学习情境一　搭建 Spark 开发环境

Spark 起源于加利福尼亚大学伯克利分校的一个研究项目。学校当时关注分布式机器学习算法的应用情况。因此，Spark 从一开始便为应对迭代式应用的高性能需求而设计。在这类应用中，相同的数据会被多次访问。该设计主要靠利用数据集内存缓存以及启动任务时的低延迟和低系统开销来实现高性能。再加上其容错性、灵活的分布式数据结构和强大的函数式编程接口，Spark 在各类基于机器学习和迭代分析的大规模数据处理任务上有广泛的应用，这也表明了其实用性。

搭建 Spark 环境是开展 Spark 学习的基础。作为一种分布式处理框架，Spark 可以部署在集群中运行，也可以部署在单机上运行。由于 Spark 是一种计算框架，不负责数据的存储和管理，因此，通常需要把 Spark 和 Hadoop 进行统一部署，由 Hadoop 中的 HDFS 和 HBase 等组件负责数据的存储，由 Spark 负责完成计算。

通过本学习情境的学习，可以掌握的知识有：读者理解 Spark 的用途、生态圈、应用场景等基本概念，以及搭建 Spark 开发环境。

1.1　典型工作环节 1：需求分析

小李正在学习大数据，想对职业能力分析大数据服务平台的职位数据进行统计分析，但对使用什么工具来统计分析大数据比较茫然，因此小李打算对大数据处理工具 Spark 进行调研，深入了解 Spark 的相关概念、应用场景等。

小李在调研后，准备使用 Spark 进行大数据开发，因此小李需要搭建一套大数据开发环境，主要包括系统的选择、准备与初始化，软件工具的下载安装与配置等。

1.2　典型工作环节 2：步骤分析

根据需求分析，本节内容主要从认识 Spark、Spark 应用场景、Spark 开发环境搭建进行学习。

第一步：认识 Spark

收集 Spark 资料，对 Spark 有整体认识，了解 Spark 的概念、生态圈以及与 Hadoop 的关系、优缺点等内容。

第二步：熟悉 Spark 应用场景

对 Spark 有了了解后，对 Spark 的用途、适合处理的数据、应用场景等进行归纳。

第三步：准备集群环境

Spark 部署模式主要有四种：Local 模式（单机模式）、Standalone 模式（使用 Spark 自带的简单集群管理器）、YARN 模式（使用 YARN 作为集群管理器）和 Mesos 模式（使用 Mesos 作为集群管理器）。本教材使用集群模式的 Spark 安装。

根据官网资料，Spark 提供了 Windows 版本，这个版本是做调试程序使用的。在实际生产环境中，需要将 Spark 部署到 Linux 系统上。Linux 系统有多个版本，比如 CentOS，RHEL，Ubuntu，Debian，Fedora，Mint，ArchLinux，Gentoo，Kali，Parrot Security OS，Deepinlinux，Manjaro。由于每个版本的适用场景不一样，并且鉴于目前互联网行业用得最多的系统是 CentOS，因此这里将选择 CentOS7.6 作为 Spark 的基础运行环境。

第四步：搭建 Spark 开发环境

由于大数据的各组件，比如 Hadoop、Hive、Hbase、Kafka 等，都是由不同组织机构开发的，存在版本不兼容的情况。因此在实际生产环境中，都不建议升级到最新版本，而是建议使用业内成熟的版本，本教材将使用 spark-2.1.1-bin-hadoop2.7.tgz 版本。

在基础设施建立完毕后，根据需要在单机模式、伪分布式模式、集群模式上搭建并运行 Spark 应用。

1.3 典型工作环节 3：认识 Spark

本环节主要通过与 Hadoop 比较，介绍 Spark 的用途和特点以及 Spark 的生态环境。能够让学习者对 Spark 形成形象的认识。

1.3.1 Spark 概述

Spark 是一个快速的通用的集群计算平台。

快速：是相对而言的，比如比 Hadoop 快。因为 Hadoop 的 Map 任务结束后将结果输出到磁盘或者 HDFS，Reduce 任务从 HDFS 获取结果，计算完后又放到 HDFS，这个过程就会牵涉到磁盘 I/O，若 Map 任务和 Reduce 任务个数特别多，那么 I/O 次数就会特别多，对性能的影响就特别大。若在 MapReduce 任务执行过程中有 Shuffle 过程，Shuffle 也需要 I/O，同时，Shuffle 过程也会伴随数据迁移，也会耗费时间。若出现迭代计算，后一个 Map 任务需要依赖前一个 Map 或 Reduce 的计算结果，那么 I/O 就会更多。Spark 相对而言，是把 MapReduce 计算过程需要的数据尽量放到内存，当达到一定阈值才往磁盘写；另外 Spark 计算引擎会根据 RDD 的依赖关系，生成 DAG（有向无环图），由于对 RDD 的计算是惰性的，在 Spark 实际执行任务的时候，才会去加载数据，这样就能做到数据的按需加载。因此，从这两个方面来讲，Spark 的性能远高于 Hadoop，如图 1-1 所示，某些情况下，Spark 甚至比 Hadoop 快出 100 倍。

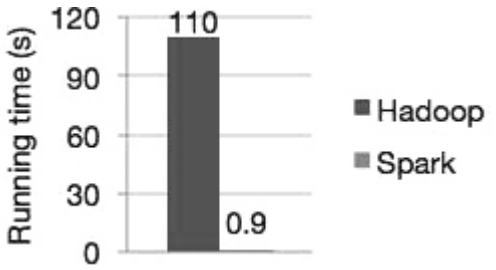


图 1-1　Spark 与 Hadoop 性能比较

易用：Spark 支持使用 Java，Scala，Python，R 和 SQL 快速编写应用程序。Spark 提供 80 多个高级接口，可以轻松构建并行应用程序。用户还可以从 Scala，Python，R 和 SQL shell 中以交互方式使用 Spark。

通用：Spark 拥有很多库，包括 SQL 和 DataFrames，用于机器学习的 MLlib，GraphX 和 Spark Streaming（图 1-2）。用户可以在同一个应用程序中无缝地组合这些库。

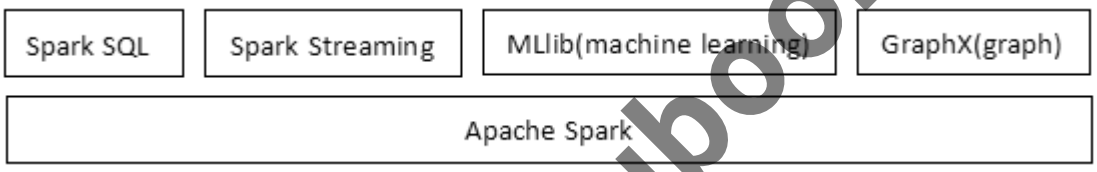


图 1-2　Spark 库

随处运行：用户可以使用其独立群集模式，EC2，Hadoop YARN，Mesos 或 Kubernetes 运行 Spark 应用。同时 Spark 还可以访问各种数据源，比如从 HDF，Alluxio，Apache Cassandra，Apache Hbase，Apache Hive 以及数百个其他数据源中获取数据。

Apache Spark 由来自 300 多家公司的众多开发人员构建。自 2009 年以来，已有超过 1200 名开发人员为 Spark 做出了贡献。活跃的社区使 Spark 的应用与发展得到保障。

1.3.2　Spark 生态圈

Spark 生态圈也称为伯克利数据分析技术栈。包含 Spark Core，Spark SQL，Spark Streaming，Mllib/ML，GraphX，SparkR，BlinkDB，资源调度器等，这里简要介绍如下。

- Spark Core：实现内存管理、任务调度、DAG 解析等功能，还提供了大量的基础 API。
- Spark SQL：操作结构化数据。比如操作数据库表，json，csv 等有格式的数据。
- Spark Streaming：操作流式数据，实现实时分析。
- Mllib/ML：实现了一些机器学习算法，比如回归分析、聚类分析。
- GraphX：用于图计算或网格计算。
- SparkR：适用于 R 语言的数据分析。
- BlinkDB：交互式的 SQL 查询引擎，用来减少查询响应时间。
- 资源调度器：Spark 本身自带资源调度器。
- YARN：Hadoop 的核心组件之一，用来调度任务。
- Mesos：资源调度器。与 YARN 不同的是：资源需要多少给多少，任务运行资源过多，Mesos 可以进行收回，是一种细粒度的调度框架；YARN 是按额度分配资源，不管任务是否需要这么多，是一种粗粒度的调度框架。

1.4 典型工作环节 4:调研 Spark 应用场景

Spark 应用场景非常广。在家电销售平台、电影售票平台、矿产开发、舆情分析、交通预警、医疗康养、金融欺诈检测等领域都有 Spark 的身影,这里简要介绍如下。

- 家电销售平台:目前大多数电商销售平台都在使用 Spark 做实时数据计算。通过 Spark Streaming 流式处理引擎,实时处理 Kafka 传递过来的数据,实现对库存的实时管控、物流的实时调度。
- 电影售票平台:使用 Spark 做离线分析,预测电影的热度。同时对大量用户数据分析,为用户个性化推荐电影。
- 矿产开发:通过对大数据的分析,根据矿山岩层结构、各化学元素含量,来分析矿山的开发价值。
- 舆情分析:利用 Spark 的 GraphX 来构建用户关联网络,从而实现网络舆情监控。

1.5 典型工作环节 5:准备集群系统

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。虽然 Spark 和 Hadoop 都可以安装在 Windows 系统中使用,但是,建议在 Linux 系统中安装和使用。

为了后续能够流畅部署 Spark 环境,需要虚拟机软硬件环境满足 Spark 环境的需求。

1.5.1 环境需求

要搭建 Spark 开发环境,需要预先安装虚拟机,可以单节点搭建伪分布式开发环境,也可以多节点搭建完全分布式环境。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(主节点),另外两台机器(节点)作为 Slave 节点(即作为 Worker 节点),主机名分别为 Slave1 和 Slave2。

学习测试,通常选择 3 个节点进行搭建。表 1-1 列出了虚拟机核心配置项。

表 1-1 虚拟机核心配置项

配置项	主节点	节点 1	节点 2
内存	4GB	2GB	2GB
CPU 内核数	2	1	1
硬盘	50GB	50GB	50GB
网络适配器	NAT	NAT	NAT
主机名	Master	Slave1	Slave2

1.5.2　配置虚拟机系统

配置虚拟机之前，需要最小化安装系统，然后进行安装前配置。

1. 禁用 SELINUX

分布式框架在进行端口通信时，SELINUX 可能会阻塞其中的通信，因此暂且将其禁用，在 CentOS7 中，使用如下命令编辑 SELINUX 配置文件：

```
永久关闭
# vi /etc/selinux/config
SELINUX=disable     #将值设置为 disable

临时关闭
setenforce 0
getenforce    #查询状态，值为 0 或 disable 则为关闭
```

将 SELINUX=enforcing 修改为 SELINUX=disable 后，保存退出。

2. 关闭防火墙

防火墙也可能导致分布式框架之间通信被拒绝，因此暂且将其关闭。

在 CentOS7 中，使用如下命令关闭防火墙：

```
systemctl stop firewalld        #停止服务
systemctl disable firewalld     #禁止开机启动
```

3. 配置网卡文件

可根据需要自动获取 IP 地址或手动配置 IP 地址。

```
vi /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static      #如果自动获取将此值改为 dhcp，删除后面的 IP 地址配置
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=b750518d-2b37-49b5-8a02-992b45837752
DEVICE=ens33
IPADDR=192.168.47.20
NETMASK=255.255.255.0
GATEWAY=192.168.47.2    #可通过 VMware 虚拟网络编辑器查看
DNS1=192.168.47.2
DNS2=114.114.114.114
ONBOOT=yes
```

在 CentOS7 中配置网卡命令如下：

编辑完成后，保存退出，重启网卡，命令如下：

```
###重启网络###
systemctl restart network
```

(4)修改 IP 地址与主机映射关系

在 CentOS7 中，添加主机映射关系命令如下：

```
vi /etc/hosts
192.168.47.20  master
192.168.47.21 slave1
192.168.47.22 slave2
```

编辑完成后，保存退出。

1.5.3 密钥配置

1. 安装 Putty

Putty 是一款免费开源的远程登录工具，提供 Telnet，SSH，rlogin 以及串口通信等功能，目前 Putty 已被移植到了多种平台。使用 Putty 可以节省操作 Linux 系统的时间。

下载 Putty 安装文件。

Putty 提供了可安装程序版本，还可直接下载 zip 版本，zip 版本不用安装，解压后即可使用。目前互联网上有很多站点提供不同版本下载，建议直接从官方网站下载最新版即可，官方网站为 https://Putty.org/，需要注意的是，请下载与操作系统对应的 32 位或 64 位版本。

下载安装完成或解压 zip 版本文件后可看到有以下文件：

PAGEANT.EXE	用于 SSH 认证代理，可以不必每次都输入密码。
PLINK.EXE	在命令行上运行，用于远程执行服务器上的命令。
PSCP.EXE	在命令行上运行，使用 SSH 传输文件。
PSFTP.EXE	命令行工具，类似于 ftp，使用 22 号端口向服务器传输文件。
PUTTY.CHM	帮助文件。
PUTTY.EXE	ssh/telnet/rlogin/serial 客户端，可以命令行或 GUI 模式运行。
PUTTYGEN.EXE	RSA 和 DSA 密钥生成和管理工具。

