

# 模块与API

BBCLOUD 林凡

[https://github.com/lvancer/course\\_python](https://github.com/lvancer/course_python)

lin029011@163.com

# 大纲

---

环境

包管理

模块

时间

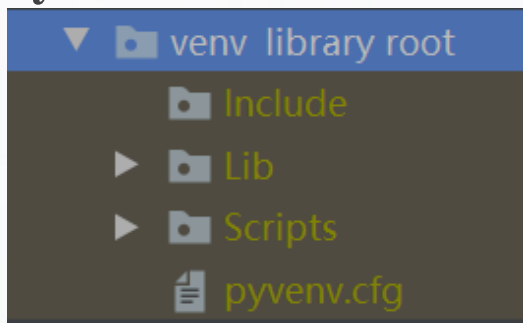
API

练习



# 环境

**Python环境**：我们回到第一次课上没有说到的**venv**这个文件夹。



项目实际执行的python在  
venv目录下

这个文件夹其实就是这个项目所使用的**Python环境目录**。

为什么不是我们安装Python时的目录？

Pycharm在默认创建项目时会根据我们安装的Python创建一个**虚拟环境**在项目下。

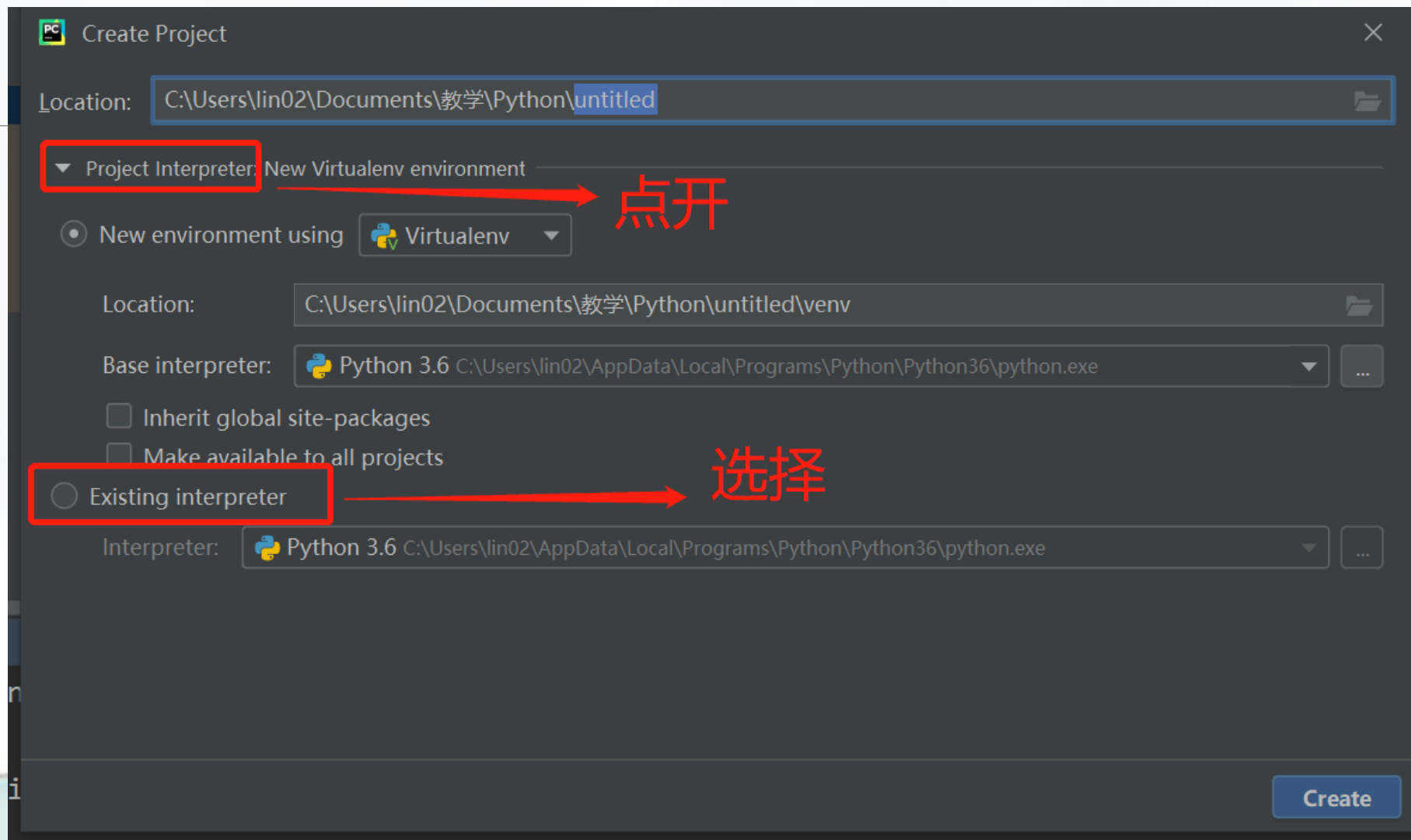
这个虚拟环境是这个**项目专用**的，是**干净**的，可以**不受**其他项目的**影响**。

