Linux和编程

1. Linux 基础

2. Linux 文件权限

3. Docker

4. 代码怎么生成 可执行文件的?

5. 数据如何在内存中 存放 ?

一、Linux基础

计算机存储单元转换

•	1	Bytes	=	8bits
•	1	KB	=	1024 Bytes
•	1	MB	=	1024 KB
•	1	GB	=	1024 MB

32位 和 64位 操作系统

本质上来说、64位操作系统就是运算速度更快

这个位数指的是CPU的位数

简单来说就是 32位的CPU 有32根线

地址总线

简单概括就是: CPU想要操作内存、就得靠这个玩意

操作指的是 读取 和 写入

简单来说就是 64位的CPU 不一定有64根线
目前各种架构的CPU、通常是42根地址线

而且32位的CPU 在早期有特殊情况的
超过了32根地址线

最小单元 Bit 是什么？

32位CPU 最大支持内存4G的存储空间
也就是 4G 个字节、实际不到4G

4G就是 2的32次方、大约42w左右的
字节空间

这就是 32位 和 64位的最大区别、同时
也导致了运算速度、64位兼容32位软件等

32位 又称 X86　　64位 又称 X86-64

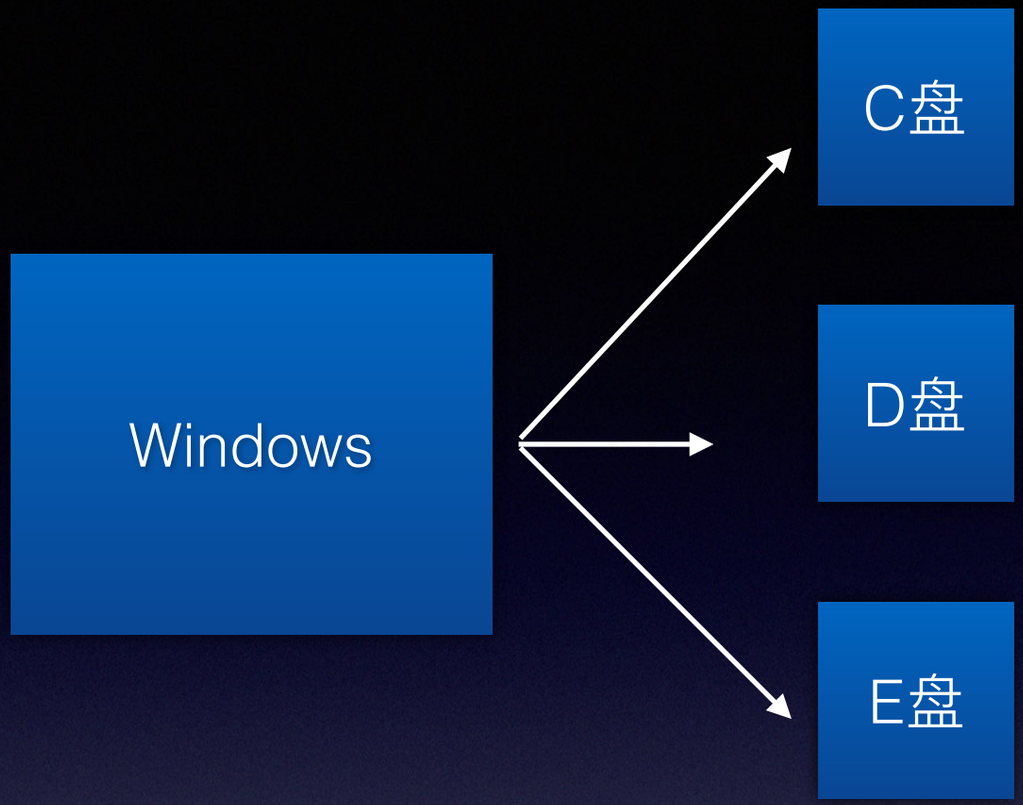
如何查看 自己电脑是32位 还是64位？

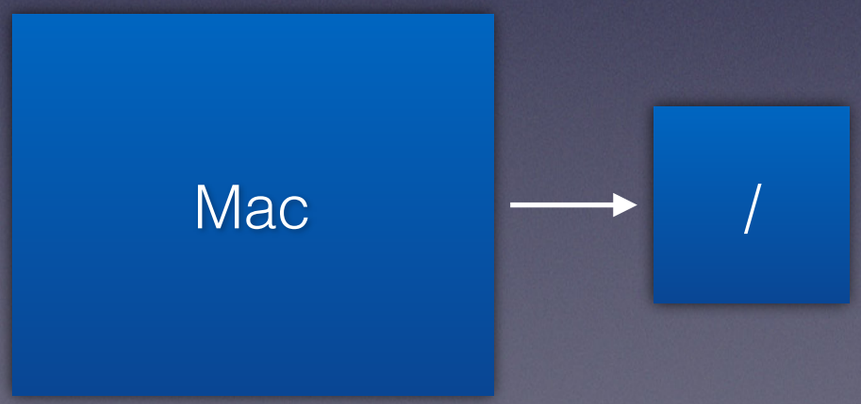
Mac 和 Linux 使用以下命令

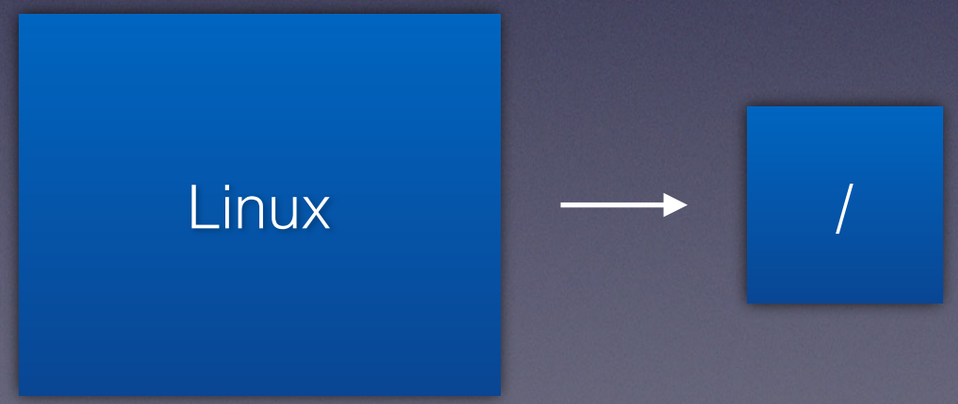
getconf LONG_BIT

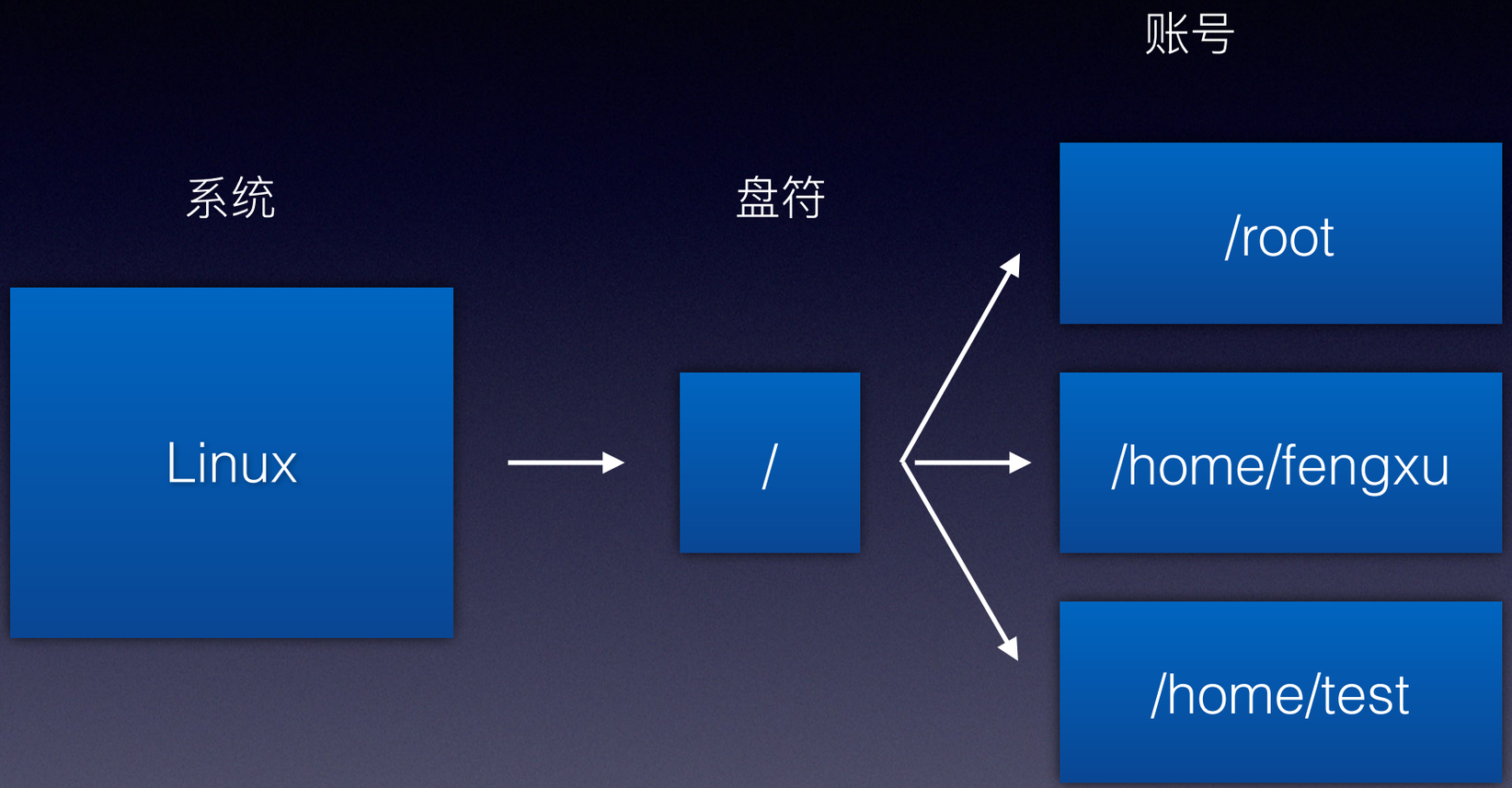
```
✘ fengxu@fengxudeMacBook-Pro ?~ ?getconf LONG_BIT
64
```

```
[root@dbctest ~]# getconf LONG_BIT
64
```