接下来设置 Putty 使用环境。

步骤 1：如图 1-3 所示，运行 Putty.exe 程序，选择默认配置，点击 Load 加载。

步骤 2：如图 1-4 所示，选择语言。

此处根据自己喜好，选择一个中文字体以及字形、字体大小，最后一定请在脚本处选择中文 GB2312 编码。

步骤 3：如图 1-5 所示，设置远端服务器的字符集环境，这里请一定选择与服务器语言环境相同的字符集编码，否则中文会显示乱码，Linux 系统默认情况下使用 UTF-8 编码。

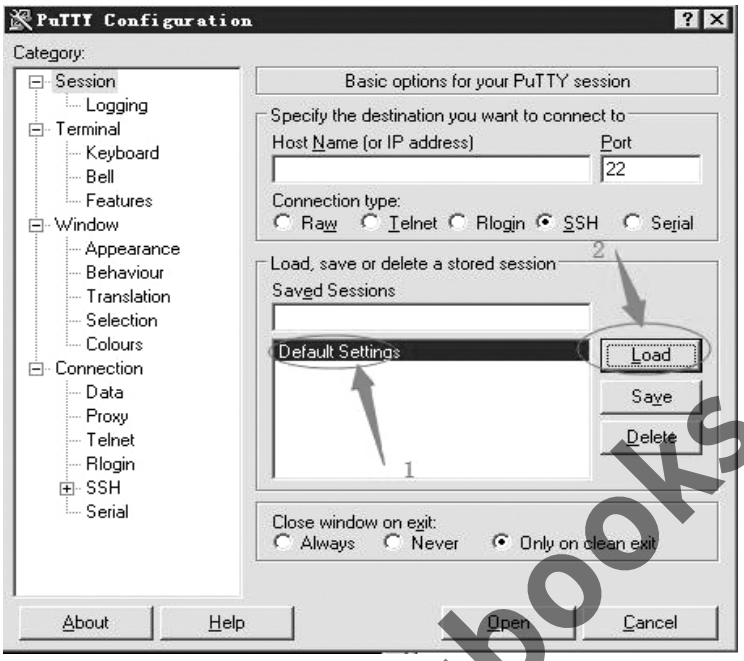


图 1-3　修改 Putty 默认设置

图 1-4　中文支持设置

步骤 4：如图 1-6 所示，如果你对 Putty 默认的黑底白字的颜色不满意，可以点击 Colours 修改前景文字和背景颜色。接下来返回到 Session 页面点击 Save，将刚才的设置保存为默认值。

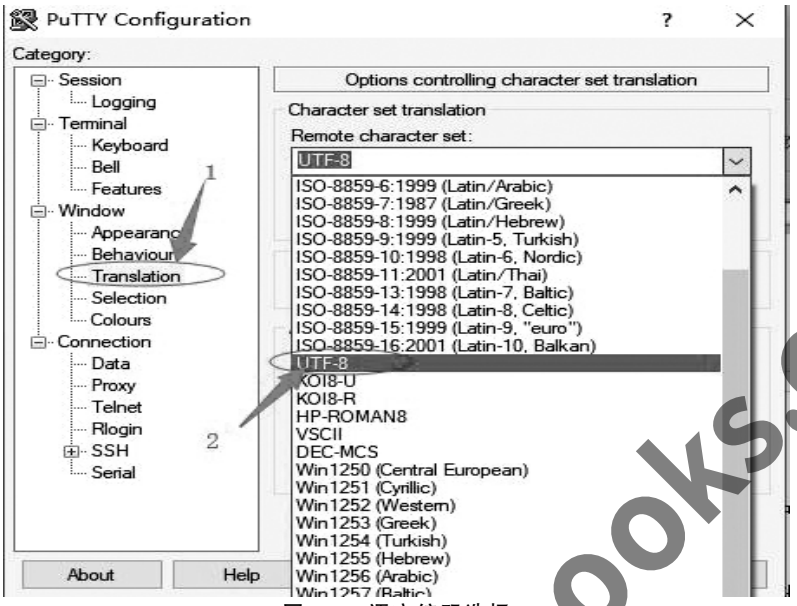


图 1-5　语言编码选择

默认配置保存完成后，可以配置服务器连接设置，在 Session 页面将服务器 IP 或域名输入，选择 SSH 协议，端口会使用默认值 22，如果你有 SSH 服务器不使用默认值 22 作为端口，请根据实际情况修改。假如你是偶尔使用服务器进行连接访问，直接点下方的 Open 即可，但如果你以后会经常性访问，可将此配置保存为一个友好名称，以后双击此名称即可进行连接。

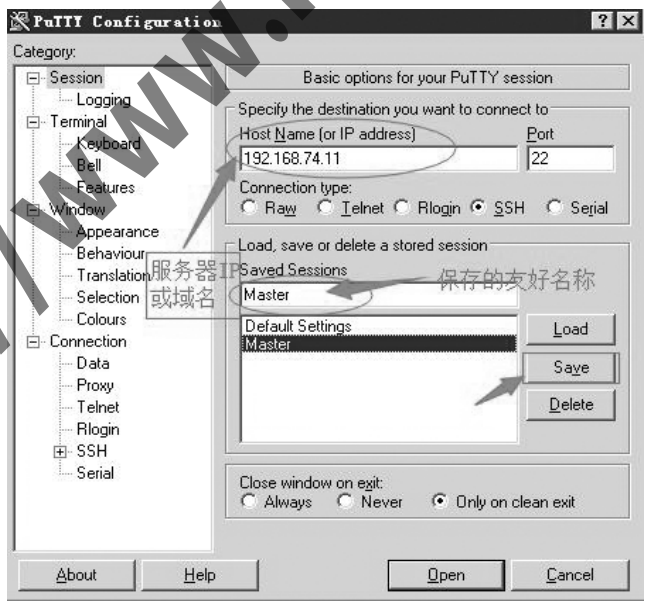


图 1-6　创建新会话连接

步骤 5：如图 1-7 所示，使用用户名密码方式登录。

点击 Open，如果与服务器间网络通信正常，出现提示 login as：时输入用户名回车，再输入账号密码即可登录系统。

图 1-7　Putty 访问服务器

步骤 6：使用证书（公钥私钥）方式登录访问系统

虽然使用 SSH 方式访问服务器，在网络中传输的数据是加密的，相对比较安全，如果使用账号密码方式访问服务器，服务器容易受到暴力破解攻击，特别是那些需要通过互联网访问的服务器。在这种情况下，大部分企业会考虑使用证书方式进行访问，服务器会关闭用户名密码方式验证，只有证书验证通过的用户才可与服务器建立连接。

使用秘钥登录的原理如图 1-8 所示。

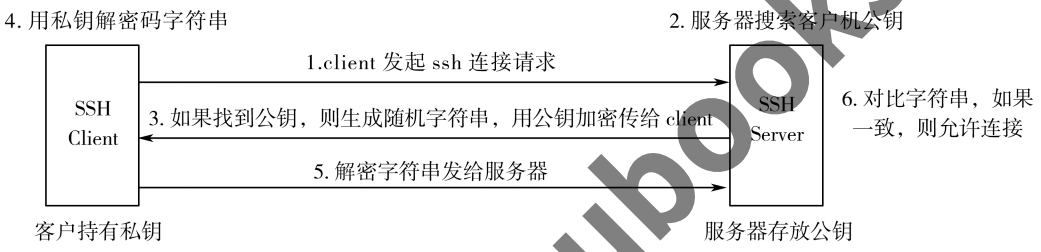


图 1-8　公钥私钥 SSH 访问原理图

由于公钥和私钥可由服务器生成，也可由 Putty 生成，需要注意的是 Linux 下生成的私钥格式 Putty 不能直接使用，需要使用 Puttygen 进行格式转换后才可以使用。

服务器生成公私钥操作：使用命令生成一对公私钥。

```
###生成一对公私钥###
[hadoop@Master ~]$ ssh-keygen
```

执行结果如图 1-9 所示。

图 1-9　密钥对生成

步骤 7：将公钥放入到 $HOME/.ssh/authorized_keys 文件。

```
[hadoop@Master ~]$ cd ~/.ssh
[hadoop@Master .ssh]$ cat id_rsa.pub >> authorized_keys
[hadoop@Master .ssh]$ chmod 600 authorized_keys
```

下载私钥到 Windows 电脑上，如图 1-10 所示。

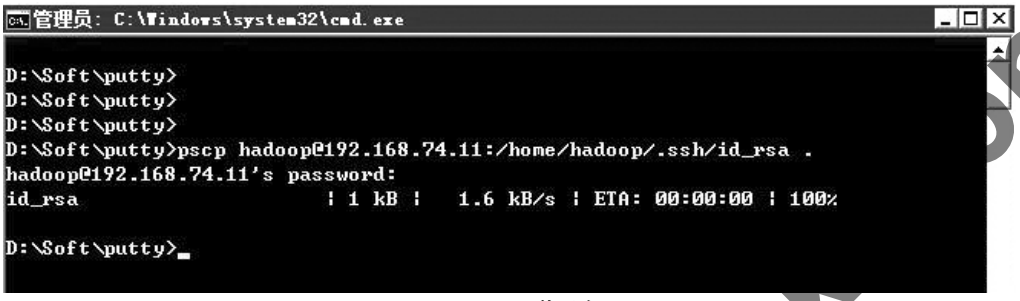


图 1-10　下载私钥

步骤 8：如图 1-11 所示，将私钥转换为 ppk 格式。运行 Puttygen 并加载上一步中下载的私钥 id_rsa。

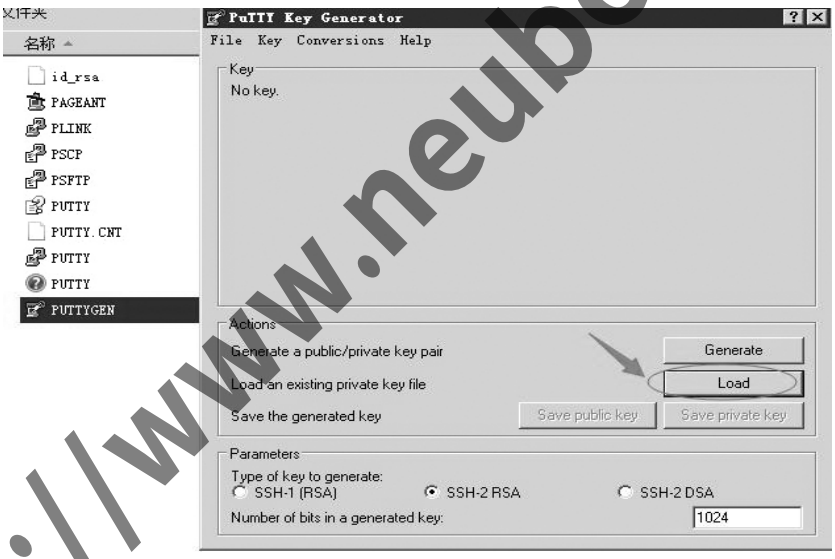


图 1-11　私钥加载

步骤 9：如图 1-12 所示，如果原私钥设置有密码，则会弹出输入私钥密码进行验证的画面。

图 1-12　私钥密码验证

步骤 10：如图 1-13 所示，设置备注及密码。

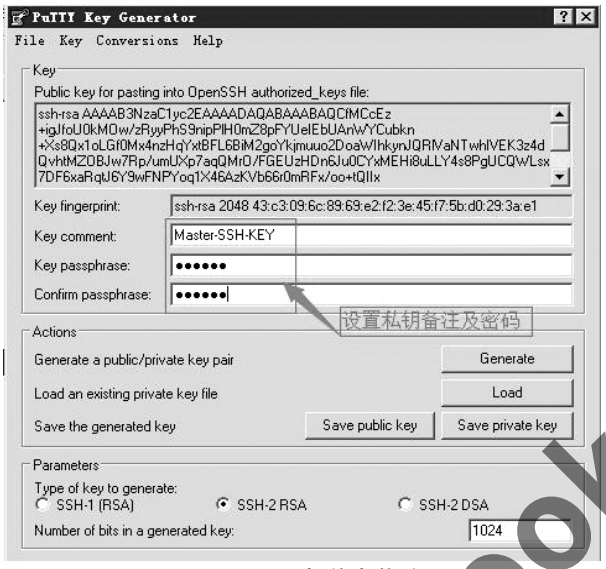


图 1-13　私钥信息修改

步骤 11：如图 1-13 所示，完成后点击“Save private key”保存私钥，即可得到 Putty 的 ppk 格式私钥（图 1-14）。

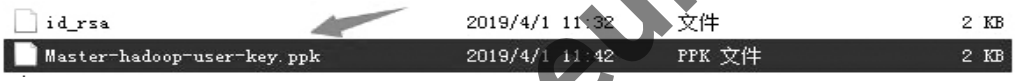


图 1-14　保存后的私钥文件

步骤 12：使用 Puttygen 生成公私钥。

Puttygen 生成公私钥后，私钥保存在本地备用，而公钥则需要上传到服务器。

操作过程如下：

运行 Puttygen，选择 RSA 类型并设置密钥长度，点击“Generate”，生成一对公私钥，生成过程中，需要在进度条下方的区域不断移动鼠标，为生成过程产生随机数，否则进度会停止不动，如图 1-15 所示。