**干净**要表达的是**最原始**的Python环境。

**互不影响**的作用体现在多个项目对环境的**依赖**不同，甚至可能**冲突**。

# 环境

我们也可以  
选择安装的  
原始环境  
进行开发。



# 包管理

包【Package】就是环境依赖中最主要的内容。

包其实就是一个程序目录，在这个目录下包含了许多模块，方便我们调用。

如上次课就使用到了一个 re 模块就是Python自带包里的一个模块。

Python自带包，在安装了Python后就可以直接使用。

除了自带包，我们还要使用到第三方包。

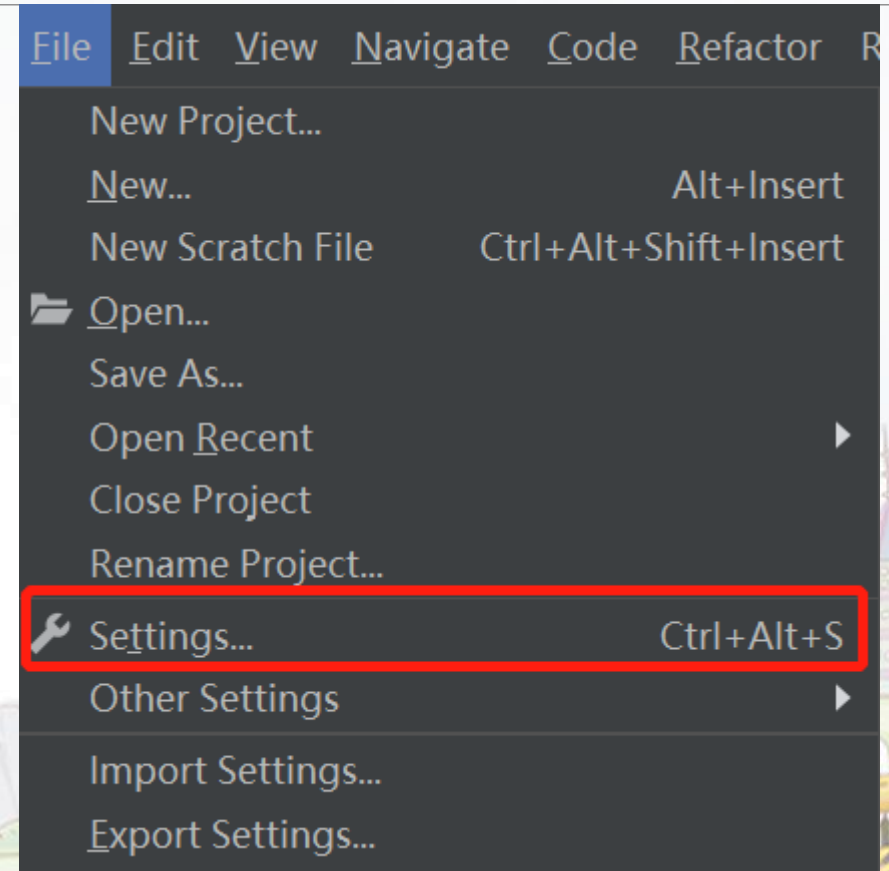
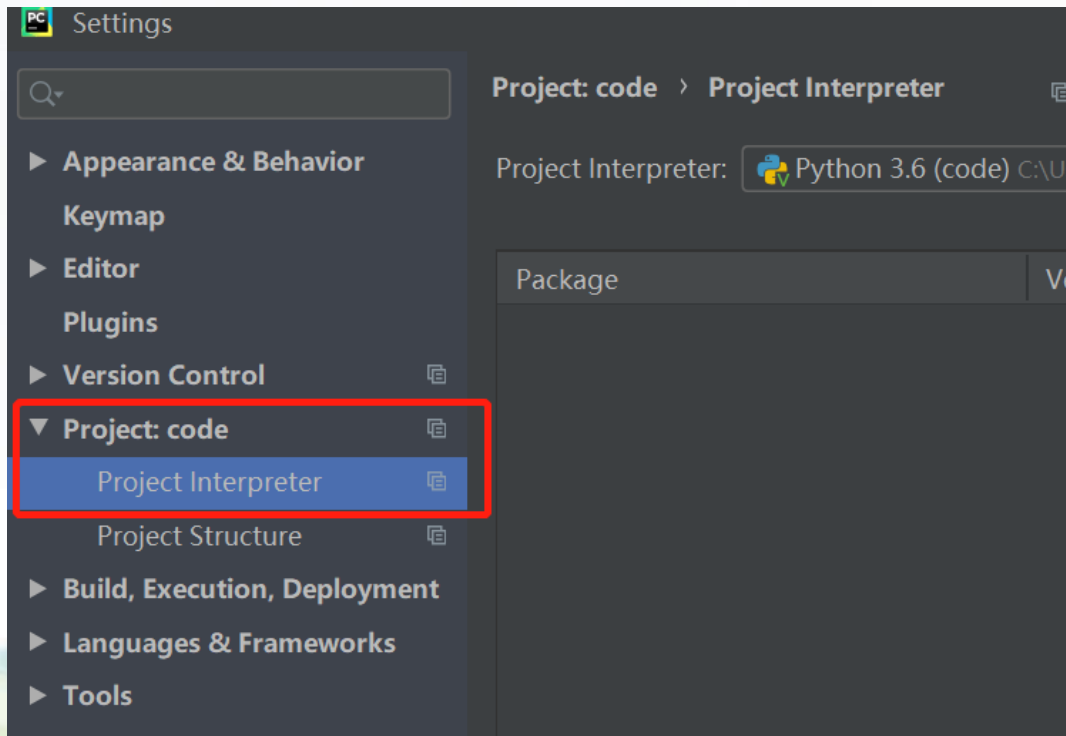
大量的第三方包是支撑Python开发的重要一环。

