了，所以只需要在hadoop02和hadoop03上安装。

安装完成后，在所有节点执行这个命令：

conda activate pyspark

然后测试一下Python环境：

python

如果能正常启动Python，就说明安装成功了。最后用exit()退出。

所有节点都要这样操作，确保Python环境一致。

슬라이드6：在集群中安装Spark

这一步相对简单。因为hadoop01节点在前面Local模式的时候已经安装过Spark了，所以不用重复安装。

但是hadoop02和hadoop03节点还没有Spark，我们等会儿会通过复制的方式来安装。

这里要提醒大家，在实际工作中，你们可能需要在每个节点都重新安装Spark，确保版本一致。

슬라이드7：配置环境变量

现在我们要配置环境变量。这一步非常重要，配置错了后面就启动不了。

在hadoop02和hadoop03节点上，编辑bashrc文件：

vim ~/.bashrc

然后添加这些环境变量：

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

大家注意几个重点：

- JAVA_HOME指定了Java的安装路径
- HADOOP_HOME和SPARK_HOME分别指定了Hadoop和Spark的安装路径
- PYSPARK_PYTHON指定了Python解释器的路径

配置完成后，执行：

source ~/.bashrc

这样配置就立即生效了。

슬라이드8：Spark配置 - workers文件

现在开始配置Spark，这是最关键的部分。

首先配置workers文件。这个文件告诉Spark Master有哪些Worker节点。

在hadoop01节点执行：

cd /usr/local/spark/

sudo mv ./conf/workers.template ./conf/workers

然后编辑workers文件，把默认的localhost替换成：

hadoop01

hadoop02

hadoop03

这样Spark就知道集群中有三个工作节点了。注意，这里用的是主机名，不是IP地址。

슬라이드9：Spark配置 - spark-env.sh文件

接下来配置spark-env.sh文件，这个文件包含了Spark的运行参数。

配置分为三部分：

PART1 - 基础环境配置：

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 - Master节点配置：

export SPARK_MASTER_HOST=hadoop01

export SPARK_MASTER_PORT=7077

export SPARK_MASTER_WEBUI_PORT=8081

注意这些端口号：

- 7077是Spark Master的通信端口
- 8081是Web UI端口

PART3 - 资源配置：

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

这里配置了每个节点的CPU核数和内存大小。在实际环境中，你们需要根据机器配置来调整这些参数。

슬라이드10：创建历史服务器日志目录

现在我们要为历史服务器(History Server)创建日志目录。

历史服务器的作用是什么呢？它可以保存Spark应用的运行历史，方便我们后续查看和分析。

首先启动HDFS：

cd /usr/local/hadoop

./sbin/start-dfs.sh

然后在HDFS中创建日志目录：

./bin/hdfs dfs -mkdir /sparklog

./bin/hdfs dfs -chmod 777 /sparklog

这个/sparklog目录就是用来存放Spark应用日志的。777权限确保所有用户都可以写入。

슬라이드11：配置spark-defaults.conf文件

这个文件配置Spark的默认参数。

首先重命名模板文件：

sudo mv spark-defaults.conf.template spark-defaults.conf

然后添加这些配置：

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

这些配置的含义：

- eventLog.enabled=true：启用事件日志记录
- eventLog.dir：指定日志保存在HDFS的哪个目录
- eventLog.compress=true：压缩日志文件，节省存储空间

슬라이드12：配置Worker节点

现在我们要把Master节点的配置复制到Worker节点。

在hadoop01节点执行：

```
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
```

这里我们先打包，然后用scp命令复制到其他节点。

在hadoop02和hadoop03节点分别执行：

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

注意最后一步chown命令，这是为了确保hadoop用户有权限访问Spark目录。

这样，所有节点的Spark配置就完全一致了。

슬라이드13：启动Spark集群

配置完成后，我们就可以启动集群了。启动顺序很重要!