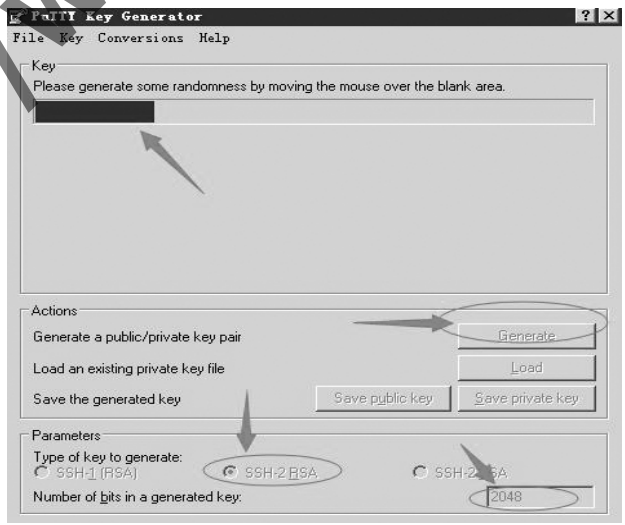


图 1-15　Putty 生成密钥对

步骤 13：如图 1-16 所示，最后分别将公钥和私钥保存为文件。

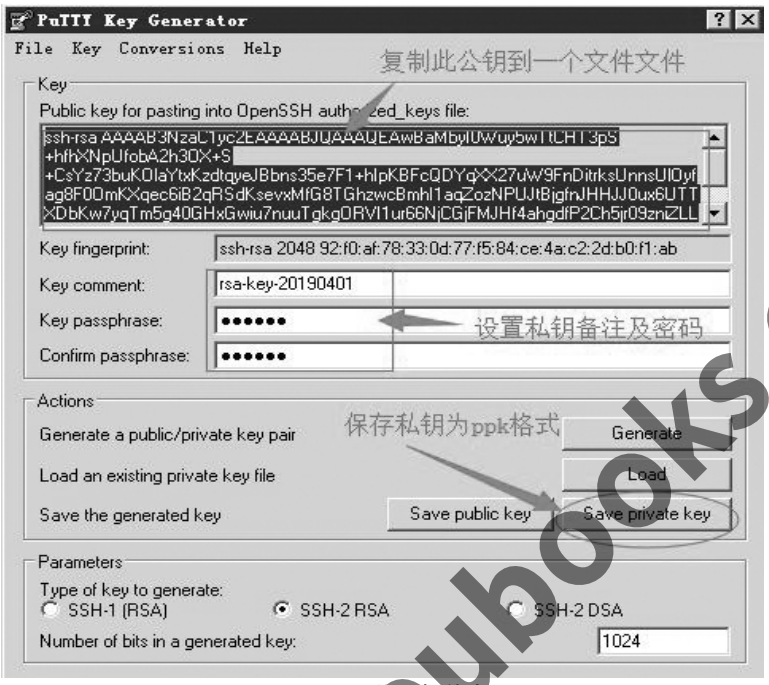


图 1-16　公私钥信息

步骤 14：如图 1-17 所示，将图中的公钥直接复制存放到一个文本文件。不要点下方的“Save public key”，这种方式生成的公钥，Linux 系统不能使用。

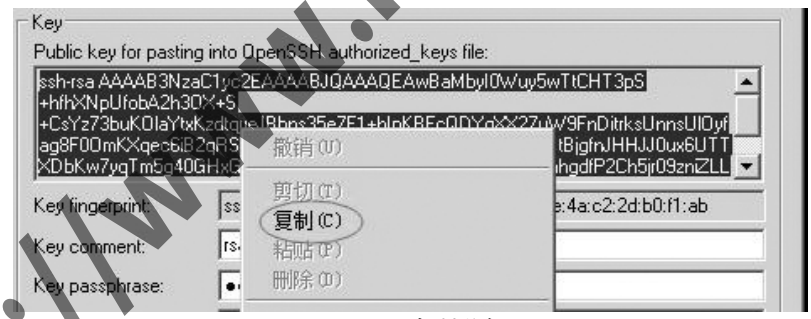


图 1-17　复制公钥

步骤 15：如图 1-18 所示，将公钥保存到文本。

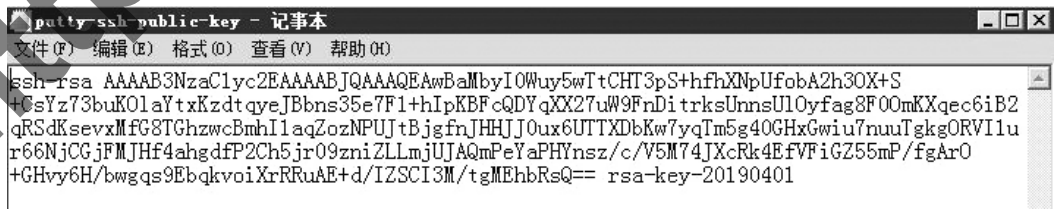


图 1-18　保存公钥

文本文件中的公钥为一行内容，这里显示多行是因为记事本开启了自动换行，请不要手工增加换行。

步骤 16：如图 1-19 所示，将生成的公钥上传到服务器。

图 1-19　上传公钥文件

步骤 17：如图 1-20 所示，将公钥追加到用户 authorized. keys 文件中。

图 1-20　公钥加入认证文件

步骤 18：如图 1-21 所示，使用 Putty 私钥登录系统。运行 Putty 软件，加载之前创建的配置 Master，也可以新创建一个连接配置。

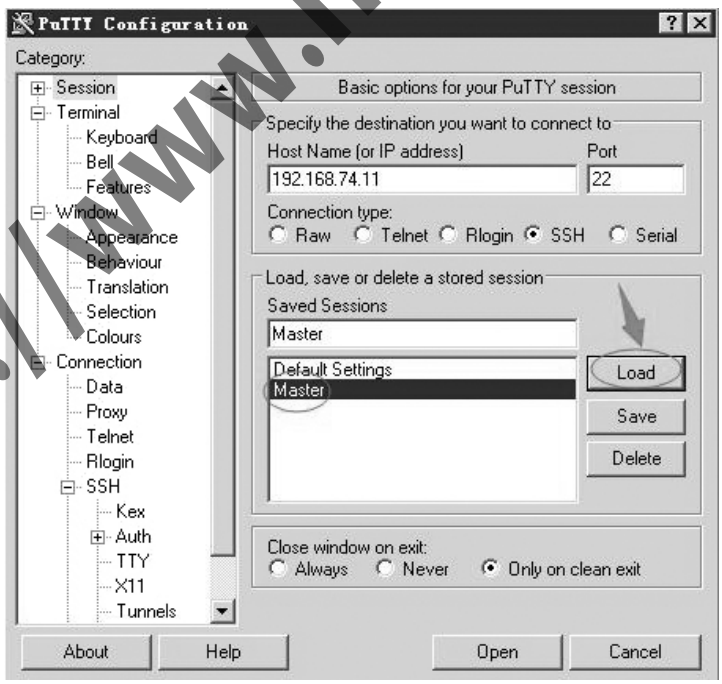


图 1-21　修改连接为私钥登录

步骤 19：如图 1-22 所示，配置 Putty。选择左侧 Connection->Data 设置自动登录名称

（可以自动带入要登录的名称）。

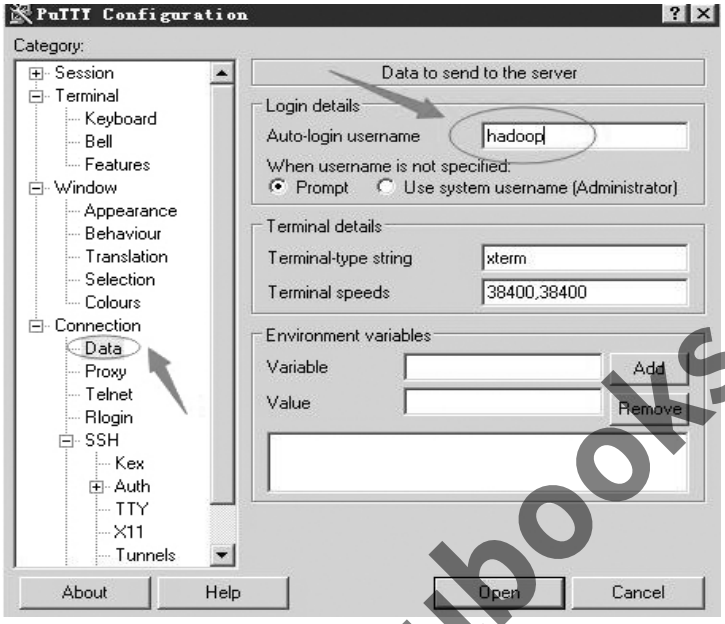


图 1-22　设置自动登录用户名

步骤 20：如图 1-23 所示，选择左侧 SSH->Auth 设置对应私钥。

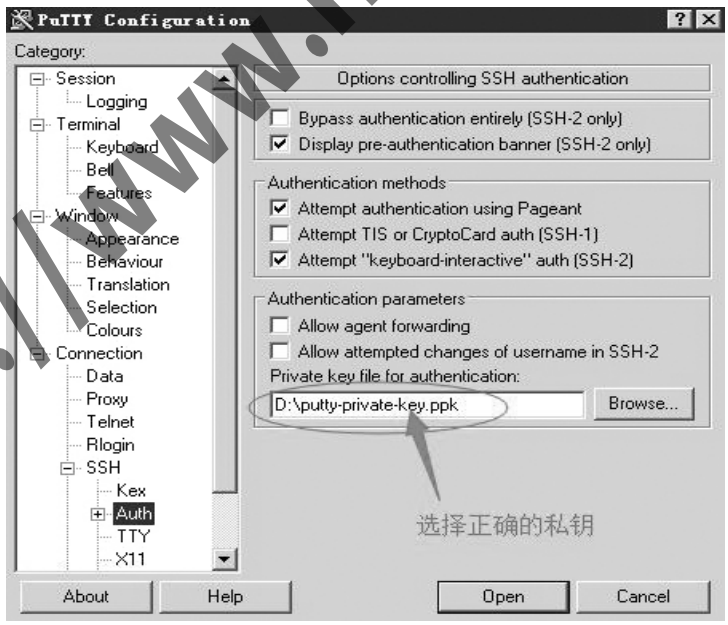


图 1-23　指定用户私钥

步骤 21：如图 1-24 所示，选择最上方的 Session，对修改进行保存，然后就可以点 Open，使用私钥进行服务器连接了。

图 1-24　私钥访问

2. 安装使用 WinSCP

当需要在 Windows 与 Linux 之间传递文件时，Putty 软件包中可以使用命令行工具 pscp，但因为在命令行使用不太方便，所以这里介绍另一款 GUI 工具 WinSCP。

WinSCP 是一个 Windows 环境下使用的 SSH 的开源图形化 SFTP 客户端，同时支持 SCP 协议。它的主要功能是在本地与远程计算机间安全地复制文件，并且可以直接编辑文件。

WinSCP 安装文件可从其官方网站进行下载，下载后按默认方式进行安装后即可开始使用。

步骤 1：运行 WinSCP 工具，如图 1-25 所示。

需要注意的是，WinSCP 能够检测到 Putty 中保存的会话连接，并可根据选择导入 WinSCP 中。

图 1-25　导入 Putty 连接信息

步骤 2：如图 1-26 所示，选择会话连接。

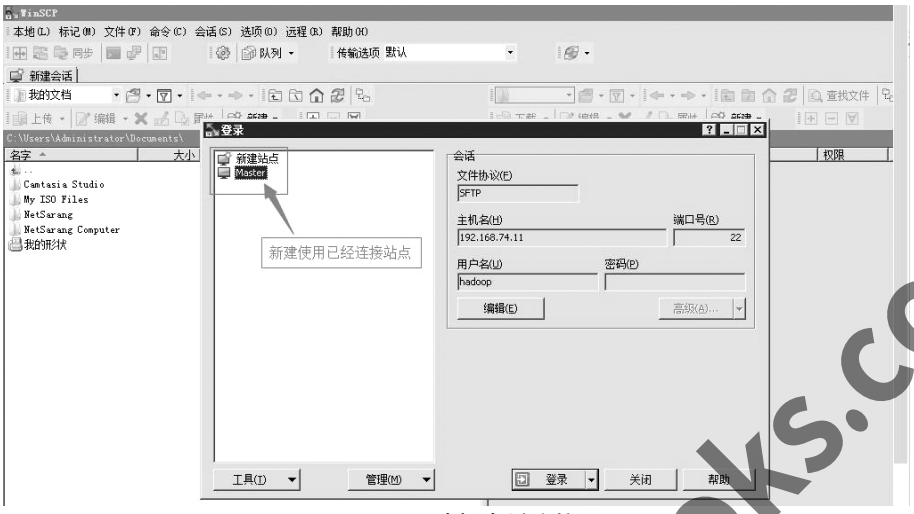


图 1-26　选择会话连接

步骤 3：如图 1-27 所示，如需要对连接选项进行修改，比如进行协议选择，可点编辑按钮。

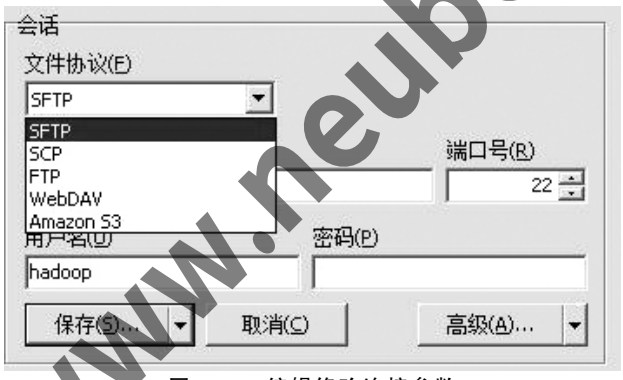


图 1-27　编辑修改连接参数

登录成功后，如果使用的是 Commander 窗口，左侧为本地资源文件，右侧窗体为远程服务器上的资源文件，只要有相应权限，两边窗体都可以切换目录，显示不同资源。如果需要上传、下载文件，可以用鼠标直接拖拽完成操作，如图 1-28 所示。