Windows
C盘
D盘
E盘


Mac
/


Linux
/


账号
系统
盘符
/root
Linux
/
/home/fengxu
/home/test

基础命令

```
[root@dbctest ~]# ls
golang-builder.tar  golang-runner.tar  go_runner.tar
```

```
[root@dbctest ~]# pwd
/root
```

```
[root@dbctest ~]# ll
总用量 1277468
-rw------- 1 root root 393220608 1月   14 2020 golang-builder.tar
-rw------- 1 root root   9935360 1月   14 2020 golang-runner.tar
-rw------- 1 root root 882158080 9月    3 09:29 go_runner.tar
drwxr-xr-x 4 root root      4096 5月    9 09:59 inquire_task
-rw------- 1 root root  22786560 2月    4 2020 nginx-runner.tar
-rw-r--r-- 1 root root         0 8月    2 09:52 sed
drwxr-xr-x 3 root root      4096 3月   18 2019 test_mysql
```

```
[root@dbctest ~]# ls
golang-builder.tar  golang-runner.tar  inquire_task  nginx-ru
[root@dbctest ~]# cp golang-runner.tar gr.rar
[root@dbctest ~]# ls
golang-builder.tar  golang-runner.tar  gr.rar  inquire_task
```

```
[root@dbctest ~]# mkdir test_mv
[root@dbctest ~]# ll
总用量 425688
-rw------- 1 root root 393220608 1月   14 2020 golang-builder.tar
-rw------- 1 root root   9935360 1月   14 2020 golang-runner.tar
-rw------- 1 root root   9935360 9月    8 19:05 gr.rar
drwxr-xr-x 4 root root      4096 5月    9 09:59 inquire_task
-rw------- 1 root root  22786560 2月    4 2020 nginx-runner.tar
-rw-r--r-- 1 root root         0 8月    2 09:52 sed
drwxr-xr-x 2 root root      4096 9月    8 19:08 test_mv
drwxr-xr-x 3 root root      4096 3月   18 2019 test_mysql
```

```
[root@dbctest ~]# touch file1
[root@dbctest ~]# ll
总用量 425688
-rw-r--r-- 1 root root         0 9月    8 19:08 file1
-rw------- 1 root root 393220608 1月   14 2020 golang-builder.tar
-rw------- 1 root root   9935360 1月   14 2020 golang-runner.tar
-rw------- 1 root root   9935360 9月    8 19:05 gr.rar
drwxr-xr-x 4 root root      4096 5月    9 09:59 inquire_task
-rw------- 1 root root  22786560 2月    4 2020 nginx-runner.tar
-rw-r--r-- 1 root root         0 8月    2 09:52 sed
drwxr-xr-x 2 root root      4096 9月    8 19:08 test_mv
drwxr-xr-x 3 root root      4096 3月   18 2019 test_mysql
[root@dbctest ~]#
```

```
[root@dbctest ~]# echo $USER
root
```

```
[root@dbctest ~]# ls
file1  golang-builder.tar  golang-runner.tar  gr.rar  inquire_task  nginx-runn
[root@dbctest ~]# cd ..
[root@dbctest /]# ls
bin   clan  dev     etc     home  lib64  lost+found  mnt  proc  run         sbi
boot  data  docker  fengxu  lib   logs   media       opt  root  sass-test   srv
[root@dbctest /]# cd ~
[root@dbctest ~]# ls
file1  golang-builder.tar  golang-runner.tar  gr.rar  inquire_task  nginx-runn
[root@dbctest ~]# cd -
/
[root@dbctest /]# ls
bin   clan  dev     etc     home  lib64  lost+found  mnt  proc  run         sbi
boot  data  docker  fengxu  lib   logs   media       opt  root  sass-test   srv
[root@dbctest /]#
```

```
[root@dbctest /]# ll
总用量 88
lrwxrwxrwx.   1 root    root     7 2月   15 2019 bin -> usr/bin
dr-xr-xr-x.   5 root    root  4096 2月   15 2019 boot
drwxr-xr-x    2 root    root  4096 9月    6 12:06 clan
drwxr-xr-x    3 polkitd root  4096 8月   25 14:20 data
drwxr-xr-x   19 root    root  2980 5月   25 00:25 dev
drwxr-xr-x   16 root    root  4096 7月   31 15:55 docker
drwxr-xr-x.  81 root    root  4096 7月   19 15:58 etc
drwxr-xr-x    3 root    root  4096 2月   18 2020 fengxu
drwxrwxrwx.   4 root    root  4096 7月   19 15:58 home
lrwxrwxrwx.   1 root    root     7 2月   15 2019 lib -> usr/lib
lrwxrwxrwx.   1 root    root     9 2月   15 2019 lib64 -> usr/lib64
drwxr-xr-x    8 root    root  4096 1月   16 2020 logs
drwx------.   2 root    root 16384 2月   15 2019 lost+found
drwxr-xr-x.   2 root    root  4096 4月   11 2018 media
drwxr-xr-x.   2 root    root  4096 4月   11 2018 mnt
drwxr-xr-x.   3 root    root  4096 3月    4 2019 opt
dr-xr-xr-x  207 root    root     0 3月    4 2019 proc
dr-xr-x---.  11 root    root  4096 9月    8 19:17 root
drwxr-xr-x   26 root    root   820 6月   21 2019 run
drwxr-xr-x    3 root    root  4096 3月   26 2019 sass-test
lrwxrwxrwx.   1 root    root     8 2月   15 2019 sbin -> usr/sbin
drwxr-xr-x.   2 root    root  4096 4月   11 2018 srv
dr-xr-xr-x   13 root    root     0 3月   13 2019 sys
drwxrwxrwt.   9 root    root  4096 9月    9 03:22 tmp
drwxr-xr-x.  13 root    root  4096 2月   15 2019 usr
drwxr-xr-x.  19 root    root  4096 2月   15 2019 var
drwxr-xr-x    5 root    root  4096 4月   19 2019 wwwroot
[root@dbctest /]# clear
```

```
[root@dbctest /]#
```

```
[root@dbctest ~]# ls
file1  golang-builder.tar  golang-runner.tar  gr.rar
[root@dbctest ~]# cat file1
文件内容
文件内容文件内容
文件内容文件内容
文件内容文件内容
文件内容文件内容
文件内容文件内容
文件内容
[root@dbctest ~]#
```

```
[root@dbctest ~]# ls
file1  golang-builder.tar  golang-runner.tar
[root@dbctest ~]# rm file1
rm: 是否删除普通文件 "file1"? y
[root@dbctest ~]#
```