第1步：启动Hadoop集群

cd /usr/local/hadoop/

sbin/start-all.sh

这会启动HDFS和YARN服务。

第2步：启动历史服务器

cd /usr/local/spark/

./sbin/start-history-server.sh

启动成功后，用jps命令可以看到HistoryServer进程。

为什么要先启动历史服务器呢？因为它需要连接HDFS来读写日志文件。

슬라이드14：启动Master和Worker节点

继续启动Spark的核心组件：

启动Master节点：

cd /usr/local/spark/

sbin/start-master.sh

启动所有Worker节点：

sbin/start-workers.sh

注意，这个命令会自动启动workers文件中配置的所有节点。

启动完成后，用jps命令查看进程：

5409 DataNode

6434 ResourceManager

7156 Master

7380 Worker

7418 Jps

5228 NameNode

7022 HistoryServer

6607 NodeManager

看到Master和Worker进程，说明启动成功了！

슬라이드15：查看集群信息

现在我们可以通过Web界面查看集群状态了。

打开浏览器，访问：http://hadoop01:8081

你们会看到Spark Master的管理界面，显示：

- 集群的基本信息
- 活跃的Worker节点数量
- CPU和内存使用情况
- 运行中的应用程序

这个界面非常有用，可以实时监控集群状态。在实际工作中，运维人员经常通过这个界面来监控Spark集群。

从界面中可以看到我们的3个Worker节点都是ALIVE状态，说明集群搭建成功了！

슬라이드16：关闭Spark集群

学会启动，也要学会关闭。关闭的顺序和启动相反：

第1步：关闭Master节点

cd /usr/local/spark

sbin/stop-master.sh

第2步：关闭Worker节点

sbin/stop-workers.sh

第3步：关闭Hadoop集群

cd /usr/local/hadoop/

sbin/stop-all.sh

为什么要这个顺序呢？因为Worker依赖Master，Spark依赖Hadoop，所以要从上往下关闭。

记住这个原则：启动时从底层到上层，关闭时从上层到底层。

슬라이드17：在集群上运行Spark应用程序

集群搭建好了，现在我们来实际运行一些程序。

我们要学习两个内容：

- 启动Spark集群
- 提交运行程序

这是使用Spark集群的基本操作，也是最重要的操作。

슬라이드18：启动Spark集群（复习）

我们先复习一下启动步骤：

启动Hadoop：

cd /usr/local/hadoop/

sbin/start-all.sh

启动Spark：

cd /usr/local/spark/

sbin/start-master.sh

sbin/start-workers.sh

这个过程大家要熟练掌握，因为每次使用集群都需要这样启动。

슬라이드19：提交运行程序

现在我们运行第一个Spark程序!

我们使用Spark自带的示例程序SparkPi，它的功能是计算π值（圆周率）。

执行命令:

```
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"
```

让我解释一下这个命令:

- spark-submit：提交Spark应用的命令
- --master spark://hadoop01:7077：指定Master节点地址
- pi.py：要运行的Python程序
- 10：程序参数，表示计算精度
- grep "Pi is roughly"：过滤输出，只显示最终结果

运行后，你们应该能看到计算出的π值，大约是3.14159。

슬라이드20：在集群中运行pyspark

除了提交程序，我们还可以交互式地使用Spark。

启动pyspark连接到集群：

cd /usr/local/spark/

bin/pyspark --master spark://hadoop01:7077

然后我们可以交互式地运行Spark代码：

>>> textFile = sc.textFile("hdfs://hadoop01:9000/README.md")

>>> textFile.count()

105

>>> textFile.first()

'# Apache Spark'

这里我们：

- 从HDFS读取文件
- 统计文件行数
- 显示第一行内容

这种交互式的方式非常适合数据探索和调试。

슬라이드21：查看集群运行信息

运行程序时，我们可以通过Web界面实时查看运行状态。

访问 http://hadoop01:8081，你们会看到：

- Running Applications：正在运行的应用
- Completed Applications：已完成的应用
- 每个应用的资源使用情况

这个界面显示：

- 当前有1个应用正在运行（PySparkShell）
- 使用了3个CPU核心
- 使用了3GB内存

这样我们就可以实时监控应用的运行状态。

슬라이드22：历史服务器界面

最后，我们来看看历史服务器的界面。

访问 http://hadoop01:18080，可以看到：

- 所有已完成应用的历史记录
- 每个应用的开始和结束时间
- 应用类型（PythonPi、PySparkShell等）

这个功能在生产环境中非常有用，可以：

- 追踪应用运行历史
- 分析性能问题
- 调试失败的任务

课程总结

好，今天的课程到这里就结束了。我们学习了：

1. **集群规划**：如何设计Spark集群架构
2. **环境准备**：Hadoop、Anaconda3的安装配置
3. **Spark配置**：workers文件、spark-env.sh等关键配置
4. **集群启动**：按正确顺序启动各个组件
5. **程序运行**：使用spark-submit和pyspark运行程序
6. **监控管理**：通过Web界面监控集群状态

大家回去后要多练习，特别是配置文件的编写和集群的启动关闭。下次课我们会学习更复杂的Spark应用开发。

有什么问题吗？没有的话，今天的课就到这里。谢谢大家！

Spark on YARN 模式 - 中文讲课脚本

课程概述

本节课我们将学习 Spark on YARN 模式的部署和使用，包括概念理解、环境配置、两种部署模式的区别，以及 PySpark 库的安装和独立应用程序开发。

슬라이드 1: 课程目录介绍

大家好！今天我们要学习第4.6章的内容，叫做 Spark on YARN 模式(Spark on YARN Mode)。

我们可以看到今天的学习内容分为五个部分：

- 首先我们会了解什么是 Spark on YARN 模式
- 然后学习如何部署这种模式
- 接着我们会实际操作，用 YARN 模式运行 pyspark
- 第四部分我们学习用 spark-submit 命令(spark-submit command)提交程序到 YARN 集群(YARN cluster)
- 最后我们会详细比较 Spark on YARN 的两种部署模式