现在几乎所有的开发领域都可以使用Python。



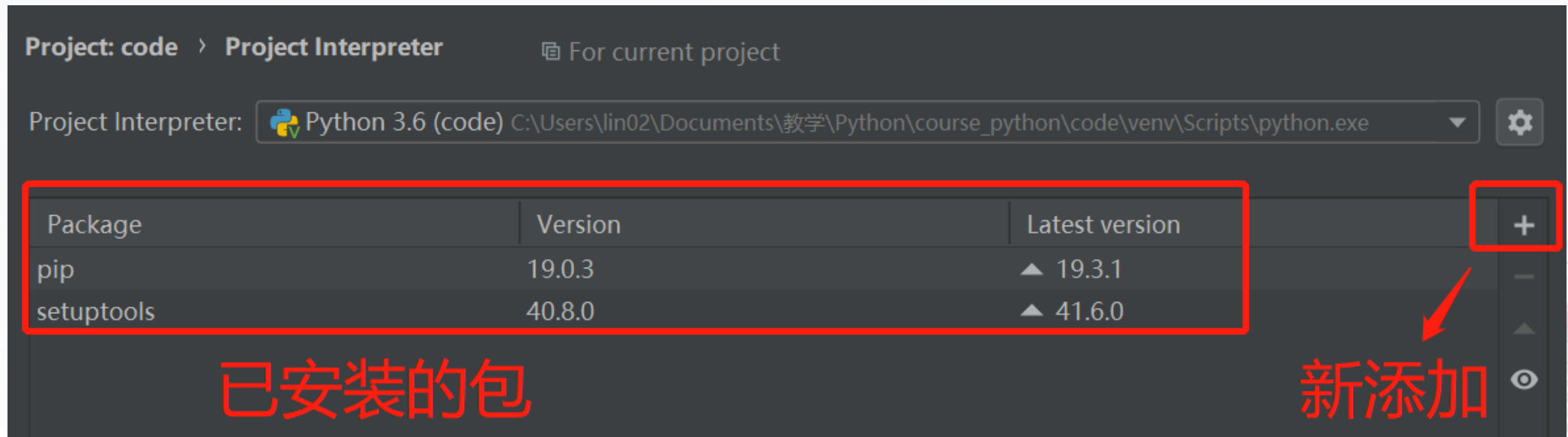
# 包管理

Pycharm中的包管理：进入配置 **File** => **Settings**  
选择**Project code** > **Project Interpreter** 进入包管理



# 包管理

## 包管理界面



The screenshot shows the 'Project Interpreter' window in PyCharm. At the top, it says 'Project: code > Project Interpreter' and 'For current project'. Below that, the 'Project Interpreter:' dropdown shows 'Python 3.6 (code)' with the path 'C:\Users\lin02\Documents\教学\Python\course\_python\code\venv\Scripts\python.exe'. A table lists installed packages:

Package	Version	Latest version
pip	19.0.3	▲ 19.3.1
setuptools	40.8.0	▲ 41.6.0

To the right of the table is a '+' button in a red box, with a red arrow pointing to it and the text '新添加' (New Add) in red. Below the table, the text '已安装的包' (Installed Packages) is written in red.

