

Python学习笔记2

十二、Linux 基础命令

1. Ubuntu 操作系统

vmware 下载地址: [VMware](#)

Ubuntu 下载地址: [Ubuntu](#)

vmware 激活秘钥: (选择其一即可)

```
NZ4RR-FTK5H-H81C1-Q30QH-1V2LA
4A4RR-813DK-M81A9-4U35H-06KND
MC60H-DWHD5-H80U9-6V85M-8280D
JU090-6039P-08409-8J0QH-2YR7F
```

主要目录说明

- `/`: 根目录
- `/bin`: 可执行二进制文件目录
- `/etc`: 系统配置文件存放目录
- `/home` 用户家目录

2. 查看目录命令

`ls`: 查看当前目录信息

`tree`: 以树状图方式显示目录信息

`pwd`: 查看当前目录路径

`clear` 清除终端内容

`ctrl+alt+t`: 快速启动命令行/终端

3. 切换目录命令

`cd ..`: 返回上一级目录

`cd 目录`: 切换到指定目录

`cd ~` / `cd`: 切换到当前用户的主目录

`cd .`: 切换到当前目录

`cd -`: 切换到上一次目录

4. 绝对路径和相对路路径

1. 绝对路径

从根目录算起的路径叫做绝对路径

例如：

- `/home/huihuia24/桌面`
- `/bin`

2. 相对路径

从当前目录算起的路径

例如：

- `./tast/hello`
- `../static/images`

3. 绝对路径和相对路径的使用

- 绝对路径切换到桌面

```
cd /home/huihuia24/桌面
```

- 在下载目录以相对路径切换到桌面

```
cd Downloads
```

```
cd ../Desktop
```

- 在桌面使用相对路径谢欢到上一级目录的上一级目录

```
cd Desktop cd ../../
```

5. 创建、删除文件及目录命令

1. 创建。删除文件及目录命令的使用

`touch` 文件名：创建指定文件

`mkdir` 目录名：创建目录/文件夹

`rm` 文件名或目录名：删除指定文件或目录，删除文件的时候得加 `-r`

`rmdir` 目录名：删除空目录

6. 复制、移动文件及目录命令

`cp`：拷贝文件、拷贝目录，拷贝文件夹的时候加 `-r`

`mv`：移动文件、移动目录、重命名

7. 查看命令帮助

`--help`：命令 `--help`

`man`：man 命令

8.1s 命令选项

`-l`：以列表方式显示，简写：`ll`

`-h`：以大小单位显示，默认是字节

`-a`：显示隐藏文件和隐藏目录

9. mkdir 和 rm 命令选项

mkdir 命令选项

`-p`：创建所依赖的文件夹

```
mkdir aa/bb/cc -p
```

rm 命令选项

`-i`：交互式提示

```
rm 1.txt -i
```

`-r`：递归删除目录及其内容

```
rm aa -r
```

`-f`：强制删除，忽略不存在的文件，无需提示

```
rm 999.txt(不存在的文件) -f // 不会报错
```

`-d`：删除空目录

```
rm aaaa -d
```

10. cp 和 mv 命令选项

cp 命令选项

`-i`：交互式提示

```
cp 1.txt aa -i
```

`-r`：递归拷贝目录及其内容

```
cp aa bb -r
```

`-v`：显示拷贝后的路径描述

```
cp 1.txt aa -v
```

`-a`：保留文件原有权限

```
cp 1.txt -a
```

mv 命令选项

-i：交互式提示

```
mv 1.txt aa -i
```

-v：显示移动后的路径描述

```
mv 1.txt aa -v
```

11. 重定向命令

重定向也称为输出重定向，把在终端执行命令的结果保存到目标文件

重定向命令使用

>：如果文件存在会覆盖原有文件内容，相当于文件操作中的 w 模式

>>：如果文件存在会追加写入文件末尾，相当于，文件操作中的 a 模式

十三、Linux 高级命令

1. 查看文件内容命令

cat：查看小型文件

more：分屏查看大型文件

gedit：打开文件，可以修改内容

2. 管道 | 命令的使用

管道 |：一个命令的输出可以通过管道作为另一个命令的输入，可以理解成是一个容器，存放在终端显示的内容

```
ls /bin | more
```

3. 链接命令

链接命令是创建链接文件，链接分为：

- 软链接
- 硬链接

ln -s：创建软链接

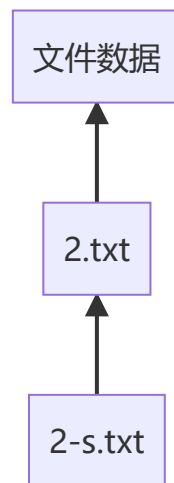
ln：创建硬链接

软链接

类似于windows下的快捷方式，当一个源文件的目录层级比较深，我们想要方便使用它可以给源文件创建一个软链接

例：为 aa/2.txt 创建软链接到桌面

```
ln -s aa/2.txt ./2-s.txt
```



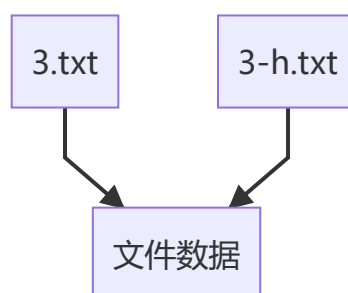
注：

- 如果软链接和源文件不在同一目录，源文件要用绝对路径，不能使用相对路径。
- 删除源文件则软链接失效
- 可以给目录创建软链接

硬链接

类似于源文件的一个别名，也就是说这两个名字指向的是同一个文件数据(相当于直接复制了一份，但是里面数据是同步的)

```
ln 3.txt 3-h.txt
```



注：

- 创建硬链接使用相对路径和绝对路径都可以
- 删除源文件，硬件链接还可以访问到数据
- 创建硬链接，硬链接数会加1，删除源文件或者硬链接，硬链接数会减1
- 创建软链接，硬链接数不会加1
- 不能给目录创建硬链接

4. 文本搜索命令

grep 命令使用

grep：文本搜索

```
grep "aa" aa/2.txt
```

-i：忽略大小写

-n：显示匹配行号

-v：显示不包含匹配文本的所有行

grep 命令结合正则表达式的使用

```
# 2.txt  
aa,bb,cc  
ssdfs  
safdsa  
adf  
asdfsdf
```

^：指定字符串开头

```
grep "^a" aa/2.txt
```

\$：指定字符串结尾

```
grep "a$" aa/2.txt
```

.

```
grep "a.d" aa/2.txt
```

5. 查找文件命令

find：在指定目录下查找文件(包括目录)

find 命令选项

-name：根据文件名(包括目录名)字查找

```
find . -name 2.txt
```

find 命令结合通配符使用

*：表示0个或多个任意字符

```
find . -name "*.txt"
```

?: 表示任意一个字符

```
find . -name "*.txt"
```

6. 压缩和解压缩

Linux 默认支持的压缩格式

- .gz
- .bz2
- .zip

说明：

- .gz 和 bz2 的压缩包需要使用 tar 命令来压缩和解压缩的
- .zip 的压缩包需要使用 zip 命令来压缩，使用 unzip 命令来解压缩

tar 命令及选项的使用

tar：压缩和解压命令

tar 命令选项

- c：创建打包文件
- v：显示打包或者解包的详细信息
- f：指定文件名称，必须放到所有选项后面
- z：压缩或解压缩(.gz)
- j：压缩或解压缩(.bz2)
- x：解包
- C：解压缩到指定目录

zip 和 unzip 命令及使用

zip：压缩成 .zip 格式文件

unzip：解压缩 .zip 格式文件

unzip 命令选项

- d：解压缩到指定目录

7. 文件权限命令

chmod 命令

chmod：修改文件权限

chmod 修改文件权限有两种方式

- 字母法
- 数字法

chmod 字母法的使用

角色说明

角色	说明
u	user，表示该文件的所有者
g	group，表示用户组
o	other，表示其他用户组
a	all，表示所有用户组

权限设置说明：

操作符	说明
+	增加权限
-	撤销权限
=	设置权限

权限说明：

权限	说明
r	可读
w	可写
x	可执行
-	无任何权限

```
chmod u-r aa/2.txt
chmod u+r aa/2.txt
chmod u+x info.py
```

chmod 数字法的使用

数字法就是 rwx 这些权限也可以使用数字来代替

权限	说明
r	可读，权限值是4
w	可写，权限值是2
x	可执行，权限值是1
-	无任何权限，权限值是0

```
chmod 000 1.txt
chmod 777 1.txt
chmod 644 1.txt
```

8. 获取管理员权限的相关命令

sudo 命令的使用

`sudo -s`：切换到root用户，获取管理员权限

`sudo`：某个命令的执行需要获取管理员权限可以在执行命令前面加上 `sudo`

`whoami`：查看当前用户

`exit`：退出登录用户

`who`：查看所有的登录用户

`passwd`：修改用户密码，不指定用户默认修改当前登录用户密码

`which`：查看命令位置

`shutdown -h now`：立即关机

`reboot`：重启

9. 用户相关操作

创建用户

`useradd`：创建(添加)用户

命令选项

`-m`：自动创建用户主目录，主目录的名字就是用户名

`-g`：指定用户所属用户组，默认不指定会自动创建一个同名的用户组

设置密码

给其他用户设置密码，需要使用 `sudo passwd 用户名`

切换用户

`su`：切换用户

语法 `su - 用户名`

修改用户信息

`usermod`：修改用户信息

选项

`-G`：设置一个附加组

`-g`：修改用户组

删除附加组

`gpasswd`：添加和删除附加组信息

选项

`-a 用户名`：给用户添加附加组

`-d 用户名`：给用户删除附加组

删除用户

`userdel`：删除用户

选项

`-r` 用户名：删除用户主目录，必须要设置，否则用户主目录不会删除

10. 用户组相关操作

`groupadd`：创建(添加)用户组

`groupdel`：删除用户组

11. 远程登录、远程拷贝命令

`ssh`：远程登录

需要ssh软件

xshell

windows下载地址：<https://www.xshell.com/zh/free-for-home-school/>

ubuntu命令

```
sudo apt-get install ssh    # 安装ssh服务
service ssh start
ps -e|grep ssh              # 启动ssh服务
```

openssh

windows下载地址：<https://www.mls-software.com/opensshd.html>

Windows链接远程 Ubuntu 在命令提示符输入

```
ssh 用户名@IP地址
```

拓展：

```
apt list
```

`scp`：远程拷贝

可以使用软件如：`xftp`

windows下载链接：<https://www.xshell.com/zh/free-for-home-school/>

在命令提示符中使用 `scp` 远程拷贝文件

```
scp 文件名 用户名@IP地址:/文件存储的绝对地址
```

例：

从本地拷至服务器

```
scp python学习.pdf huihuia24@192.168.11.129:/home/huihuia24/桌面
```

从服务器拷至本地

```
scp -r huihuia24@192.168.11.129:/home/huihuia24/桌面/aa .
```

十四、多任务编程

1. 编辑器 vim

介绍

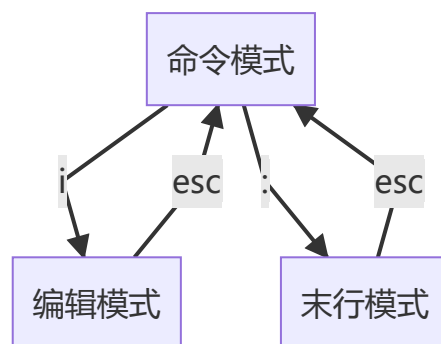
vim 是一款功能强大的文本编辑器，也是早年 vi 编辑器的加强版，它的最大特色就是使用命令进行编辑完全脱离了鼠标的操作。

vim 工作模式

- 命令模式
- 编辑模式
- 末行模式

注：： vim 打开文件进入的是命令模式

安装： `sudo apt install vim`



注：编辑模式和末行模式之间不能之间进行切换，都需要通过命令模式来完成

vim 的末行模式命令

- `:wq`：保存退出
- `:w`：保存
- `:x`：保存退出
- `:q!`：强制退出

vim 的常用命令

命令	说明
<code>yy</code>	复制光标所在行
<code>p</code>	粘贴
<code>dd</code>	删除/剪切当前行
<code>v</code>	按行选中
<code>u</code>	撤销
<code>ctrl+r</code>	反撤销
<code>>></code>	往右缩进
<code><<</code>	往左缩进
<code>:/搜索的内容</code>	搜索指定内容
<code>:%s/要替换的内/替换后的内容/g</code>	全局替换
<code>:开始行数,结束行数s/要替换的内容/替换后的内容</code>	局部替换
<code>.</code>	重复上一次命令操作
<code>G</code>	回到最后一行
<code>gg</code>	回到第一行
<code>数字+G</code>	回到指定行
<code>shift+6</code>	回到当前行的行首
<code>shift+4</code>	回到当前行的行末
<code>ctrl+f</code>	下一屏
<code>ctrl+b</code>	上一屏

2. 软件安装

软件安装的两种方式：

- 离线安装(`deb` 文件格式安装)
- 在线安装(`apt-get` 方式安装)

拓展：各系统安装包后缀

Windows： `.exe`

MacOS： `.dmg`

Ubuntu： `.deb`

`deb` 文件格式安装

`deb` 是 Ubuntu 的安装包格式，可以使用 `dpkg` 命令进行软件的安装和卸载

`dpkg`：安装和卸载 deb 安装包

命令选项

`-i`：离线安装 deb 安装包

`apt-get` 方式安装

在线安装deb软件包的命令，主要用于在线从互联网的软件仓中**搜索、安装、升级、卸载**软件

在线安装软件命令格式：

- `sudo apt-get install 安装包`，表示在线安装deb安装包

3. 更改镜像源

更改镜像源的方式

1. 可视化方式更改镜像源
2. 手动方式更改镜像源

可视化方式更改镜像源

设置-->系统-->软件更新

手动方式

备份软件源列表（以防后面使用过程中源有问题更换回来）

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list_backup
```

打开软件源列表文件

```
sudo vim /etc/apt/sources.list
```

保存后就可以开始更新软件列表信息了，输入以下指令

```
sudo apt-get update
```

这个命令，会访问源列表里的每个网址，并读取软件列表，然后保存在本地电脑。我们在软件包管理器里看到的软件列表，都是通过update命令更新的。

update后，可能需要upgrade一下。

```
sudo apt-get upgrade
```

总而言之，`update` 是更新软件列表，`upgrade` 是更新软件

4. 软件卸载

卸载方式有两种：

- 离线安装包的卸载(deb文件格式卸载)
- 在线安装包的卸载和(apt-get方式卸载)

deb 文件格式卸载

命令格式：

