网络编程与自动化

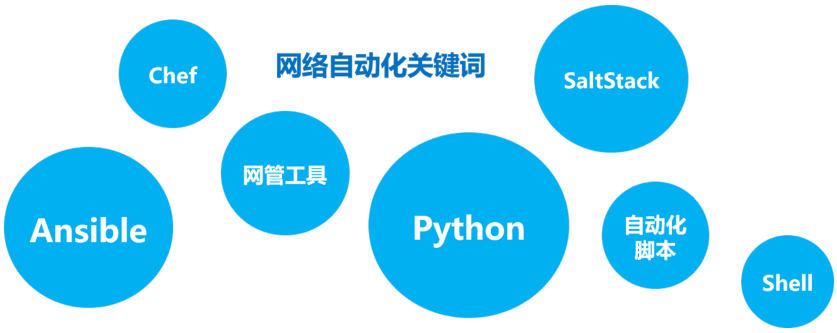
一、背景 — 传统网络运维困境

传统的网络运维工作需要网络工程师手动登录网络设备，人工查看和执行配置命令，肉眼筛选配置结果。这种严重依赖"人"的工作方式操作流程长，效率低下，而且操作过程不易审计

二、网络自动化

1、网络自动化，通过工具实现网络自动化地部署、运行和运维，逐步减少对"人"的依赖。这能够很好地解决传统网络运维的问题

2、业界有很多实现网络自动化的开源工具，例如Ansible、SaltStack、Puppet、Chef等。从网络工程能力构建的角度考虑，更推荐工程师具备代码编程能力



三、基于编程实现的网络自动化

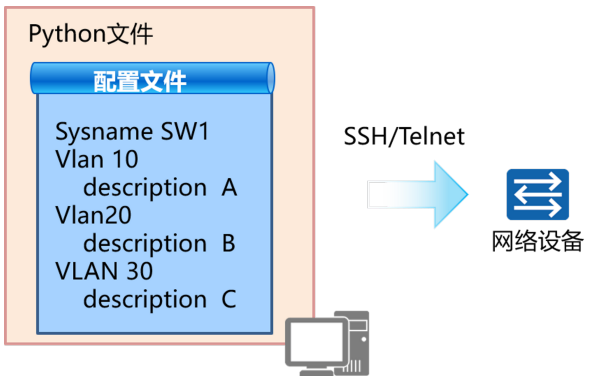
1、近几年随着网络自动化技术的兴起，以Python为主的编程能力成为了网络工程师的新技能要求

2、Python编写的自动化脚本能够很好的执行重复、耗时、有规则的操作

3、编程的过程可分为两个步骤：

3.1、编写配置文件

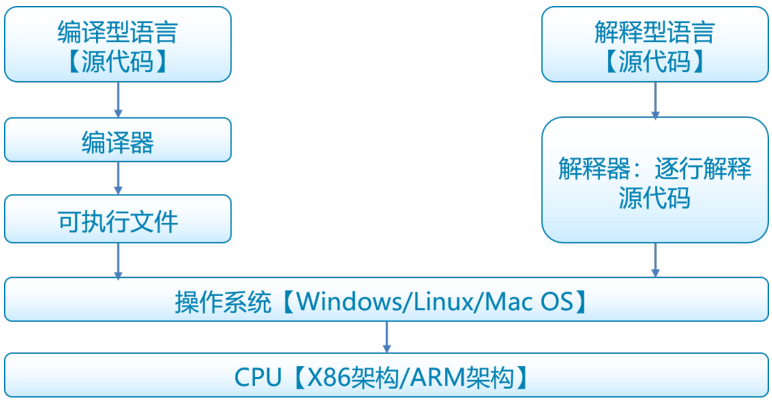
3.2、编写Python代码将配置文件推送到设备上



四、编程语言

1、编程语言【Programming Language】是一种用于编写计算机程序的语言，用于控制计算机的行为

2、按照语言在执行之前是否需要编译区分，可以将编程语言分为需要编译的编译型语言【Compiled Language】，不需要编译的解释型语言【Interpreted Language】