已安装的包显示了包的名称，当前版本【Version】，以及最新的版本【Lastest version】。默认情况下会有pip和setuptools两个包，这两个是Python用来管理包的包，必须存在。有最新版本时，可以进行升级。

# 包管理

添加包：

点击加号【+】

我们输入requests

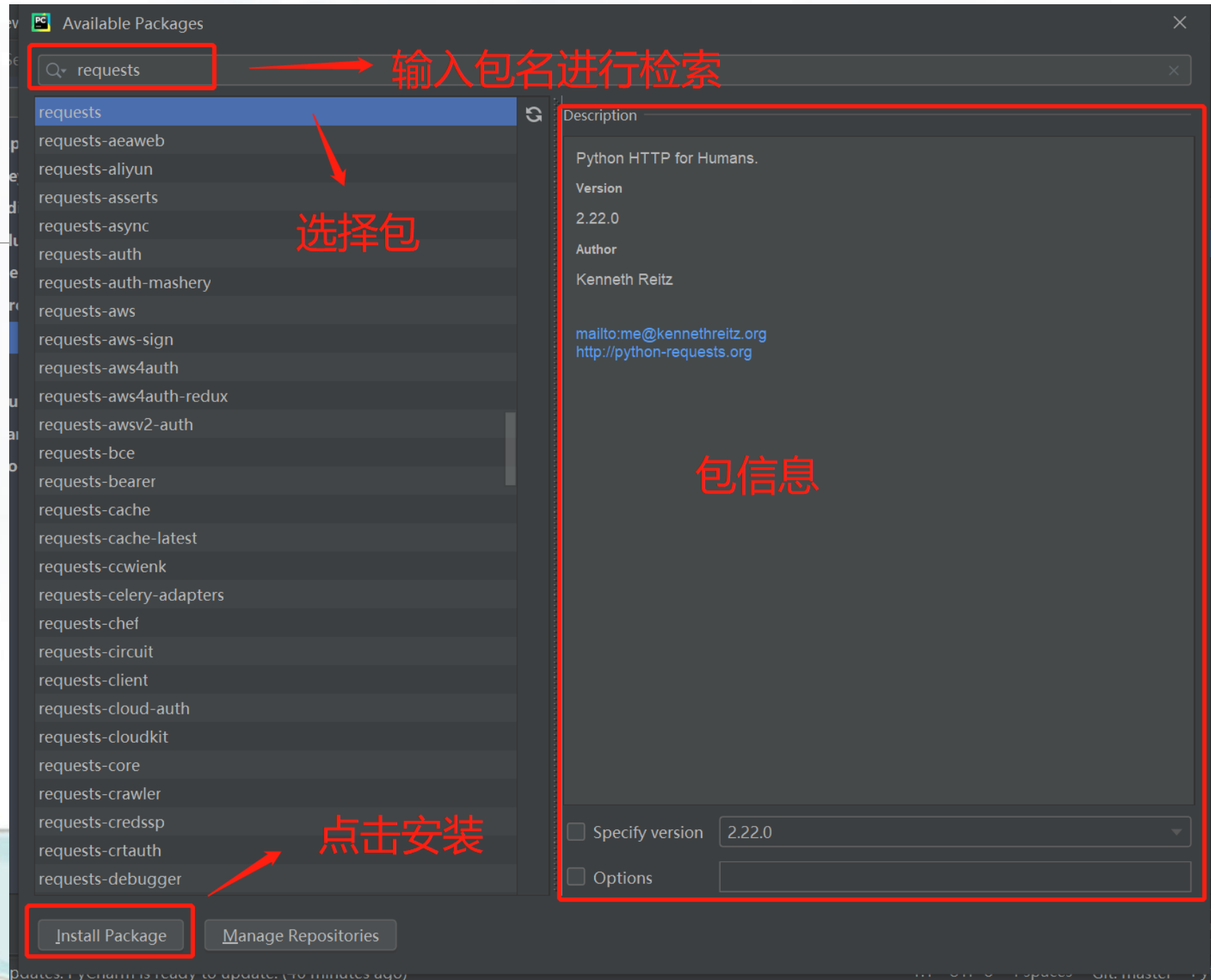
一个用于请求的包

选择正确的包后

点击Install Package

进行安装

requests (installing)





# 包管理

安装成功

Package 'requests' installed successfully

Package	Version	Latest version	
certifi	2019.9.11		+
chardet	3.0.4		-
idna	2.8		▲
pip	19.0.3		👁
requests	2.22.0		
setuptools	40.8.0		
urllib3	1.25.7		