useradd: 新增用户

passwd: 修改密码等操作

userdel: 删除用户等操作

groupadd: 新增用户组等操作

groupmod: 修改用户组等操作

groupdel: 删除用户组等操作

kill: 停止进程等操作

top: 显示进程信息等操作

systemctl: 管理服务的方式

netstat: 显示网络状态

```
[root@dbctest ~]# systemctl redis start
Unknown operation 'redis'.
[root@dbctest ~]# systemctl redis stop
Unknown operation 'redis'.
[root@dbctest ~]# systemctl enable redis
Failed to execute operation: No such file or directory
[root@dbctest ~]#
```

```
[root@dbctest ~]# systemctl nginx status
Unknown operation 'nginx'.
[root@dbctest ~]# systemctl nginx stop
Unknown operation 'nginx'.
[root@dbctest ~]# systemctl enable nginx
Failed to execute operation: No such file or directory
```

```
[root@dbctest ~]# netstat  -anp  |grep 8081
tcp6       0      0 :::8081                 :::*                    LISTEN      5562/docker-proxy
[root@dbctest ~]#
```

```
[root@dbctest ~]# netstat  -anp  |grep 8081
tcp6       0      0 :::8081                 :::*
[root@dbctest ~]#
```

三、Linux文件权限

用户系统

Linux 账户

用户组

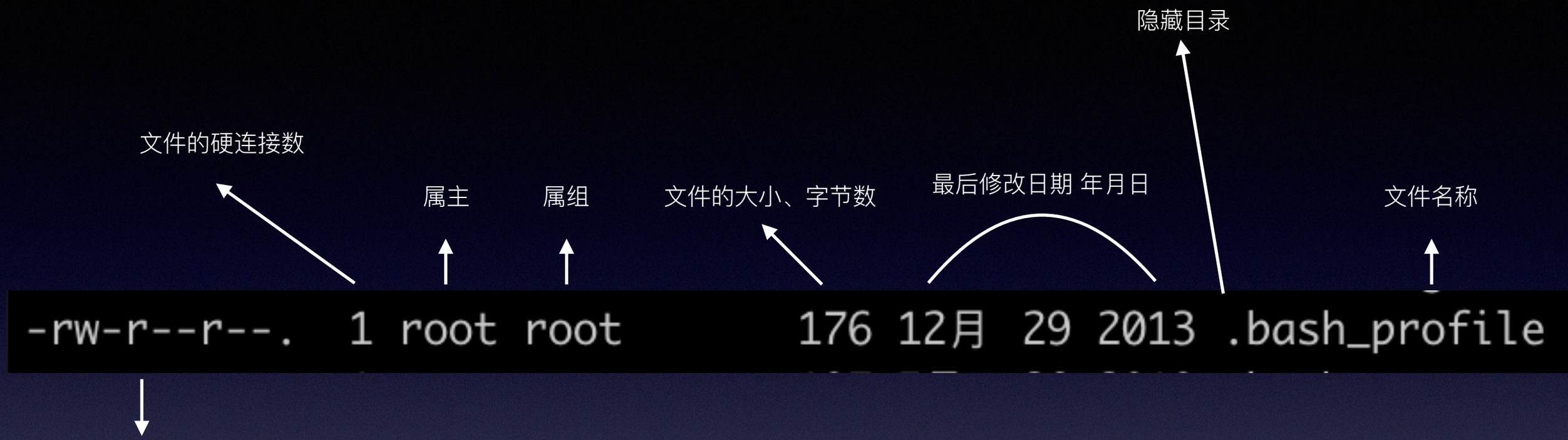
用户

列出所有文件

```
[root@dbctest ~]# ls -la
总用量 425776
dr-xr-x---. 11 root root      4096 9月   10 11:22 .
dr-xr-xr-x. 25 root root      4096 8月   29 15:10 ..
-rw-------   1 root root     11738 9月    9 20:10 .bash_history
-rw-r--r--.  1 root root        18 12月 29 2013 .bash_logout
-rw-r--r--.  1 root root       176 12月 29 2013 .bash_profile
-rw-r--r--.  1 root root       197 5月   28 2019 .bashrc
drwxr-xr-x   3 root root      4096 2月   15 2019 .cache
-rw-r--r--.  1 root root       100 12月 29 2013 .cshrc
drwx------   2 root root      4096 4月   18 2019 .docker
-rw-r--r--   1 root root        93 8月   21 2019 .gitconfig
-rw-------   1 root root       164 9月   10 11:16 .git-credentials
-rw-------   1 root root 393220608 1月   14 2020 golang-builder.tar
-rw-------   1 root root   9935360 1月   14 2020 golang-runner.tar
-rw-------   1 root root   9935360 9月    8 19:05 gr.rar
drwxr-xr-x   4 root root      4096 5月    9 09:59 inquire_task
-rw-------   1 root root  22786560 2月    4 2020 nginx-runner.tar
drwxr-xr-x   2 root root      4096 2月   15 2019 .pip
drwxr-----   3 root root      4096 3月    4 2019 .pki
-rw-r--r--   1 root root       205 3月    4 2019 .pydistutils.cfg
-rw-r--r--   1 root root         0 8月    2 09:52 sed
drwxr-xr-x   2 root root      4096 8月   20 17:12 .ssh
-rw-r--r--.  1 root root       129 12月 29 2013 .tcshrc
drwxr-xr-x   2 root root      4096 9月    8 19:08 test_mv
drwxr-xr-x   3 root root      4096 3月   18 2019 test_mysql
-rw-------   1 root root      8941 9月   10 11:22 .viminfo
drwxr-xr-x   5 root root      4096 1月   10 2020 .vscode-server
```