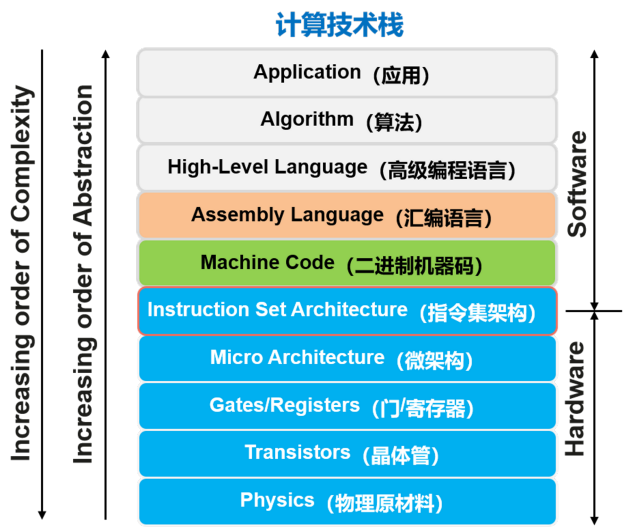


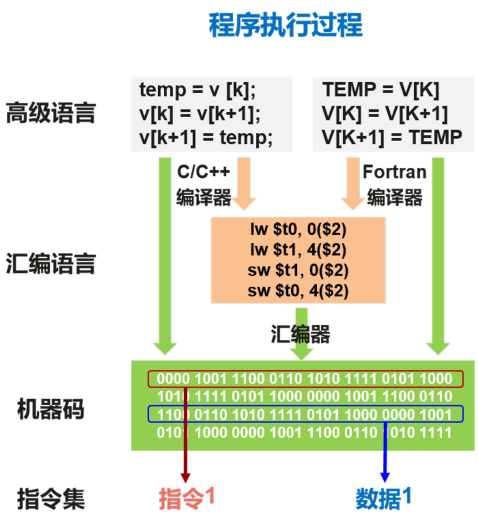
五、计算技术栈与程序执行过程

1、对于计算机的技术栈和程序执行的过程。左侧是计算的技术栈，我们可以看到硬件的最底层，是物理材料、晶体管来实现门电路和寄存器，再组成CPU的微架构。CPU的指令集是硬件和软件的接口，应用程序通过指令集中定义的指令驱动硬件完成计算

2、应用程序通过一定的软件算法完成业务功能。程序通常使用如C/C++/Java/Go/Python等高级语言开发。高级语言需要编译成汇编语言，再由汇编器按照CPU指令集转换成二进制的机器码

3、一个程序在磁盘上存在的形式，是一堆指令和数据所组成二进制机器码，也就是我们通常说的二进制文件

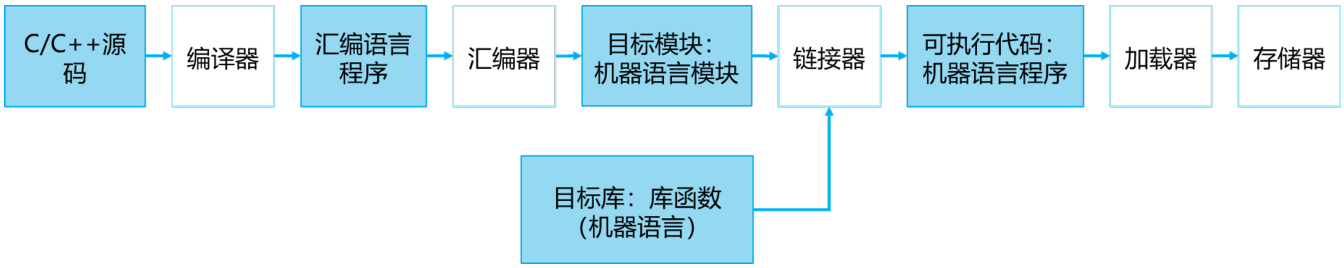




六、高级编程语言 — 编译型语言

1、编译型语言：编译型语言的程序在执行之前有一个编译过程，把程序编译成为机器语言的文件。运行时不需要重新翻译，直接使用编译的结果。典型的如C/C++/Go语言，都属于编译型语言