```
sudo dpkg -r 安装包名
```

-r 选项表示安装的卸载

apt-get 方式卸载

命令格式：

```
sudo apt-get remove 安装包名
```

5. 多任务的介绍

多任务最大的好处是**充分利用CPU资源，提高程序的执行效率**

多任务是指在**同一时间内执行多个任务**，例如：现在电脑安装的操作系统都是多任务操作系统，可以同时运行着多个软件。

多任务的执行方式

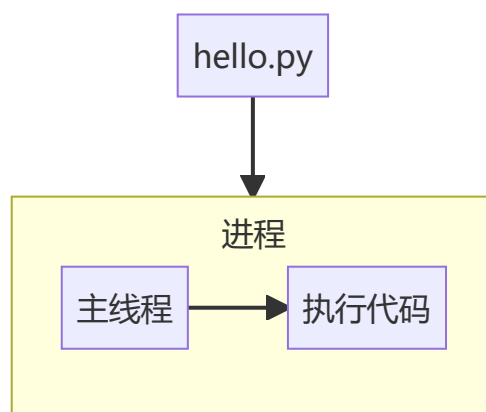
- 并发
在一段时间内交替去执行任务
- 并行
对于多核CPU处理多任务，操作系统会给CPU的每个内核安排一个执行的软件，**多个内核是真正的一起执行软件**。这里需要注意**多核CPU是并行的执行多任务，始终有多个软件一起执行**。

6. 进程

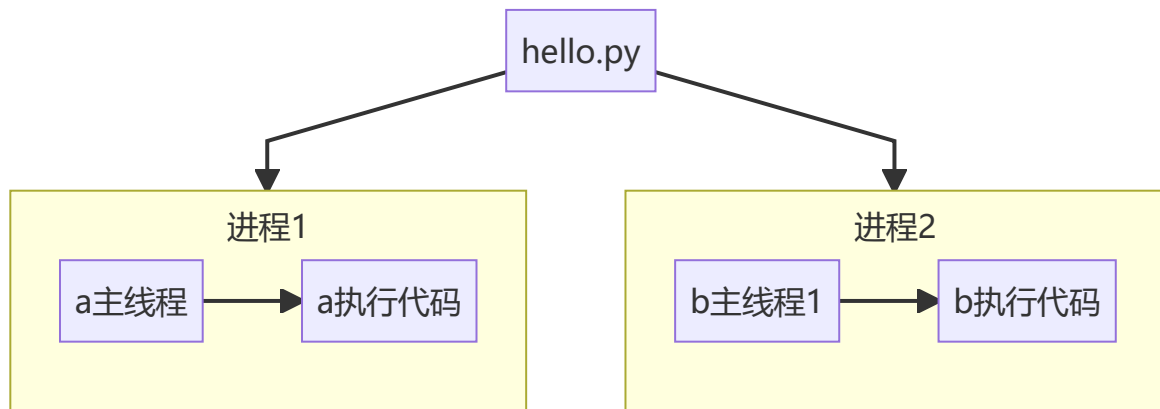
注：一个程序运行后至少有一个进程，一个进程默认有一个线程，进程里面可以创建多个线程，**线程是依附在进程里面的，没有进程就没有线程**。

进程的作用

单进程效果图



多线程效果图



7. 多进程的使用

导入进程包

```
import multiprocessing
```

process 进程类的说明

```
process([group[,target[,name[,args[,kwargs]]]])
```

- `group`: 指定进程组，目前只能使用None
- `target`: 执行的目标任务名
- `name`: 进程名字
- `args`: 以元组的方式给执行任务传参
- `kwargs`: 以字典方式给执行任务传参

process 创建实例对象的常用方法

- `start()`: 启动子进程实例(创建子进程)
- `join`: 等待子进程执行结束
- `terminate`: 不管任务是否完成，立即终止子进程

process 创建实例对象的常用属性

`name`: 当前进程的别名，默认为Process-N，N为从1开始递增的整数

实战:

```
# 导入进程包
import multiprocessing
# 跳舞任务
def dance():
    for i in range(3):
        print('跳舞')

# 唱歌
def change():
    for i in range(3):
        print('唱歌')

# 创建子进程
dance_process = multiprocessing.Process(target = dance)
change_process = multiprocessing.Process(target = change)
```

```
# windows 系统下使用 multiprocessing 时，必须将进程创建 / 启动的代码放在 if __name__ ==
"__main__": 代码块中。
if __name__ == '__main__':
    # 启动进程，执行对应的任务
    dance_process.start()
    change_process.start()
    # 主进程执行唱歌任务
    change()
```

注：进程执行是无序的，具体哪个进程先执行是由操作系统调度决定

8. 获取进程编号

获取进程编号的目的

获取进程编号的目的是验证主进程和子进程的关系，可以得知子进程是由那个主进程创建出来的。

获取进程编号的两种操作：

- 获取当前进程编号

`os.getpid()` 表示获取当前的进程编号

```
# 导入进程包
import multiprocessing
import os
# 跳舞任务
def dance():
    for i in range(3):
        print('跳舞')

# 唱歌
def change():
    for i in range(3):
        print('唱歌')

# 创建子进程
dance_process = multiprocessing.Process(target = dance)
change_process = multiprocessing.Process(target = change)

# 获取当前进程编号
main_process_id = os.getpid()
# 获取当前进程对象，查看当前代码是由那个进程执行的：multiprocessing.current_process()
print(main_process_id, multiprocessing.current_process())

# windows 系统下使用 multiprocessing 时，必须将进程创建 / 启动的代码放在 if
__name__ == "__main__": 代码块中。
if __name__ == '__main__':
    # 启动进程，执行对应的任务
    dance_process.start()
    change_process.start()

    # 主进程执行唱歌任务
    change()
```

- 获取当前父进程编号

`os.getppid()` 表示获取当前父进程编号

```
# 导入进程包
import multiprocessing
import os

# 跳舞任务
def dance():
    # 获取当前父进程编号
    dance_process_id = os.getppid()
    print(dance_process_id)
    for i in range(3):
        print('跳舞')

# 唱歌
def changge():
    # 获取当前子进程
    changge_process_id = os.getpid()
    print(changge_process_id)
    for i in range(3):
        print('唱歌')
    # 拓展，更具进程编号强制杀死
    os.kill(changge_process_id, 9)

# 创建子进程
dance_process = multiprocessing.Process(target = dance)
changge_process = multiprocessing.Process(target = changge)

# 获取当前进程编号
main_process_id = os.getpid()
# 获取当前进程对象，查看当前代码是由那个进程执行的: multiprocessing.current_process()
print(main_process_id, multiprocessing.current_process())

# windows 系统下使用 multiprocessing 时，必须将进程创建 / 启动的代码放在 if
__name__ == "__main__": 代码块中。
if __name__ == '__main__':
    # 启动进程，执行对应的任务
    dance_process.start()
    changge_process.start()

    # 主进程执行唱歌任务
    changge()
```

9. 进程执行带有参数的任务

`Process` 类执行任务并给任务传参数有两种方式：

- `args`：以元组的方式给执行任务传参
- `kwargs`：以字典方式给执行任务传参

`args` 参数的使用

```
import multiprocessing

def show_info(name,age):
    print(f"姓名{name}今年{age}岁。")

if __name__ == '__main__':
    pool = multiprocessing.Process(target=show_info, args=('张三',20))
    pool.start()
```

注：以元组方式传参，元组里面的元素顺序要和函数的参数顺序保持一致

kwargs 参数使用

```
import multiprocessing

def show_info(name,age):
    print(f"姓名{name}今年{age}岁。")

if __name__ == '__main__':
    pool = multiprocessing.Process(target=show_info, kwargs={'name':'张三','age':20})
    pool.start()
```

注：以字典方式传参，字典里面的key名要和函数的参数名保持一致，没有顺序要求

使用两种方式同时传参

```
import multiprocessing

def show_info(name,age):
    print(f"姓名{name}今年{age}岁。")

if __name__ == '__main__':
    pool = multiprocessing.Process(target=show_info, args=("张三",), kwargs={'age':20})
    pool.start()
```

10. 进程的注意点

1. 进程之间不共享全局变量

```
import multiprocessing

list1 = []

def add_data():
    for i in range(3):
        list1.append(i)
        print(f'添加: {i}')
    print(list1)

def read_data():
    print('子进程二中的列表内容: ',list1)

if __name__ == '__main__':
```

```

add = multiprocessing.Process(target=add_data)
read = multiprocessing.Process(target=read_data)
add.start()
# 当前进程（主进程）等待添加数据的进程执行完成后再继续往下执行
add.join()
read.start()

```

注：创建子进程其实就是对主进程资源进行拷贝，子进程其实就是主进程的一个副本

2. 主进程会等待所有子进程执行结束再结束

```

import multiprocessing
from time import sleep

def task():
    for i in range(10):
        print(i)
        sleep(0.2)

if __name__ == '__main__':
    add = multiprocessing.Process(target=task)
    add.start()
    sleep(0.5)
    print('结束')

```

解决办法：

1. 让子进程设置成为守护主进程，主进程退出子进程销毁，子进程会依赖主进程

```

import multiprocessing
from time import sleep

def task():
    while True:
        print(1)
        sleep(0.2)

if __name__ == '__main__':
    add = multiprocessing.Process(target=task)
    # 把子进程设置成为守护主进程，以后主进程退出子进程直接销毁
    add.daemon = True
    add.start()
    sleep(0.5)
    print('结束')

```

2. 让主进程退出之前先让子进程销毁

```

import multiprocessing
from time import sleep

def task():
    while True:
        print(1)
        sleep(0.2)

```

```

if __name__ == '__main__':
    add = multiprocessing.Process(target=task)
    add.start()
    sleep(1)
    # 主进程退出之前先让子进程销毁
    add.terminate()
    print('结束')

```

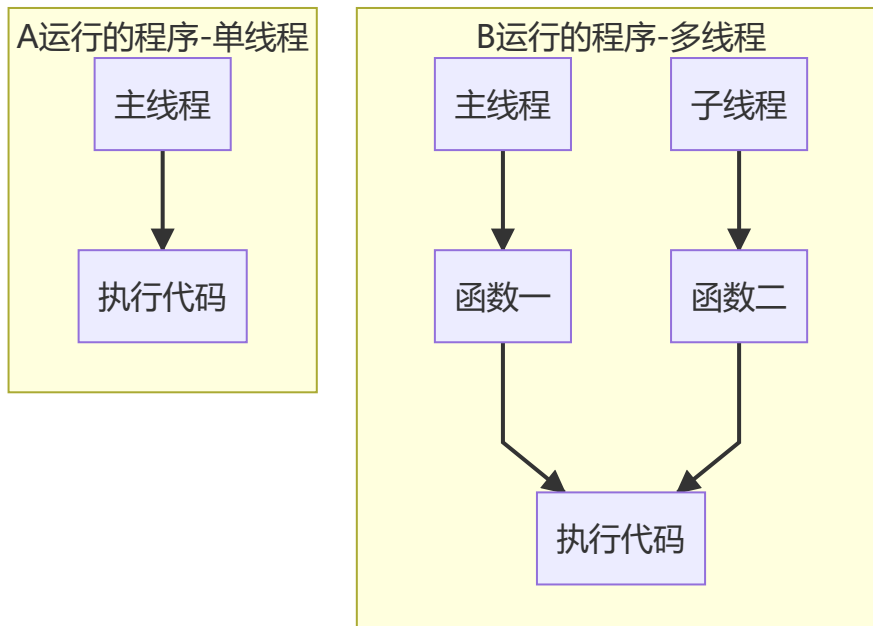
十五、多任务编程-2

1. 线程

线程是进程中执行代码的一个分支，每个执行分支(线程)要想工作执行代码需要CPU进行调度，也就是说线程是CPU调度的基本单位，每个进程至少都有一个线程，而这个线程就是我们通常说的主线程。

线程的作用

多线程效果图：



注：程序启动默认会有一个主线程，程序员自己创建的线程可以称为子进程，多线程可以完成多任务

2. 多线程的使用

导入线程模块

```
import threading
```

线程类 Thread 参数说明

```
Thread([group [,target [, name [, kwargs]]]])
```

- `group`：线程组，目前只能使用None
- `target`：执行的目标任务名
- `args`：以元组的方式给执行任务传参
- `kwargs`：以字典的方式给执行任务传参
- `name`：线程名，一般不设置

启动线程

启动线程使用 start 方法

多线程完成多任务实战

```
import threading

def sing():
    # 获取当前线程
    sing_id = threading.current_thread()
    print(sing_id)
    for i in range(1,6):
        print(i, '唱歌。。')

def wu():
    # 获取当前线程
    wu_id = threading.current_thread()
    print(wu_id)
    for i in range(1,6):
        print(i, '跳舞。。')

if __name__ == '__main__':
    main_id = threading.current_thread()
    print(main_id)
    a = threading.Thread(target=sing)
    b = threading.Thread(target=wu)
    a.start()
    b.start()
```

3. 线程执行带有参数的任务

Thread类执行任务并给任务传参数有两种方式：

- args：以元组的方式给执行任务传参
- kwargs：以字典的方式给执行任务传参

args 参数使用

```
import threading

def user_info(name, age):
    print(f'我叫{name}, 今年{age}岁。')

if __name__ == '__main__':
    user_info = threading.Thread(target=user_info, args=('张三', 10))
    user_info.start()
```

kwargs 参数使用

```
import threading

def user_info(name, age):
    print(f'我叫{name}, 今年{age}岁。')

if __name__ == '__main__':
    user_info = threading.Thread(target=user_info, kwargs={'name': '张三', 'age': 18})
    user_info.start()
```

同时使用

```
import threading

def user_info(name, age):
    print(f'我叫{name}, 今年{age}岁。')

if __name__ == '__main__':
    user_info = threading.Thread(target=user_info, args=('张三',), kwargs={'age': 18})
    user_info.start()
```

4. 线程的注意点

1. 线程之间执行是无序的

```
import threading

def task():
    print(threading.current_thread())

if __name__ == '__main__':
    for i in range(10):
        threading.Thread(target=task).start()
```

2. 主线程会等待所有子线程执行结束再结束

```
import threading
import time

def task():
    for i in range(100):
        print(i)
        time.sleep(0.5)

if __name__ == '__main__':
    threading.Thread(target=task).start()
    time.sleep(1)
    print('退出')
    exit()
```

解决方法

创建子线程守护主线程

```
import threading
import time

def task():
    for i in range(100):
        print(i)
        time.sleep(0.2)

if __name__ == '__main__':
    # 创建子线程守护
    threading.Thread(target=task, daemon = True).start()
    time.sleep(1)
    print('退出')
```

3. 线程之间共享全局变量

```
import threading
import time

list1 = []
def add():
    for i in range(4):
        list1.append(i)
        print('添加: ',i)

def pin():
    print(list1)
if __name__ == '__main__':
    add_info = threading.Thread(target=add)
    pin_info = threading.Thread(target=pin)
    add_info.start()
    # 当前线程等待添加数据的子线程执行完成以后代码在继续执行
    add_info.join()
    pin_info.start()
```

4. 线程之间共享全局变量数据出现错误问题

需求:

1. 定义两个函数，实现循环100万次，每循环1次给全局变量加1
2. 常见两个子线程执行对应的两个函数，查看计算后的结果

