day02 数据类型

本节设计内容范围：基础 ~ 数据类型

周末直播课的目标： 强化知识 + 实战案例 + 名企面试题 + 收集学员问题 + 答疑

今日概要：

- C语言基本知识
  - 字符和字符串（数组）
  - 指针（内存地址）
  - 结构体（链表）
- Python内存管理机制
- git入门
- 实战案例（7个）
- 企业面试题
- 学员答疑

1.聊聊C语言

1.1 字符和字符串

- 字符，用一个字节来存储字符。

```
char v1 = 'w'
```

- 字符串，用字符数组构造出来的一个字符串。

```
char v2[8] = {'w','u','p','e','i','q','i','\0'}  //
v1 = "wupeiqi"

char v3[] = "wupeiqi"
```

在C语言中字符串是由多个字符串构成的字符数组搞出来。

在C语言中创建字符数组时，内存地址已创建好了（并且是连续的）。

```
char v2[8] = {'w','u','p','e','i','q','i','\0'}

char v3[] = "wupeiqi"
```

```
假设：
    0x7ffee1b2b752
    0x7ffee1b2b753
    0x7ffee1b2b754
    0x7ffee1b2b755
    0x7ffee1b2b756
    0x7ffee1b2b757
    ...
    0x7ffee1b2b758
```

如果c语言中修改字符串：

- `wupeiqi` -> `wubeiqi`，可以
- `wupeiqi` -> `wupeipeipeipeiqi`，不支持直接字符串修改，而是新创建一份。

在Python中的字符串：

```
name = "alex"
data = name.upper()

v1 = "灿老师谢新雪"
v2 = v1.replace("新","旧的")
```

注意：Python中的字符串不是简单的像C语言存储。

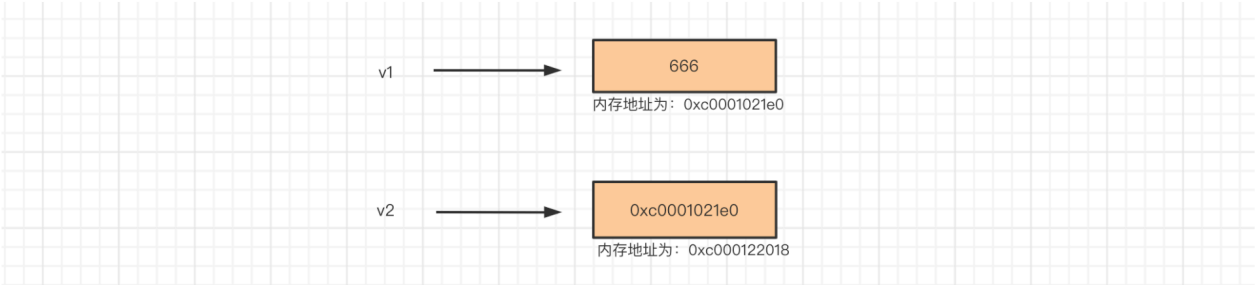
1.2 指针

Python的语法中没有提供指针。

想要在Python中创建一个指针（不可以）。

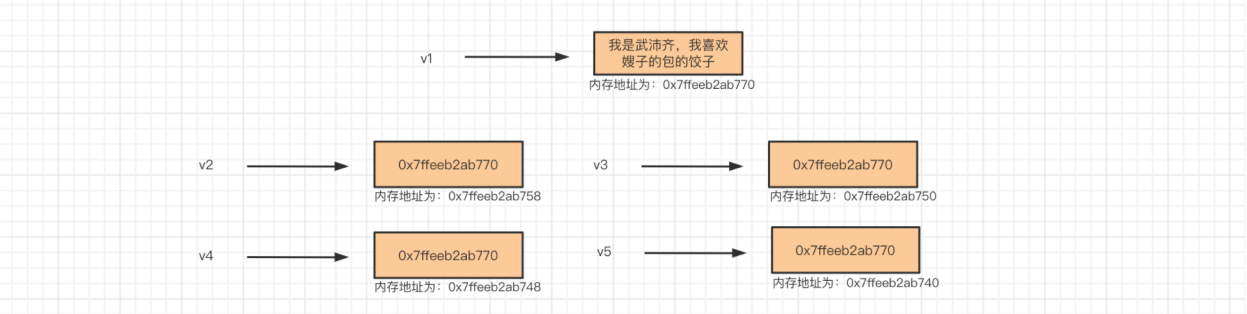
Python中的很多东西都是基于C语言的指针实现。

```
int v1 = 666;
int* v2 = &v1; // 指针
```