2、从源码到程序的过程：源码需要由编译器、汇编器翻译成机器指令，再通过链接器链接库函数生成机器语言程序。机器语言必须与CPU的指令集匹配，在运行时通过加载器加载到内存，由CPU执行指令

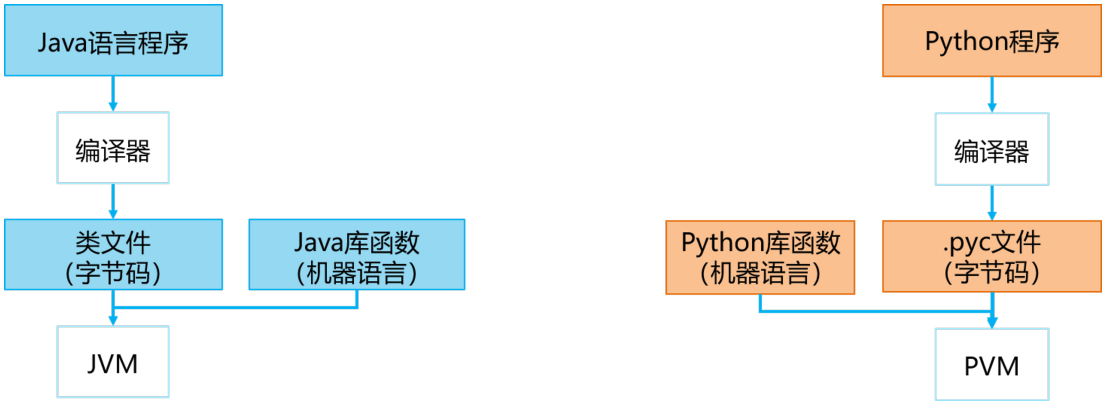


七、高级编程语言 — 解释型语言

1、解释型语言：解释型语言的程序不需要在运行前编译，在运行程序的时候才逐行翻译。典型的如Java/Python语言，都属于解释型语言

2、从源码到程序的过程：解释型语言的源代码由编译器生成字节码，然后再由虚拟机（JVM/PVM）解释执行。虚拟机将不同CPU指令集的差异屏蔽，因此解释型语言的可移植性相对较好

注：JVM：Java虚拟机、PVM：Python虚拟机



八、什么是Python

1、Python是一门完全开源的高级编程语言，它的作者是Guido Van Rossum

2、由于Python具有非常丰富的第三方库，加上Python语言本身的优点，所以Python可以在非常多的领域内使用：人工智能、数据科学、APP、自动化运维脚本等

3、Python的优点：

3.1、Python拥有优雅的语法、动态类型具有解释性质。能够让学习者从语法细节的学习中抽离，专注于程序逻辑

3.2、Python同时支持面向过程和面向对象的编程

3.3、Python拥有丰富的第三方库

3.4、Python可以调用其他语言所写的代码，又被称为胶水语言

4、Python的缺点：运行速度慢。Python是解释型语言，不需要编译即可运行。代码在运行时会逐行地翻译成CPU能理解的机器码，这个翻译过程非常耗时

九、Python代码执行过程

1、在操作系统上安装Python和运行环境

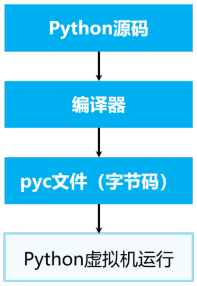
2、编写Python源码

3、编译器运行Python源码，编译生成pyc文件【字节码】

4、Python虚拟机将字节码转换为机器语言

5、硬件执行机器语言

Python程序编译运行的过程



十、初识Python代码 — 交互式运行

1、Python有两种运行方式，交互式运行和脚本式运行

2、交互式编程不需要创建脚本文件，是通过Python解释器的交互模式编写代码

```
C:\Users\Richard>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] ::
Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("hello world")
hello world
>>> a = 1
>>> b = 2
>>> print ( a + b )
3
>>>
```