删除  
更新

我们就看到了requests包，这时你会发现还多了很多的包。

这些包是requests包里需要调用的包【**依赖包**】，要使用requests也必须安装这些包。

lin029011@163.com

# 包管理

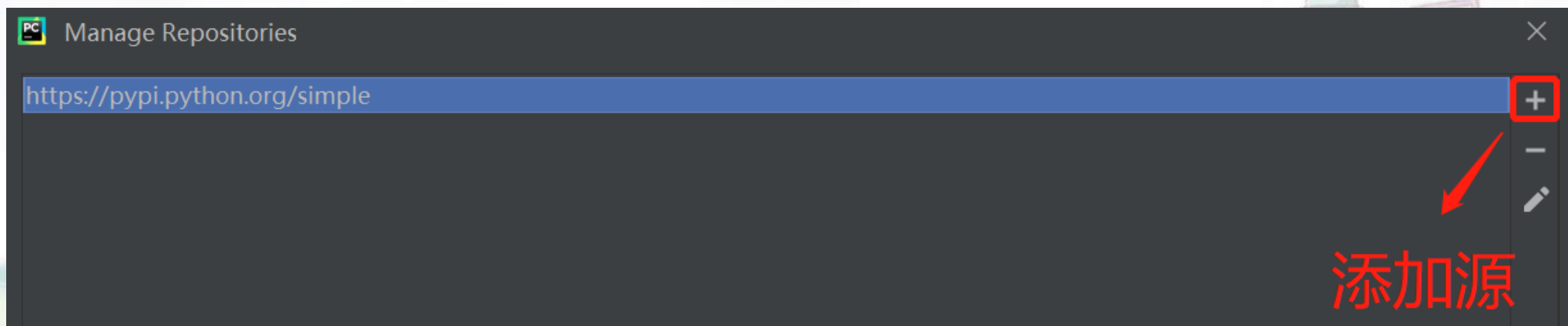
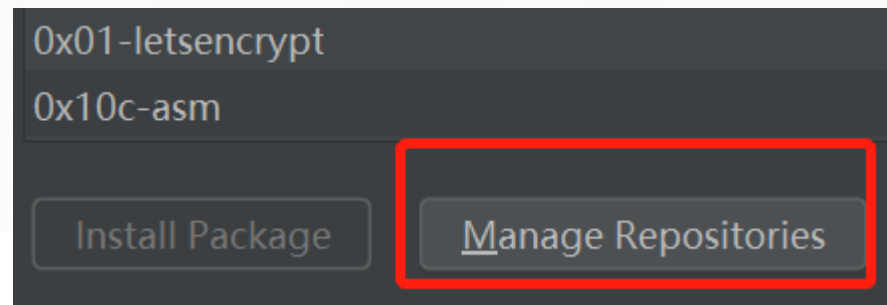
**国内源**：在安装时，你可能会觉得速度很慢，这时我们就要使用国内源。

国内源就是设在国内的服务器，访问速度比默认的要快。【默认服务器在国外】

在安装包页面，点击**Manage Repositories**

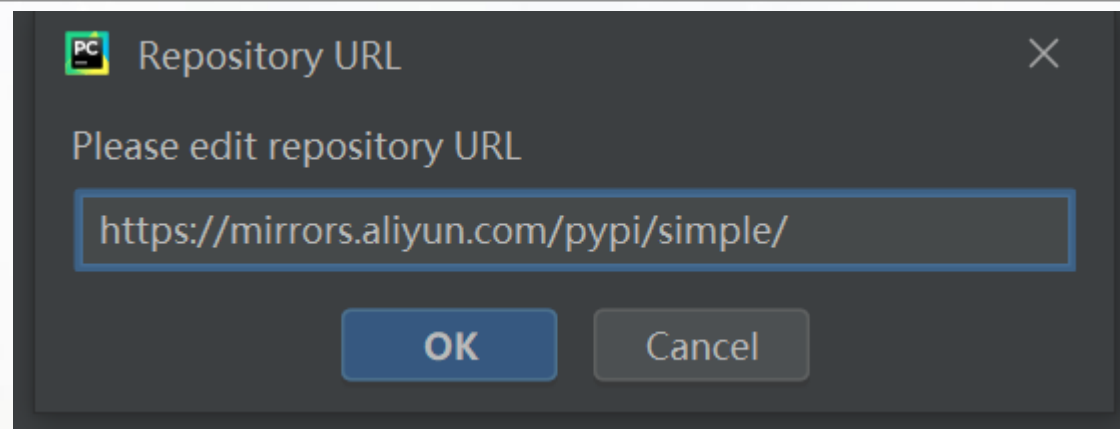
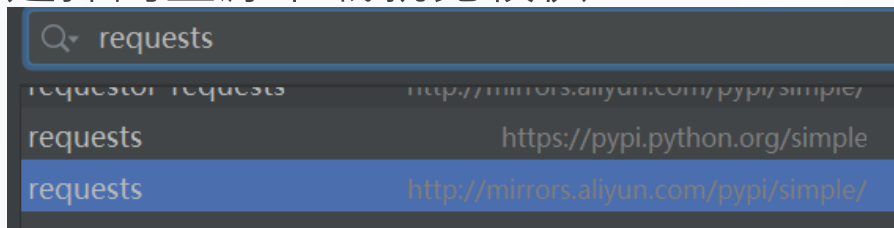
默认的地址就是<https://pypi.python.org/simple>