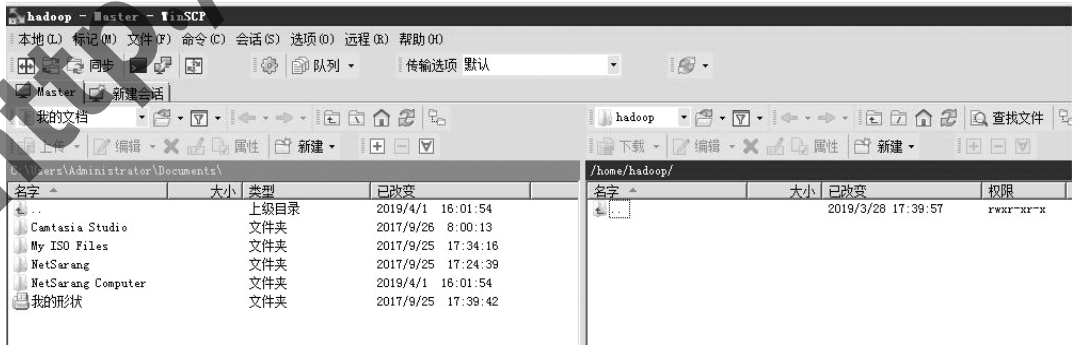


图 1-28　建立会话连接

1.6　典型工作环节 6:搭建 Spark 平台环境

Spark 可以部署在 Windows 系统和 Linux 系统中。在企业应用中,通常是将 Spark 部署在 Linux 系统中。因此,本节内容主要介绍如何在 Linux 环境下部署分布式 Spark 开发平台。根据 Spark 不同的应用场景需求,可选择的 Spark 环境有:单机模式、伪分布式模式、完全分布式模式。

1.6.1　搭建 Spark 平台单机模式

由于 Spark 设计的是主从结构,即便是单机,也是一个单机的集群。因此,为避免 Spark 的进程在相互访问时需要输入系统登录密码,因此需要安装 SSH。这里的 SSH 与 Putty,Xshell 的公钥原理是一致的。

单机模式是指不启动 Spark 的各个进程,通过调用 Spark 的 API 进行计算的模式。该模式一般用于测试。

1. 安装 SSH

步骤 1:使用命令安装 SSH。

```
#####安装 openssh####
yum install openssh*
```

安装过程中提示确认安装信息,如图 1-29 所示。

```
Transaction Summary
================================================================================
Install  5 Packages

Total download size: 585 k
Installed size: 645 k
Is this ok [y/d/N]:
```

图 1-29　确认安装信息

步骤 2:输入 y,继续安装。安装完毕后提示 Complete!,如图 1-30 所示。

```
Installed:
  openssh-askpass.x86_64 0:7.4p1-16.el7                openssh-cavs.x86_64
  openssh-server-sysvinit.x86_64 0:7.4p1-16.el7

Complete!
```

图 1-30　安装完毕

步骤 3:将 SSH 设置成跟随系统启动的服务。

```
#####添加开机自动启动服务#####
systemctl enable sshd
```

步骤 4:启动 SSH 服务。

```
systemctl start sshd
```

步骤 5：使用 SSH 连接本机。

```
ssh localhost
```

连接过程需要输入 yes，敲回车，然后输入账户登录密码，如图 1-31 所示。

```
[root@datanode01 ~]# ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:/rKL2jGnhAGZxGqiDkQCpxc5040/YGKWtk0x1ZwQm1Q.
ECDSA key fingerprint is MD5:fe:2f:e5:93:d4:5f:44:c6:e3:bb:36:95:89:88:8c:63.
Are you sure you want to continue connecting (yes/no)? yes^H^H
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
root@localhost's password:
```

图 1-31　登录本机

登录成功提示最后一次登录时间，如图 1-32 所示。

```
root@localhost's password:
Last failed login: Thu Apr 18 15:11:09 CST 2019 from localhost on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Thu Apr 18 14:42:56 2019 from 192.168.13.37
```

图 1-32　输入密码

步骤 6：在本机上生成秘钥。

```
cd ~/
ssh-keygen -t rsa -P '' -f ~/.ssh/id_dsa
```

秘钥指纹如图 1-33 所示。

```
[root@datanode01 ~]# ssh-keygen -t rsa -P '' -f ~/.ssh/id_dsa
Generating public/private rsa key pair.
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
SHA256:0owewIfni0e27ez23oHPo1IAeymfbmNAJM2z8Bj/3EQ root@datanode01.lab.hwadee.com
The key's randomart image is:
+---[RSA 2048]----+
|    o            |
|  +.*.  E        |
|   O+=oo         |
|  ..B*++.        |
|  ..o*==S        |
|   o..=oo.       |
|    oo... .      |
|   . +B  +..     |
|    .*==+.+.     |
+----[SHA256]-----+
```

图 1-33　生成秘钥

步骤 7：将秘钥保存到文件。

```
cd .ssh
cat id_dsa.pub >> authorized_keys
```

此时在当前目录下会自动创建 authorized_keys 文件，秘钥也被自动保存到 authorized_keys 文件中，如图 1-34 所示。

```
[root@datanode01 .ssh]# cat id_dsa.pub >> authorized_keys
[root@datanode01 .ssh]# ls
authorized_keys  id_dsa  id_dsa.pub  known_hosts
```

图 1-34　保存秘钥文件

2. 安装 Java

由于 Spark2.X 版本后自带 Scala，无须单独安装。但是 Spark 的运行需要依赖 JDK，因此需要安装 Java 环境。

步骤 1：输入命令安装 JDK。

```
yum search openjdk
yum install java-1.8.0-openjdk.x86_64
yum install java-1.8.0-openjdk-devel.x86_64
```

步骤 2：配置 Java 环境变量。Java 环境变量可以针对整个系统所有用户配置，也可只对使用的用户进行设置。对全局所有账号配置，可将环境变量放入/etc/profile 或/etc/bashrc 文件中，如对单个用户进行设置，可将环境变量放入 $HOME/.bash_profile 或 $HOME/.bashrc 中。

```
vi  .bash_profile
```

如图 1-35 所示，添加如下内容。

```
# User specific environment and startup programs
JAVA_HOME=/usr/lib/jvm/java
JRE_HOME=/usr/lib/jvm/jre

export JAVA_HOME
export JRE_HOME

PATH=$PATH:$HOME/.local/bin:$HOME/bin:$JAVA_HOME/bin:$JRE_HOME/bin

export PATH
alias vi=vim
```

图 1-35　设置环境变量

步骤 3：为让环境变量生效，可退出系统，然后重新登录，登录时将会自动加载新设置的环境变量，如想让环境变量不退出系统而生效，可采用手工加载方式完成。

```
source  .bash_profile
```

步骤 4：验证环境变量是否生效。

```
env | grep HOME
env | grep PATH
```

执行结果如图 1-36 所示。

```
[hadoop@Master ~]$
[hadoop@Master ~]$ source .bash_profile
[hadoop@Master ~]$
[hadoop@Master ~]$ env | grep HOME
JRE_HOME=/usr/lbi/jvm/jre
JAVA_HOME=/usr/lib/jvm/java/
HOME=/home/hadoop
[hadoop@Master ~]$ env | grep PATH
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/hadoop/.local/bin:/home/hadoop/bin:/
vm/jre/bin
```

图 1-36　验证环境变量

3. 安装 Spark

从官网下载 spark-2.1.1-bin-hadoop2.7.tgz 到本地，接下来开始安装。

步骤 1：如图 1-37 所示，使用 WinSCP 上传到服务器 opt 目录下。

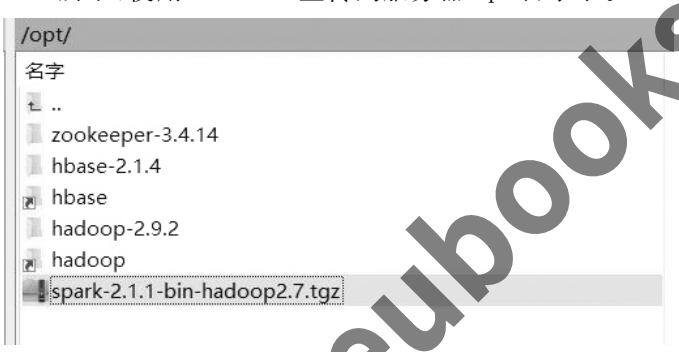


图 1-37　将 Spark 上传到服务器

步骤 2：使用如下命令解压文件。

```
tar -zxf /opt/spark-2.1.1-bin-hadoop2.7.tgz -C /usr/local/
```

切换到/usr/local/，查看解压后的文件。

```
cd /usr/local/
ls
```

如图 1-38 所示。

```
[root@datanode01 local]# ls
bin  etc  games  include  lib  lib64  libexec  sbin  share  spark-2.1.1-bin-hadoop2.7
```

图 1-38　Spark 文件

步骤 3：将文件重命名。

```
mv spark-2.1.1-bin-hadoop2.7 spark
```

步骤 4：复制 spark-env.sh.template 文件。

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

步骤 5：添加 Spark 环境变量。

```
vi ~/.bashrc
```

添加如下内容：

```
export SPARK_HOME=/usr/local/spark
export PYTHONPATH = $SPARK _ HOME/python: $SPARK _ HOME/python/lib/py4j-0. 10. 4-src.zip: $PYTHONPATH
export PYSPARK_PYTHON=python3
export PATH= $HADOOP_HOME/bin: $SPARK_HOME/bin: $PATH
```

步骤 6：使变量修改生效。

```
source ~/.bashrc
```

步骤 7：验证安装。

```
cd /usr/local/spark
bin/run-example SparkPi 2>&1 | grep "Pi is"
```

命令执行结果如图 1-39 所示。

```
[root@datanode01 spark]# cd /usr/local/spark
[root@datanode01 spark]# bin/run-example SparkPi 2>&1 | grep "Pi is"
Pi is roughly 3.143515717578588
```

图 1-39　验证 Spark 安装

1.6.2　搭建 Spark 平台伪分布式模式

伪分布模式是指 Spark 的各个进程独立运行，但是都运行在同一个 Java 虚拟机中。伪分布式模式使用的是 Spark 自带的集群管理工具，需要做如下配置。

步骤 1：将 spark-defaults.conf.template 重命名：spark-defaults.conf。

```
cd /usr/local/spark/conf
mv spark-defaults.conf.template spark-defaults.conf
```

步骤 2：配置 slave。重命名 slaves.template。

```
mv slaves.template slaves
```

将文件底部的 localhost 修改为 datanode01.lab.hwadee.com。

步骤 3：使用如下命令启动 Spark 集群。

```
cd $SPARK_HOME
./sbin/start-all.sh
```

步骤 4：验证启动状态，输入命令。

```
jps
```

命令执行结果如图 1-40 所示，出现框中的进程，表示集群正常启动。

```
21105 Worker
21285 Jps
21017 Master
16586 DataNode
16699 JournalNode
9006 QuorumPeerMain
14190 NodeManager
```

图 1-40　Spark 守护进程

在浏览器输入地址：

```
http://datanode01.lab.hwadee.com:8080/
```

可以看到 SparkUI 界面，如图 1-41 所示。

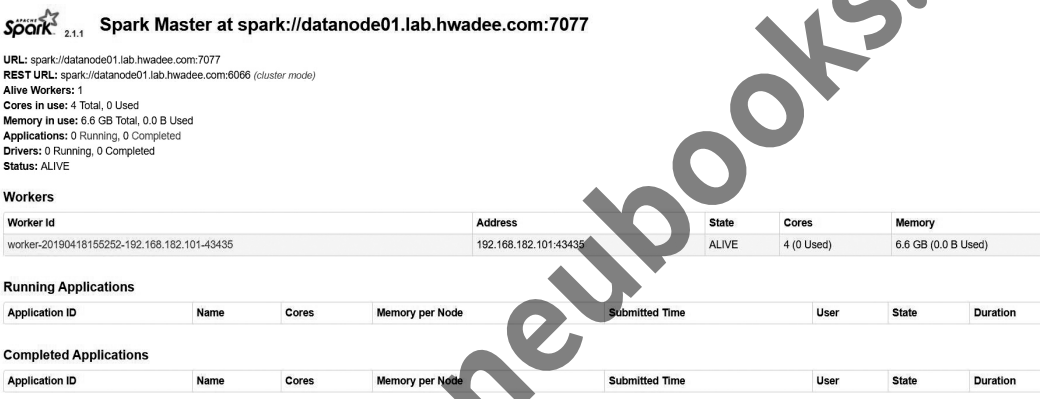


图 1-41　Spark 集群页

步骤 5：提交应用到集群。

```
./bin/spark-submit --master spark://datanode01.lab.hwadee.com:7077 --class org.apache.spark.examples.SparkPi /usr/local/spark/examples/jars/spark-examples_2.11-2.1.1.jar
```

如图 1-42 所示，Spark 创建了应用，ID 为 app-20190418160620-0000，Name 是 Spark Pi，State 是 FINISHED。

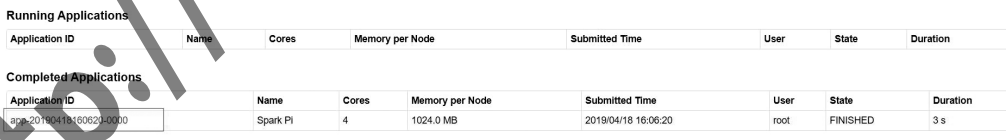


图 1-42　应用信息

1.6.3　搭建 Spark 平台完全分布式模式

使用已安装的 Linux 虚拟机，搭建 Spark 完全分布式模式，并在集群上提交应用。

步骤 1：编辑 slaves 文件，添加集群节点。

```
vi /usr/local/spark/conf/slaves
```

添加如下内容。注意，这是相互能够连通，使用 SSH 工具能互相登录的两台 Linux 服务器。其中，master.lab.hwadee.com 作为主节点，datanode01.lab.hwadee.com 作为工作节点。

```
master.lab.hwadee.com
datanode01.lab.hwadee.com
```