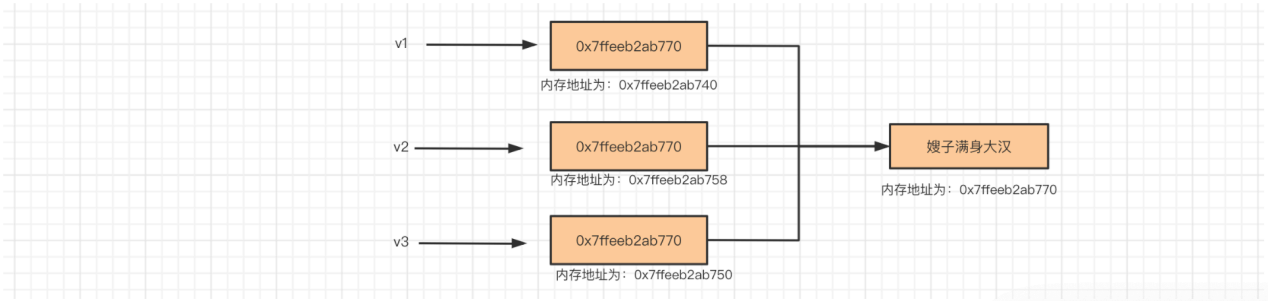
指针的好处节省内存的开销：

```
char v1[] = "我是武沛齐，我喜欢嫂子的包的饺子";

char *v2 = &v1[0];
char *v3 = &v1[0];
char *v4 = &v1[0];
char *v5 = &v1[0];
```



在Python中我会写：

```
v1 = "嫂子满身大汉"
v2 = v1
v3 = v1
```



```
v1 = "嫂子满身大汉"
v2 = v1
v3 = v1
v4 = "嫂子满身大汉"

# 理论上：在内存中再开辟一个空间去存储。
# 优化机制：重用这个字符串。（Python优化机制）
# 如果这个时候定义v4=嫂子满身大汉，是不是会另外开新的地址？
```

```
>>> v1 = 'wupeiqi'
>>> v2 = v1
>>>
>>>
>>> v3 = "alex"
>>> v4 = "alex"

>>> v5 = [11,22,33]
>>> v6 = [11,22,33]
>>> id(v5)
140231354254848
>>> id(v6)
140231354254912
>>>
>>> v7 = "alex"
>>> v8 = v7
>>> v7[0] = "A" # 不支持对字符串的元素进行修改。

>>> v9 = [11,22,33]
>>> v10 = v9
>>> v9[0] = 666
```

注意：重新赋值？还是修改内部元素。

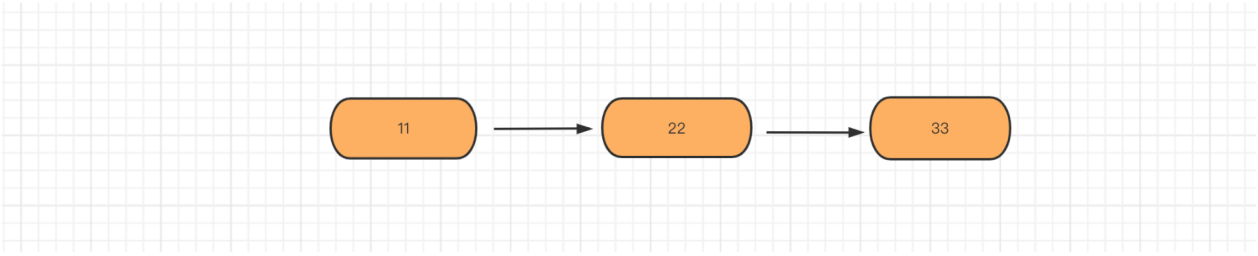
```
v7 = "alex"
v8 = "ALEX"
v9 = v7.upper()
```