我们点击加号【+】，添加源。



# 包管理

添加源：输入地址，我选择了阿里的源  
再次搜索包就会出现两个  
选择阿里源下载就比较快



几个常用的国内源

清华：<https://pypi.tuna.tsinghua.edu.cn/simple>

阿里云：<https://mirrors.aliyun.com/pypi/simple>

中国科技大学 <https://pypi.mirrors.ustc.edu.cn/simple>

豆瓣：<https://pypi.douban.com/simple>

# 模块

模块就是一个Python文件。文件中包含方法和类，或者其他内容。

我们之前所写的程序都是在一个文件中实现，其实这这也是一个模块。

为了实现代码复用，我们会将方法和类放到模块中去，并由另一个执行文件来调用。

创建一个模块：我们编写一个utils工具模块。

创建utils.py文件，写入两个方法。与之前没有任何的不同。

```
def warning(content):  
    print('【warning】{}'.format(content))  
  
def error(content):  
    print('【error】{}'.format(content))
```

# 模块

模块调用：`import`、`from`。

```
import utils
utils.warning('s')
utils.error('ss')
```

直接import来的模块，其实会将该模块执行一遍  
你们可以在utils模块中打印内容，观察会不会被执行

`import`后跟上**模块名**，就完成了模块的导入。

然后我们就可以**通过模块名调用**到该方法。

另一种方式是**from导入**，特点是导入后不需要用模块名调用，直接使用方法或类。

`from ... import` 表示从某个模块中导入方法或类。

星号【\*】代表导入**所有**的方法或类。

或者**直接指定**要导入的方法或类，多个用**逗号**隔开。

```
from utils import *
warning('s')
error('ss')
```

```
from utils import warning, error
```

```
from utils import warning
warning('ss')
```

lin029011@163.com

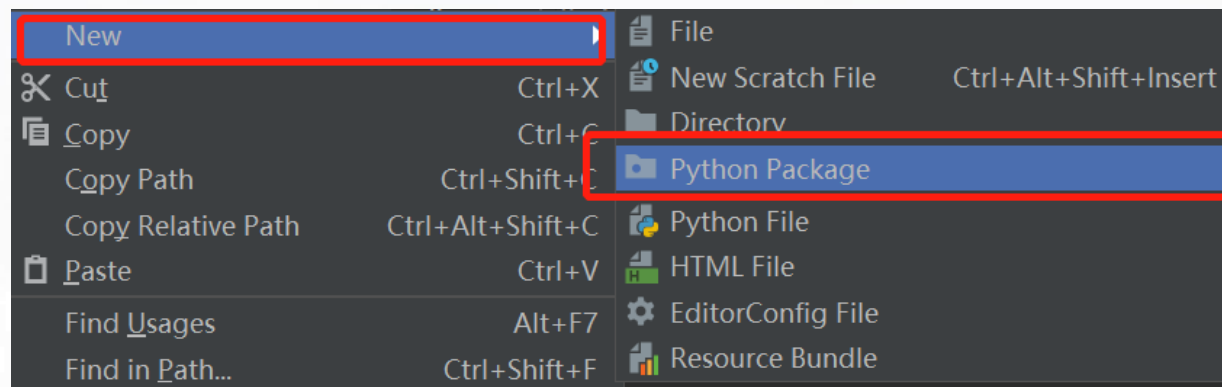
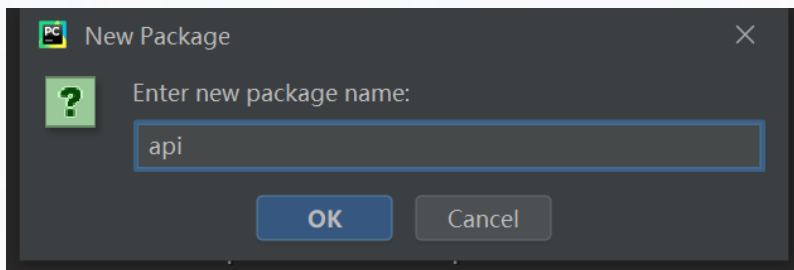