除了这些，我们还会学习安装 PySpark 库(PySpark library)，以及开发 Spark 独立应用程序。

好，那我们现在开始第一部分。

슬라이드 2: Spark on YARN 模式概述（第一部分）

现在我们来看 Spark on YARN 模式的概述。

同学们，首先我要问大家一个问题：如果我们想要一个稳定的 Spark 生产环境(production environment)，最好的选择是什么呢？

理论上来说，最好的选择是 Standalone HA 模式，也就是 Standalone High Available 模式。但是呢，在实际的企业环境中，我们会遇到一个问题。

这个问题是什么呢？就是服务器资源总是很有限的。许多企业不管做什么业务，基本上都会搭建 Hadoop 集群(Hadoop cluster)，因为 Hadoop 是大数据处理的基础平台。既然搭建了 Hadoop 集群，那么就会有 YARN 集群。

那么问题来了，如果企业已经有了 YARN 集群，再单独准备一个 Standalone 集群，这样做的话，资源利用率就不高了，对不对？所以呢，在企业中，在多数场景下，我们会将 Spark 运行到 YARN 集群中。

슬라이드 3: Spark on YARN 模式概述（第二部分）

继续我们刚才的话题。

YARN 本身是什么呢？YARN 本身是一个资源调度框架(resource scheduling framework)，它负责对运行在内部的计算框架进行资源调度管理。比如说 Spark 就是一个典型的计算框架(computing framework)。

所以 Spark 一般是直接运行在 YARN 中，并且接受 YARN 的资源调度。

这样的话，对于 Spark on YARN 模式而言，我们就不需要部署 Spark 集群了。我们只需要找一台服务器，充当 Spark 客户端(Spark client)，就可以提交任务到 YARN 集群中运行。

在 Spark on YARN 模式中，我们来看看各个角色是怎么分配的：

- Master 角色由 YARN 的 ResourceManager 来担任
- Worker 角色由 YARN 的 NodeManager 来担任
- Driver 角色运行在 YARN 容器(YARN container)中，或者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，大家记一下：

第一，已经安装了 Hadoop 集群，这意味着已经同时安装了 YARN 集群。 第二，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。 第三，需要有被提交的程序。

슬라이드 4: Spark on YARN 模式的部署

好，现在我们来看第二部分，Spark on YARN 模式的部署。

同学们，我要告诉大家一个好消息，Spark on YARN 模式的部署非常简单！

为什么说简单呢？因为我们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。

具体怎么做呢？在 hadoop01 节点上打开 "/usr/local/spark/conf/spark-env.sh" 文件，确保里面要包含这两个环境变量的配置：

第一个是 HADOOP_CONF_DIR，设置为 /usr/local/hadoop/etc/hadoop 第二个是 YARN_CONF_DIR，设置为 /usr/local/hadoop/etc/hadoop

大家注意看，这两个路径是一样的，都指向 Hadoop 的配置目录。

这样配置的目的是什么呢？让 Spark 知道 Hadoop 和 YARN 的配置文件在哪里，这样 Spark 就能和 YARN 正确地通信了。

슬라이드 5: 采用 YARN 模式运行 pyspark

现在我们来实际操作一下，用 YARN 模式运行 pyspark。

首先，我们需要启动 YARN 集群。在 hadoop01 节点上执行这些命令：

$ cd /usr/local/hadoop

$ ./sbin/start-yarn.sh # 启动 YARN 集群

等 YARN 集群启动完成后，我们就可以用 YARN 模式启动 pyspark 了：

$ cd /usr/local/spark

$ ./bin/pyspark --master yarn

大家注意这里的关键参数 --master yarn，这个参数告诉 pyspark 我们要使用 YARN 作为集群管理器(cluster manager)。

启动成功以后，我们可以在浏览器中输入 "http://hadoop01:8088" 打开 YARN 集群管理页面。在这个页面上，我们就可以查看当前运行的 pyspark 的信息。