1.3 结构体

```
int age = 18;
char name[30] = "武沛齐";
```

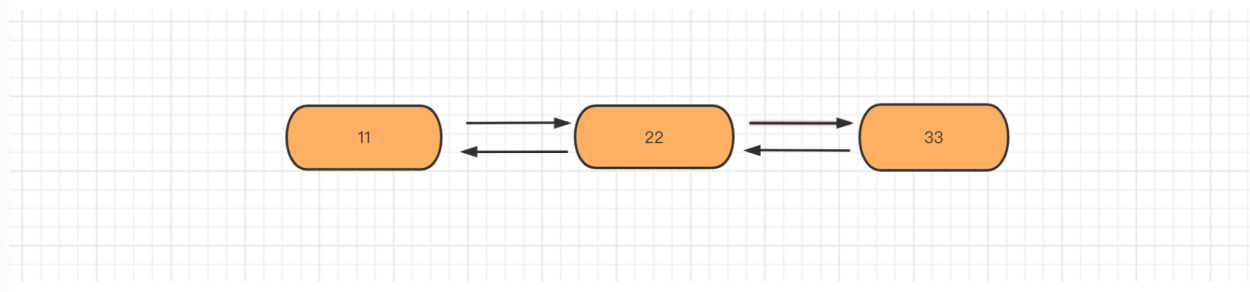
```
struct Person{
    char name[10];
    int age
}

struct Person p1 = {"苍老师",29};
struct Person p2 = {"有阪老师",21};
struct Person p3 = {"大桥老师",23};
p1.name
p1.age
```

1.单向链表



```
struct Node{
    int data;
    struct Person *next; // 指针，存储内存地址
}

struct Node n3 = {33};
struct Node n2 = {22, &n3};
struct Node n1 = {11, &n2};

n1.data
n1.next.data
n1.next.next.data
```

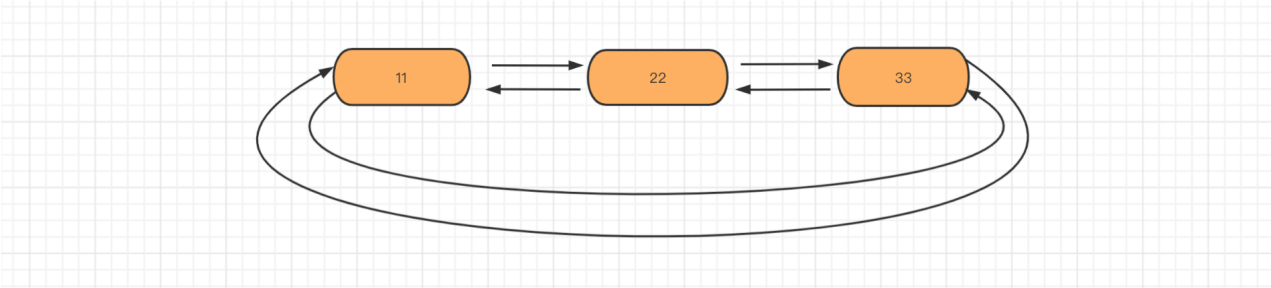
2.双向链表




```
struct Person
{
    int data;
    struct Person *next;
    struct Person *prev;
};

struct Person p3 = { 33 };
struct Person p2 = { 22 };
struct Person p1 = { 11 };

p1.next = &p2;
```

```
p2.next = &p3;
p2.prev = &p1;

p3.prev = &p2;
```

3.双向环状链表




```
struct Person
{
    int data;
    struct Person *next;
    struct Person *prev;
};

struct Person p3 = { 33 };
struct Person p2 = { 22 };
struct Person p1 = { 11 };

p1.next = &p2;
p1.prev = &p3;

p2.next = &p3;
p2.prev = &p1;

p3.prev = &p2;
p3.next = &p1;
```