# 模块

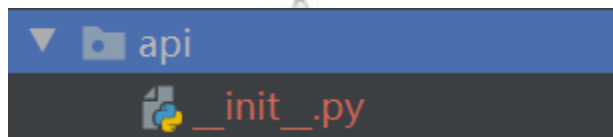
创建包：通过创建包，将模块集合起来管理。

右键 => New => Python Package

我们创建一个api模块。



创建好的包显示为



包下面默认有个 `__init__.py` 文件，是Python用来标识该目录为包的。

我们在api包下创建几个模块，来编写我们下面要学习的API请求功能。

lin029011@163.com

# 时间

在编写API之前，我们先学习使用一个常用的**内置模块**【时间】：**datetime**。

**时间模块**是用来处理时间的Python自带的包。我们可以方便的对时间进行操作。

**导入包**：从datetime包中导入datetime模块

```
from datetime import datetime
```

时间模块在导入上一般使用from的模式，这样在调用时比较简单。

由于在时间处理上一般都用该包，所以一般也不会有同名模块造成的冲突。

**获取当前时间**：使用datetime模块下的**now**方法获得当前时间。

```
now = datetime.now()  
print('现在是' + str(now))
```

现在是2019-11-18 09:18:34.551736

lin029011@163.com

# 时间

时间信息：我们获得了时间，但是打印时并不是我们想要的格式。

通过获取时间的年、月、日、时、分、秒、毫秒等信息，来组合我们要的时间。

```
year = now.year          # 年
month = now.month        # 月
day = now.day            # 日
hour = now.hour          # 时
minute = now.minute      # 分
second = now.second       # 秒
microsecond = now.microsecond # 毫秒
t = int(now.timestamp())  # 时间戳
```

时间戳是一个10位的数字，代表了从1970年1月1日0点开始所经历的秒数，是开发过程中常用的计量方式。

自定义格式

```
print(str(year) + '-' + str(month) + '-' + str(day))
print('{}-{}-{}'.format(year, month, day))
```

lin029011@163.com

# 时间

时间格式化：`strftime`。时间的格式化与字符串不同，使用百分号%起头的标准替换符。

%y	年份【两位数】
%Y	年份【四位数】
%m	月份
%d	月内第几天
%H	小时【24小时制】
%I	小时【12小时制】
%M	分钟
%S	秒

%j	年内第几天
%U	年内第几周【周日起】
%W	年内第几周【周一起】
%w	星期几【0表示周日】
%Z	时区
%p	AM PM
%x	日期表示【本地】
%X	时间表示【本地】

%a	星期名称【简】
%A	星期名称【全】
%b	月份名称【简】
%B	月份名称【全】
%c	日期时间表示【本地】

尝试所有的替换符的效果

```
import locale
locale.setlocale(locale.LC_CTYPE, 'chinese') # 解决中文问题
print(now.strftime('%Y年%m月%d日 第%W周 星期%w'))
```

1@163.com



# 时间

时间创建：一种方式我们可以通过反格式化【`strptime`】来进行。

```
date = datetime.strptime('2019年3月16日', '%Y年%m月%d日')
print(date)
```

`datetime`的`strptime`方法第一个参数是要转换的时间，第二个参数是时间格式。

```
try:
    birthday = input('输入你的生日(例: 2019-03-16)? ')
    birthday = datetime.strptime(birthday, '%Y-%m-%d')
    print(birthday)
except Exception as e:
    print('时间格式输入错误')
```



# 时间

**对象创建**：对象创建是另一种时间创建方式，是**面向对象**的编程方式。

**datetime**其实是一个**类**，可以通过直接创建来生成。

```
birthday = datetime(year=2019, month=3, day=16)
```

如上的代码，直接调用datetime类，就可以创建出**时间对象**。

通过**control**键，我们可以进入其所调用的具体方法内容。

```
def __init__(self, year: int, month: int, day: int, hour: int = ...,  
             minute: int = ..., second: int = ..., microsecond: int = ...,  
             tzinfo: Optional[_tzinfo] = ..., *, fold: int = ...)
```

**\_\_init\_\_**就是调用的类**构造方法**，**self**暂时不管，后面的参数就是该方法的参数。

我们就可以创建任意时间了。

面向对象的内容，  
下次课会详细介绍

# 时间

时间计算：`timedelta`。

除了`datetime`模块，我们还需要引入`timedelta`模块。

```
from datetime import datetime, timedelta
```

计算使用加减号进行。

```
yesterday = now - timedelta(days=1)    # 昨天，减去一天  
delta = now - yesterday                 # 时间差  
print(delta.days)                       # 相差几天
```