步骤 2：编辑 spark-env.sh 文件，设置主节点的 IP 和主机名称。

```
vi /usr/local/spark/conf/spark-env.sh
```

在文件顶部添加内容。

```
export SPARK_MASTER_IP=192.168.182.11
export SPARK_MASTER_HOST=master.lab.hwadee.com
```

步骤 3：将整个 Spark 目录同步到 master.lab.hwadee.com 机器同一目录下。

```
scp -r /usr/local/spark master.lab.hwadee.com:/usr/local/
```

步骤 4：启动集群。登录到 master.lab.hwadee.com 机器上的 spark 目录下，运行命令。

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

然后在浏览器打开地址：

```
http://master.lab.hwadee.com:8080/
```

如图 1-43 所示，可以看到集群中有两个节点。

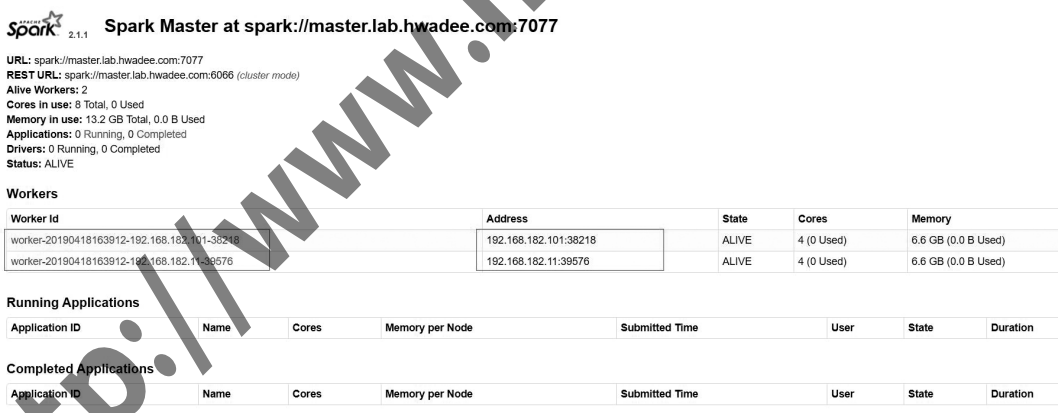
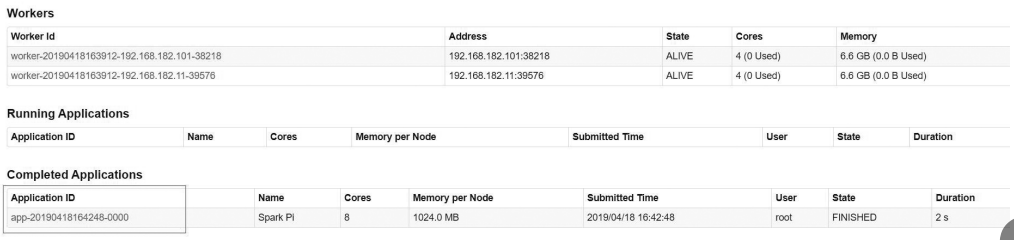


图 1-43　集群页面

步骤 5：提交任务到集群。

```
./bin/spark-submit --master spark://master.lab.hwadee.com:7077 --class org.apache.spark.
examples.SparkPi  /usr/local/spark/examples/jars/spark-examples_2.11-2.1.1.jar
```

刷新 Web 页面，可以看到新的应用已经提交到 Spark 集群上，如图 1-44 所示。



Workers

Worker Id	Address	State	Cores	Memory
worker-20190418163912-192.168.182.101-38218	192.168.182.101:38218	ALIVE	4 (0 Used)	6.6 GB (0.0 B Used)
worker-20190418163912-192.168.182.11-39576	192.168.182.11:39576	ALIVE	4 (0 Used)	6.6 GB (0.0 B Used)

Running Applications

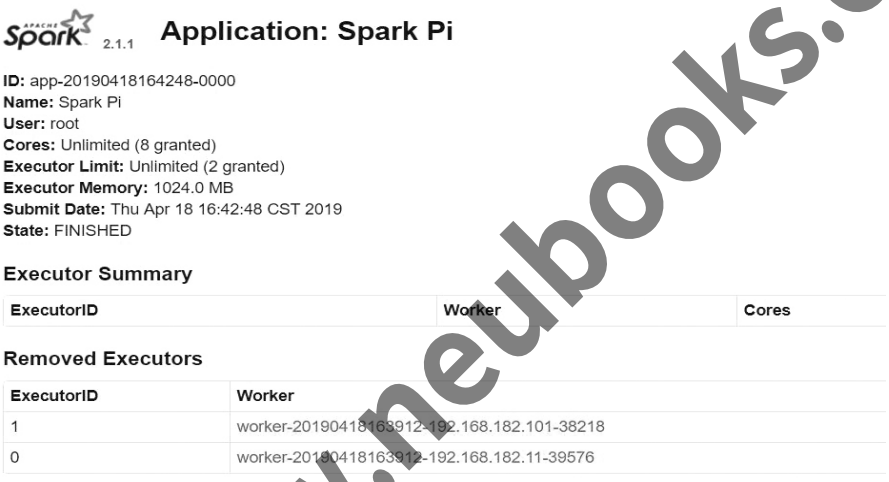
Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20190418164248-0000	Spark Pi	8	1024.0 MB	2019/04/18 16:42:48	root	FINISHED	2 s

图 1-44　集群上的应用

点击应用 ID 的链接，切换到应用详情页，如图 1-45 所示，可以看到两个节点分别都在执行 Spark 应用，这就是完全集群模式下的应用提交。



Application: Spark Pi

ID: app-20190418164248-0000
Name: Spark Pi
User: root
Cores: Unlimited (8 granted)
Executor Limit: Unlimited (2 granted)
Executor Memory: 1024.0 MB
Submit Date: Thu Apr 18 16:42:48 CST 2019
State: FINISHED

Executor Summary

ExecutorID	Worker	Cores

Removed Executors

ExecutorID	Worker
1	worker-20190418163912-192.168.182.101-38218
0	worker-20190418163912-192.168.182.11-39576

图 1-45　应用详情

1.7　归纳总结与拓展提高

本工作情景主要介绍了 Spark 平台环境的搭建。从认识 Spark 开始，到 Spark 生态、应用场景等，逐步深入理解 Spark。然后做了搭建 Spark 平台的准备工作，对单机模式、伪分布式模式、完全分布式模式的 Spark 平台环境进行了介绍。

1.8　课后练习

选择题

1. 以下关于 Spark 说法错误的是(　　)。

A. Spark 专为取代 Hadoop 而设计的大数据平台

B. Spark 与 Hadoop 的 MapReduce 类似，主要负责数据处理过程

C. 大部分情况下 Spark 的数据处理速度远远高于 MapReduce

D. Spark 有比较完整的生态环境，可以与多种快速编程语言配合进行数据处理

2. Spark 可能运行在多种操作系统上，以下相关描述正确的是(　　)。

A. 一般情况下 Windows 环境下安装 Spark 用于测试环境

B. Spark 运行在 CentOS 系统上具有最佳性能

C. Spark 运行于 Linux 下时，只能使用 CentOS

D. Linux 内核中有对 Spark 进行专门优化，所以一般生产环境的操作系统采用 Linux

3. Linux 远程使用 SSH 访问时，说法正确的是(　　)。

A. 只能使用用户名密码方式进行认证登录

B. 只能使用 RSA 公钥私钥方式进行认证登录

C. RSA 私钥不能设置密码，所以用户可以免密码登录

D. SSH 连接远程主机时，用户与主机间的数据是加密传输的

4. 以下关于 RSA 密钥说法错误的是(　　)。

A. RSA 密钥对可在 Linux 系统中制作，也可以在 Windows 系统下的客户端软件中制作

B. 使用 SSH 客户端软件制作 RSA 密钥时，需要将公钥上传到服务器

C. SSH 访问时，客户端配置私钥，服务器端配置对应的公钥

D. Linux 系统中一个用户只能对应一份 RSA 密钥对

5. 以下关于 Spark 运行模式正确的是(　　)。

A. Spark 只能运行于集群模式，依赖于 Hadoop 的 hdfs 分布式文件系统

B. Spark 伪分布模式，各个进程都运行在同一个 java 虚拟机中

C. Spark 伪分布模式，需要多个主机配合来实现

D. Spark 只能独立运行，不能使用 HDFS 中的资源

6. Spark 有哪些特点？[多选](　　)

A. Speed：快速高效　　B. Ease of Use：简洁易用

C. Generality：全栈式数据处理　　D. Runs Everywhere：兼容

7. 下面哪个不是 Spark 的四大组件？(　　)

A. Spark Streaming　　B. Mlib

C. Graphx　　D. Spark R

8. 下面哪个不是 Spark 服务的端口？(　　)

A. 8080　　B. 4040

C. 8090　　D. 18080

9. 以下哪种方式是分布式部署？[多选](　　)

A. Spark on local　　B. Spark on mesos

C. Spark on YARN　　D. Standalone

学习情境二　使用 Scala 统计平台数据

Spark 是一个通用的分布式并行计算框架，可以支持采用 Scala，Java，Python 和 R 语言开发应用程序。由于 Spark 内核是由 Scala 语言开发的，因此 Scala 语言是开发 Spark 应用程序首选语言。

通过本学习情境的学习，可以掌握的知识有：Scala 语言的基础知识，包括基本数据类型和变量、常用容器类型、输入/输出和控制结构、函数、类、异常处理等。

2.1　典型工作环节 1：需求分析

新时期的高职院校作为高素质技能型人才培养的摇篮，每年都有大批毕业生走向就业岗位，为了让毕业生在人才市场上精准就业，毕业生需要了解最新行业动态信息、就业岗位的要求、薪资水平等信息。现某学院利用前沿的大数据技术采集三大就业门户网的行业就业信息，然后进行统计，实时显示给毕业生，助其精准就业。

该平台需采集如下信息：最新行业动态信息、就业岗位的要求、薪资水平等。然后对该城市职位类型最高薪酬、平均薪酬进行统计。

2.2　典型工作环节 2：步骤分析

第一步：优选系统开发语言。

第二步：搭建开发环境。

第三步：使用程序设计语言统计平台数据，编写程序，收集三大网站某城市 IT 行业全部岗位需求信息：岗位名称、地区、岗位类型、薪资。

第四步：统计该城市职位类型最高薪酬、平均薪酬。

2.3　典型工作环节 3:优选系统开发语言

Spark 开发目前主要可以使用三种语言:Scala,Java,Python。一般研发团队选择各开发语言对比优势如下:

1. Java 与 Scala

(1)当涉及大数据 Spark 项目场景时与 Python 和 Scala 相比,Java 太冗长了,1 行 Scala 可能需要 10 行 Java 代码。

(2)当开发大数据项目时,Scala 支持 Scala-shell,这样可以更容易地进行原型设计,并帮助初学者轻松学习 Spark,而无需全面的开发周期。但是 Java 不支持交互式的 shell 功能。

2. Python 与 Scala

虽然两者都有很简洁的语法,两者都是面向对象加功能,两者都有活跃的社区。

(1)Python 通常比 Scala 慢,Scala 会提供更好的性能。

(2)Scala 是 static typed。错误在编译阶段就抛出,它使在大型项目中开发过程更容易。

(3)Scala 基于 JVM,因为 Spark 是基于 Hadoop 的文件系统 HDFS 的。Python 与 Hadoop 服务交互非常糟糕,因此开发人员必须使用第三方库(如 Hadoopy)。Scala 通过 Java 中的 Hadoop API 来与 Hadoop 进行交互。

通常在 Scala 中编写本机 Hadoop 应用程序非常简单,另外 Scala 在 JVM 上运行,这使得更容易集成 Hadoop、YARN 等框架,因此优选 Scala 语言进行开发。

2.4　典型工作环节 4:了解 Scala 语言

编程语言的流行主要归功于其技术上的优势以及其对某种时代需求的适应性。随着近几年大数据技术、物联网技术的快速发展,对于高并发性、异构性以及快速开发等应用场景,函数式编程语言流行起来,而传统的面向对象编程的统治地位还没有结束,因此,能够将两者结合起来的 Scala 语言开始流行起来。

Scala 是由瑞士洛桑联邦理工学院的 Martin Odersky 教授于 2001 年设计的。Scala 语言的名称来自“scalable language”,就是“可扩展”的语言。Scala 的“可扩展”是因为它融合了函数式编程和面向对象编程的思想,前者让它可以很方便快速地构建可用程序,后者则让其具有构建大型可扩展系统的能力。

Scala 语言是面向对象编程语言,又无缝地结合了命令式和函数式的编程风格。它也可以访问现存的数之不尽的 Java 类库。基于其函数式编程特性,Scala 在大数据时代越来越受欢迎,甚至有人认为它会是下一代 Java。其特点如下:

1. 面向对象

Scala 是一门纯粹的面向对象的语言。Scala 运行于 Java 虚拟机(JVM)之上,并且兼容现有的 Java 程序,可以与 Java 类进行互操作,包括调用 Java 方法、创建 Java 对象、集成 Java 类和实现 Java 接口。

在 Scala 中,每个值都是对象。对象的数据类型以及行为由类和特质描述。类抽象机制的扩展有两种途径:一种途径是子类继承,另一种途径是灵活的混入机制,这两种途径能避免多重继承的一些问题。