2.Python内存管理机制

> 引用计数器为主、标记清楚和分代回收为辅。

- Python中的所有的东西最终都会根据底层C代码来实现。

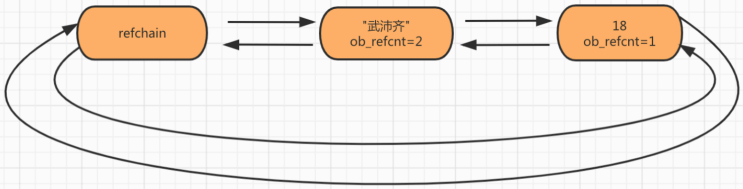
```
v1 = 0.9  # 浮点型 --> 结构体（4个值）     {1, 0.9, 上一个, 下一个 }
v2= 1     # 整型   -->  结构体（4个值）    {1, 1, 上一个, 下一个 }
```

- 环状双向链表 refchain，用于保存我们在代码中创建的所有值。
- 创建一个值

在refchain中的所有对象内部都有一个 ob_refcnt 用来保存当前对象的引用计数器，顾名思义就是自己被引用的次数，例如：

```
age = 18
name = "武沛齐"
nickname = name
```

上述代码表示内存中有 18 和 "武沛齐" 两个值，他们的引用计数器分别为：1、2 。



- 删除一个值

```
age = 18
name = "武沛齐"
nickname = name

del name       # name变量删除；引用计数器obj_refcnt减 1
del nickname   # name变量删除；引用计数器obj_refcnt减 1
===> 0，如果引用计数器=0则认为他没用，就是垃圾。

# 垃圾回收
#    1. 在环状链表中摘除。
#    2. 在内存中去销毁（缓存不销毁），以后如果再用的话。
```

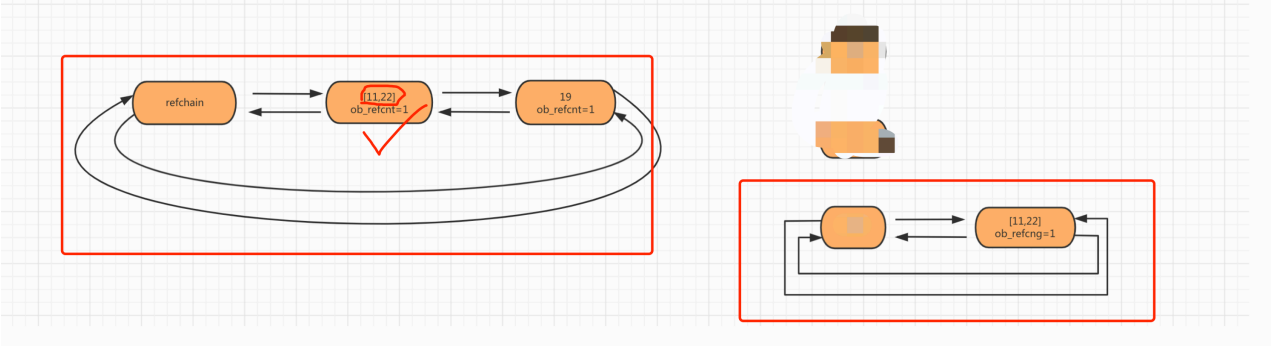
上述都是基于引用计数器实现的，如果Python仅用这种是不行的，因为：会存在循环引用导致有些数据永远无法销毁的情况。

```
v1 = [11,22,33] # [11,22,33,v2]
v2 = [33,44,55] # [11,22,33,v1]
v1.append(v2)
v2.append(v1)

# 两个列表数据内部的引用计数器都是 2
del v1
del v2
# 两个列表数据内部的引用计数器都是 1

=====》 以后代码中再也无法使用v1和v2《======
```