```
import threading

num = 0

def add_num():
    for i in range(1000000):
        global num
        num += 1
        print(f'1.结果是:{num}')

def add_num1():
```

```

for i in range(1000000):
    global num # 表示声明要修改全局变量的内存地址
    num += 1
print(f'2.结果是:{num}')

if __name__ == '__main__':
    add_num = threading.Thread(target=add_num)
    add_num1 = threading.Thread(target=add_num1)
    add_num.start()
    # 解决方法
    add_num.join()
    add_num1.start()

```

5. 互斥锁

互斥锁：对共享数据进行锁定，保证同一时刻只能有一个线程去操作

注：

- 互斥锁是多个线程一起去抢，抢到锁的线程执行，没有抢到锁的线程需要等待，等互斥锁使用完释放后，其它等待的线程再去抢这个锁。

互斥锁的使用

`threading` 模块中定义了Lock变量，这个变量本质上是一个函数，通过调用这个函数可以获得一把互斥锁

```

# 创建互斥锁
num = threading.Lock()

# 上锁
num.acquire()

# 释放锁
num.release()

```

注：

- `acquire` 和 `release` 方法之间的代码同一时刻只能有一个线程去操作
- 如果在调用 `acquire` 方法的时候 其他线程已经使用了这个互斥锁，那么此时 `acquire` 方法会堵塞，直到这个互斥锁释放后才能再次上锁。

解决循环100万次数据错误问题

```

import threading

num = 0
# 创建互斥锁
app = threading.Lock()

def add_num():
    # 上锁
    app.acquire()
    for i in range(1000000):
        global num

```

```

        num += 1
    print(f'1.结果是:{num}')
    # 释放锁
    app.release()

def add_num1():
    # 上锁
    app.acquire()
    for i in range(1000000):
        global num # 表示声明要修改全局变量的内存地址
        num += 1
    print(f'2.结果是:{num}')
    # 释放锁
    app.release()

if __name__ == '__main__':
    add_num = threading.Thread(target=add_num)
    add_num1 = threading.Thread(target=add_num1)
    add_num.start()
    add_num1.start()

```

互斥锁可以保证同一时刻只有一个线程去执行代码，能够保证全局变量的数据没有问题

线程等待和互斥锁都是把多任务改成单任务去执行，保证了数据的准确性，但是执行性能会下降

6. 死锁

```

import threading

lock = threading.Lock()

def pas(num):
    # 上锁
    lock.acquire()
    list1 = [3,4,99,7]
    if num >= len(list1):
        print('越界',num)
        # 取值不成功，也需要释放互斥锁，不要影响后面的线程去取值
        lock.release()
        return

    value = list1[num]
    print(value)
    # 释放锁
    lock.release()

if __name__ == '__main__':
    for i in range(10):
        das = threading.Thread(target= pas,args=(i,))
        das.start()

```

7. 进程和线程的对比

1. 关系对比

1. 线程是依附在进程里面的，没有进程就没有线程
2. 一个进程默认提供一条线程，进程可以创建多个线程

2. 区别对比

1. 进程之间不共享全局变量
2. 线程之间共享全局变量，但是要注意资源竞争问题，解决办法：互斥锁或者线程同步
3. 创建进程的资源开销比创建线程的资源开销要大
4. 进程是操作系统资源分配的基本单位，线程是CPU调度的基本单位
5. 线程不能独立执行，必须依存在进程中
6. 多进程开发比单进程多线程稳定性要强

3. 优缺点对比

- 进程优缺点
 - 优点：可以用多核
 - 缺点：资源开销大
- 线程优缺点
 - 优点：资源开销小
 - 缺点：不能使用多核

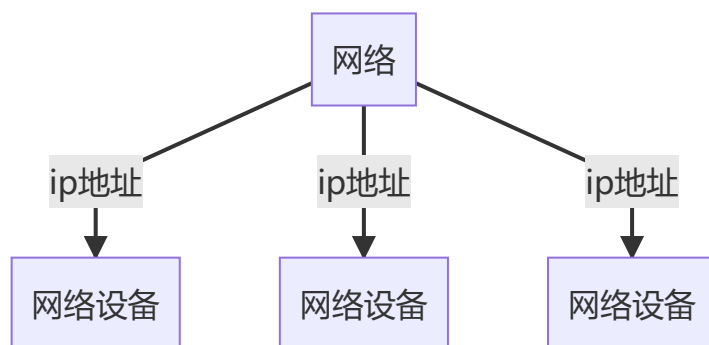
十六、网络编程

1. IP地址的介绍

IP地址的概念

IP地址就是**标识网络中设备的一个地址**

网络中的设备效果图



IP地址的表现形式

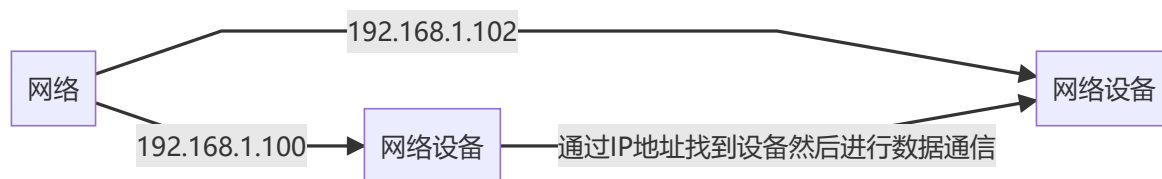
说明

- IP地址分为两类：**IPv4**和**IPv6**
- IPv4是目前使用的IP地址
- IPv6是未来使用的IP地址
- IPv4是由点分十进制组成
- IPv6是由冒号十六进制组成

IP地址的作用

IP地址的作用是**标识网络中唯一的一台设备的**，也就是说通过IP地址能够找到网络中的某台设备

IP地址作用效果图



查看IP地址

- Linux和mac OS使用 `ifconfig` 这个命令
- Windows使用 `ipconfig` 这个命令

说明：

- 192.168.1.107是设备在网络中的IP地址
- 127.0.0.1表示本机地址，提示：如果和自己电脑通信就可以使用该地址
- 127.0.0.1该地址对应的域名是 `localhost` 域名是IP地址的别名，通过域名解析出一个对应的IP地址

检查网络是否正常

- 检查网络是否正常使用 `ping` 命令

说明：

- `ping www.baidu.com` 检查是否能上公网
- `ping` 当前局域网IP地址，检查是否在同一网段
- `ping 127.0.0.1` 检查本地网卡是否正常

2.端口和端口号的介绍

每运行一个网络程序都会有一个端口，想要给对应的程序发送数据，找到对应的端口即可

什么是端口

端口是传输数据的通道，是数据传输的必经之路

其实，**每个端口都会有对应的端口号**，想要找到端口通过端口号即可

什么是端口号

操作系统为了统一管理这么多端口，**就对端口进行了编号**，这就是端口号，**端口号其实就是一个数字**

端口号有65536个

端口和端口号的关系

端口号可以标识唯一的一个端口

端口号的分类

- 知名端口号

知名端口号是指众所周知的端口号，范围从0到1023

这些端口号一般固定分配给一些服务，比如21端口分配给FTP(文件传输协议)服务，25端口分给SMTP(简易邮件传输协议)服务，80端口分配给HTTP服务，3389端口(远程连接协议)

- 动态端口号

一般程序员开发应用程序使用端口号称为动态端口号，范围是1024到65535

- 如果程序员开发的程序没有设置端口号，操作系统会在冬天端口号这个范围随机生成一个给开发的应用程序使用
- 当运行一个程序默认会有一个端口号，当这个程序退出时，所占用的这个端口号就会被释放

3. TCP的介绍

TCP的概念

TCP 的英文全拼(Transmission Control Protocol)简称**传输控制协议**，它是一种面向**连接的**、**可靠的**、**基于字节流的传输层通信协议**。

TCP通信步骤

1. 创建连接
2. 传输数据
3. 关闭连接

TCP的特点

1. 面向连接
 - 通信双方必须先建立好连接才能进行数据的传输，数据传输完成后，双方必须断开此连接，以释放系统资源。
2. 可靠传输
 - TCP 采用发送应答机制
 - 超时重传
 - 错误校验
 - 流量控制和阻塞管理

4. socket的介绍

socket的概念

socket (简称 套接字)是**进程之间通信一个工具**，好比现实生活中的插座，所有的家用电器要想工作都是基于插座进行，**进程之间想要进行网络通信需要基于这个socket**

socket的作用

负责进程之间的网络数据传输，好比数据的搬运工

socket使用场景

网络相关的应用程序或软件都用到了socket

总结:

进程之间网络数据的传输可以通过 socket 来完成, **socket 就是进程间网络数据通信的工具**

5. TCP网络应用程序开发流程

TCP网络应用程序开发流程的介绍

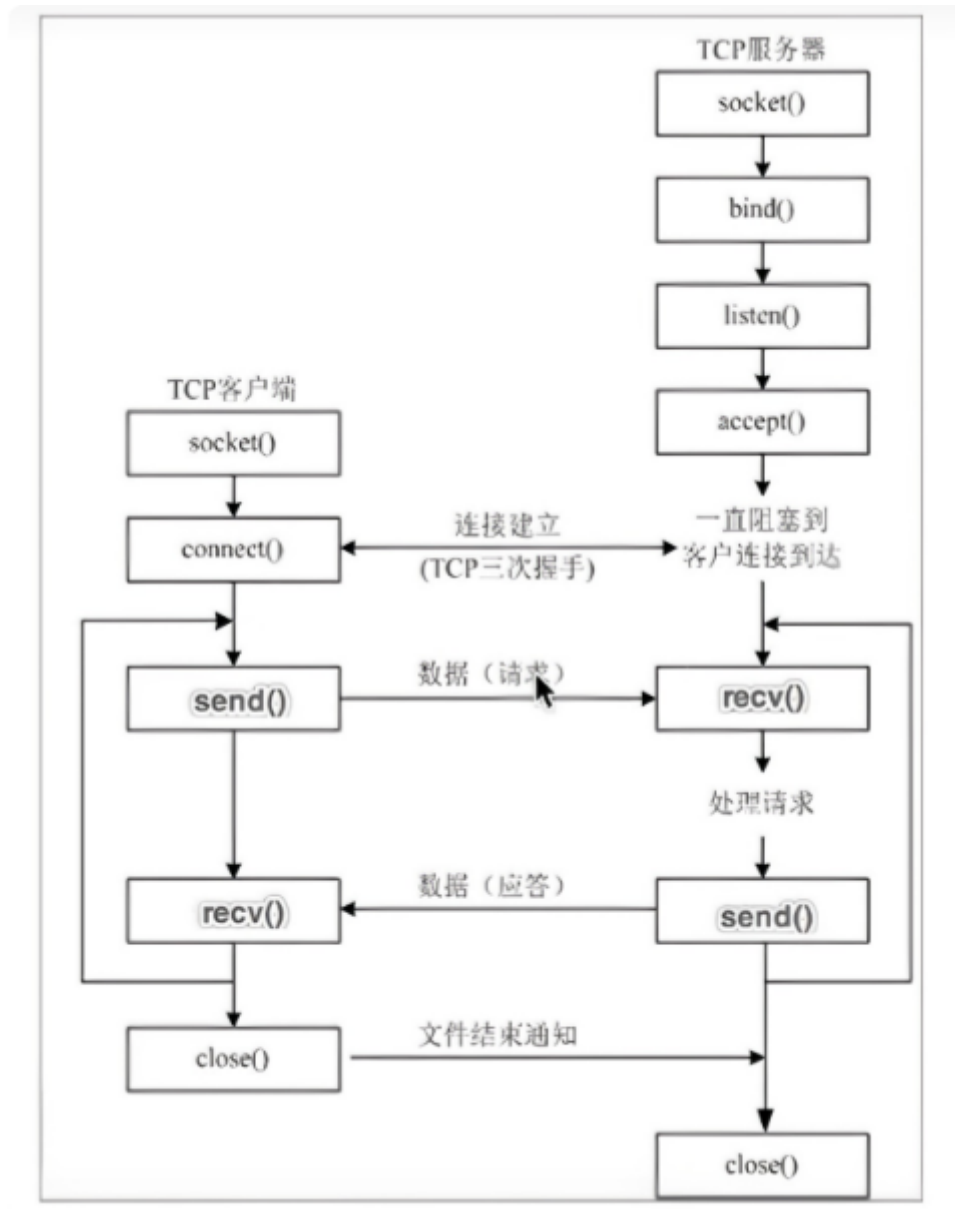
TCP网络应用程序开发分为:

- TCP客户端程序开发
- TCP服务端程序开发

说明:

客户端程序是指运行在用户设备上的程序, 服务端程序是指运行在服务器设备上的程序, 专门为客户端提供数据服务。

TCP客户端程序开发流程介绍

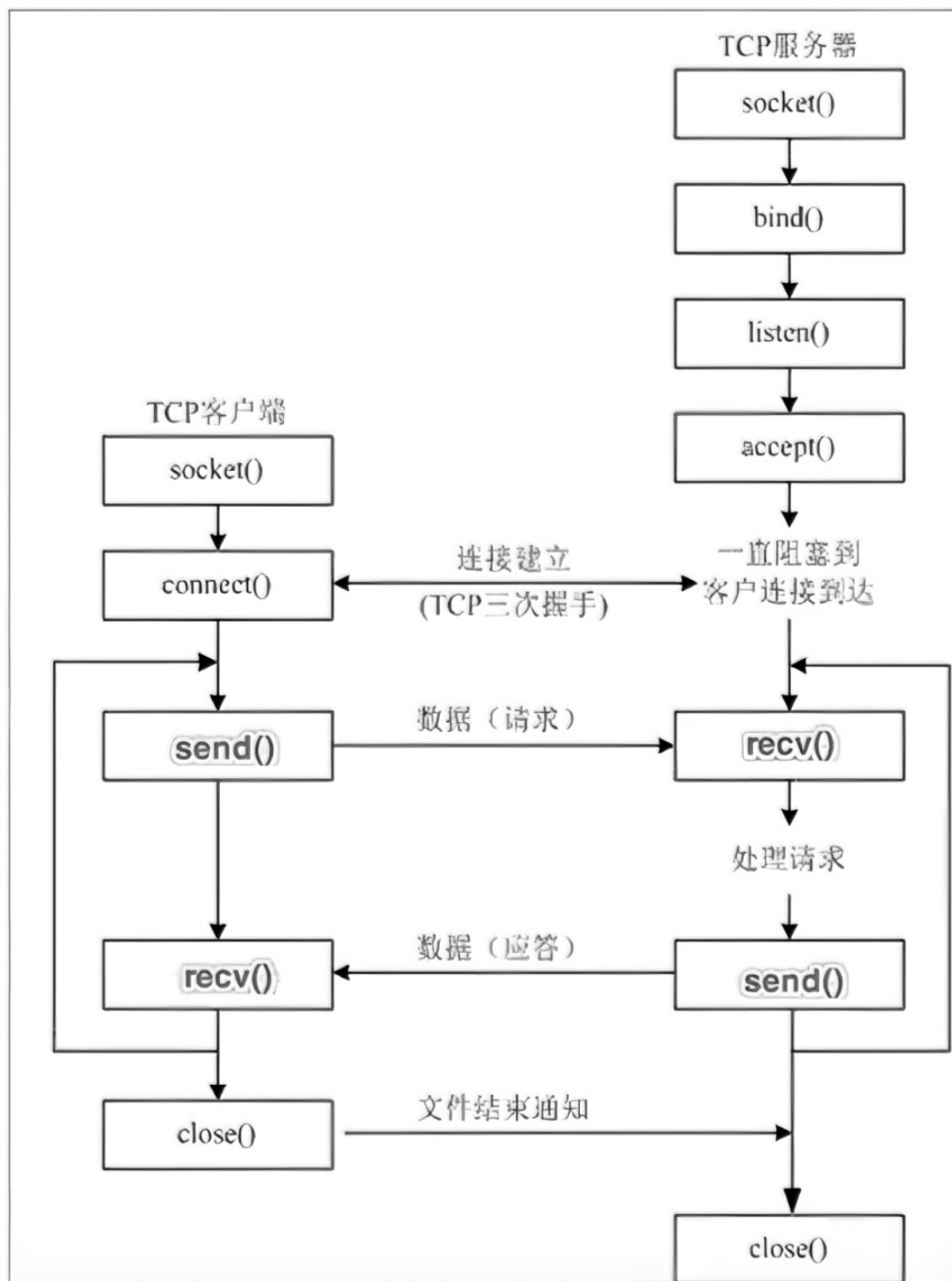


步骤说明

1. 创建客户端套接字对象
2. 和服务端套接字建立连接

3. 发送数据
4. 接收数据
5. 关闭客户端套接字

TCP服务端程序开发流程介绍



步骤说明

1. 创建服务端套接字对象
2. 绑定端口号
3. 设置监听
4. 等待接受客户端的连接请求
5. 接收数据
6. 发送数据
7. 关闭套接字

总结

1. TCP网络应用程序开发分为**客户端和服务端**
2. **主动发起建立连接请求**的是客户端程序
3. **等待接受连接请求**的是服务端程序

6. TCP客户端程序开发

socket类的介绍

导入socket模块