2. 函数式编程

Scala 也是一种函数式语言,其函数也能当成值来使用。Scala 提供了轻量级的语法用以定义匿名函数,支持高阶函数,允许嵌套多层函数,并支持柯里化。Scala 的 case class 及其内置的模式匹配相当于函数式编程语言中常用的代数类型。

2.5 典型工作环节 5:搭建开发环境

尽管 Scala 与 Java 兼容,同样也运行在 Java 虚拟机之中,但是它使用的是自身的开发接口,因此需要单独安装相关的库,同时需要提前安装 Java 虚拟机。

编写 Scala 程序使用 Eclipse,ENSIME,NetBeans,Vim,IntelliJ IDEA 等作为开发工具,但在生产环节中,根据用户量和趋势,以及官方推荐,本书将采用 IntelliJ IDEA 作为主要开发环境。安装步骤如下:

第一步,安装 Scala SDK。

第二步,安装 IntelliJ IDEA。

第三步,安装 IntelliJ IDEA Scala 开发插件。

2.5.1 安装 Scala SDK

从官网下载 Scala SDK。注意,选择版本有讲究,需要和 Spark 配套。根据本书 Spark 的版本,这里选择 2.11.12 版本。

```
scala-2.11.12.msi
```

下载完毕后就可以进行安装。

步骤 1:双击软件名称,打开欢迎界面,如图 2-1 所示,然后单击 Next。

步骤 2:如图 2-2 所示,接受许可确认界面。勾选接受许可复选框,然后单击 Next。

步骤 3:如图 2-3 所示,选择安装路径。保持默认,然后单击 Next。

步骤 4:如图 2-4 所示,进入安装确认界面,然后单击 Install。

步骤 5:如图 2-5 所示,等待安装完毕。

步骤 6:如图 2-6 所示,安装完毕,单击 Finish 按钮关闭对话框。

图 2-1　Scala 欢迎界面

图 2-2　接受许可

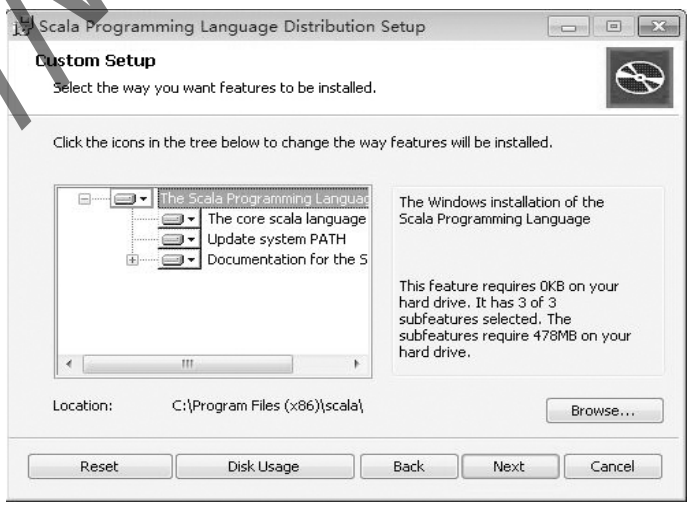


图 2-3　选择安装路径

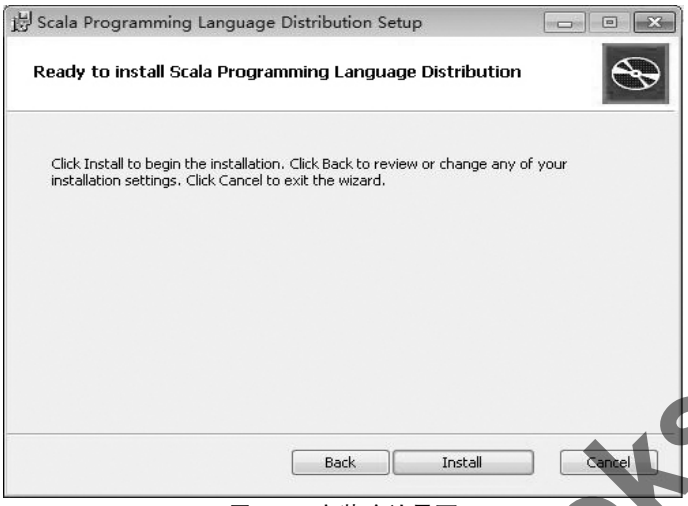


图 2-4　安装确认界面

图 2-5　安装等待

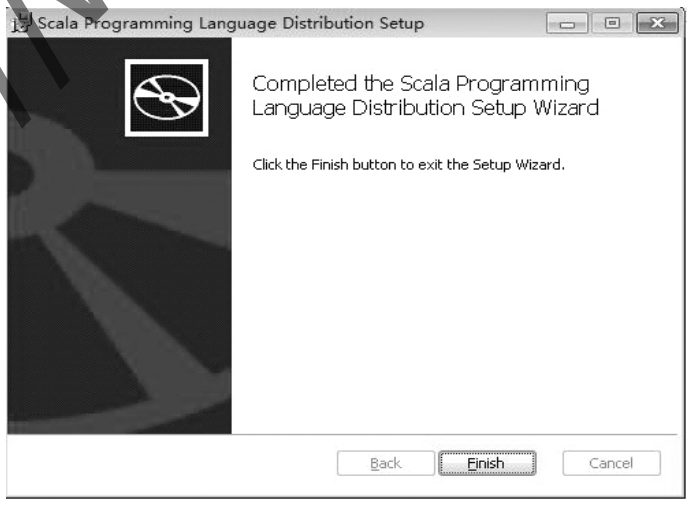


图 2-6　安装完毕

步骤 7：右键单击计算机，选择属性菜单，如图 2-7 所示，选择高级系统设置。

图 2-7　设置环境变量

步骤 8：如图 2-8 所示，将 Scala 路径加入环境变量。

图 2-8　设置环境变量

步骤 9：如图 2-9 所示，在命令行窗口中输入 scala，能输出对应版本号，表示正常安装。

图 2-9　验证安装

2.5.2　安装 IntelliJ IDEA

从官网下载 ideaIU-2019.1.1.exe。

步骤 1：双击软件名称，打开欢迎界面，如图 2-10 所示，然后单击 Next。

图 2-10　Idea 欢迎界面

步骤 2：如图 2-11 所示，选择安装路径，然后单击 Next。

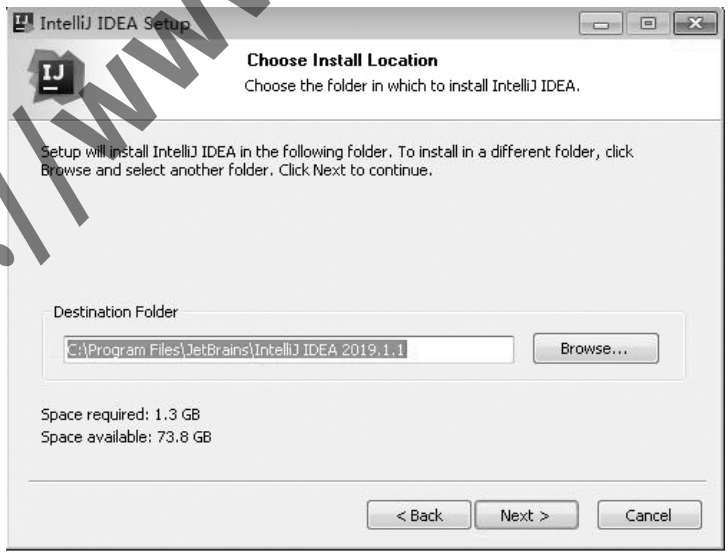


图 2-11　设置安装路径

步骤 3：如图 2-12 所示，这里需要根据系统的位数来勾选 32-bit 或者 62-bit。其余保持默认，继续点击 Next。

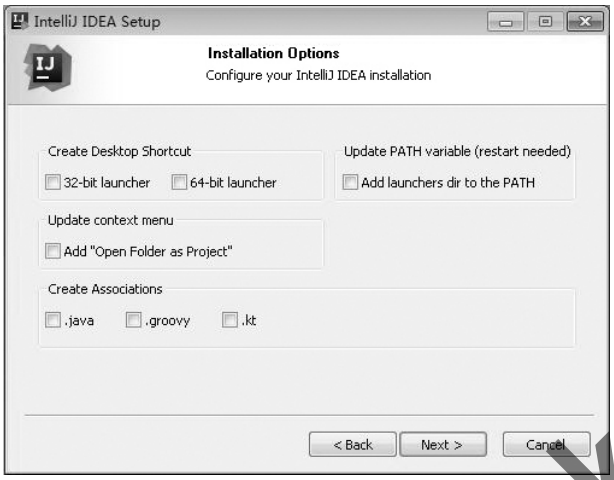


图 2-12　安装配置

步骤 4：如图 2-13 所示，设置开始菜单目录名称，这里保持默认，然后点击 Install 进行安装。

图 2-13　创建开始菜单目录

步骤 5：如图 2-14 所示，等待安装结束。

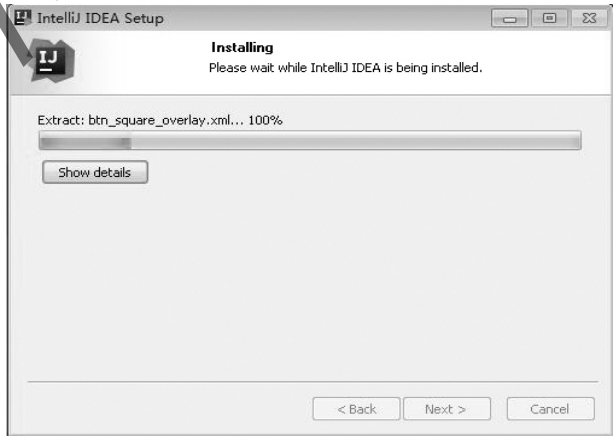


图 2-14　等待安装

步骤 6：如图 2-15 所示，单击 Finish 按钮，关闭安装界面。

图 2-15　安装完成界面

2.5.3　安装 IntelliJ IDEA Scala 开发插件

步骤 1：双击 IDEA 桌面图标，启动开发工具。如图 2-16 所示，保持默认点击 OK。

图 2-16　导入配置