大家可以看到，在应用程序列表中，我们的 PySparkShell 应用已经在运行了，应用类型显示为 SPARK。

슬라이드 6: YARN 模式下的 pyspark 操作示例

现在我们来做一个简单的测试。

假设 HDFS 的根目录下已经存在一个文件 README.md，我们在 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'
```

大家看，第一行代码我们从 HDFS 中读取文件，注意这里用的是 HDFS 的完整路径。 第二行我们统计文件有多少行，结果是 105 行。 第三行我们获取文件的第一行内容，结果是 "# Apache Spark"。

这个例子说明我们的 Spark on YARN 模式配置成功了，可以正常访问 HDFS 中的数据。

슬라이드 7: 通过 spark-submit 命令提交程序到 YARN 集群

接下来我们学习第四部分，通过 spark-submit 命令提交程序到 YARN 集群。

除了交互式的 pyspark 环境，我们还可以用 spark-submit 命令提交完整的程序到 YARN 集群中运行。

命令格式是这样的：

$ 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"

这里我们用的是 Spark 自带的一个示例程序 pi.py，这个程序用蒙特卡洛方法(Monte Carlo method)计算圆周率。

参数 --master yarn 指定我们使用 YARN 作为集群管理器。 10 是传递给程序的参数，表示迭代次数。 最后的 grep 命令是为了过滤输出，只显示包含 "Pi is roughly" 的行。

执行完上述操作以后，我们可以在 YARN 集群管理器 Web 界面查看应用的运行情况。

슬라이드 8: Spark on YARN 的两种部署模式概述

现在我们来学习非常重要的第五部分，Spark on YARN 的两种部署模式。

同学们，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哪里呢？主要区别在于 Driver 的运行位置。

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

Client 模式(Client Mode)： 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。

这个区别很重要，因为它会影响到应用的性能、稳定性和日志查看方式。

슬라이드 9: 两种部署模式的详细比较

现在我们来详细比较一下这两种模式。我制作了一个对比表，大家看一下：

Driver 运行位置：

- Cluster 模式：在 YARN 容器内部
- Client 模式：在客户端进程内

通讯效率：

- Cluster 模式：高
- Client 模式：低于 Cluster 模式

为什么 Cluster 模式通讯效率更高呢？因为 Driver 和 Executor 都在集群内部，网络通讯距离更短。

日志查看：

- Cluster 模式：日志输出在容器内，不方便查看
- Client 模式：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：

- Cluster 模式：推荐使用
- Client 模式：不推荐使用

稳定性：

- Cluster 模式：稳定
- Client 模式：受到客户端进程影响

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。

슬라이드 10: Client 模式实操演示

现在我们来实际操作一下 Client 模式。

执行以下命令:

```
$ 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"
```

大家注意这里的关键参数 --deploy-mode client，这个参数明确指定我们使用 Client 模式。

如果没有添加 --deploy-mode client 参数，实际上默认也是采用 Client 模式。

上面代码执行结束以后，会在屏幕上直接输出 "Pi is roughly 3.141280"。

这就是 Client 模式的特点，结果直接显示在客户端的终端上，非常方便调试和查看结果。

슬라이드 11: Cluster 模式实操演示

现在我们来试试 Cluster 模式：

$ 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"

大家注意这里的参数变成了 --deploy-mode cluster。

执行这个命令后，大家会发现一个问题：屏幕上不会输出 "Pi is roughly 3.141280"！

为什么呢？因为在 Cluster 模式下，Driver 运行在集群的容器中，程序的输出也在那个容器里，不会显示在我们的客户端终端上。

那我们怎么查看结果呢？需要到 YARN 的 Web 界面中查找日志。

슬라이드 12-14: YARN 集群管理界面操作指导

现在我来教大家如何在 YARN Web 界面中查看 Cluster 模式的运行结果。

第一步： 在浏览器中输入 "http://hadoop01:8088"，会出现图4-7所示的界面。大家可以看到应用程序列表。

第二步： 点击界面中 "ID" 下方的链接，会出现图4-8所示的界面。这里显示了应用的详细信息。

第三步： 点击界面中 "Logs" 下方的链接，会出现图4-9所示的界面。这里列出了各种日志文件。

第四步： 点击界面中 "stdout:....." 链接，会出现图4-10所示的界面。

在最后这个界面里，大家就可以看到我们要找的日志信息 "Pi is roughly 3.138240"。

这个过程虽然麻烦一些，但这就是 Cluster 模式的特点。在生产环境中，我们通常会配置专门的日志收集系统来简化这个过程。

슬라이드 15: PySpark 库安装概述

现在我们进入第4.7章，学习安装 PySpark 库(PySpark library)。

这一章分为两个部分：

- 首先我们要理解库和框架的区别
- 然后学习如何安装 PySpark 库

这个内容很重要，因为很多同学经常搞混 PySpark 库和 Spark 框架的概念。

슬라이드 16: 库与框架的区别详解

现在我来详细解释一下库和框架的区别，这是一个很重要的概念。

什么是库(Library)? 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。

比如说，pandas 就是 Python 的一个第三方库。我们在编写 Python 程序时，可以导入 pandas 库，用它来编写数据清洗程序代码。

什么是框架(Framework)? 框架是可以独立运行并提供编程结构的一种软件产品。

比如说，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。

那么 PySpark 是什么呢? PySpark 就是一个 Python 的库，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 库和 "bin/pyspark" 是完全不同的两个概念：

- "bin/pyspark" 是一个客户端程序，不是库
- 它提供了交互式的 Python 客户端用于编写基于 Spark API 的程序

슬라이드 17: PySpark 库与 Spark 框架对比表

我们来看一个详细的对比表：

底层语言：

- PySpark：Python
- Spark：Scala

上层语言支持：

- PySpark：仅 Python
- Spark：Python/Java/Scala/R

集群化分布式运行：

- PySpark：不支持，仅支持单机
- Spark：支持

定位：

- PySpark：Python 库（客户端）
- Spark：标准框架（客户端和服务端）

是否可以以守护进程方式运行：

- PySpark：不可以
- Spark：可以

应用场景：

- PySpark：本地开发调试 Python 程序
- Spark：生产环境集群化运行

这个对比帮助大家理解什么时候用 PySpark 库，什么时候用 Spark 框架。

슬라이드 18: PySpark 库的安装步骤

现在我们来实际安装 PySpark 库。

第一步： 在 hadoop01 节点中打开一个终端，执行如下命令激活环境：

$ conda activate pyspark

第二步： 执行如下命令安装 PySpark 库：

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

这里我们指定了版本号 3.4.0，确保和我们的 Spark 框架版本一致。使用清华大学的镜像源可以加快下载速度。

第三步： 安装结束后，进入 Python 环境，执行 "import pyspark" 命令测试是否成功。

如果没有报错，就表示安装成功了。

第四步： 在 hadoop02 和 hadoop03 节点上也分别执行上面同样的操作。

这样我们就在所有节点上都安装了 PySpark 库。

슬라이드 19: Spark 独立应用程序开发概述

现在我们进入第4.8章，学习开发 Spark 独立应用程序。

这一章包括两个部分：

- 编写程序
- 通过 spark-submit 运行程序

这是一个非常实用的技能，因为在实际工作中，我们经常需要开发独立的 Spark 应用程序。

슬라이드 20: 编写 Spark 程序

现在我们来学习如何编写 Spark 程序。

首先我要告诉大家一个好消息：使用 Python 进行 Spark 编程比使用 Java 和 Scala 要简单得多！

为什么呢？

- 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

现在我们来创建一个代码文件 "/usr/local/spark/mycode/python/WordCount.py"，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()
```

让我解释一下这个程序的逻辑：

- 第1行：导入必要的 Spark 类
- 第2行：创建 Spark 配置，使用本地模式
- 第3行：创建 SparkContext
- 第4行：指定要处理的文件
- 第5行：读取文件并缓存

- 第6-7行：分别统计包含字母 'a' 和 'b' 的行数
- 第8行：打印结果
- 第9行：停止 SparkContext

슬라이드 21: 直接运行 Python 程序

这个 Python 代码，我们可以在 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

这说明在 README.md 文件中，包含字母 'a' 的行有72行，包含字母 'b' 的行有39行。

这种方式适合开发和测试阶段，可以快速验证程序的正确性。

슬라이드 22: spark-submit 命令详解

现在我们学习用 spark-submit 命令提交应用程序，这是生产环境中最常用的方式。

spark-submit 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

主要参数说明：

- --master：指定集群管理器，可以是 local、spark://、yarn 等
- --deploy-mode：部署模式，可以是 client 或 cluster
- <application-file>：要执行的 Python 文件
- [application-arguments]：传递给程序的参数

我们可以执行 "spark-submit --help" 命令，获取完整的选项列表。

슬라이드 23: 使用 spark-submit 提交程序

最后，我们来实际使用 spark-submit 提交程序。

假设我们采用了 Standalone 模式的集群，对于前面的 WordCount.py，可以通过 spark-submit 提交到 Spark 中运行：

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

这里：

- --master spark://hadoop01:7077 指定我们使用 Standalone 模式的集群
- 最后是我们的 Python 程序文件路径

执行该命令以后，可以得到如下结果：

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

结果和直接运行 Python 程序是一样的，但这次程序是在 Spark 集群上运行的。

课程总结

好，今天的课程内容就到这里。我们学习了：

1. **Spark on YARN 模式的概念和优势**
2. **如何配置和部署 Spark on YARN 模式**
3. **Client 模式和 Cluster 模式的区别和使用场景**
4. **PySpark 库的安装和使用**
5. **如何开发和提交 Spark 独立应用程序**