```
import socket
```

创建客户端socket对象

```
socket.socket(AddressFamily, Type)
```

参数说明：

- `AddressFamily` 表示IP地址类型，分为IPv4和IPv6
- `Type` 表示传输协议类型

方法说明：

- `connect(host, port)` 表示和服务端套接字建立连接
 - `host` 是服务器IP地址
 - `port` 是应用程序的端口号
- `send(data)` 表示发送数据
 - `data` 是二进制数据
- `recv(buffer size)` 表示接收数据
 - `buffer size` 是每次接收数据的长度

TCP客户端程序开发示例

网络调试助手下载地址：[NetAssistant 网络调试助手](#)

```
import socket

if __name__ == '__main__':
    # 创建TCP客户端套接字
    # socket.AF_INET = IPv4类型, socket.SOCK_STREAM = TCP协议
    tcp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 和服务端建立连接
    tcp_client_socket.connect(('192.168.10.100', 8080))
    # 对字符串进行编码成为二进制数据
    sent_content = input("输入你要发送的数据: ")
    sent_data = sent_content.encode('utf-8')
    # 发送数据到服务端
    tcp_client_socket.send(sent_data)
    # 接收服务端数据
    # 1024表示每次接收的最大字节数
    recv = tcp_client_socket.recv(1024)
    print(recv.decode('UTF-8'))
```

```
# 关闭套接字
tcp_client_socket.close()
```

7. TCP服务端程序开发

socket类的介绍

导入socket模块

```
import socket
```

创建客户端socket对象

```
socket.socket(AddressFamily, Type)
```

参数说明：

- `AddressFamily` 表示IP地址类型，分为IPv4和IPv6
- `Type` 表示传输协议类型

方法说明：

- `bind(host, port)` 表示绑定端口号
 - `host` 是IP地址
 - `port` 是端口号
 - IP地址一般不指定，表示本机的任何一个IP地址都可以
- `listen(backlog)` 表示设置监听
 - `backlog` 参数表示最大等待建立连接的个数
- `accept()` 表示等待接受客户端的连接请求
- `send(data)` 表示发送数据
 - `data` 是二进制数据
- `recv(buffer size)` 表示接收数据
 - `buffer size` 是每次接收数据的长度

TCP服务端程序开发示例

```
import socket

if __name__ == '__main__':
    while True:
        # 创建TCP服务端套接字
        tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # 绑定端口号
        # 第一个参数表示ip地址，一般不用指定，表示本机的任何一个ip即可
        tcp_server_socket.bind(('', 8080))

        # 设置监听
        # 默认128
        # 128：表示最大等待建立连接的个数
        tcp_server_socket.listen(128)
```

```

# 等待接受客户端连接请求
new_client, ip_port = tcp_server_socket.accept()
# 注意点:每次当客户端和服务端建立连接成功都会返回一个新的套接字
print(ip_port)
# 接收数据
recv_data = new_client.recv(1024)
print(recv_data.decode('utf-8'))

# 发送数据
new_client.send(b'hello\n')

# 关闭服务于客户端的套接字
new_client.close()
# 关闭套接字
tcp_server_socket.close()

```

注: 当客户端和服务端建立连接后, 服务端退出程序后, 端口号不会立即释放, 需要等待大概1-2分钟

解决方法:

1. 更换端口号
2. 设置端口号复用(推荐)也就是说让服务端退出程序后端口号立即释放

设置端口号复用代码如下

```

# 设置端口号复用
# 参数1: 表示当前套接字; 参数2: 设置端口号复用选项; 参数3: 设置端口号复用选项对应的值
tcp_server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
True)

```

8. TCP网络应用程序的注意点

1. 当 TCP 客户端程序想要和 TCP 服务端程序进行通信的时候必须要先**建立连接**
2. TCP 客户端程序一般不需要绑定端口号, 因为客户端是主动发起建立连接的。
3. **TCP 服务端程序必须绑定端口**, 否则客户端找不到这个 TCP 服务端程序。
4. listen 后的套接字是被动套接字, **只负责接收新的客户端的连接请求, 不能收发消息。**
5. 当 TCP 客户端程序和 TCP 服务端程序连接成功后, TCP 服务器端程序会产生一个新的套接字, 收发客户端消息使用该套接字。
6. **关闭 accept 返回的套接字意味着和这个客户端已经通信完毕。**
7. **关闭 listen 后的套接字意味着服务端的套接字关闭了, 会导致新的客户端不能连接服务端, 但是之前已经接成功的客户端还能正常通信。**
8. **当客户端的套接字调用 close 后, 服务器端的 recv 会解阻塞, 返回的数据长度为0**, 服务端可以通过返回数据的长度来判断客户端是否已经下线, **反之服务端关闭套接字, 客户端的 recv 也会解阻塞, 返回的数据长度也为0。**

9.案例-多任务版TCP服务端程序

```

import socket
import threading

# 处理客户端请求的任务
def user(ip_port, new_client):

```

```

# 注意点:每次当客户端和服务端建立连接成功都会返回一个新的套接字
print(ip_port)
# 接收数据
while True:
    recv_data = new_client.recv(1024)
    if recv_data:
        print(recv_data.decode('utf-8'))

        # 发送数据
        new_client.send(b'hello\n')
    else:
        print('下线了',ip_port)
        break

# 关闭服务于客户端的套接字
new_client.close()

if __name__ == '__main__':
    # 创建TCP服务端套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 设置端口号复用
    # 参数1: 表示当前套接字; 参数2: 设置端口号复用选项; 参数3: 设置端口号复用选项对应的值
    tcp_server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # 绑定端口号
    # 第一个参数表示ip地址, 一般不用指定, 表示本机的任何一个ip即可
    tcp_server_socket.bind(('', 8080))

    # 设置监听
    # 默认128
    # 128: 表示最大等待建立连接的个数
    tcp_server_socket.listen(128)

    while True:
        # 等待接受客户端连接请求
        new_client, ip_port = tcp_server_socket.accept()
        # 当客户端和服务端建立连接成功, 创建子线程, 让子线程专门负责接收客户端的消息
        user_info = threading.Thread(target=user, args=
(ip_port,new_client),daemon=True)
        user_info.start()

    # 关闭套接字
    # tcp_server_socket.close() # 因为服务端的程序需要一直运行, 所以关闭服务端套接字的代码
    可以省略

```

10. socket之send和recv原理剖析

TCP socket的发送和接收缓冲区

当创建一个TCP socket对象的时候会有一个**发送缓冲区**和一个**接收缓冲区**, 这个发送和接收缓冲区指的就是**内存中的一片空间**。

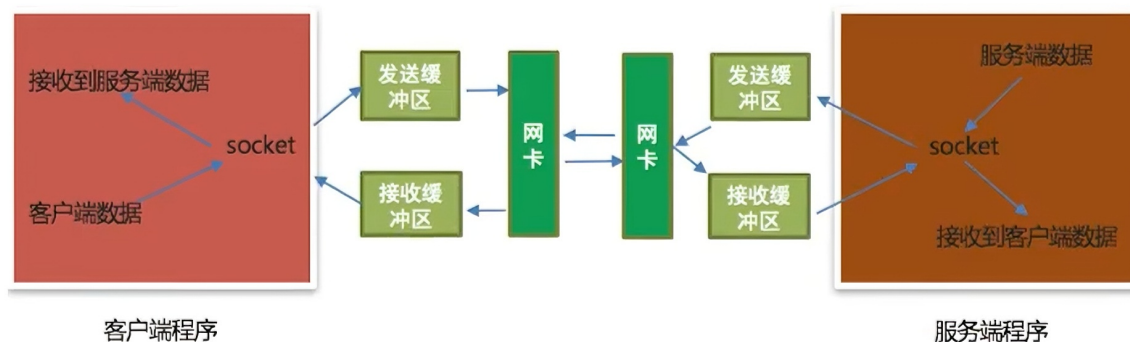
send原理剖析

send要想发数据，必须得**通过网卡发送数据**，应用程序是无法直接通过网卡发送数据的，它需要调用操作系统接口，也就是说，应用程序把发送的数据先写入到**发送缓冲区**(内存中的一片空间)，再由**操作系统控制网卡**把发送缓冲区的数据发送给服务端网卡。

recv原理剖析

recv从客户端接收数据，**应用软件是无法直接通过网卡接收数据的**，它需要调用操作系统接口，**由操作系统通过网卡接收数据**，把接收的数据写入到**接收缓冲区**(内存中的一片空间)，应用程序再从**接收缓存区**获取客户端发送的数据。

send和recv原理剖析图



说明：

- 发送数据是发送到发送缓冲区
- 接收数据是从接收缓冲区获取

十七、HTTP协议和静态Web服务器

1. HTTP协议

介绍

HTTP：超文本传输协议

HTTP 协议设计之前目的是传输网页数据的现在允许传输任意类型的数据。

传输 HTTP 协议格式的数据是基于 TCP 传输协议的，发送数据之前需要先建立连接。

HTTP协议的作用

它规定了浏览器和 Web 服务器通信数据的格式，也就是说浏览器和web服务器通信需要使用 HTTP 协议。

总结：

- HTTP协议是一个超文本传输协议
- HTTP是一个基于TCP传输协议传输数据的
- HTTP协议规定了浏览器和Web服务器通信数据的格式

2. URL

URL概述

统一资源定位符，通俗理解就是网络资源地址，也就是我们常说的网址

URL的组成

URL的样子

- <https://www.huihuia24.com/database/mysql>

URL的组成部分

1. 协议部分: `https://` 端口号: 443、`http://` 端口号: 80、`ftp://` 端口号: 21、20
2. 域名部分: www.huihuia24.com
3. 资源路径部分: `/database/mysql`

域名

域名就是**IP地址的别名**，它是用点进行分割使用英文字母和数字组成的名字，**使用域名目的就是方便的记住某台主机IP地址。**

URL的扩展

<https://news.163.com/hello.html?page=1&count=1>

- 查询参数部分: `?page=1&count=1`

参数说明:

- ?后面的 `page` 表示第一个参数，后面的参数都使用 `&` 进行连接

3. 查看HTTP协议的通信过程

Google浏览器开发者工具的使用

- 谷歌浏览器的开发者工具是查看HTTP协议的通信过程利器，通过Network标签选项可以查看每一次请求和响应的通信过程，调出开发者工具的通用方法是在网页右选择检查。
- 开发者工具的Headers选项总共有三部分组成：
 1. General: 主要信息
 2. Response Headers: 响应头
 3. Request Headers: 请求头
- Response选项是查看响应体信息的

4. HTTP请求报文

HTTP请求报文介绍

HTTP最常见的请求报文有两种

1. GET方式的请求报文
2. POST方式的请求报文

说明:

- GET: 获取web服务器数据
- POST: 向web服务器提交数据

HTTP GET 请求报文分析

HTTP GET 请求报文效果

```
--请求行--
GET / HTTP/1.1 => GET: 请求方法; /: 请求资源路径; HTTP/1.1: HTTP协议版本
--请求头--
Host: www.itcast.cn      => 服务器的主机IP地址和端口号
```

Connection: keep-alive => 和服务端保持长连接，当客户端和服务端有一段时间没有进行通信，那么服务端程序会主动向客户端断开连接

Upgrade-Insecure-Requests: 1 => 让客户端升级不安全请求，以后要使用HTTPS

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36 => 用户代理，客户端程序的名称，当后续讲爬虫的时候可以更具是否有User-Agent进行反爬机制

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3 => 告诉服务端程序可以接受的数据类型

Accept-Encoding: gzip, deflate => 告诉服务端程序支持的压缩算法

Accept-Language: zh-CN,zh;q=0.9,en;q=0.8 => 告诉服务端程序支持的语言

Cookie: UM_distinctid=169f06ab9d21a-0a52d93224ce3-12376d57-13c680-169f06ab9d571a;
accessId=22bdcd10-6250-11e8-917f-9fb8db4dc43c;
bad_id22bdcd10-6250-11e8-917f-9fb8db4dc43c=5c8a5a11-5811-11e9-975d-5932d6370ce7;
parent_qimo_sid_22bdcd10-6250-11e8-917f-9fb8db4dc43c=69928890-5811-11e9-ac5b-eb2506ccad9a;
CNZZDATA4617777=cnzz_eid%3D324231225-1554516145-%26ntime%3D1554521548;
href=http%3A%2F%2Fwww.itcast.cn%2F;
pageViewNum=5; Hm_lvt_0cb375a2e834821b74efffa6c71ee607=1554516722,1554523431;
Hm_lpvt_0cb375a2e834821b74efffa6c71ee607=1554523431;
nice_id22bdcd10-6250-11e8-917f-9fb8db4dc43c=fc5f90f1-5820-11e9-b896-dba72e2578c0;
openChat22bdcd10-6250-11e8-917f-9fb8db4dc43c=true => 客户端用户身份的标识

--空行--
\r\n

注：每项信息之间都需要一个\r\n，是由HTTP协议规定的

格式：