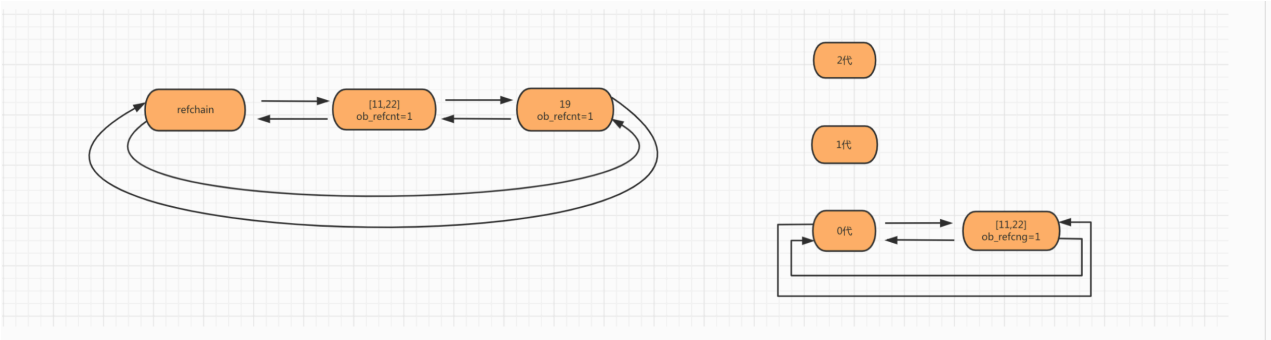
所以搞了一个`标记清楚`的技术。

- 专门维护一个链表去存储 "可变类型"（列表、元组、字典、集合、自定义类等对象）。

- 内部会监测是否出现了循环引用，如果出现循环应用就会让各自的引用计数器-1，再检查是否是0；垃圾。



为了提高扫描的效率，Python又搞了一个 `分代回收` 的技术。

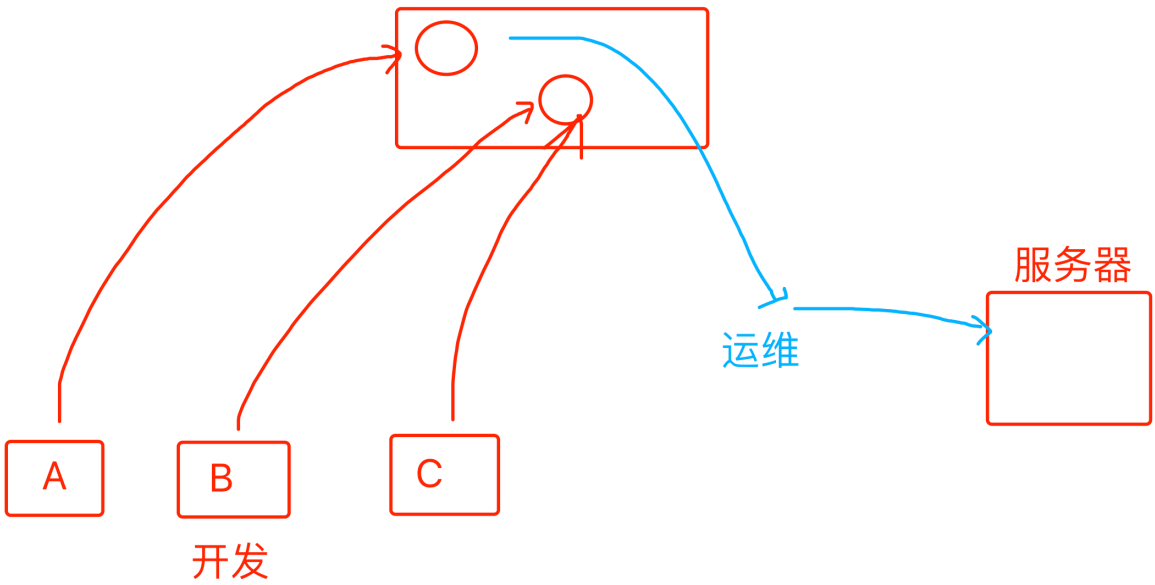



```
// 分代的c源码
#define NUM_GENERATIONS 3
struct gc_generation generations[NUM_GENERATIONS] = {
    /* PyGC_Head,
threshold,    count */
    {{(uintptr_t)_GEN_HEAD(0), (uintptr_t)_GEN_HEAD(0)},
  700,        0}, // 0代
    {{(uintptr_t)_GEN_HEAD(1), (uintptr_t)_GEN_HEAD(1)},
  10,         0}, // 1代
    {{(uintptr_t)_GEN_HEAD(2), (uintptr_t)_GEN_HEAD(2)},
  10,         0}, // 2代
};
```