Linux下面一切皆文件，所以列出目录下的内容，也相当于列出目录文件里面的内容，进入目录，就相当于进入目录文件里面

- chgrp： 改变档案所属群组
- chown： 改变档案所属人
- chmod： 改变档案的属性、 等等的特性



档案属性、共10个属性

- 代表这个档案的类型

后面9位 每三位是一组
可读 可写 可执行

rw-属主 r—属组 r—其它用户的操作

当为[d]则是目录，例如上表的第 11 行；

当为[-]则是档案，例如上表的第 5 行；

若是[l]则表示为连结档(link file)；

若是[b]则表示为装置文件里面的可供储存的接口设备；

若是[c]则表示为装置文件里面的串行端口设备，例如键盘、鼠标。

三、Docker

What is docker ?

通常情况下 大家都理解为虚拟机、轻量级的虚拟机。和虚拟机是两回事

虚拟机是 虚拟出来一个操作系统、里面各种资源都是虚拟出来的

而Docker 虚拟出来的是容器 而且还不完整

所有的容器共用 宿主机上面的资源

容器和虚拟机的隔离不在一个等级

传统应用上线、部署

不使用docker、使用服务器部署应用

网络 使用的系统网络

端口 直接绑定的
系统网卡上的端口、
不易扩展

文件 直接在linux系统上
一旦应用OOM了、服务器
也承载不了、这么大压力
应用会相互影响

Docker 使用

映射、资源按隔离

网络

资源(文件)

端口

Docker 资源隔离

一个没有资源限制的容器、基本可以使用宿主机器的所有资源、如果可以是用所有资源、那么和 直接部署在宿主机 没什么区别
Docker 控制 内存、cpu、io、提出了资源隔离

资源隔离借住了、 Linux系统内核的机制

namespace

cgroup

Linux 内核 namespace

文件系统需要是被隔离的

网络也是需要被隔离的

进程间的通信也要被隔离

针对权限，用户和用户组也需要隔离

进程内的PID也需要与宿主机中的PID进行隔离

容器也要有自己的主机名

Namespace 可以做到这些、但是有很多资源不能被隔离
比如 容器内的时间、和宿主机的时间、就隔离不了、
再比如 任务的挂起 恢复等操作

cgroup - 内核工具

cgroups 为资源管理 提供了一个统一接口、
从单个资源的控制、到操作系统层面的虚拟化

资源限制、比如使用内存的上限、CPU的上限

任务控制 比如 容器的暂停 就是使用这个

资源统计 CPU使用时长 内存用量等

总的来说、 Docker 是通过 内核的 namespace、cgroup
实现的 资源隔离、 资源限额、包括一定的虚拟化技术

namespace使得容器像一台独立的计算机，
namespace实现容器间资源隔离

cgroup 使用限制 容器使用的 内存、CPU、IO写入

Docker 使用

docker --help	查看帮助文档 命令
docker ps	查看在运行的容器
docker stop	停止一个容器
docker rm	删除一个容器
docker rmi	删除一个镜像
docker search xxx	搜索一个镜像
docker pull	拉取一个镜像
docker restart	重启一个容器