1. Input --
2. Output --
3. Input --
4. Input --
5. Input --
6. Output --

十一、初识Python代码 — 脚本式运行

脚本模式里的代码可以在各种Python编译器或者集成开发环境上运行。例如Python自带的IDLE、Atom、Visual Studio、Pycharm和Anaconda等

demo.py文件

```
print ("hello world")
a = 1
b = 2
print ( a + b )
```

1. Input --
2. Output --
3. Output --

```
C:\Users\Richard>python demo.py
hello world
3
```

❶ 编写Python脚本文件(.py)　　❷ 执行脚本文件

十二、Python编码规范

1、编码规范是使用Python编写代码时应遵守的命名规则、代码缩进、代码和语句分割方式等。良好的编码规范有助于提高代码的可读性，便于代码的维护和修改

2、例如分号、圆括号、空行和空格的使用规范建议如下：

分号

- Python程序允许在行尾添加分号，但是不建议使用分号隔离语句
- 建议每条一句单独一行

空行

- 不同函数或语句块之间可以使用空格来分隔。用以区分两段代码，提高代码可读性

圆括号

- 圆括号可用于长语句的续行。一般不使用不必要的括号

空格

- 不建议在括号内使用空格
- 对于运算符，可以按照个人习惯决定是否在两侧加空格

十三、Python编码规范 — 标识符命名

1、Python标识符用于表示常量、变量、函数以及其他对象的名称

2、标识符通常由字母、数字和下划线组成，但不能以数字开头。标识符大小写敏感，不允许重名。如果标识符不符合规则，编译器运行代码时会输出SyntaxError语法错误

1. 数值赋值 --
2. 数值赋值 --
3. 字符串赋值 --
4. 数值赋值 --
5. 错误的标识符 --

```
User_ID = 10
user_id = 20
User_Name = 'Richard'
Count = 1 + 1
4_passwd = "Huawei"
```

```
print ( User_ID )
print ( user_id )
print ( User_Name )
print ( Count )
print ( 4_passwd )
```

print()为Python内置的函数，作用是输出括号内的内容

十四、Python编码规范 — 代码缩进

1、在Python程序中，代码缩进代表代码块的作用域。如果一个代码块包含两个或更多的语句，则这些语句必须具有相同的缩进量。对于Python而言代码缩进是一种语法规则，它使用代码缩进和冒号来区分代码之间的层次

2、编写代码时候，建议使用4个空格来生成缩进。如果程序代码中使用了错误的缩进，则会在运行中发出IndentationError错误信息

正确缩进 --

正确缩进 --

错误缩进 --

```
if True:
    print ("Hello")
else:
    print (0)

a = "Python"
    print (a)
```

十五、Python编码规范 — 使用注释

1、注释就是在程序中添加解释说明，能够增强程序的可读性。在Python程序中，注释分为单行注释和多行注释

2、单行注释以 # 字符开始直到行尾结束
3、多行注释内容可以包含多行，这些内容包含在一对三引号内（'''...'''或者"""..."""）

单行注释 --
多行注释 --

```
#将字符串赋值给a
a = “Python”
print (a)

“““
运行输出结果为Python
”””
```

十六、Python编码规范 — 源码文件结构
1、一个完整的Python源码文件一般包含几个组成部分：解释器和编码格式声明、文档字符串、模块导入和运行代码
2、如果会在程序中调用标准库或其他第三方库的类时，需要先使用import或from... import语句导入相关的模块。导入语句始终在文件的顶部。在模块注释或文档字符串【docstring】之后