3. git

Git软件，对文件夹中的文件版本控制（本地）。

代码仓库，保存所有代码，例如：github[公有/私有]、gitee[公有/私有]、gitlab（自己公司）。



3.1 Git

- 下载

  https://git-scm.com/

  https://git-scm.com/downloads

- 安装

- 使用它做版本控制

3.2 本地版本控制

1. 今日要管理的文件件
2. 初始化

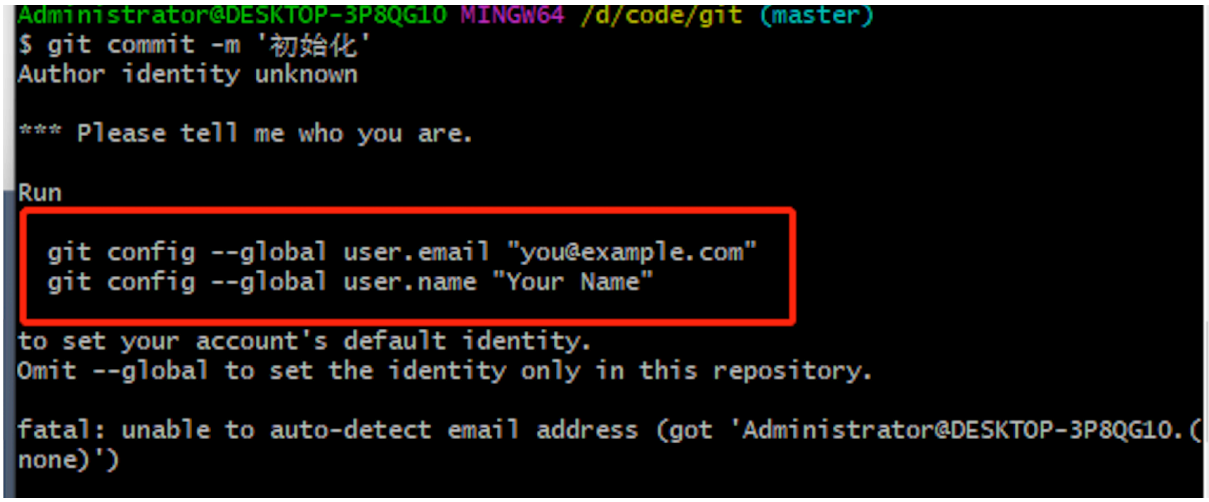
```
git init
```

3. 开发程序、写代码和修改代码....
4. 让git帮助我们将文件夹下的所有文件生成一个版本

```
git add .                    # 将当前文件夹中所有变化（新增、修改内容、删除）暂存起来。【红色->绿色】
git commit -m "日志"      # 生成一个提交记录
```

第一次执行git commit 时，提示你要git config 配置（姓名&邮箱），在生成记录的时候要写下。

```
git config ...
```



5. 继续开发、写代码和修改代码....
6. 再生成一个版本

```
git add .                  # 将当前文件夹中所有变化（新增、修改内容、删除）暂存起来。【红色->绿色】
git commit -m "日志"       # 生成一个提交记录
```

7. 继续开发、写代码和修改代码....
8. 再生成一个版本

```
git add .                  # 将当前文件夹中所有变化（新增、修改内容、删除）暂存起来。【红色->绿色】
git commit -m "日志"       # 生成一个提交记录
```

如果说已经开发了10个版本，如果想要回滚到某个之前的版本。

- 查看所有的版本记录

```
git log
```

```
commit 171ff4152a248d3ec49b52cdbe10129637cbd14a (HEAD
-> master)
Author: wupeiqi <wupeiqi@live.com>
Date:   Sun Mar 14 14:10:56 2021 +0800

    开发了新功能直播

commit 90db1d73717bce421e93fcd14f542bc93ee83f70
Author: wupeiqi <wupeiqi@live.com>
Date:   Sun Mar 14 14:08:13 2021 +0800

    初始化所有的文件
```