Docker 使用

docker logs	查看容器的日志
docker exec	进去容器的指令
docker export	导出容器
docker import	导入容器
docker rm	删除一个容器
docker rmi	删除一个镜像
docker search xxx	搜索一个镜像
docker pull	拉取一个镜像
docker restart	重启一个容器

```
[root@dbctest ~]# docker ps
CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STATUS              PORTS
            NAMES
58b44dfef631        shop_mobile_back:mirror                "./shop_mobile_back"     6 days ago          Up 6 days           0.0.0.0:9193->9193/tcp
            mirror_shop_mobile_back_1
1b3bf726bc95        saas_operation_front:test              "nginx -g 'daemon of…"   6 days ago          Up 6 days           80/tcp, 0.0.0.0:8082->8082/tcp
            saas_operation_front_operation_front_1
470109e32aba        test_image                             "/bin/sh -c ./run-de…"   7 days ago          Up 7 days           0.0.0.0:8081->8081/tcp
            saas_op_backend_dev
b3e2dfa52f86        shop_admin_back:mirror                 "./shop_admin_back"      12 days ago         Up 12 days          0.0.0.0:9191->9191/tcp
            mirror_shop_admin_back_1
b2b0ec3dddc0        test_image                             "bee run -gendoc=tru…"   4 weeks ago         Up 3 hours          0.0.0.0:9093->9093/tcp
            shop_mobile_back_dev
b7065802b64c        shop_admin_back:test                   "./shop_admin_back"      2 months ago        Up 2 months         0.0.0.0:9091->9091/tcp
            test_shop_admin_back_1
4ba41a98c9a2        account_server:test                    "./account_server"       5 months ago        Up 5 months         443/tcp, 0.0.0.0:8001->80/tcp
            test_account_server_1
2bd5c123a282        saas_pc_gateway:test                   "./saas_pc_gateway"      5 months ago        Up 5 months         0.0.0.0:8003->80/tcp
            test_saas_pc_gateway_1
ad1dc21011cf        shop_uni_front:mirror                  "nginx -g 'daemon of…"   6 months ago        Up 6 months         80/tcp, 0.0.0.0:1448->1448/tcp
            mirror_cashier_desk_front_1
88036fdf9e94        shop_uni_front:test                    "nginx -g 'daemon of…"   6 months ago        Up 6 months         80/tcp, 0.0.0.0:1444->1444/tcp
            shop_mobile_uni_cashier_desk_front_1
f151e782b44c        test_image                             "go run main.go"         7 months ago        Up 7 months         0.0.0.0:13301->13301/tcp
            saas_pc_server_dev
ad8fc1e345c9        test_image                             "bee run -gendoc=tru…"   7 months ago        Up 5 months         0.0.0.0:9797->9797/tcp
            cashier_desk_back_dev
b2b63d909a53        cashier_desk_front:test                "nginx -g 'daemon of…"   7 months ago        Up 7 months         80/tcp, 0.0.0.0:1443->1443/tcp
            cashier_desk_front_cashier_desk_front_1
123ee26a4082        sms_server:test                        "./sms_server"           9 months ago        Up 9 months         443/tcp, 0.0.0.0:8000->80/tcp
            test_sms_server_1
19efc31a916e        swaggerapi/swagger-editor              "sh /usr/share/nginx…"   14 months ago       Up 14 months        0.0.0.0:8083->8080/tcp
            flamboyant_banach
3f31cd61a0fc        rabbitmq:3.7.15-management             "docker-entrypoint.s…"   14 months ago       Up 14 months        4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp, 25672/tcp, 0.0.0
>15672/tcp  elastic_thompson
877b9dd8d0fa        wurstmeister/zookeeper                 "/bin/sh -c '/usr/sb…"   15 months ago       Up 15 months        22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp
            zookeeper
8e07811f270f        openresty/openresty                    "/usr/bin/openresty …"   15 months ago       Up 15 months        0.0.0.0:3001->80/tcp
            openresty_1
9117a7d5a97c        redis                                  "docker-entrypoint.s…"   15 months ago       Up 15 months        0.0.0.0:6379->6379/tcp
            redis
7d410c1e892d        nginx:latest                           "nginx -g 'daemon of…"   16 months ago       Up 2 months         0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:8080->8080/tcp
            nginx-latest
a4eaf4aa8c82        39.96.27.29:5000/php/php_image:php56   "docker-php-entrypoi…"   17 months ago       Up 2 weeks          9000/tcp
            php-56
fcdbc02085d0        mysql:5.7                              "docker-entrypoint.s…"   18 months ago       Up 3 days           33060/tcp, 0.0.0.0:3307->3306/tcp
            mysqltest2
```

Example

方式一

[root@dbctest saas_operation_backend]# ls

saas_operation_backend up.sh

[root@dbctest saas_operation_backend]# cat up.sh

docker run -d -v "$(pwd)":/go/src/ -w /go/src/saas_operation_backend -p 8081:8081 -e export BEEGO_RUNMODE=dev -e export GO111MODULE=auto -e export GOPROXY=https://goproxy.cn --name saas_op_backend_dev test_image /bin/sh -c 'go mod tidy \n bee -gendoc=true -downdoc=true'

docker run　　-d 后台运行　　-v 文件挂载　　-w 容器工作目录

-p 指定端口 主机端口:容器端口　　-e 设置环境变量　　- - name 设置容器名称

test_image 指定使用镜像　　sh -c 执行shell脚本

方式二

File1: saas_operation_front.dockerfile

File2: docker-compose.yml

File3: nginx.conf

docker-compose up --build -d

2　　1　　3

```
[root@dbctest saas_operation_front]# cat docker-compose.yml
version: "3.5"
services:
  operation_front:
    restart: always
    build:
      context: .
      dockerfile: saas_operation_front.dockerfile
    image: saas_operation_front:test
    ports:
      - 8082:8082
    # volumes:
    #   - ./dist/:/html/
    networks:
      - saas_operation_network

networks:
  saas_operation_network:
    driver: bridge
```

restart: always 不管退出状态码是什么始终重启容器

```
[root@dbctest saas_operation_front]# cat saas_operation_front.dockerfile
FROM 39.96.27.29:5000/library/react:builder as builder
WORKDIR /root
RUN git clone http://f[redacted]@39.96.27.29/wangzl/saas_operation_forent.git
WORKDIR /root/saas_operation_forent
RUN yarn && npm run build && cp -r ./dist /release

FROM 39.96.27.29:5000/library/nginx:runner

COPY --from=builder /release /html
ADD ./nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 8082

CMD ["nginx", "-g", "daemon off;"]
```