步骤 2：如图 2-17 所示，选择工具主题，这里选择 Light。然后点击右下角的 Next 按钮。

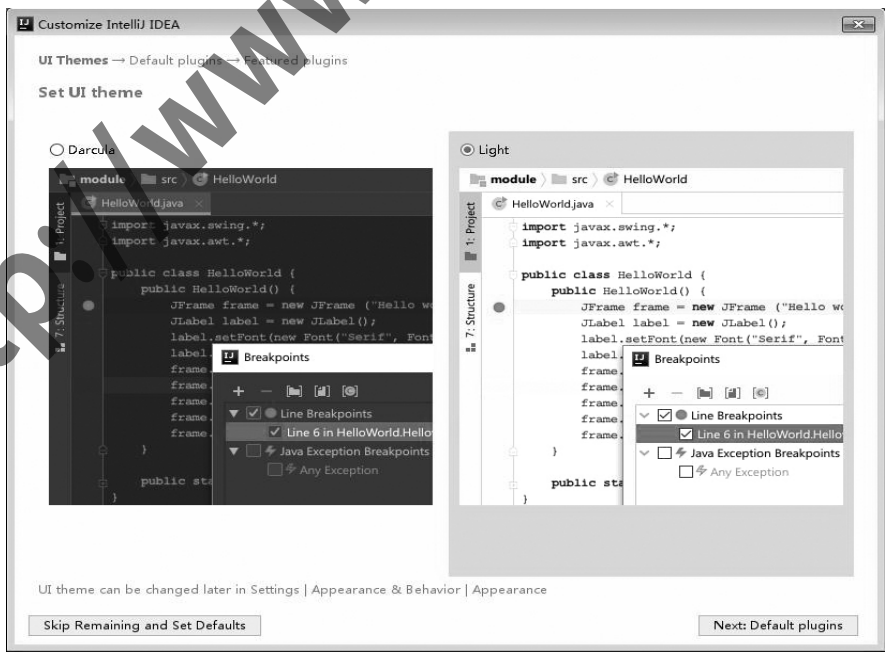


图 2-17　选择主题

步骤 3：如图 2-18 所示，这里可以安装额外的组件。点击左下角的 Skip 按钮，跳过安装。

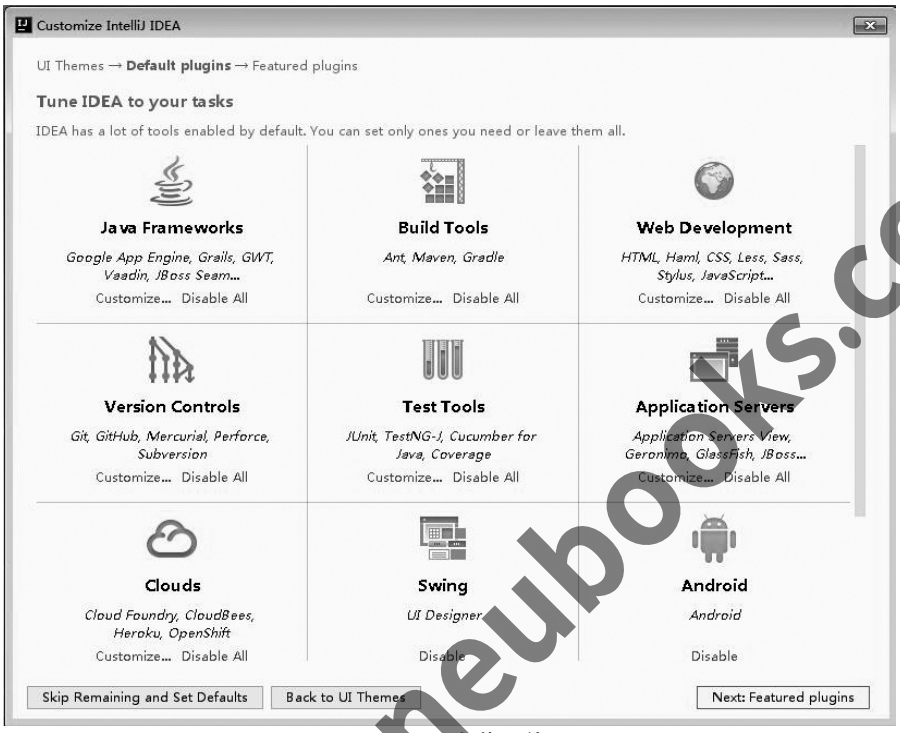


图 2-18　安装组件

步骤 4：如图 2-19 所示，需要输入账户密码。因此这里选择“Evaluate for free”，选择评估版本即可。

图 2-19　账户验证

步骤 5：点击“Evaluate”按钮，进入启动界面，如图 2-20 所示。

图 2-20　启动界面

步骤 6：选择 Scala 插件，点击 Install 进行安装，如图 2-21 所示。

图 2-21　安装 Scala 插件

步骤 7：安装完毕后点击 Restart，重启 IDEA，如图 2-22 所示。

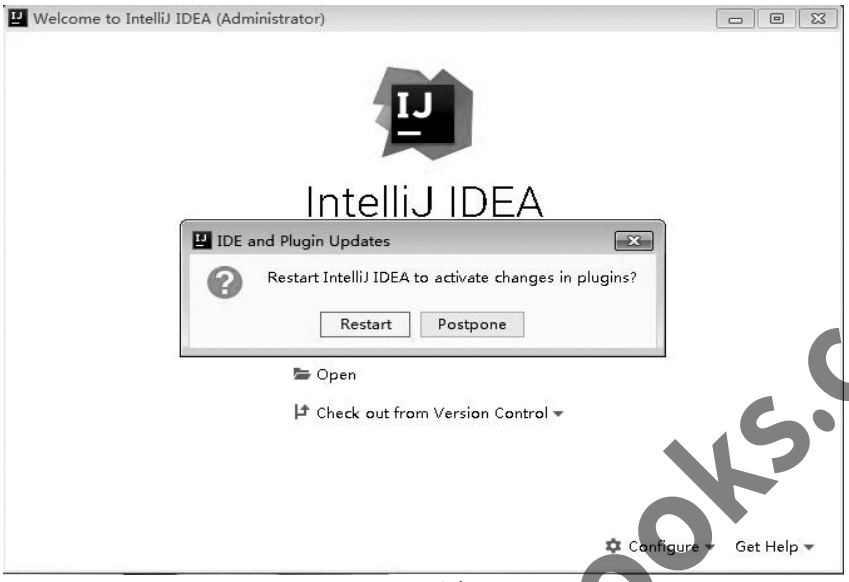


图 2-22　重启 Idea

步骤 8：如图 2-23 所示，选择 Scala，创建 IDEA 项目。

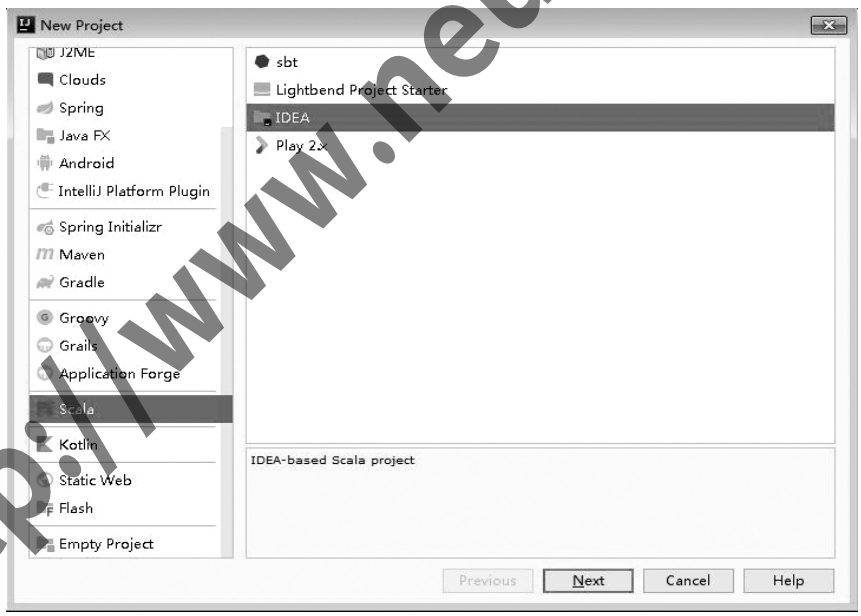


图 2-23　创建 Scala 项目

步骤 9：如图 2-24 所示，首先选择 JDK，然后在 Scala SDK 后面点击 Create 按钮，弹出对话框，此时 IDEA 会自动检测已经安装好的 Scala SDK。这里保持默认，然后点击 OK。

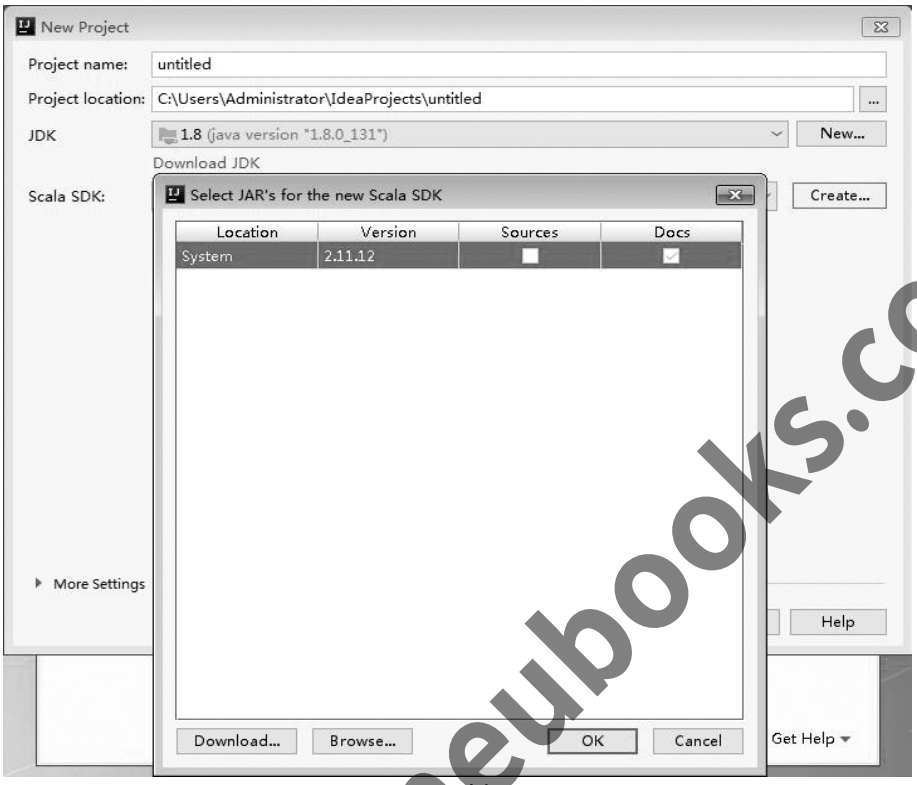


图 2-24　选择 SDK

步骤 10：如图 2-25 所示，配置完毕后点击 Finish 按钮。

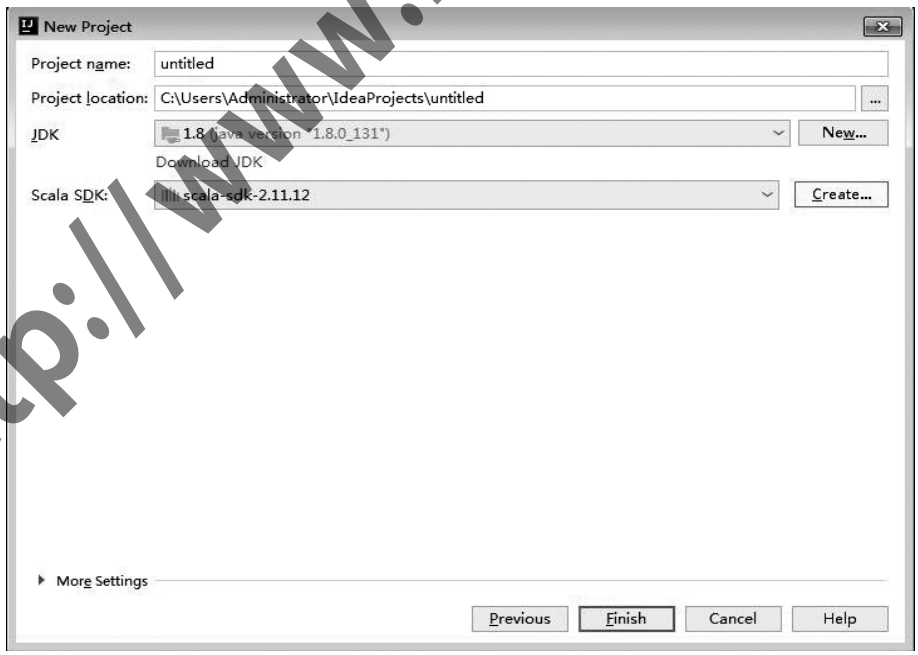


图 2-25　配置 JDK 和 SDK

步骤 11：如图 2-26 所示，点击 Close，关闭欢迎窗口。

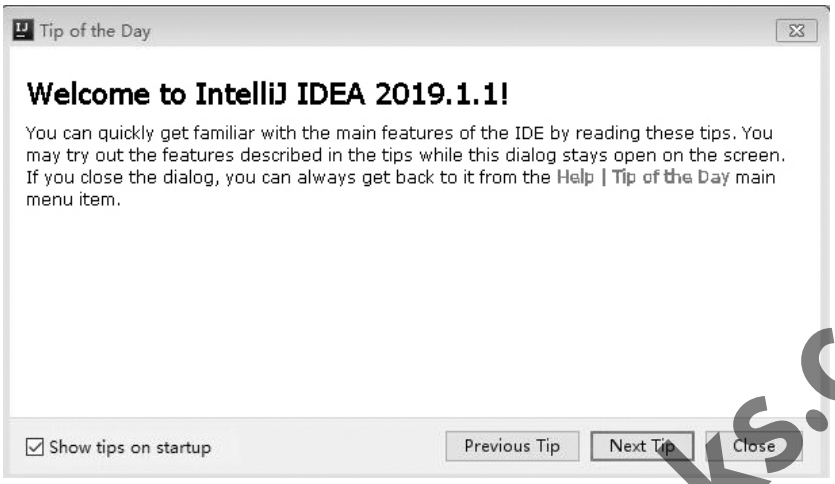


图 2-26　欢迎窗口

步骤 12：如图 2-27 所示，这里需要创建 Scala Class。

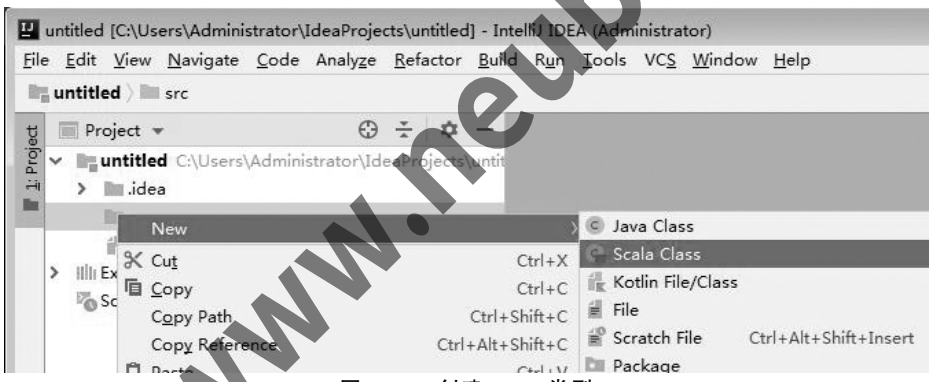


图 2-27　创建 Scala 类型

步骤 13：如图 2-28 所示，输入类名，并在 Kind 下拉框中选择 Object，然后点击 OK。

图 2-28　创建类

步骤 14：如图 2-29 所示，在编辑器中输入如下内容，点击右键，运行程序。

图 2-29　运行程序

运行结果如图 2-30 所示，在控制台输出 hello world。

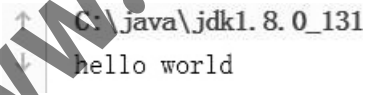


图 2-30　输出 hello world

2.6　典型工作环节 6：学习 Scala 语言

2.6.1　Scala 编程基础

1. Scala 语言简介

Scala 是 Scalable Language 的简写，是一门多范式的编程语言。联邦理工学院洛桑的 Martin Odersky 于 2001 年基于 Funnel 的工作开始设计 Scala。Odersky 先前的工作内容是 Generic Java 和 javac(Sun Java 编译器)。Java 平台的 Scala 于 2003 年底、2004 年初发布。.NET 平台的 Scala 发布于 2004 年 6 月。该语言第二个版本v2.0，发布于 2006 年 3 月。Scala 的发展历史见表 2-1。

表 2-1　　Scala 的发展历史

时间	描述
2001 年	Scala 的设计在 EPFL 开始
2004 年初	Java 平台的 Scala 发布
2004 年 6 月	.NET 平台的 Scala 发布
2006 年 3 月	Scala 2.0 Java 版发布
2012 年	官方停止维护 Scala .NET 版
2014 年	Scala 2.11.2 发布
2019 年	Scala 2.13.0 发布