```
请求行\r\n
请求头\r\n
空行\r\n
```

HTTP POST请求报文分析

格式：

```
请求行\r\n
请求头\r\n
空行\r\n
请求体\r\n
```

请求体：浏览器发送给服务器的数据通过请求体来完成

总结：

- 应该HTTP请求报文可以由**请求行**、**请求头**、**空行**和**请求体**4个部分组成
- 请求行是由三个部分组成
 1. **请求方式**
 2. **请求资源路径**
 3. **HTTP协议版本**
- GET方式的请求报文没有请求体，只有请求行、请求头、空行组成

- POST方式的请求报文可以有请求行、请求头、空行和请求体四个部分组成，注意：POST方式可以允许没有请求体，但是这种格式很少见

5. HTTP响应报文

HTTP响应报文是web服务器程序发送给浏览器的HTTP协议的数据

```
--响应行（状态行）--
HTTP/1.1 200 OK      => HTTP协议版本 状态码 状态描述
--响应头--
Server: Tengine      => 服务器名称
Content-Type: text/html; charset=UTF-8  => 服务器发送给浏览器的内容类型及编码格式
Transfer-Encoding: chunked  => 服务器发送给客户端程序的数据不确定数据长度，数据发送完成的接收标识: 0/r/n 或者 Content-Length: 200(字节) 两者只能出现其一，不可同时出现
Connection: keep-alive  => 和客户端保持长连接
Date: Sat, 06 Apr 2019 08:49:57 GMT => 服务器的时间
--以下都是自定义响应头信息--
Accept-Ranges: bytes
Ali-Swift-Global-Savetime: 1554540597
Via: cache45.l2nu29-1[3,200-0,M], cache11.l2nu29-1[4,0], kunlun2.cn249[30,200-0,M], kunlun2.cn249[33,0]
X-Cache: MISS TCP_MISS dirn:-2:-2
X-Swift-Savetime: Sat, 06 Apr 2019 08:49:57 GMT
X-Swift-Cachetime: 0
Timing-Allow-Origin: *
EagleId: 2a51041615545405973986157e
--空行--
\r\n
--响应体--真正意义上给浏览器解析使用的数据
网页数据
```

提示：对于请求头和响应头信息程序员都可以进行自定义，按照客户端和服务端约定好的方式来制定即可

HTTP响应报文格式

```
响应行\r\n
响应头\r\n
空行\r\n
响应体\r\n
```

HTTP状态码

HTTP状态码是用于表示web服务器响应状态的三位数字代码

状态码	说明
200	请求成功
307	重定向
400	错误的请求，请求地址或者参数有错误
404	请求资源在服务器不存在
500	服务器内部源代码出现错误

6. 搭建Python自带静态Web服务器

如何搭建Python自带的静态web服务器

搭建Python自带的静态Web服务器使用 `python3 -m http.server 端口号`。

`-m`：表示运行包里的模块，执行这个命令的时候需要进入你自己指定静态文件的目录，然后通过浏览器就能访问对应的HTML文件

总结

- 静态Web服务器是为发出请求的浏览器提供静态文档的程序
- 搭建Python自带的Web服务器使用 `python -m http.server 端口号` 这个命令即可，端口号不指定默认是8000

7. 静态Web服务器-返回固定页面

开发自己的静态web服务器

步骤

1. 编写一个TCP服务端程序
2. 获取浏览器发送的HTTP请求报文数据
3. 读取固定页面数据，把页面数据组装成HTTP响应报文数据发送给浏览器
4. HTTP响应报文数据发送完成后，关闭服务与客户端的套接字

静态Web服务器-返回固定页面数据

```
import socket
import threading

# 定义user_info多线程
def user_info(new, port):
    print(f'已与{port}建立连接')
    while True:
        # 接收TCP客户端程序数据（最大4096字节）
        user_data = new.recv(4096)

        if user_data:
            # 对客户端发送的数据进行转换
            print(user_data)

            # 打开文件
            # 因为Windows默认格式为GBK，这里需要指定以UTF-8格式进行读取文件数据
```

```

        with open("HTTP静态测试/index.html", "r", encoding='UTF-8') as file: #
file表示打开文件的对象
            file_data = file.read()
            # with open 读取数据后关闭文件不需要程序员来完成，系统自己关闭

            # 把数据封装成HTTP响应报文格式
            # 响应行
            line = 'HTTP/1.1 200 OK\r\n'
            # 响应头
            head = 'Server: PWS/1.0\r\n'
            # 响应体
            http_body = file_data
            # 合并
            rdr_info = line + head + "\r\n" + http_body
            rdr_info_data = rdr_info.encode("utf-8")
            # 发送响应报文数据
            new.send(rdr_info_data)

            # 断开连接
        else:
            print(f'{port}已下线...')
            break

    # 关闭服务于客户端的套接字
    new.close()

if __name__ == '__main__':
    # 编写一个TCP服务端程序
    # 创建TCP服务端套接字
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 绑定端口
    server_socket.bind(('', 8000))
    # 设置端口复用
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # 设置监听（最大128位）
    server_socket.listen(128)
    while True:
        # 与TCP客户端建立连接
        new, port = server_socket.accept()
        # 创建多线程运行
        user = threading.Thread(target=user_info, args=(new, port), daemon=True)
        # 启动线程
        user.start()

```

注：在打开文件读取数据的时候，Windows默认使用 GBK 编码读取数据，而HTML文件通常是 UTF-8 编码，所以需要在读取数据时要指定文件编码格式 `encoding='UTF-8'`

8. 静态Web服务器-返回指定页面

- 指定路径
- 对客户端返回值进行分割

```

import socket
import threading

# 定义user_info多线程

```

```

def user_info(new, port):
    print(f'已与{port}建立连接')
    while True:
        # 接收TCP客户端程序数据（最大4096字节）
        user_data = new.recv(4096)

        if user_data:
            print(user_data)
            # 对客户端发送的数据进行转换
            user_data_info = user_data.decode("utf-8")
            # 对数据按照空格进行分割
            user_list = user_data_info.split(' ', maxsplit=2)
            # 获取请求路径
            user_name = user_list[1]

            if user_name == '/':
                # 指定路径
                user_name = '/index.html'
            # 打开文件
            # 为了兼容图片使用rb打开
            with open(f"HTTP静态测试{user_name}", "rb") as file: # file表示打开文件
                file_data = file.read()
            # with open 读取数据后关闭文件不需要程序员来完成，系统自己关闭

            # 把数据封装成HTTP响应报文格式
            # 响应行
            line = 'HTTP/1.1 200 OK\r\n'
            # 响应头
            head = 'Server: PWS/1.0\r\n'
            # 响应体
            http_body = file_data
            # 合并
            rdr_info = (line + head + "\r\n").encode("utf-8") + http_body
            # 发送响应报文数据
            new.send(rdr_info)

            # 断开连接
        else:
            print(f'{port}已下线...')
            break
    # 关闭服务于客户端的套接字
    new.close()

# noinspection DuplicatedCode
if __name__ == '__main__':
    # 编写一个TCP服务端程序
    # 创建TCP服务端套接字
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 绑定端口
    server_socket.bind(('', 8000))
    # 设置端口复用
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # 设置监听（最大128位）

```

的对象

```

server_socket.listen(128)
while True:
    # 与TCP客户端建立连接
    new,port = server_socket.accept()
    # 创建多线程运行
    user = threading.Thread(target=user_info, args=(new,port),daemon=True)
    # 启动线程
    user.start()

```

如何返回404页面

```

import socket

# 定义user_info多线程
def user_info():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 绑定端口
    server_socket.bind(('', 8000))
    # 设置端口复用
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # 设置监听（最大128位）
    server_socket.listen(128)
    while True:
        new, port = server_socket.accept()

        print(f'已与{port}建立连接')

        # 接收TCP客户端程序数据（最大4096字节）
        user_data = new.recv(4096)

        # 判断是否为空数据
        if len(user_data)==0:
            print(f'{port}已下线...')
            new.close()
            return

        # 对客户端发送的数据进行转换
        user_data_info = user_data.decode("utf-8")
        # 对数据按照空格进行分割
        user_list = user_data_info.split(' ',maxsplit=2)
        # 获取请求路径
        user_name = user_list[1]

        if user_name == '/':
            # 指定路径
            user_name = '/index.html'

        # 查看有无这个文件
        try:
            # 打开文件
            # 为了兼容图片使用rb打开
            with open(f"HTTP静态测试{user_name}", "rb") as file: # file表示打开文件
                # with open 读取数据后关闭文件不需要程序员来完成，系统自己关闭
                file_data = file.read()

```

的对象

```

except Exception as e:
    # 到这里说明请求的没有该文件，返回404
    # 响应行
    not_line = 'HTTP/1.1 404 Not Found\r\n'
    # 响应头
    not_head = 'Server: PWS/1.0\r\n'
    with open("HTTP静态测试/404.html", "rb") as file:
        not_file = file.read()

    not_info = (not_line + not_head + "\r\n").encode("utf-8") + not_file
    new.send(not_info)
else:
    # 把数据封装成HTTP响应报文格式
    # 响应行
    line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    head = 'Server: PWS/1.0\r\n'
    # 响应体
    http_body = file_data
    # 合并
    rdr_info = (line + head + "\r\n").encode("utf-8") + http_body
    # 发送响应报文数据
    new.send(rdr_info)
finally:
    # 断开连接
    new.close()
    print(f'{port}已下线...')

if __name__ == '__main__':
    user_info()

```

9. 静态Web服务器-多任务版

```

import socket
import threading

# 处理客户端请求
def process(new, port):
    print(f'已与{port}建立连接')

    # 接收TCP客户端程序数据（最大4096字节）
    user_data = new.recv(4096)

    # 判断是否为空数据
    if len(user_data) == 0:
        print(f'{port}已下线...')
        new.close()
        return

    # 对客户端发送的数据进行转换
    user_data_info = user_data.decode("utf-8")
    # 对数据按照空格进行分割
    user_list = user_data_info.split(' ', maxsplit=2)
    # 获取请求路径

```

```

user_name = user_list[1]

if user_name == '/':
    # 指定路径
    user_name = '/index.html'

# 查看有无这个文件
try:
    # 打开文件
    # 为了兼容图片使用rb打开
    with open(f"HTTP静态测试{user_name}", "rb") as file: # file表示打开文件的对象
        # with open 读取数据后关闭文件不需要程序员来完成，系统自己关闭
        file_data = file.read()

except Exception as e:
    # 到这里说明请求的没有该文件，返回404
    # 响应行
    not_line = 'HTTP/1.1 404 Not Found\r\n'
    # 响应头
    not_head = 'Server: PWS/1.0\r\n'
    with open("HTTP静态测试/404.html", "rb") as file:
        not_file = file.read()

    not_info = (not_line + not_head + "\r\n").encode("utf-8") + not_file
    new.send(not_info)
else:
    # 把数据封装成HTTP响应报文格式
    # 响应行
    line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    head = 'Server: PWS/1.0\r\n'
    # 响应体
    http_body = file_data
    # 合并
    rdr_info = (line + head + "\r\n").encode("utf-8") + http_body
    # 发送响应报文数据
    new.send(rdr_info)
finally:
    # 断开连接
    new.close()
    print(f'{port}已下线...')

# 定义user_info多线程
def user_info():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 绑定端口
    server_socket.bind(('', 8000))
    # 设置端口复用
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # 设置监听（最大128位）
    server_socket.listen(128)
    while True:
        new, port = server_socket.accept()

```

```

        sub_process = threading.Thread(target=process, args=(new, port),
daemon=True)
        sub_process.start()

if __name__ == '__main__':
    user_info()

```

10. 静态Web服务器-面向对象开发

```

import socket
import threading

# HTTP协议的Web服务器类
class HttpWebServer(object):
    def __init__(self):
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # 绑定端口
        server_socket.bind(('', 8000))
        # 设置端口复用
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
        # 设置监听（最大128位）
        server_socket.listen(128)
        # 把TCP服务器套接字作为web服务器对象属性
        self.server_socket = server_socket

    # 处理客户端请求
    @staticmethod
    # @staticmethod 是 Python 类中的静态方法装饰器，作用是定义一个与类和实例都无关的方法——
    # 它既不需要访问类属性，也不需要访问实例属性。
    def process(new, port):
        print(f'已与{port}建立连接')

        # 接收TCP客户端程序数据（最大4096字节）
        user_data = new.recv(4096)

        # 判断是否为空数据
        if len(user_data) == 0:
            print(f'{port}已下线...')
            new.close()
            return

        # 对客户端发送的数据进行转换
        user_data_info = user_data.decode("utf-8")
        # 对数据按照空格进行分割
        user_list = user_data_info.split(' ', maxsplit=2)
        # 获取请求路径
        user_name = user_list[1]

        if user_name == '/':
            # 指定路径
            user_name = '/index.html'

        # 查看有无这个文件
        try:

```

的对象

```
# 打开文件
# 为了兼容图片使用rb打开
with open(f"HTTP静态测试{user_name}", "rb") as file: # file表示打开文件

    # with open 读取数据后关闭文件不需要程序员来完成，系统自己关闭
    file_data = file.read()

except Exception as e:
    # 到这里说明请求的没有该文件，返回404
    # 响应行
    not_line = 'HTTP/1.1 404 Not Found\r\n'
    # 响应头
    not_head = 'Server: PWS/1.0\r\n'
    with open("HTTP静态测试/404.html", "rb") as file:
        not_file = file.read()

    not_info = (not_line + not_head + "\r\n").encode("utf-8") + not_file
    new.send(not_info)
else:
    # 把数据封装成HTTP响应报文格式
    # 响应行
    line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    head = 'Server: PWS/1.0\r\n'
    # 响应体
    http_body = file_data
    # 合并
    rdr_info = (line + head + "\r\n").encode("utf-8") + http_body
    # 发送响应报文数据
    new.send(rdr_info)
finally:
    # 断开连接
    new.close()
    print(f'{port}已下线...')