Nginx 官方仓库解释

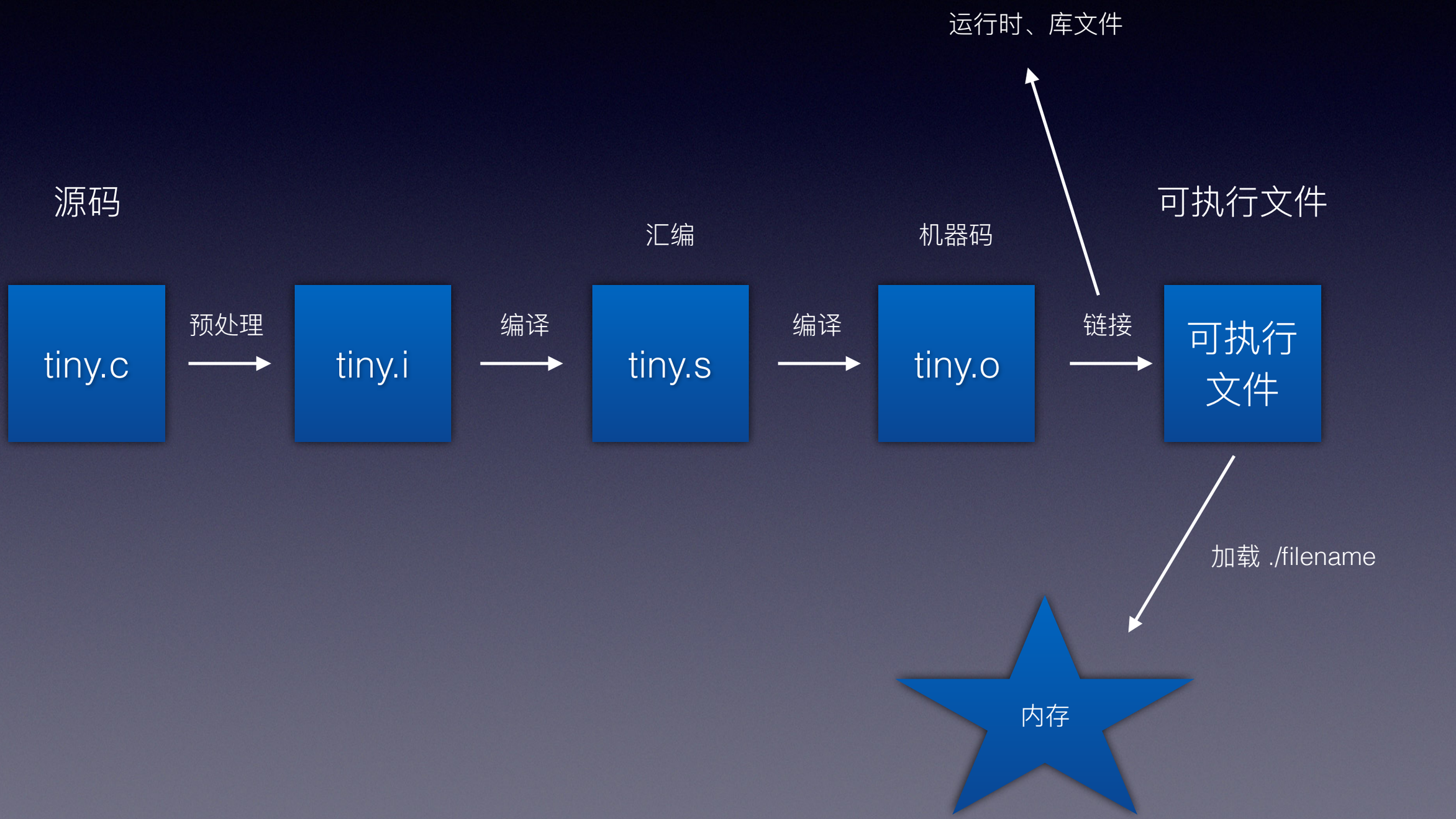
Docker 容器启动时，默认会把容器内部第一个进程，也就是pid=1的程序，作为docker容器是否正在运行的依据，如果 docker 容器pid=1的进程挂了，那么docker容器便会直接退出。

Docker未执行自定义的CMD之前，nginx的pid是1，执行到CMD之后，nginx就在后台运行，bash或sh脚本的pid变成了1。

所以一旦执行完自定义CMD，nginx容器也就退出了

四、代码怎么生成 可执行文件的?

使用C语言 看一下 代码怎么一步步的编程可执行文件



汇编语言是有很多种的

汇编语言跟CPU绑定
主流的有 ARM 和 AMD

ARM 编译出来的、不能在AMD的机器上跑

AMD 编译出来的、不能在ARM的机器上跑

Example

五、数据如何存放到内存中的?