`timedelta`的参数还有`weeks`、`seconds`、`minutes`、`hours`等。

# API

API请求是开发过程常用的一个功能。我们使用刚刚安装的`requests`包来完成。

**实例：**时间模块中，我们只能知道某一天是周几来判断是不是放假。

实际中我们还需要知道某天是不是放假的需求。这时我们就需要API的帮助了。

我们选择了聚合数据【<https://www.juhe.cn/>】这个第三方平台的免费API进行讲解。

万年历【<https://www.juhe.cn/docs/api/id/177>】就是我们要使用的API。

注册、认证、申请该API成功后。

免费会员100次/天

数据中心

- 我的接口
- 我的数据包
- 安全中心
  - IP白名单管理
  - 登录保护
  - 密码修改
  - 预置号码管理

+ 申请新数据

已申请数据检索,如:违章

搜索

数据	申请时间	剩余次数	购买	操作
AppKey: [REDACTED]				
31 万年历	2019-11-18	—	免费	<a href="#">接口</a> <a href="#">统计</a> <a href="#">测试</a>

@163.com

# API

我们可以在【测试】中进行该接口的调试，或者使用Postman进行。

聚盒子 > 生活服务 > 万年历 > 接口测试

我的数据:

[177] 万年历

接口名称:

获取当天的详细信息

接口地址:

http://v.juhe.cn/calendar/day

请求方式:

POST

请求参数:

参数名	类型	是否必填	说明	值
date	string	是	指定日期,格式为YYYY-MM-DD,如月份和日期小于10,则取个位,如:2012-1-1	2012-1-1

发送请求

(\*为了准确性,所有请求均基于真实环境请求,请悉知)

调试完成后我们就可以开始写代码了。

lin029011@163.com

# API

模块编写：

先导入需要的包，并定义两个变量聚合平台的URL地址和申请到的appkey。

```
from datetime import datetime    # 时间包
import requests                  # 请求包

appkey = 'xxxxxxxxxxxxxxxxxxxx'  # 申请到的appkey
juhe_url = 'http://v.juhe.cn/'   # 聚合数据的网址
```

requests的基本使用：

```
response = requests.get(url=url, headers=headers, params=params) # get请求
response = requests.post(url=url, headers=headers, data=params)  # post请求
```

**params**和**data**都是字典类型，表示两种请求中的参数。**headers**也是字典，表示http请求头。  
lin029011@163.com



# API

编写方法：

在请求完成后，

`response.status_code`

是请求返回的状态码，

`200`表示请求正常。

`response.content`

是返回内容。

```
def calendar_day(date=datetime.now()):  
    url = juhe_url + 'calendar/day'  
    params = {  
        'date': date.strftime('%Y-%m-%d'),  
        'key': appkey  
    }  
    response = requests.get(url=url, params=params)  
    if response.status_code == 200:      # 请求成功  
        return response.content        # 返回结果  
    else:  
        return None                    # 请求失败
```

当状态码不是200时，返回空值`None`表示请求失败。

返回的内容是`JSON`形式，我们暂时还无法直接使用。

# API

**JSON**：该接口返回内容【如图】就是**json**。

Json是一种**数据格式**，用于**数据的传输**。

其与Python的字典列表组合的结构是一样的。

所以我们可以将**json**和**python**格式**互相转换**。

```
import json
# json格式转换为python格式
data = json.loads(content)
print(data)
# python格式转换为json格式
content2 = json.dumps(data)
print(content2)
```

```
{
  "reason": "Success",
  "result": {
    "data": {
      "holiday": "元旦",
      "avoid": "伐木.纳畜.破",
      "animalsYear": "兔",
      "desc": "2012年1月1日",
      "weekday": "星期日",
      "suit": "祭祀.开光.理发",
      "lunarYear": "辛卯年",
      "lunar": "腊月初八",
      "year-month": "2012-1",
      "date": "2012-1-1"
    }
  },
  "error_code": 0
}
```

# API

解析JSON：`return json.loads(response.content)`

或者使用封装好的方法`response.json`也很方便。`return response.json()`

调用该模块：编写了一个`is_holiday`方法来判断

```
from api import juhe          # 从api包导入juhe模块
from datetime import datetime
def is_holiday(date=datetime.now()):
    info = juhe.calendar_day(date)    # 获得数据
    if info is not None and info['reason'] == 'Success':    # 验证数据正确
        result = info['result']
        if 'holiday' in result:
            return True    # 有holiday时，表示时假日
        else:
            return False
    return None
```

# 练习

---

- 1、选择一个API进行请求模块的练习。
- 2、对于is\_holiday这个方法，我们免费的请求次数只有每天100次，假设我们的用户可能有很多，每天的请求量远不止100次。我们现在只需要今天的数据，如何可以使我们不付费？