这些知识点在实际工作中都非常重要。建议大家课后多练习，特别是两种部署模式的使用，这样才能真正掌握 Spark on YARN 的精髓。

有什么问题可以随时问我。下节课我们将学习更高级的 Spark 编程技巧。

谢谢大家！

PyCharm安装和使用 - Spark程序开发讲义脚本

第4.9节 PyCharm安装和使用

슬라이드1: 课程介绍

讲师说：

同学们好！今天我们来学习第4.9节的内容——PyCharm的安装和使用。这是一个非常实用的内容，因为PyCharm是我们开发Spark(Apache Spark)程序最常用的集成开发环境(IDE)之一。

今天的内容主要分为两个部分：

- 首先是4.9.1，我们会学习如何安装PyCharm
- 然后是4.9.2，我们会学习如何使用PyCharm来开发Spark程序

为什么要学习PyCharm呢？虽然我们之前学过用命令行来写Spark程序，但是在实际开发中，使用专业的IDE会让我们的效率大大提升。好，我们开始吧！

슬라이드2: PyCharm下载和安装

讲师说：

首先我们来看PyCharm的安装过程。大家注意看屏幕上的这些命令。

首先，我们需要到PyCharm的官网去下载安装包。这个网址是https://www.jetbrains.com/pycharm/download。我们下载的是社区版(Community Edition)，它是免费的，对我们学习来说完全够用了。

下载完成后，我们得到一个压缩包文件，文件名是pycharm-community-2022.2.3.tar.gz。

现在我来演示安装步骤：

第一步，进入下载目录：

cd ~/Downloads

第二步，解压缩到/usr/local目录：

sudo tar -zxvf pycharm-community-2022.2.3.tar.gz -C /usr/local

第三步，重命名文件夹，这样比较方便：

cd /usr/local

sudo mv pycharm-community-2022.2.3 pycharm

第四步，修改权限，让hadoop用户可以使用：

sudo chown -R hadoop pycharm

最后，启动PyCharm：

cd /usr/local/pycharm

./bin/pycharm.sh

大家看，这样就启动了PyCharm。第一次启动可能会比较慢，大家耐心等待一下。

슬라이드3: PyCharm初始配置

讲师说：

好，PyCharm启动后，我们看到了欢迎界面。现在我们需要做一些基本的配置。

大家看到屏幕上这个界面了吗？左边有几个选项，我们点击"Customize"。

在右边的设置里，我们可以看到"Color theme"，也就是颜色主题。这里我建议大家选择"IntelliJ Light"，这是一个比较清爽的浅色主题，适合长时间编程使用。当然，如果你喜欢深色主题，也可以选择其他的。

这个设置主要是为了让我们的开发环境看起来更舒服。好的开发体验从选择一个喜欢的主题开始！

슬라이드4: 创建新项目

讲师说：

配置好主题后，我们来创建第一个项目。大家看，我现在点击左边的"Projects"选项。

在右边的界面上，我们可以看到几个选择：

- "New Project" - 创建新项目
- "Open" - 打开现有项目
- "Get from VCS" - 从版本控制系统获取

我们要创建一个新的Spark项目，所以点击"New Project"这个蓝色的按钮。

这里还有一个"Take a quick onboarding tour"，如果大家是第一次使用PyCharm，可以点击这个来快速了解界面。不过今天时间有限，我们直接进入项目创建。

슬라이드5: 项目配置设置

讲师说：

现在我们进入了项目配置页面。这个页面很重要，我们需要仔细设置。

首先看"Location"这一栏，这里是设置项目的保存路径和名称。我建议大家设置为：

/home/hadoop/PycharmProjects/PySpark

这样设置的好处是：

- PycharmProjects是PyCharm的默认项目目录
- PySpark这个名字一看就知道是Spark项目

接下来是关键的部分 - Python解释器(Python Interpreter)的设置。大家看到这里有几个选项：

- "New environment using" - 创建新环境
- "Previously configured interpreter" - 使用之前配置的解释器

我们选择"Previously configured interpreter"，因为我们要使用之前安装好的pyspark环境。

然后点击"Add interpreter"，再选择"Add Local Interpreter"。

슬라이드6: Conda环境配置

讲师说：

好，现在我们进入了添加Python解释器的界面。这个界面左边有几个选项：

- Virtualenv Environment
- Conda Environment
- System Interpreter
- Pipenv Environment
- Poetry Environment