# 启动服务器方法
def start(self):
    while True:
        new, port = self.server_socket.accept()
        sub_process = threading.Thread(target=self.process, args=(new,
port), daemon=True)
        sub_process.start()

# 启动!!!
def main():
    http = HttpwebServer()
    http.start()

if __name__ == '__main__':
    main()
```

11. 静态Web服务器-命令行启动动态绑定端口号

```
import socket
import threading
import sys

# HTTP协议的Web服务器类
class HttpWebServer(object):
    def __init__(self, port):
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # 绑定端口
        server_socket.bind(('', port))
        # 设置端口复用
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
        # 设置监听（最大128位）
        server_socket.listen(128)
        # 把TCP服务器套接字作为web服务器对象属性
        self.server_socket = server_socket

    # 处理客户端请求
    @staticmethod
    # @staticmethod 是 Python 类中的静态方法装饰器，作用是定义一个与类和实例都无关的方法——它既不需要访问类属性，也不需要访问实例属性。
    def process(new, port):
        print(f'已与{port}建立连接')

        # 接收TCP客户端程序数据（最大4096字节）
        user_data = new.recv(4096)

        # 判断是否为空数据
        if len(user_data) == 0:
            print(f'{port}已下线...')
            new.close()
            return

        # 对客户端发送的数据进行转换
        user_data_info = user_data.decode("utf-8")
        # 对数据按照空格进行分割
        user_list = user_data_info.split(' ', maxsplit=2)
        # 获取请求路径
        user_name = user_list[1]

        if user_name == '/':
            # 指定路径
            user_name = '/index.html'

        # 查看有无这个文件
        try:
            # 打开文件
            # 为了兼容图片使用rb打开
            with open(f"HTTP静态测试{user_name}", "rb") as file: # file表示打开文件的对象
                # with open 读取数据后关闭文件不需要程序员来完成，系统自己关闭
                file_data = file.read()
```

```

except Exception as e:
    # 到这里说明请求的没有该文件, 返回404
    # 响应行
    not_line = 'HTTP/1.1 404 Not Found\r\n'
    # 响应头
    not_head = 'Server: PWS/1.0\r\n'
    with open("HTTP静态测试/404.html", "rb") as file:
        not_file = file.read()

    not_info = (not_line + not_head + "\r\n").encode("utf-8") + not_file
    new.send(not_info)
else:
    # 把数据封装成HTTP响应报文格式
    # 响应行
    line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    head = 'Server: PWS/1.0\r\n'
    # 响应体
    http_body = file_data
    # 合并
    rdr_info = (line + head + "\r\n").encode("utf-8") + http_body
    # 发送响应报文数据
    new.send(rdr_info)
finally:
    # 断开连接
    new.close()
    print(f'{port}已下线...')

# 启动服务器方法
def start(self):
    while True:
        new, port = self.server_socket.accept()
        sub_process = threading.Thread(target=self.process, args=(new,
port), daemon=True)
        sub_process.start()

def main():
    # 获取终端命令行参数
    parms = sys.argv

    if len(parms) != 2:
        print('格式错误! 如下: ')
        print('python xxx.py 端口号(数字)')
        return

    # 判断第二个参数是否都是由数字组成
    if not parms[1].isdigit():
        print('格式错误! 如下: ')
        print('python xxx.py 端口号(数字)')
        return

    port = int(parms[1])
    print(f'成功! 你的端口号是{port}')
    # 创建web服务器
    http = HttpwebServer(port)

```

```
# 启动!!!
http.start()

if __name__ == '__main__':
    main()
```

十八、html+css 基础

1. HTML的介绍

HTML的定义

HTML：**超文本标记语言**，标记就是**标签**，`<标签名称></标签名称>`，比如 `<html></html>`、`<h1></h1>` 等，标签大多数都是成对出现的

超文本的两层含义：

1. 因为网页中还可以图片、视频、音频等内容(超越文本限制)
2. 它还可以在网页中跳转到另一个网页，与世界各地主机的网页链接(超链接文本)

HTML的作用

HTML是用来开发网页的，它是开发网页的语言

2. HTML的基本结构

结构代码

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>网页标题</title>
</head>
<body>
    网页显示内容
</body>
</html>
```

1. `<!DOCTYPE html>` 是文档声明，用来指定页面所使用的 html 的版本，这里声明的是一个 html5 的文档。
2. `<html>...</html>` 标签是开发人员在告诉浏览器,整个网页是从 `<html>` 这里开始的，到 `</html>` 结束，也就是 html 文档的开始和结束标签。
3. `<head>...</head>` 标签用于定义文档的头部，是负责对网页进行设置标题、编码格式以及引入 css 和 js 文件的。
4. `<body>...</body>` 标签是编写网页上显示的内容。

浏览网页文件

网页文件的后缀是 `.html` 或者 `.htm`，一个HTML文件就是一个网页，可以用文本的方式编辑它，如果用浏览器打开，浏览器会按照标签描述内容将文件渲染成网页。

3.VScode的基本使用

软件使用: [VScode](#)

安装教程请参阅: [VScode的下载安装](#)

插件: `html / css / JavaScript / open in browser`

```
<!-- 文档声明:HTML5 -->
<!DOCTYPE html>
<!-- 当前网页语言是中文, 默认不指定是中文=>lang="zh" -->
<html lang="en">
<head>
  <!-- 指定网页编码格式 -->
  <meta charset="UTF-8">
  <!-- 在移动设备浏览网页时, 网页不缩放 -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- 网页的标题 -->
  <title>Document</title>
</head>
<body>
  hell
</body>
</html>
```

4. 常用的HTML标签

成对出现

```
<h1>h1标题</h1>
<div>div盒子容器</div>
<p>段落标签</p>
```

单个出现

```
<br>    <!--换行-->
<img src='./aaa/aaa.png' alt='图片'>
<hr>    <!--水平分割线-->
```

带属性的标签

```
<img src='./aaa/aaa.png' alt='图片'>
<a href='https://www.baidu.com/'>百度</a>
```

标签的嵌套

```
<div>
  <img src='./aaa/aaa.png' alt='图片'>
  <a href='https://www.baidu.com/'>百度</a>
</div>
```

注:

1. 标签不区分大小写，但是推荐使用小写
2. 根据标签的书写形式，标签分为双标签(闭合标签)和单标签(空标签)
 - 双标签是指由开始标签和结束标签组成的一对标签，这种标签允许嵌套和承载内容，比如：`div` 标签
 - 单标签是一个标签组成，没有标签内容，比如：`img` 标签

5.资源路径

相对路径

从当前操作的HTML的文档所在目录算起的路径叫相对路径

```
<img src='./aaa/aaa.png' alt='图片'>
<img src='aaa/aaa.png' alt='图片'>
```

绝对路径

从根目录算起的路径叫绝对路径

```
<img src='c:\demo\aaa\aaa.png' alt='图片'>
```

注：资源路径一般会使用相对路径，更加通用和方便

6.列表标签

有序列表(`ul` 标签)

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

无序列表(`ol` 标签)

```
<ol>
  <li>测试项一</li>
  <li>测试项二</li>
  <li>测试项三</li>
</ol>
```

7.表格标签

表格标签

- `<table>`：表示一个表格
 - `<tr>`：表示表格中的一行
 - `<td>`：表示表格中的列
 - `<th>`：表示表格中的表头

示例：

```
<table style="border: 1px solid black; border-collapse: collapse;">
  <tr>
    <th style="border: 1px solid black;">姓名</th>
    <th style="border: 1px solid black;">年龄</th>
  </tr>
  <tr>
    <td style="border: 1px solid black;">张三</td>
    <td style="border: 1px solid black;">28</td>
  </tr>
  <tr>
    <td style="border: 1px solid black;">李四</td>
    <td style="border: 1px solid black;">34</td>
  </tr>
</table>
```

`border-collapse: collapse;`: 表格边线合并

8. 表单标签

表单用于搜集不同类型的用户输入(用户输入的数据), 然后可以把用户数据提交到web服务器

表单标签的使用

`<form>`: 表示表单标签, 定义整体的表单区域

`<label>`: 表示单元元素的文字标注标签, 定义文字标注

`<input>`: 表示单元元素的用户输入标签, 定义不同类型的用户输入数据方式

- `type` 属性
 - `type="text"` 定义单行输入文本
 - `type="password"` 定义密码输入框
 - `type="radio"` 定义单选框
 - `type="checkbox"` 定义复选框
 - `type="file"` 定义上传文件
 - `type="submit"` 定义提交按钮
 - `type="reset"` 定义重置按钮
 - `type="button"` 定义一个普通按钮

`<textarea>` 表示单元元素多行文本输入框标签, 定义多行文本输入框

`<select>` 表示单元元素的下拉列表标签, 定义下拉列表

- `<option>` 标签与 `<select>` 标签配合, 定义下拉列表中的选项

示例:

```
<form>
  <p>
    <!-- for 更具id名给指定的id设置光标 -->
    <label for="name">用户名: </label>
    <input type="text" id="name">
  </p>
  <p>
    <label for="password">密码: </label>
```

```
<input type="password" id="password">
</p>
<p>
  <label>性别: </label>
  <input type="radio">男
  <input type="radio">女
</p>
<p>
  <label>爱好: </label>
  <input type="checkbox">篮球
  <input type="checkbox">足球
  <input type="checkbox">排球
</p>
<p>
  <label>照片: </label>
  <input type="file">
</p>
<p>
  <label>自我介绍: </label>
  <textarea></textarea>
</p>
<p>
  <label>籍贯: </label>
  <select>
    <option>北京</option>
    <option>上海</option>
    <option>广州</option>
  </select>
</p>
<p>
  <input type="submit" value="提交">
  <input type="reset" value="重置">
  <input type="button" value="按钮">
</p>
</form>
```

9. 表单提交

表单属性设置

`<form>` 表示表单标签，定义整体的表单区域

- `action` 属性：设置表单数据的提交地址
- `method` 属性：设置表单提交的方式，一般有 `GET` 方式和 `POST` 方式，不区分大小写

表单元素属性设置

`name` 属性：设置表单元素的名称，该名称是提交时的参数名

`value` 属性：设置表单元素的值，该值是提交数据时参数名所对应的值

示例：

注：

- get方式提交数据到web服务器以地址栏的方式提交给服务器，不安全，能看到提交的数据，严谨的说是以查询参数的方式提交给web服务器
- post方式提交数据，表单数据会放到请求体里面

```
<form action="https://www.baidu.com" method="post">

  <p>
    <!-- for 更具id名给指定的id设置光标 -->
    <label for="name">用户名: </label>
    <input type="text" id="name" name="username">
  </p>
  <p>
    <label for="password">密码: </label>
    <input type="password" name="password">
  </p>
  <p>
    <label>性别: </label>
    <input type="radio" name="sex" value="0">男
    <input type="radio" name="sex" value="1">女
  </p>
  <p>
    <label>爱好: </label>
    <input type="checkbox" name="hobby" value="basketball">篮球
    <input type="checkbox" name="hobby" value="football">足球
    <input type="checkbox" name="hobby" value="volleyball">排球
  </p>
  <p>
    <label>照片: </label>
    <input type="file" name="pic">
  </p>
  <p>
    <label>自我介绍: </label>
    <textarea name="dsc"></textarea>
  </p>
  <p>
    <label>籍贯: </label>
    <select name="position">
      <option value="1">北京</option>
      <option value="2">上海</option>
      <option value="3">广州</option>
    </select>
  </p>
  <p>
    <input type="submit" value="提交">
    <input type="reset" value="重置">
    <input type="button" value="按钮">
  </p>
</form>
```

10. CSS 的介绍

CSS 的定义

CSS：层叠样式表，它是用来美化页面的一种语言

CSS 的作用

1. 美化页面，比如：设置标签文字大小、颜色、字体加粗等样式
2. 控制页面布局，比如：设置浮动，定位等样式

CSS 的基本语法

```
选择器{  
  样式规则  
}  
样式规则：  
属性名:属性值;  
属性名:属性值;  
....
```

选择器是用来选择标签的，选出来后给标签添加样式

示例：

```
div{  
  width: 100px;  
  height: 100px;  
  background: gold;  
}
```

说明：

CSS 是由两个主要的部分构成：**选择器和一条或者多条样式规则**；

注：样式规则需要放到大括号里面

11. CSS 的引入方式

内行式

直接在标签的 style 属性中添加 CSS 样式

```
<div style="width: 100px; height: 100px; background: red;">hello</div>
```

优点：方便、直观

缺点：缺乏可重用性

内嵌式(内部样式)

在 `<head>` 标签内加入 `<style>` 标签, 在 `<style>` 标签中编写 `css` 代码

```
<head>
  <style>
    h3{
      color: blue;
    }
  </style>
</head>
```

优点: 在同一个页面内部便于复用和维护

缺点: 在多个页面之间的可重用性不够高

外链式

将 `css` 代码写在一个单独的 `.css` 文件中, 在 `<head>` 标签中使用 `<link>` 标签直接引入该文件到页面中。

```
<link rel="stylesheet" href="aaa.css">
```

优点: 使得 `css` 样式与HTML页面分离, 便于整个页面系统的规划和维护, 可重用性高

缺点: `css` 代码由于分离到单独的 `css` 文件中, 容易出现 `css` 代码过于集中, 若维护不当, 极易造成混乱

css 引入方式选择

1. 内行样式几乎不用
2. 内嵌式在学习 `css` 样式阶段使用
3. 外链式在公司开发的阶段使用, 可以对 `css` 样式和 `html` 页面分别进行开发

12. CSS 选择器

定义

`css` 选择器是用来选择标签的, 选出来以后给标签加样式

标签选择器

根据标签来选择标签, 以**标签开头**, 此种选择器影响范围大, 一般用来做一些通用设置

```
p {
  color: red;
}
```

类选择器

更具类名来选择标签, 并以 `.` 开头, 一个类选择器可用于多个标签上, 一个标签上也可以使用多个类选择器, 多个类选择器需要使用空格分割, 应用灵活, 可复用, 是 `css` 中应用最多的一种选择器

```

<head>
  <style type="text/css">
    .red{color: red;}
    .big{font-size: 30px;}
    .box{width: 200px; height: 200px; background: blue;}
  </style>
</head>
<body>
  <div class="blue">这是一个</div>
  <h3 class="big blue box">这个是一个标题</h3>
  <p class="big">这是一个段落</p>
</body>

```

层级选择器(后代选择器)

根据层级关系选择后代标签，以选择器1选择器2开头，主要应用在嵌套的结构中，减少命名

```

<head>
  <style>
    div p {
      color: red;
    }
  </style>
</head>
<body>
  <div>
    <a href="www.baidu.com">百度</a>
    <p>测试段落</p>
  </div>
  <div>
    <h2>标题</h2>
  </div>
</body>

```

注：层级选择器不一定必须是父子关系，祖孙关系的子标签页可以找到，完成添加样式的操作

id选择器

根据id选择标签，以#开头，元素的id名称不能重复，所以id选择器只能应对于页面上的一个元素，不能复用，id名一般给程序使用，所以不推荐使用id作为选择器

```

<head>
  <style>
    #box{
      color: skyblue;
    }
  </style>
</head>
<body>
  <p id="box">这是一个段落</p>
  <p>我的房子</p>
  <p>我的车子</p>
</body>