- 回滚到指定版本

```
git reset --hard
90db1d73717bce421e93fcd14f542bc93ee83f70
```

- 又让我跳回到之前最新的版本。

```
git reflog
git reset --hard 版本号
```

3.3 代码仓库

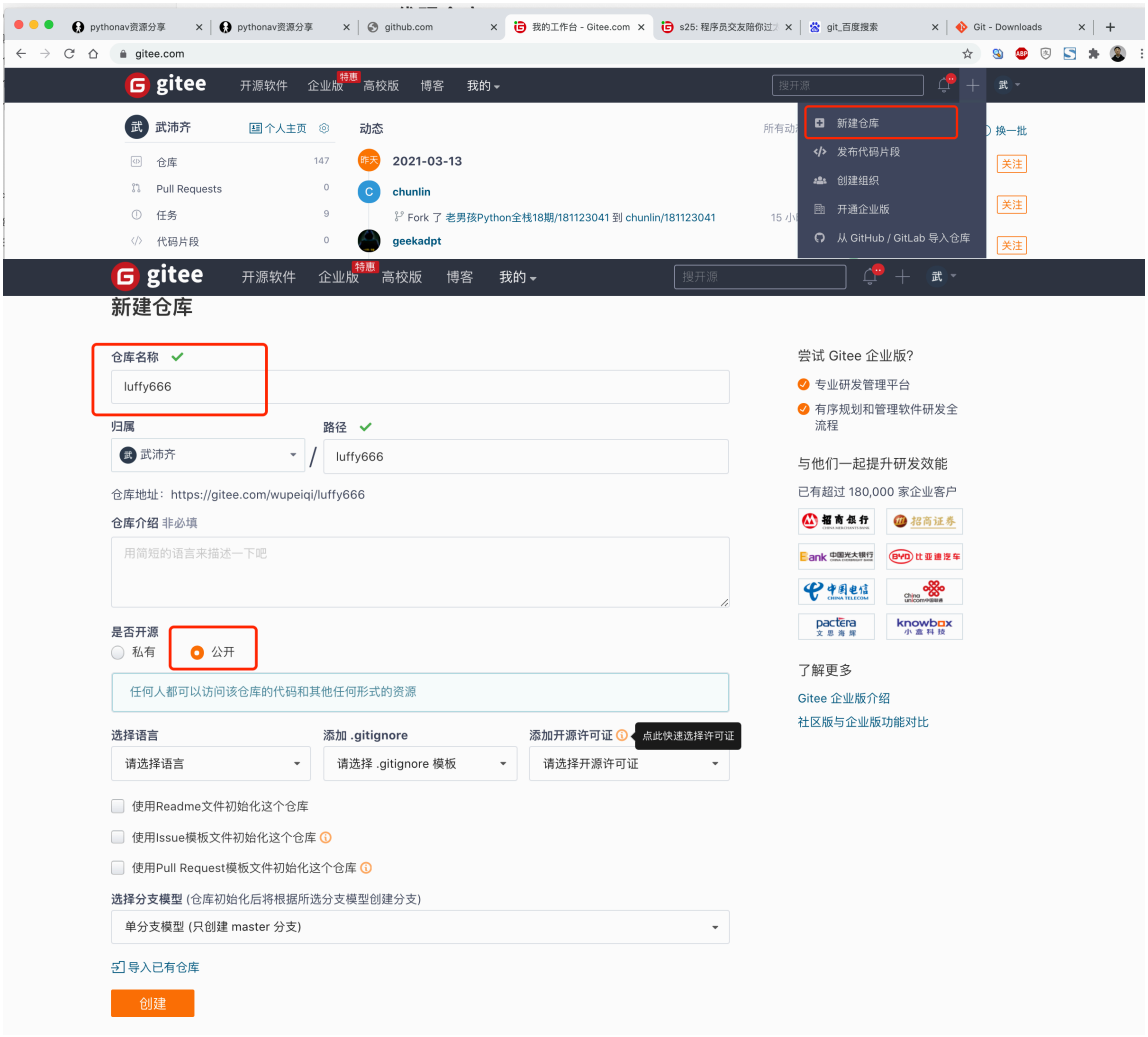
建议：github（全球程序员集散地&交友平台）【以后写功能/找项目，去github找找】