聊聊编程语言？

编程语言 =

可读性

+

组织的数据类型

+

流程控制

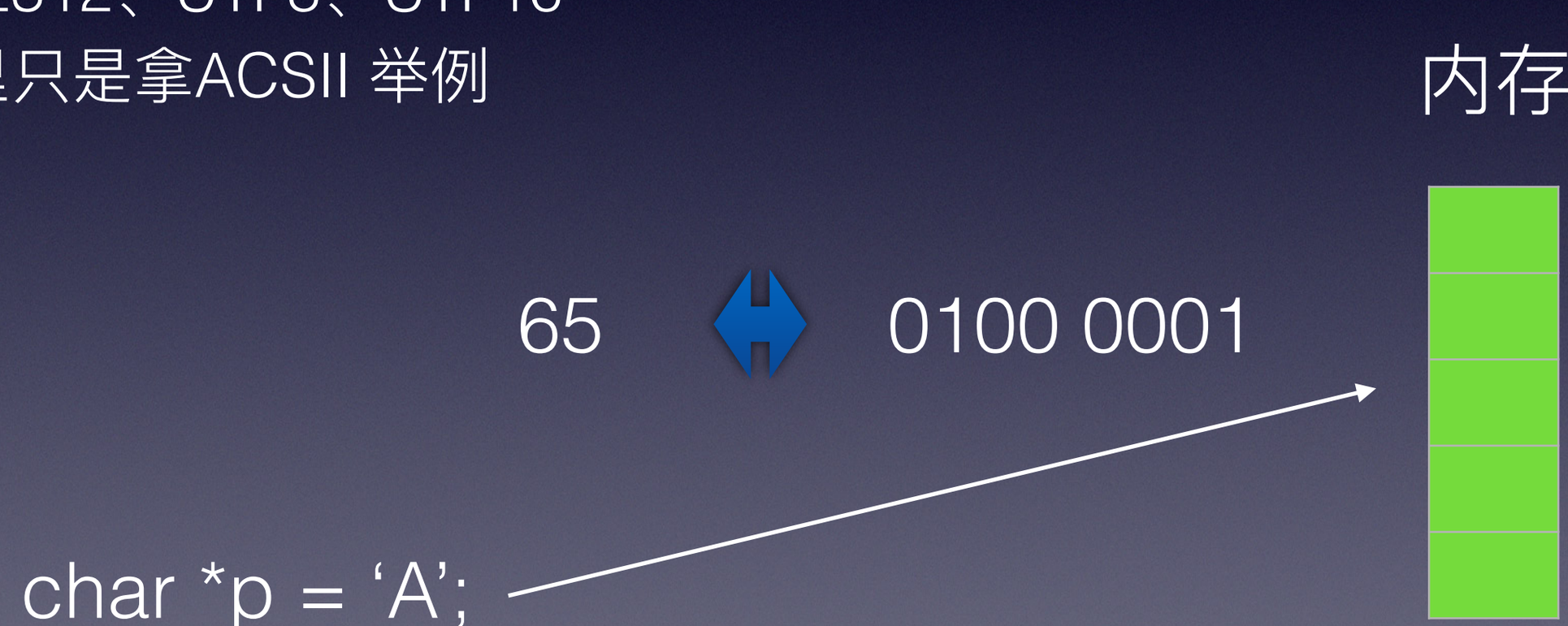
无论是代码 还是数据、或者文件、
在内存中存放的数据、肯定是 二进
制

计算机 运算、认识 乘法、除法、取余、次方的运算吗？

不认识、它只认识加法

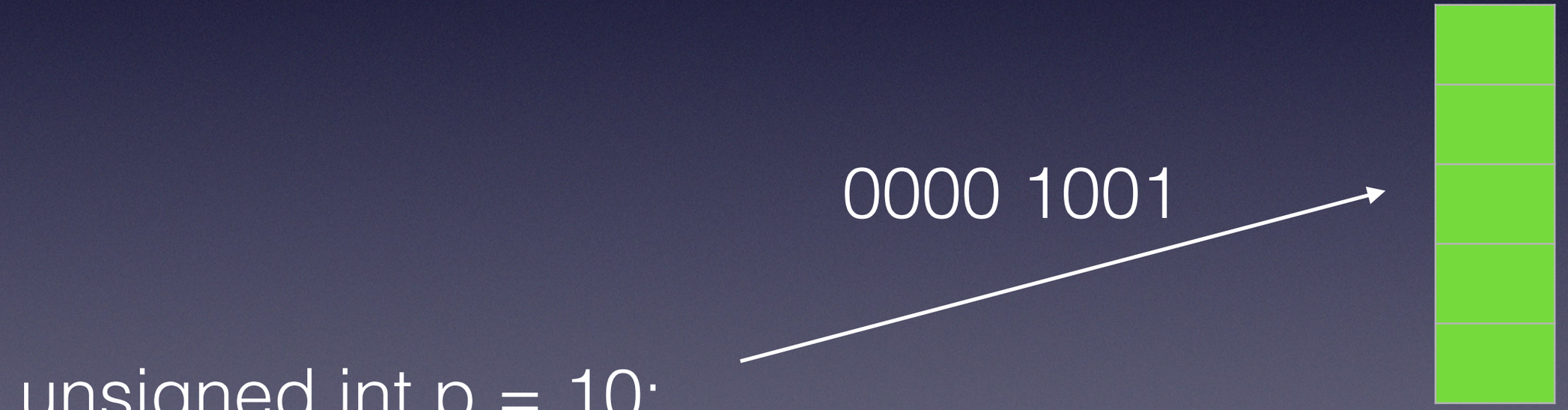
字符型的数据、是把对应的 ACSII码的值、存放到内存中的、
这个转换是操作系统做的、对用户(也就是程序猿)隐藏的

ACSII码 只是一种编码类型
GB2312、UTF8、UTF16
这里只是拿ACSII 举例



无符号、没有符号、 全部都是正整数、是整数、直接二进制编码

内存



Example

计算机中只有加法

1 - -1

运算时 拿数据的补码进行 运算

> 一个整数的补码、就是它的原码、将这个数转换为二进制

> 一个负数的补码、就是这个负数的绝对值的原码、按位取反加1

0000, 0000, 0000, 0000, 0000, 0000, 0000, 0001　1的补码（原码) 补码就是原码

1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111　-1的补码 (按位取反 加1)

1, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000　最后结果0

〔1减去 -1、往前进一位 但是计算机只有32位、所以溢出了、最后相加的结果都是0〕

Tthanks