```

注：虽然给其他标签设置 `id=box` 也可以设置样式，但不推荐这样做，因为id是唯一的，以后js通过id只能获取一个唯一的标签对象

组选择器

根据组合的选择器选择不同的标签，以`,`分割开，如果有公共的样式设置，可以使用组选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .box1, .box2, .box3{width: 100px; height: 100px;}
    .box1{background: red;}
    .box2{background: blue;}
    .box3{background: green;}
  </style>
</head>
<body>
  <div class="box1"></div>
  <div class="box2"></div>
  <div class="box3"></div>
</body>
</html>
```

快捷方法：`div.box1*3` 创建三个div并且指定class属性是box1

注：在css里面出现同样的选择器，后面的不会覆盖前面的，后面的选择器会在前面选择器的基础上进行追加样式

伪类选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .box{width: 100px; height: 100px; background: gold;}
    .box:hover{width: 150px; height: 150px; background: red;}
  </style>
</head>
<body>
  <div class="box">这个是一个盒子</div>
</body>
</html>
```

`hover` 当光标接触时启动此样式

13. CSS 属性

布局常用样式属性

`width`：设置元素(标签)的宽度，如：`width:100px;`

`height`：设置元素(标签)的高度，如：`height:100px;`

`background`：设置元素背景色或者背景图片，如：`background:gold`;设置元素的背景色，`background:url(images/logo.png);`，设置元素的背景图片

`border`：设置元素四周的边框，如 `boder:1px solid black`;设置元素四周边框是1像素的黑色实线

`border-top`：设置顶边边框，如：`border-top:10px solid red`;

`border-left`：设置左边边框，如：`border-left:10px solid blue`;

`border-right`：设置右边边框，如：`border-right:10px solid green`;

`border-bottom`：设置底边边框，如：`border-bottom:10px solid pink`;

```
div{
  width:100px;
  height:100px;
  background:url("imgs/logo.png") on-repeat;
  /* no-repeat: 表示图片不重复显示图片，拉伸当前显示的图片 */
}
```

`padding`：设置元素包含的内容和元素边框的距离，也叫内边距，如：`padding:20px`;，`padding` 是同时设置4个边的，也可以像 `border` 一样拆分成分别设置四个边：`padding-top`、`padding-left`、`padding-right`、`padding-bottom`。

`margin`：设置元素和外界的距离，也叫外边距，如：`margin:20px`;margin是同时设置4个边的，也可以像border一样拆分成分别设置四个边：`margin-top`、`margin-left`、`margin-right`、`margin-bottom`。

`float`：设置元素浮动，浮动可以让块元素排列在一行，浮动分为左浮动：`float:left`;右浮动：`float:right`;

- 以后设置 `div` 在一行显示，可以设置浮动

快捷方式：`div>div*2` 创建一个父 `div`，在父 `div` 里面创建两个子 `div`

文本常用样式属性

`color`：设置文字的颜色，如：`color:red`;

`font-size`：设置文字的大小，如：`font-size:12px`;

`font-family`：设置文字的字体，如：`font-family:"微软雅黑"`;，为了避免中文字不兼容，一般写成：`fontfamily:'Microsoft Yahei'`;

`font-weight`：设置文字是否加粗，如：`font-weight:bold`;，设置加粗 `font-weight:normal`;设置不加粗

`line-height`：设置文字的行高，如：`line-height:24px`;表示文字高度加上文字上下的间距是24px，也就是每一行占有的高度是24px

`text-decoration`：设置文字的下划线，如：`text-decoration:none`;将文字下划线去掉

- `underline`：下划线
- `line-through`：删除线
- `overline`：上划线

`text-align`：设置文字水平对齐方式，如：`text-align:center`;设置文字水平居中

- `left`: 左对齐
- `center`: 中齐
- `right`: 右对齐

`text-indent`: 设置文字首行缩进, 如: `text-indent:24px`; 设置文字首行缩进 24px (一个字占 30px)

注: `span` 标签可以给文本中的一小段内容设置样式

14. CSS 元素 溢出

什么是元素溢出

当子元素(标签)的尺寸超过父元素(标签)的尺寸时, 此时需要设置父元素显示溢出的子元素的方式, 设置的方法是通过 `overflow` 属性来完成

`overflow` 的设置项

- `visible`: 默认值, 显示子标签溢出部分
- `hidden`: 隐藏子标签溢出部分
- `auto`: 如果子标签溢出, 则可以滚动查看其余的内容

示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .box1{
      width: 100px;
      height: 100px;
      background: red;
      /* 不显示溢出内容 */
      /* overflow: hidden; */
      /* 显示溢出内容, 出现滚动条 */
      /* overflow: auto; */
      /* 显示溢出内容, 内容直接溢出 */
      overflow: visible;
    }
    .box2{
      width: 200px;
      height: 50px;
      background: black;
    }
  </style>
</head>
<body>
  <div class="box1">
    <div class="box2"></div>
  </div>
</body>
</html>
```

注：设置元素溢出，需要在父标签的选择器上进行设置

15. CSS 显示特性

学习视频：[CSS 显示特性](#)

display 属性的使用

- `none`：元素隐藏且不占位置
- `inline`：元素以行内元素显示
- `block`：元素以块元素显示

示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    div {
      width: 100px;
      height: 100px;
      background: red;
      /* 完全隐藏 */
      /* display: none; */
      /* 行内 */
      /* display: inline; */
      /* 块级 */
      display: block;
    }
    a {
      /* 设置单独占一行，不和其他标签在一行显示 */
      display: block;
    }
  </style>
</head>
<body>
  <div>
    哈哈
  </div>
  <p>嘻嘻</p>
  <a href="https://www.baidu.com">百度</a>
</body>
</html>
```

注：设置 `div` 可以和其他元素在一行显示，以后就不能设置宽高；如果既设置 `div` 的宽高，又设置子啊一行显示，可以通过浮动完成

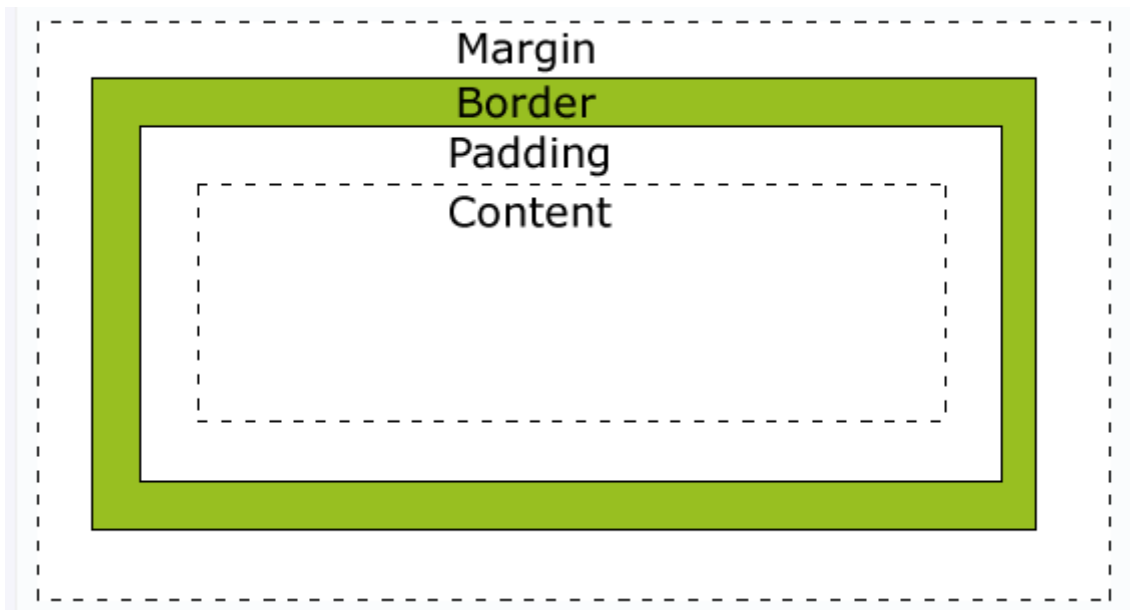
16. CSS 盒子模型

学习地址: [盒子模型基础](#)

定义

所有HTML元素可以看作盒子，CSS 盒模型本质上是一个盒子，封装周围的HTML元素，它包括：**外边距** (margin)、**内边距** (padding)、**边框** (border)、**内容** (content)，盒模型允许我们在**其它元素和周围元素边框之间的空间放置元素**。

盒子模型示意图：



盒子模型相关样式属性

- 盒子的内容宽度 (width)，注意：不是盒子的宽度
- 盒子的内容高度 (height)，注意：不是盒子的高度
- 盒子的边框 (border)
- 盒子内的内容和边框之间的间距 (padding)

示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .box1 {
      width: 200px;
      height: 200px;
      background: green;
      border: 5px solid blue;
      /* padding-top: 10px;
      padding-left: 10px; */
      padding: 10px;
    }
    .box2 {
```

```

        width: 100px;
        height: 150px;
        background-color: lightcoral;
    }
    .box3 {
        width: 200px;
        height: 200px;
        background-color: lightpink;
        margin-top: 20px;
    }

</style>
</head>
<body>
    <!-- 盒子模型 -->
    <div class="box1">
        <div class="box2"></div>
    </div>
    <div class="box3"></div>
</body>
</html>

```

盒子的真实尺寸

盒子的width和height值固定时，如果盒子增加border和padding，盒子的整体尺寸会发生变化所以盒子的真实尺寸为：

- 盒子的宽度=width + padding左右 + border左右
- 盒子的高度=height + padding上下 + border上下

17. CSS 弹性盒子模型 Flex

学习地址：[弹性盒子模型](#)

Flex：弹性布局，用来为盒状模型提供最大的灵活性

```

.box{
    display: flex;
}

```

行内元素也可以使用 Flex 布局

```

.box1{
    display: inline-flex;
}

```

Webkit 内核的浏览器，必须加上 -webkit 前缀

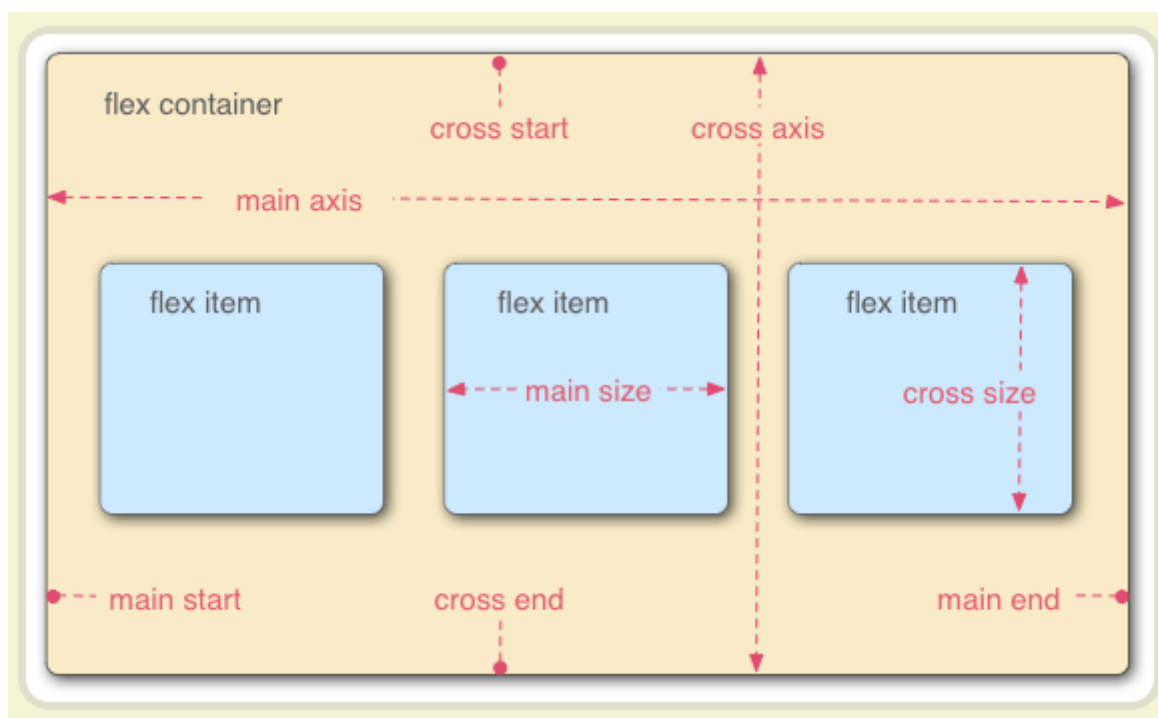
```

.box{
    display: -webkit-flex; /* safari */
    display: flex;
}

```

基本概念

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称“容器”。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称“项目”。



容器默认存在两根轴：**水平的主轴**（main axis）和**垂直的交叉轴**（cross axis）。主轴的开始位置（与边框的交叉点）叫做 `main start`，结束位置叫做 `main end`；交叉轴的开始位置叫做 `cross start`，结束位置叫做 `cross end`。

项目默认沿主轴排列。单个项目占据的主轴空间叫做 `main size`，占据的交叉轴空间叫做 `cross size`。

容器的属性

- `flex-direction`：属性决定主轴的方向（即项目的排列方向）。
 - `row`（默认值）：主轴为水平方向，起点在左端。
 - `row-reverse`：主轴为水平方向，起点在右端。
 - `column`：主轴为垂直方向，起点在上沿。
 - `column-reverse`：主轴为垂直方向，起点在下沿。
- `flex-wrap`：属性定义，如果一条轴线排不下，如何换行。
 - `nowrap`（默认）：不换行。
 - `wrap`：换行，第一行在上方。
 - `wrap-reverse`：换行，第一行在下方。
- `flex-flow`：是 `flex-direction` 属性和 `flex-wrap` 属性的简写形式，默认值为 `row nowrap`。
- `justify-content`：定义了项目在主轴上的对齐方式。
 - `flex-start`（默认值）：左对齐。
 - `flex-end`：右对齐。
 - `center`：居中。
 - `space-between`：两端对齐，项目之间的间隔都相等。
 - `space-around`：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。
- `align-items`：定义项目在交叉轴上如何对齐。

- `flex-start`：交叉轴的起点对齐。
- `flex-end`：交叉轴的终点对齐。
- `center`：交叉轴的中点对齐。
- `baseline`：项目的第一行文字的基线对齐。
- `stretch`（默认值）：如果项目未设置高度或设为`auto`，将占满整个容器的高度。
- `align-content`：定义了对多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用
 - `flex-start`：与交叉轴的起点对齐。
 - `flex-end`：与交叉轴的终点对齐。
 - `center`：与交叉轴的中点对齐。
 - `space-between`：与交叉轴两端对齐，轴线之间的间隔平均分布。
 - `space-around`：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
 - `stretch`（默认值）：轴线占满整个交叉轴。

项目属性

- `order`：定义项目的排列顺序。数值越小，排列越靠前，默认为0
- `flex-grow`：定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大

注：如果所有项目的`flex-grow`属性都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的`flex-grow`属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍。
- `flex-shrink`：定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小

注：如果所有项目的`flex-shrink`属性都为1，当空间不足时，都将按比例缩小。如果一个项目的`flex-shrink`属性为0，其他项目都为1，则空间不足时，前者不缩小。
- `flex-basis`：定义了项目在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为`auto`，即项目的本来大小。

注：它可以设为跟`width`或`height`属性一样的值（比如350px），则项目将占据固定空间
- `flex`：是`flex-grow`、`flex-shrink`和`flex-basis`的简写，默认值为`0 1 auto`。后两个属性可选
 - 该属性有两个快捷值：`auto`（`1 1 auto`）和`none`（`0 0 auto`）。
- `align-self`：允许单个项目有与其他项目不一样的对齐方式，可覆盖`align-items`属性。默认值为`auto`，表示继承父元素的`align-items`属性，如果没有父元素，则等同于`stretch`。
 - 该属性可能取6个值，除了`auto`，其他都与`align-items`属性完全一致

十九、JavaScript

1. JavaScript 的定义

JavaScript 是运行在浏览器端的脚本语言，是由浏览器解释执行的，简称 js，它能够让网页和用户有交互功能，增加良好的用户体验效果

前端开发三大块：

1. `html`：负责网页结构
2. `css`：负责网页样式
3. `JavaScript`：负责网页行为，比如：网页与用户的交互效果

2. JavaScript 的使用方式

行内式(主要用于事件)