2. Scala 的特性

(1)面向对象特性

Scala 是一种纯面向对象的语言,每个值都是对象。对象的数据类型以及行为由类和特质描述。

(2)函数式编程

Scala 也是一种函数式语言,其函数也能当成值来使用,这一点又和 Python 类似。因此我们说 Scala 集成了 Java 和 Python 的特性。Scala 提供了轻量级的语法,用以定义匿名函数,支持高阶函数,允许嵌套多层函数,并支持柯里化。

(3)语言简洁优雅

如果你写过 Java,就会对这一点体会深刻,Scala 几行代码就能完成 Java 一个很复杂的操作,就代码量而已,Scala 会少很多。

(4)Scala 商业成功

Spark 和 Kafka 都是使用 Scala 语言编写的,因此 Scala 广泛应用于大数据领域。

Scala 是一门脚本语言。开始学习 Scala 语言最简单的方法是使用 Scala REPL(Read,Eval,Print,Loop 的缩写)。像 Python 的解释器那样,在命令行输入一个表达式后回车,直接计算并输出表达式的值,如图 2-31 所示。Scala REPL 是一个运行 Scala 表达式和程序的交互式 Shell,在 Linux 系统中打开一个终端(组合键),输入 Scala 命令启动 Scala REPL。

如果一条语句需要占用多行,只需要以一个不能合法结尾的字符(比如未封闭的括号与引号中间的字符)结束,则 REPL 会自动在下一行以“|”开头,提示用户继续输入。

图 2-31　命令行编写 Scala

在解释器里编写程序,一次只能运行一行,如果运行多行则需要编写脚本文件,在 Linux Shell 中用“scala 文件名”即可运行,或者在 Scala 解释器终端用“:load 文件名”的形式执行。在 Scala SDK 安装完毕后,安装目录下有一个编译与解析的工具,通过该工具可以执行 Scala 脚本文件,如图 2-32 所示。

图 2-32　执行 Scala 脚本

①Scala 源文件以“.scala”为扩展名。

②Scala 程序的执行入口是 main() 函数。

③Scala 语言严格区分大小写。

④Scala 方法由一条条语句构成，每个语句后不需要分号，加上也不错。

⑤如果在同一行有多条语句，除了最后一条语句不需要分号，其他语句需要分号。

3. 单例对象 Singleton

在 Scala 中，使用 class 定义的 scala 对象，称为类。如果用 object 替换 class 关键字，那么这个就叫单例对象。

```
object hello extends App{
  println("hello world")
}
```

区分大小写：Scala 是大小写敏感的，这意味着标识 Hello World 和 hello world 在 Scala 中会有不同的含义。

类名：对于所有的类名的第一个字母要大写。如果需要使用几个单词来构成一个类的名称，每个单词的第一个字母要大写。

示例：

```
class MyFirstScalaClass
```

单例对象与类的区别如下：

● 单例对象与类同名时，这个单例对象被称为这个类的伴生对象，而这个类被称为这个单例对象的伴生类。伴生类和伴生对象要在同一个源文件中定义，伴生对象和伴生类可以互相访问其私有成员。不与伴生类同名的单例对象称为孤立对象。

● 类和单例对象的一个差别是，单例对象是在第一次访问的时候初始化，不可以初始化，不能带参数，而类可以初始化，可以带参数。

● 使用伴生对象的主要目的：(1)可以编写独立运行的 Scala 程序。(2)模拟静态方法、静态类。

【例 2-1】 演示了类与单例对象的区别与联系

代码如下：

```
class TestScalaA {
  def show(): Unit = {
    println("这是 TestScalaA 伴生类")
    //伴生类可以访问伴生对象的私有方法
    TestScalaA.showA()
    TestScalaA.showB()
```

```
  }

  private def showA(): Unit = {
    println("这是 TestScalaA 伴生类 private 方法")
  }

  protected def showB(): Unit = {
    println("这是 TestScalaA 伴生类 protected 方法")
  }

}

object TestScalaA {
  def show(): Unit = {
    println("这是 TestScalaA 伴生对象")
    var a = new TestScalaA
    //    在伴生对象中,可以访问伴生类中的私有、保护、公有方法
    a.showA()
    a.showB()
  }

  private def showA(): Unit = {

  }

  protected def showB(): Unit = {

  }
}

object TestScala extends App {
  var a = List(1, 2, 2)
  var c = new TestScalaA
  c.show()
}
```

4. 可运行的 Scala 程序

要想编写能独立运行的 Scala 程序,就必须创建含有 main 方法的单例对象。main 方法将作为程序入口:

【例 2-2】 包含 main 函数的单例对象

代码如下:

```
object ChecksumAccumulator {
  def main(args: Array[String]): Unit = {


  }
}
```

在【例 2-1】中，我们发现没有定义 main 函数，仍然能执行。那是因为 TestScala 继承自 App 对象。App 对象中已经包含 main 方法。

App 的定义如图 2-33 所示。

```
2  package scala
3  trait App extends scala.AnyRef with scala.DelayedInit {
4    @scala.deprecatedOverriding("executionStart should not be overridden", "2.11.0")
5    val executionStart : scala.Long = { /* compiled code */ }
6    @scala.deprecatedOverriding("args should not be overridden", "2.11.0")
7    protected def args : scala.Array[_root_.scala.Predef.String] = { /* compiled code */ }
8    @scala.deprecated("the delayedInit mechanism will disappear", "2.11.0")
9    override def delayedInit(body : => scala.Unit) : scala.Unit = { /* compiled code */ }
10   @scala.deprecatedOverriding("main should not be overridden", "2.11.0")
11   def main(args : scala.Array[_root_.scala.Predef.String]) : scala.Unit = { /* compiled code */ }
12 }
```

图 2-33　App 定义

- 方法名称：所有的方法名称的第一个字母用小写。如果若干单词被用于构成方法的名称，则每个单词的第一个字母应大写。

示例：

```
def myMethodName()
```

- 程序文件名：程序文件的名称应该与对象名称完全匹配。
- 示例：假设“HelloWorld”是对象的名称。那么该文件应保存为“HelloWorld.scala”。
- main 函数：

```
def main(args: Array[String])
```

- Scala 程序从 main()方法开始处理，这是每一个 Scala 程序的强制程序入口部分。
- 函数返回值：Scala 方法返回值时，可以不用 return 字段，在函数代码块最后一行直接写上 value 就可以，编译器自动推断类型；若使用 return，返回值就必须明确写好是什么类型。
- 默认访问：默认情况下类、成员、方法都是公有的。

2.6.2　Scala 基本语法

本节主要介绍 Scala 基础知识，包括标识符、关键字、基本数据类型、变量和常量、运算符和表达式等。

1. 标识符和关键字

Scala 对变量、方法、函数等命名时使用的字符序列称为标识符。其命名规则如下：

- Scala 中的标识符声明基本和 Java 一致，但是细节上会有所不同。
- 首字符是字母，不能是数字，后续字符可以是字母、数字、美元符号或下划线。
- 操作符标识符由一个或多个操作符字符组成。
- 操作符（如＋、－、＊、/）不能在标识符中间和最后。

● 用反引号包括的任意字符串，即使是关键字也可以。

定义的变量名不能与关键字重复。这里列出 Scala 的关键字，如表 2-2 所示。

表 2-2　　关键字

名称	描述	名称	描述
abstract	创建抽象类	if	条件判断
def	定义函数	else	条件判断
false	布尔值	match	模式匹配
true	布尔值	override	覆盖
forSome	模糊的类型	return	返回
lazy	定义惰性变量	throw	抛出异常
object	创建单例对象	type	声明类型
protected	访问限制	with	创建复合类型
this	当前对象	class	定义类
final	防止派生类	extends	继承基类
import	导入	try	捕获异常
null	空值	catch	捕获异常
private	访问限制	finally	捕获异常后最终要执行的代码
super	调用父类成员	implicit	隐式转换
new	创建实例	yield	迭代器
package	包名称	case	判断选择
sealed	密封类	do	do…while…循环
trait	特质	while	循环
var	定义变量	for	循环
val	定义常量		

2. 基本数据类型

Scala 与 Java 是兼容的，因此 Scala 的数据类型与 Java 是类似的。表 2-3 列出了 Scala 主要的数据类型。

表 2-3　　Scala 主要的数据类型

序号	类型名称	描述
1	Byte	字节
2	Short	16 位整数
3	Int	32 位整数
4	Long	64 位整数
5	Float	32 位单精度浮点数
6	Double	64 位双精度浮点数

（续表）

序号	类型名称	描述
7	Char	字符序列
8	String	字符串序列
9	Boolean	取值 true 或 false
10	Unit	无值，与 void 等效。在函数无返回值的使用 Unit 表示。Unit 只有一个值：()
11	Null	空值
12	Nothing	Scala 类继承链的最底端，是所有类型的子类型
13	Any	Scala 类继承链的最顶端，是所有类型的父类型
14	AnyRef	AnyRef 类是所有引用类型的基类
15	AnyVal	AnyVal 类是所有值类型的基类

3. 变量和常量

Scala 有两种方式定义变量：val 与 var。val 创建的是只读类型的变量，称为常量，不可修改；var 定义的则是变量，可随处修改。

【例 2-3】 定义变量

代码如下：

```
object TestScala extends App {
  val a = 5
  var b = 6
  a = 10 //出错
}
```

Scala 定义变量基本格式如下：

```
var 变量名称:变量类型=值
```

【例 2-4】 定义变量实例。

代码如下：

```
object TestScala extends App {
  //  岗位名称
  var job_name: String = "大数据开发工程师"
  var job_name1: String = "scala" +
    "开发工程师"
  //  工资
  var salary: Int = 8888
  //  个人所得税
  var tax: Float = 188.88f
  //  公积金
  var accumulation_fund = 388.88
```

```
    // 公司名称
    var company_name = null
    //  是否参加社保
    var is_buy_social_insurance = true
    //  学历要求
    var degree: Any = "专科"
    //  工作经验
    var work_experience: AnyVal = 5
    var work_experience1: AnyRef = "5 年以上"
    // 技能要求
    val skill1, skill2 = "大数据"

}
```

Scala 可以自动推断数据类型，在定义变量时可以不必指定数据基本类型。

【例 2-5】 输出 Scala 变量类型。

代码如下：

```
object TestScala extends App {
    val a = 5
    println("a 的类型是:"+a.getClass())
    var b = 6
    println("b 的类型是:"+b.getClass())
}
```

运行结果如图 2-34 所示。

```
a的类型是: int
b的类型是: int
```

图 2-34　输出类型

4. 运算符

(1)算术运算符

如表 2-4 所示，它是 Scala 支持的算术运算符。

表 2-4　　算术运算符

序号	名称	描述
1	+	两个数据相加
2	−	两个数据相减
3	*	两个数据相乘
4	/	两个数据相除
5	%	两个数据取模

【例 2-6】 基本的算术运算

代码如下：

```
object TestScala extends App {

  var a = 10
  var b = 15

  var c = a + b
  println("a+b="+ c)

  c = a - b
  println("a-b="+ c)

  c = a * b
  println("a*b="+ c)

  c = a / b
  println("a/b="+ c)

  c = a % b
  println("a%b="+ c)

}
```

运行结果如图 2-35 所示。

```
a+b=25
a-b=-5
a*b=150
a/b=0
a%b=10
```

图 2-35 输出运算结果

(2)比较运算符

如表 2-5 所示，是 Scala 支持的比较运算符。

表 2-5 比较运算符

序号	名称	描述
1	>	大于
2	<	小于
3	>=	大于等于
4	<=	小于等于
5	==	等于
6	!=	不等于

【例 2-7】　比较运算符使用实例。

代码如下：

```
object TestScala extends App {

  var a = 10
  var b = 15



  var c = a > b
  println("a > b 计算结果:" + c)

  c = a < b
  println("a < b 计算结果:" + c)

  c = a >= b
  println("a >= b 计算结果:" + c)

  c = a <= b
  println("a <= b 计算结果:" + c)

  c = a == b
  println("a==b 计算结果:" + c)

  c = a != b
  println("a! =b 计算结果:" + c)
}
```

运行结果如图 2-36 所示，输出各比较运算方式的值。

```
a > b计算结果: false
a < b计算结果: true
a >= b计算结果: false
a <= b计算结果: true
a==b计算结果: false
a!=b计算结果: true
```

图 2-36　比较运算

(3)赋值运算符

如表 2-6 所示，是 Scala 支持的赋值运算符。

表 2-6　　赋值运算符

运算符名称	描述
=	赋值
+=	加等
-=	减等
*=	乘等
/=	除等
%=	模等

【例 2-8】　赋值运算符使用实例。

代码如下：

```
object TestScala extends App {

  var a = 10
  var b = 15
  var c = a + b
  println("经过 a + b 计算结果后 a 的值为:" + a)

  a += b
  println("经过 a += b 计算结果后 a 的值为:" + a)

  a -= b
  println("经过 a -= b 计算结果后 a 的值为:" + a)

  a *= b
  println("经过 a *= b 计算结果后 a 的值为:" + a)

  a /= b
  println("经过 a /= b 计算结果后 a 的值为:" + a)

  a %= b
  println("经过 a %= b 计算结果后 a 的值为:" + a)
}
```

运行结果如图 2-37 所示，输出各赋值运算方式的值。

```
经过a + b计算结果后a的值为：10
经过a += b计算结果后a的值为：25
经过a -= b计算结果后a的值为：10
经过a *= b计算结果后a的值为：150
经过a /= b计算结果后a的值为：10
经过a %= b计算结果后a的值为：10
```

图 2-37　赋值运算