解释器声明 --
编码格式声明 --
模块注释或文档字符串 --
导入模块time --
运行代码 --

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

“““本文档的说明（docstring）

本文档作用是…
”””

import time
…
```

十七、Python的函数与模块
1、函数【Function】是组织好的、可重复使用的一段代码。它能够提高程序的模块化程度和代码利用率。函数使用关键字【def】定义
2、模块【Module】是一个保存好的Python文件。模块可以由函数或者类组成。模块和常规Python程序之间的唯一区别是用途不同：模块用于被其他程序调用。因此，模块通常没有main函数

demo.py文件

```
def sit():      #定义函数
    print ('A dog is now sitting')

sit()           #调用函数
```

运行结果：

```
A dog is now sitting.
```

1 编写Python文件(.py)

test.py文件

```
import demo     #导入模块

demo.sit()      #调用函数
```

运行结果：

```
A dog is now sitting.
A dog is now sitting.
```

2 调用模块

十八、Python的类与方法
1、类【Class】是用来描述具有一类相同的属性和方法的集合。类的定义使用关键字 class
2、被实例化的类的”函数”被称作方法【Method】。类定义方法时候必须携带 self 关键字，它表示类的实例本身

demo.py文件

```
class Dog():      # 定义类
    def sit(self):  # 定义方法
        print("A dog is now sitting.")

Richard = Dog()    #实例化类
print (type(Richard.sit)) #实例化后类型为方法
print (type(Dog.sit))     #类型为函数
```

运行结果：

```
<class 'method'>
<class 'function'>
```

1 编写Python文件(.py)

test.py文件

```
import demo

demo.Dog.sit
```

运行结果：

```
A dog is now sitting.
<class 'method'>
<class 'function'>
```

2 调用模块

十九、telnetlib介绍

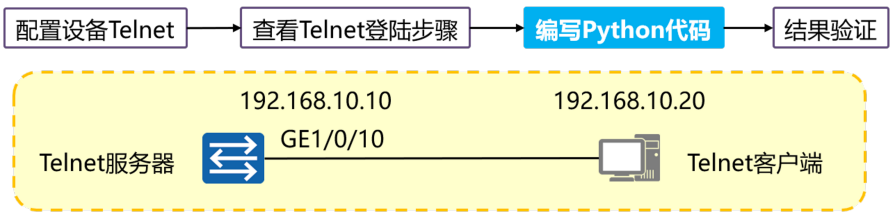
1、telnetlib是Python标准库中的模块。它提供了实现Telnet功能的类telnetlib.Telnet

2、这里通过调用telnetlib.Telnet类里的不同方法实现不同功能

```
导入telnetlib模块Telnet类  --  from telnetlib import Telnet
Telnet连接到指定服务器上  --  tn = Telnet(host=None, port=0[, timeout])
调用read_all()方法  --  tn.read_all()
                        …
```

方法	功能
Telnet.read_until（expected, timeout=None）	读取直到给定的字符串expected或超时秒数
Telnet.read_all ()	读取所有数据直到EOF(End Of File)。阻塞直到连接关闭
Telnet.read_very_eager()	读取从上次IO阻断到现在所有的内容，返回字节串。连接关闭或者没有数据时触发EOFError异常
Telnet.write(buffer)	写入数据。在套接字(Socket)上写一个字节串，加倍任何IAC(Interpret As Command)字符
Telnet.close()	关闭连接

二十、使用telnetlib登陆设备




```
导入模块                          --  import telnetlib
定义登录设备IP                    --  host = '192.168.10.10'
定义登录设备密码                  --  password = 'Huawei@123'

Telnet登录到主机                  --  tn = telnetlib.Telnet(host)
读取直到回显信息为" Password:"    --  tn.read_until(b'Password:')
输入编码为ASCII的密码并换行       --  tn.write(password.encode('ascii') + b"\n")
输出读取直到到" <Huawei>" 的信息  --  print (tn.read_until(b'<Huawei>').decode('ascii'))
关闭Telnet连接                    --  tn.close()
```

二十一、网络编程与自动化的配置

详细配置见实验手册