```
<input type="button" onclick="alert('滚 🐶')" value="点我试试">
```

内嵌式

```
<script>
    alert("测试页面");
</script>
```

外链式

```
<script src="./测试.js"></script>
```

3. 变量和数据类型

定义变量

JavaScript 是一种弱类型语言，也就是说不需要指定变量的类型，JavaScript 的变量类型由他的值来决定，定义变量需要用关键字 `var`，一条JavaScript 语句应该以 `;` 结尾

定义变量的语法格式

```
var 变量名 = 值
```

例：

```
var age = 13;
var name = "张三";
```

同时定义多个变量可以用“`,`”隔开，公用一个 `var` 关键字

```
var wad = 283 , bad = 382 ;
```

JavaScript 注释

JavaScript的注释分为：

1. 单行注释

```
// 单行注释
```

2. 多行注释

```
/*
    多行
    注释
*/
```

数据类型

js中有6种数据类型，包括五种基本数据类型和一种复杂数据类型(object)

5种基本数据类型

1. number 数字类型

```
var iOne = 10.1;
```

2. string 字符串类型

```
var sStr = '1234';
```

3. boolean 布尔类型 true 或 false

```
var bIsTrue = false;
```

4. undefined undefined 类型，变量声明未被初始化，它的值就是undefined

```
var unData;
```

5. null null 类型，表示空对象，如果定义的变量将用来准备保存对象，可以将变量初始化为null，在页面上获取不到对象，返回值就是null

```
var nullData = null;
```

1种复合类型

1. object 后面学习的JavaScript对象属于复合类型

```
var oObj = {  
  name: '隔壁老王',  
  age: 88  
}
```

获取变量的类型

```
var type = typeof(oObj);  
alert(type);  
// 获取对象的name属性  
alert(oObj.name);
```

注： null 属于 null 类型，由于js历史原因，设计之前没有 null 类型， null 类型归属于object类型，后续语言迭代也没有进行修改

变量命名规范

1. 区分大小写
2. 第一个字符必须是字母、下划线(_)、或者美元符号(\$)
3. 其他字符可以是字母，下划线、美元符号或数字

匈牙利命名风格

对象o Object 比如: `oDIV`

数组a Array 比如: `aItems`

字符串s String 比如: `sUserName`

整数i Integer 比如: `iItemCount`

布尔值b Boolean 比如: `bIsComplete`

浮点数f Float 比如: `fPrice`

函数fn Function 比如: `fnHandler`

4.函数的定义和调用

函数定义

函数就是可以重复使用的代码块, 使用关键字 `function` 定义函数

```
function fuAlert(){
    alert('hello! ');
}
```

函数调用

函数调用就是函数名加小括号, 比如: `函数名(参数[参数可选])`

```
// 定义函数
function fnAlert(){
    alert('hello! ');
}
// 调用函数
fnAlert();
```

定义有参数有返回值的函数

定义函数时, 函数如果有参数, 参数放到小括号里面, 函数如果有返回值, 返回值通过 `return` 关键字来返回

```
function fnAdd(iNum1,iNum2){
    var iRs = iNum1 + iNum2;
    return iRs;
}

var iCount = fnAdd(10,20);
alert(iCount); // 弹出30
```

函数中 `return` 关键字的作用:

1. 返回函数中的值
2. 执行完 `return` 函数执行结束

5.变量作用域

变量作用域就是变量的使用范围

局部变量

局部变量就是在函数内使用的变量，只能在函数内部使用

```
<script>
    function fnShow() {
        var iNumber = 5;
        alert(iNumber);
    }
    fnShow();
</script>
```

全局变量

全局变量就是在函数外定义的变量，可以在不同函数内使用

```
<script>
    var iNumber = 10;
    function fnShow() {
        alert(iNumber);
        iNumber = 5;
        // ++ 等价于 +=1
        iNumber++;
        alert('修改后的全局变量: ' + iNumber);
    }
    fnShow();
    // 在js中可以将数字类型和字符串直接进行相加，把数字自动转成字符串
    alert('全局变量: ' + iNumber);
</script>
```

6.条件语句

条件语句语法

1. `if` 语句：只有当指定条件为 `true` 时，使用该语句来执行代码
2. `if...else` 语句：当条件为`true`时执行代码，当条件为`false`时执行其他代码
3. `if...else if...else` 语句：使用该语句来判断多条件，执行条件成立的语句

比较运算符

假如 `x=5`，查看比较后的结果

比较运算符	描述	例子
==	等于	x == 8 为 false
===	全等(值和类型)	x === 5 为 true; x === '5' 为 false
!=	不等于	x != 8 为 true
>	大于	x > 8 为 false
<	小于	x < 8 为 true
>=	大于等于	x >= 8 为 false
<=	小于等于	x <= 8 为 true

示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    var iNum = 2;
    var iStr = '2';
    // 在js中如果数字和字符串进行比较(==、>、<、>=、<=、!=)，会自动将字符串类型转换成数字类型再进行比较
    if (iNum == iStr) {
      alert('条件成立');
    }

    if (iNum > iStr) {
      alert('条件成立');
    } else {
      alert('条件不成立')
    }

    var iScore = 85;

    if (iScore >= 90) {
      alert('优秀');
    } else if (iScore >= 80) {
      alert('良好');
    } else if (iScore >= 60) {
      alert('及格');
    } else {
      alert('不及格');
    }
  </script>
</head>
<body>

</body>
</html>
```

逻辑运算符

假如 `x=6` , `y=3` , 查看比较后的结果

比较运算符	描述	例子
<code>&&</code>	and	<code>(x > 10 && y > 1)</code> 为true
<code> </code>	or	<code>(x == 5 y == 5)</code> 为false
<code>!</code>	not	<code>!(x == y)</code> 为 true

示例:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    var x = 6;
    var y = 3;

    if (x < 10 && y > 1) {
      alert("都大于");
    } else {
      alert("至少有一个不大于");
    }

    if (x > 5 || y < 7) {
      alert("至少有一个成立");
    } else {
      alert("都不成立");
    }

    if (!(x == y)) {
      alert("条件不成立");
    } else {
      alert("条件成立");
    }
  </script>
</head>
<body>

</body>
</html>
```

7. 获取标签元素

获取标签元素

可以使用**内置对象** `document` 上的 `getElementById` **方法**来获取页面上设置了id属性的标签元素，获取到的是一个 `html` 对象，然后将他赋值给一个变量，比如：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    // 如果获取的对象是null，表示标签没有获取成功
    var oDiv1 = document.getElementById("div1");
    alert(oDiv1);
  </script>
</head>
<body>
  <div id="div1">这是一个测试页面</div>
</body>
</html>
```

说明：

上面的代码，如果把JavaScript写在元素的上面，就会出错，因为页面是从上往下加载执行的，JavaScript去页面上获取 `div1` 元素的时候，元素 `div1` 还没有加载

解决方法：

第一种方法：将 javascript 放到页面的最下面

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

</head>
<body>
  <div id="div1">这是一个测试页面</div>
</body>
</html>
<script>
  var oDiv1 = document.getElementById("div1");
  alert(oDiv1);
</script>
```

注：这种写法不推荐，知道即可

第二种方法：设置页面加载完成的函数，再执行函数里面获取元素标签

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    window.onload = function() {
      var oDiv1 = document.getElementById("div1");
      alert(oDiv1);
    }
  </script>
</head>
<body>
  <div id="div1">这是一个测试页面</div>
</body>
</html>

```

说明：

onload 是页面所有元素加载完成的时间，给 onload 设置函数时，当事件触发就会执行设置的函数。

8.操作标签元素属性

属性的操作

首先获取得页面标签元素，然后就可以对页面标签元素的属性进行操作，属性的操作包括：

- 属性的读取
- 属性的设置

属性名在 js 中的写法

1. html 的属性和 js 里面属性大多数写法一样，但是 class 属性写成 className
2. style 属性里面的属性，有横杠的改成驼峰式，比如：font-size 改成 style.fontSize

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .btnstyle {
      background: yellow;
      width: 80px;
      height: 50px;
    }
  </style>
  <script>
    window.onload = function() {
      // 根据id获取标签对象
      var oBtn = document.getElementById('btn1');
      // 获取标签属性
      alert(oBtn.type);
      alert(oBtn.value);
      // 设置标签属性
    }
  </script>

```

```

        oBtn.name = 'username';
        // oBtn.style.background = 'red';
        // 相当于设置标签的class属性
        oBtn.className = 'btnstyle';

        oBtn.style.fontSize = '20px';
    }
</script>
</head>
<body>
    <input type="button" value="按钮" id="btn1">
</body>
</html>

```

innerHTML

innerHTML 可以读取或者设置标签包裹的内容

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        window.onload = function() {
            var oDiv1 = document.getElementById("div1");
            oDiv1.innerHTML = "还睡! 收你来啦~";
        }
    </script>
</head>
<body>
    <div id="div1">
        哈哈哈, 我是测试页面
    </div>
</body>
</html>

```

9. 数组及操作方法

数组的定义

```

// 实例化对象方法创建
var aList = new Array(1,2,3,4,5);

// 字面量方式创建, 推荐使用
var aList2 = [1,2,3,4,5];

```

多维数组

多维数组指的是数组的成员也是数组，把这样的数组叫做多维数组

```
var aList = [[1,2,3,4,5],['a','b','c','d','e'],[true,false]];
```

数组的操作

1. 获取数组的长度

```
var aList = [1,2,3,4,5];  
alert(aList.length);  
// 弹出 5
```

2. 根据下标取值

```
var aList = [1,2,3,4,5];  
alert(aList[2]);  
// 弹出 3  
  
// 下标修改数据  
alert(aList[1]) = 26;
```

注：在js中不支持负数下标

3. 从数组最后添加和删除数据

```
var aList = [1,2,3,4,5];  
aList.push(6);  
alert(aList);  
// 弹出 1,2,3,4,5,6  
var oValue = aList.pop();  
alert(aList);  
// 弹出 1,2,3,4,5
```

注：pop() 不支持根据下标删除

4. 根据下标添加和删除元素

```
arr.splice(start,num,element1,.....elementN)
```

参数解析：

1. start：必需，开始删除的索引
2. num：可选：删除数组元素的个数
3. elementN：可选，在 start 索引位置要插入的新元素

此方法会删除从 start 索引开始的 num 个元素，并将 elementN 参数插入到 start 索引位置

```
var aList = [1,2,3,4,5];
// 插入数据
// 在下标为2的位置插入一个元素'苹果'
aList.splice(2,0,'苹果');
alert(aList);

// 删除数据
// 从下标3删除2个数据
aList.splice(3,2);
alert(aList);
```

10.循环语句

循环语句的介绍

循环语句就是让一部分代码重复执行，javascript中常用的循环语句有：

- for
- while
- do-while

for 循环

```
var aList = [1,2,3,4,5];

// for循环遍历数组
for(var i = 0; i < aList.length; i++) {
    // 根据下标获取数据
    console.log(aList[i]);
}
```

while 循环

```
var aList = [1,2,3,4,5];

// while循环遍历数组
var i = 0;
while(i < aList.length){
    console.log(aList[i]);
    i++;
}
```

do-while 循环

```
var aList = [1,2,3,4,5];

// do-while循环遍历数组
var i = 0;
do {
    // 这里需要完成取值操作,不管条件是否成立,do至少会执行一次
    console.log(aList[i]);
    i++;
} while(i < aList.length);
```

11.字符串拼接

字符串拼接使用“+”运算符

```
var iNum = 10;
var fNum = 20.5;
var sStr = "Hello, world!";

result = iNum + fNum + sStr;
alert(result);
// 弹出 30.5Hello, world!

// +=
iNum += sStr
alert(iNum)
// 弹出 10Hello, world!
```

说明:

数字和字符串拼接会自动进行类型转换(隐式类型转换),把数字类型直接转成字符串类型进行拼接

12.定时器

定时器的介绍

定时器就是在一段特定的时间后执行某段程序代码

定时器的使用

js 定时器有两种创建方式

1. `setTimeout(func[,delay,param1,param2,...])`: 以指定的时间间隔(以毫秒计)调用一次函数的定时器
2. `setInterval(func[,delay,param1,param2,...])`: 以指定的时间间隔(以毫秒计)重复调用一个函数的定时器

`setTimeout` 函数的参数说明:

- 第一个参数 `func`, 表示定时器要执行的函数名
- 第二个参数 `delay`, 表示时间间隔, 默认是0, 单位是毫秒
- 第三个参数 `param1`, 表示定时器执行函数的第一个参数, 以此类推传入多个执行函数对应的参数

调用一次

```
function fnShow(name,age){
    alert("OK!")
    alert(name + age)

    // 打印定时器的id
    console.log(oTid);
    // 销毁定时器
    clearTimeout(oTid);
}
var oTid = setTimeout(fnShow, 1000, '张三',18);
```

多次调用，点击按钮停止

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function fnShow(name,age){
            alert("OK!")
            alert(name + age)

            // 打印定时器的id
            console.log(oTid);
        }
        // 销毁定时器
        function fnDown(){
            clearInterval(oTid);
        }
        var oTid = setInterval(fnShow, 10000, '张三',18);
    </script>
</head>
<body>
    <input type="button" value="停止定时器" onclick="fnDown();">
</body>
</html>
```