我们要选择"Conda Environment"，因为我们之前是用Anaconda安装的PySpark环境。

选择Conda Environment后，右边会显示一些配置选项。我们需要点击右边的"..."按钮来浏览文件系统，找到我们的Python解释器。

슬라이드7: 选择Python解释器路径

讲师说：

现在我们看到了文件浏览器。我们需要找到之前创建的pyspark环境中的Python解释器。

路径是：

/home/hadoop/anaconda3/envs/pyspark/bin/python

大家看屏幕上的文件列表，我们可以看到各种文件：

- lzmainfo, lzmore - 这些是压缩工具
- openssl - SSL工具
- pip, pip3 - Python包管理工具
- pydoc, pydoc3 - Python文档工具
- pyspark - 这就是我们的Spark命令
- python - 这就是我们要选择的Python解释器

选择python文件，然后点击"OK"。

슬라이드8: 全局设置选项

讲师说：

很好！现在我们回到了解释器配置界面。大家注意看，这里有一个很重要的选项："Make available to all projects"。

我强烈建议大家勾选这个选项。为什么呢？因为这样设置后，以后我们创建其他Spark项目时，就不需要重新配置解释器了，直接就能使用这个pyspark环境。

这是一个很实用的功能，可以为我们节省很多时间。勾选后，点击"OK"。

슬라이드9: 完成项目创建

讲师说：

现在我们回到了项目创建的最后一步。大家看屏幕上的配置总结：

- Location: /home/hadoop/PycharmProjects/PySpark
- Python interpreter: Python 3.8 (pyspark)

这里显示"Python 3.8 (pyspark)"说明我们的配置是正确的。

下面还有一个选项"Create a main.py welcome script"，这个默认是勾选的，意思是创建一个欢迎脚本。我们保持勾选状态，这样PyCharm会为我们创建一个示例文件。

一切准备就绪，点击"Create"按钮！

슬라이드10: 项目界面介绍

讲师说：

太好了！我们的项目创建成功了。现在大家看到的就是PyCharm的主界面。

让我来介绍一下这个界面的主要部分：

- 左边是项目树(Project Tree)，显示我们项目中的所有文件
- 中间是代码编辑区域，现在显示的是main.py文件
- 下面是控制台区域，用来显示程序运行结果

PyCharm已经为我们创建了一个main.py文件，里面有一些示例代码。我们可以右键点击代码区域，在弹出的菜单中选择"Run 'main'"来运行这个程序。

这样我们就可以测试Python解释器是否工作正常。

슬라이드11: 测试运行结果

讲师说：

很好！大家看到下面的控制台输出了"Hi, PyCharm"。这说明什么？说明我们的Python解释器配置完全正确！

同时我们还可以看到一些运行信息：

- 使用的Python解释器路径
- 程序执行时间
- 进程结束状态

这些信息告诉我们，我们的开发环境已经搭建成功了。如果大家看到类似的输出，就说明配置没有问题。

슬라이드12: 检查解释器设置

讲师说：

现在我们来验证一下我们的Spark环境是否正确配置。大家看屏幕下方的状态栏，有一个"Python3.8(pyspark)"的显示。

我们点击这个显示，会弹出一个菜单。选择"Interpreter Settings..."，这样我们就可以查看Python解释器的详细信息。

这是一个很好的习惯 - 在开始开发之前，先确认我们的开发环境配置是否正确。

슬라이드13: 验证PySpark安装

讲师说：

太棒了！现在我们看到了解释器设置界面。在左边选择"Python Interpreter"，右边就会显示所有已安装的Python包。

大家注意看右边的包列表，我们可以看到：

- 各种系统包，比如pip、setuptools等
- 最重要的是，我们看到了"pyspark"包，版本是3.4.0

看到pyspark包的存在，说明我们的PySpark库已经成功安装并且可以使用了。这意味着我们可以开始开发Spark程序了！

如果大家在自己的环境中没有看到pyspark包，说明安装有问题，需要重新安装PySpark。

第4.9.2节 使用PyCharm开发Spark程序

슬라이드14: 创建Spark程序文件

讲师说：

好，现在我们的开发环境已经配置完成了，接下来我们来创建第一个Spark程序。

我们要创建一个经典的WordCount程序。在项目名称"PySpark"上右键点击，会弹出一个菜单。我们选择"New"，然后在子菜单中选择"Python File"。

这里有多种文件类型可以选择：

- Python File - 普通Python文件
- Python unit test - Python单元测试文件
- Python stub - Python存根文件
- Directory - 目录
- HTML File - HTML文件

我们选择"Python File"。

슬라이드15: 命名Python文件

讲师说：

现在系统询问我们文件名。我们输入"WordCount"。

注意，我们不需要输入".py"扩展名，PyCharm会自动为我们添加。

输入完成后按回车键，PyCharm就会为我们创建一个名为WordCount.py的文件。

WordCount是Spark的经典入门程序，就像其他编程语言的"Hello World"一样。通过这个程序，我们可以学习Spark的基本概念和操作。

슬라이드16: 编写WordCount代码

讲师说：

现在我们来编写WordCount程序的代码。我会一行一行地解释：

coding:utf8

这一行指定文件编码为UTF-8，这样我们可以在代码中使用中文注释。

from pyspark import SparkConf, SparkContext

这里我们导入了两个重要的类：

- SparkConf：用于配置Spark应用
- SparkContext：Spark的入口点

if __name__ == '__main__':