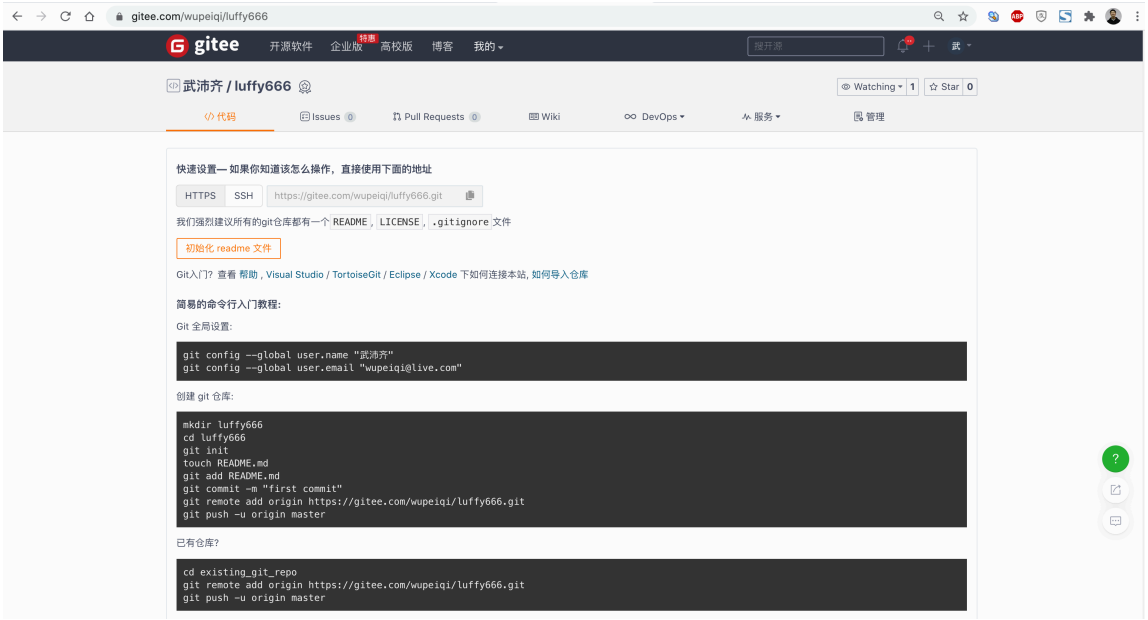
建议：码云 gitee（开源中国）。

注意：千万不要把公司代码放上去（乌云）。

以码云为例：

- 注册码云
- 在码云：
  - 创建一个仓库（文件夹），一般与项目同名。





- 在自己电脑：
  - 没创建项目

    ```
    创建项目文件夹
    进入项目的文件夹
    在终端执行 git init
    创建/删除/修改文件....
    git add .
    git commit -m 'xxxxxx'

    git remote add origin
    https://gitee.com/wupeiqi/luffy666.git（写自己项目的仓库地址）
    git push origin master
    ```

  - 已经有的项目

    ```
    git remote add origin
    https://gitee.com/wupeiqi/luffy2.git          【只需要配置一次就行了】
    git push origin master

    注意：git push origin master  是将本地代码推送到代码仓库。
    ```

如果想要继续开发代码。

```
wupeiqi@wupeiqideMBP luffy666 % git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        "oo\347\232\204\345\211\257\346\234\254.py"
        "oo\347\232\204\345\211\257\346\234\2542.py"

nothing added to commit but untracked files present (use "git add" to track)
wupeiqi@wupeiqideMBP luffy666 % git add .
wupeiqi@wupeiqideMBP luffy666 % git commit -m '开发了一些工鞯呢'
[master 30dc4c0] 开发了一些工鞯呢
 2 files changed, 4 insertions(+)
 create mode 100644 "oo\347\232\204\345\211\257\346\234\254.py"
 create mode 100644 "oo\347\232\204\345\211\257\346\234\2542.py"
wupeiqi@wupeiqideMBP luffy666 % git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 300 bytes | 300.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0)
remote: Powered by GITEE.COM [GNK-5.0]
To https://gitee.com/wupeiqi/luffy666.git
   05a9483..30dc4c0  master -> master
wupeiqi@wupeiqideMBP luffy666 %
```

现场作业：

- 码云创建一个仓库
- 本地随便创建项目
- 本地项目推送到码云的残酷