这是Python的标准写法，确保代码只在直接运行时执行。

conf = SparkConf().setMaster("local[*]").setAppName("My App")

创建Spark配置：

- local[*]：表示在本地模式运行，使用所有可用的CPU核心
- "My App"：应用程序的名称

sc = SparkContext(conf = conf)

创建SparkContext，这是我们与Spark集群交互的入口。

logFile = "file:///usr/local/spark/README.md"

logData = sc.textFile(logFile, 2).cache()

读取文件并创建RDD(弹性分布式数据集)，cache()表示将数据缓存在内存中。

numAs = logData.filter(lambda line: 'a' in line).count()

numBs = logData.filter(lambda line: 'b' in line).count()

这两行代码统计包含字母'a'和'b'的行数。

print('Lines with a: %s, Lines with b: %s' % (numAs, numBs))

输出统计结果。

sc.stop()

最后要记得停止SparkContext，释放资源。

슬라이드17: 启动集群服务

讲师说：

在运行我们的Spark程序之前，我们需要先启动Hadoop和Spark集群。

为了节省系统资源和提高实验速度，虽然我们之前配置了3台虚拟机，但今天我们只启动hadoop01这一台就够了。

首先启动Hadoop集群，在hadoop01上执行：

cd /usr/local/hadoop

./sbin/start-dfs.sh

注意，这里我们只启动了HDFS（Hadoop分布式文件系统），没有启动YARN资源管理器。

然后启动Spark集群：

cd /usr/local/spark

./sbin/start-all.sh

这个命令会同时启动Spark的Master和Worker进程。

等一下，我们可以通过jps命令来检查服务是否启动成功。

슬라이드18: 运行Spark程序

讲师说：

好，现在我们的集群已经启动了，可以运行我们的WordCount程序了。

在PyCharm中，我们可以直接右键点击WordCount.py文件，选择"Run 'WordCount'"，或者点击编辑器上方的绿色运行按钮。

大家看控制台的输出，我们可以看到：

1. 首先是一些Spark的启动日志
2. 然后是我们程序的运行结果："Lines with a: 72, Lines with b: 39"
3. 最后显示"Process finished with exit code 0"，表示程序正常结束

这个结果告诉我们，在Spark的README.md文件中，包含字母'a'的行有72行，包含字母'b'的行有39行。

슬라이드19: 读取HDFS文件

讲师说：

刚才我们的程序读取的是本地文件系统的文件。在实际的生产环境中，我们更多时候需要读取HDFS上的文件。

如果要读取HDFS文件，我们只需要修改一行代码：

将：

logFile = "file:///usr/local/spark/README.md"

改为：

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

不过要注意，在修改之前，我们需要先把本地的README.md文件上传到HDFS中：

hdfs dfs -put /usr/local/spark/README.md /

这样就可以从HDFS读取文件了。这种方式在处理大数据时非常有用，因为HDFS可以存储TB甚至PB级别的数据。

슬라이드20: 本章总结

讲师说：

好，今天的课程就要结束了。让我来总结一下我们学到的内容：

首先，我们学习了Spark的多种部署模式。在学习和开发阶段，单机模式是最方便的选择。

我们详细学习了：

1. **Spark单机环境的安装配置** - 这是入门的基础
2. **pyspark交互式编程环境** - 这是快速测试和学习的好工具
3. **Spark集群环境的搭建** - 这是生产环境的基础
4. **Spark on YARN模式** - 这是企业中最常用的部署方式
5. **PySpark库的安装和使用** - 这是Python开发者的必备技能
6. **PyCharm开发工具的使用** - 这可以大大提高我们的开发效率

通过今天的学习，大家应该已经掌握了：

- 如何安装和配置PyCharm开发环境
- 如何创建和运行Spark程序
- 如何在本地和集群环境中运行Spark应用

这些技能将为大家后续的大数据学习和开发打下坚实的基础。

课后作业：

1. 请大家回去后自己安装配置PyCharm环境
2. 尝试修改WordCount程序，统计其他字符的出现次数
3. 预习下一章节的内容

今天的课就到这里，大家有什么问题吗？

讲师结束语：

好的，看来大家都理解了。记住，工具只是手段，重要的是理解Spark的核心概念。PyCharm可以让我们更高效地开发，但我们也要理解底层的工作原理。

下课！