

Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心



Web项目开发方法

硕士研究生：杨宗源

导师：罗森林

2022年12月18日

- 背景简介
- 基础概念
- 方法介绍
- 总结
- 参考文献

- 预期收获
 - 1. 理解两种web开发模式
 - 2. 了解常见的web框架
 - 3. 理解项目开发中的知识产权保护方法

- 1991年，Web之父 蒂姆·伯纳斯·李建立了世界上第一个网站
<http://info.cern.ch/>
- 据Internet Live Stats统计，截至2022年11月，互联网上估计有近**20亿**个网站，其中真正活跃的网站也有5亿
- 网站的应用极大促进了人类社会的信息化进程

<http://info.cern.ch> - home of the first website

From here you can:

- [Browse the first website](#)
- [Browse the first website using the line-mode browser simulator](#)
- [Learn about the birth of the web](#)
- [Learn about CERN, the physics laboratory where the web was born](#)



- Web开发
 - 设计、构建和维护网站的过程
 - 简单的单一静态界面
- ↓
- 复杂的web应用程序
 - 基本作用
 - 实验室系统开发
 - 大平台、思智明理……
 - 外部合作工程项目
 - DSP、ICT……
 - 未来发展方向
 - 网站开发



- Web开发框架
 - 支持动态网站、网络应用程序及网络服务的开发框架
 - 减轻网页开发**共通性活动**的工作负荷
- 框架基本功能
 - 路由管理
 - 状态管理
 - 模板渲染
 - 静态资源
 - 日志记录
 - 数据库管理
 -

```
user = auth.authenticate(username=username, password=password) # 数据库通信

if user is not None and user.is_active:
    auth.login(request, user) # 登录
    request.session['username'] = user.username # session管理
    request.session['is_login'] = True # session管理
    add_loginlog(request) # 日志记录
    return redirect('/competition/cmpt_list/')

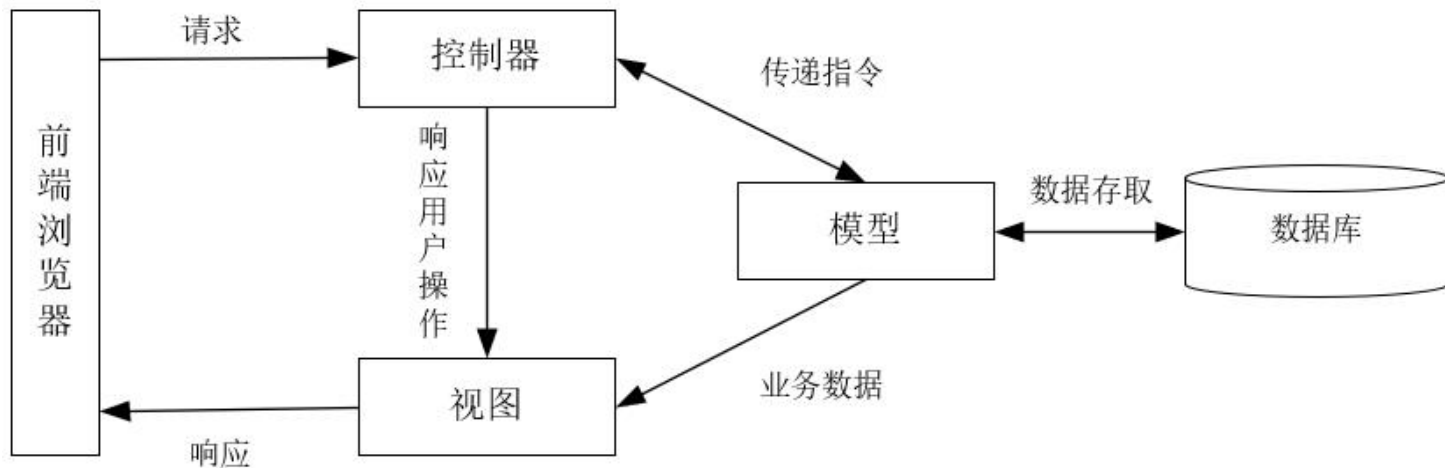
else: # 登录失败
    infor = {'message': "密码错误!"}
    return render(request, 'users/login.html', infor) # 模板渲染
```

- MVC模式

- MVC 模式是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：
模型（Model）、视图（View）和控制器（Controller）

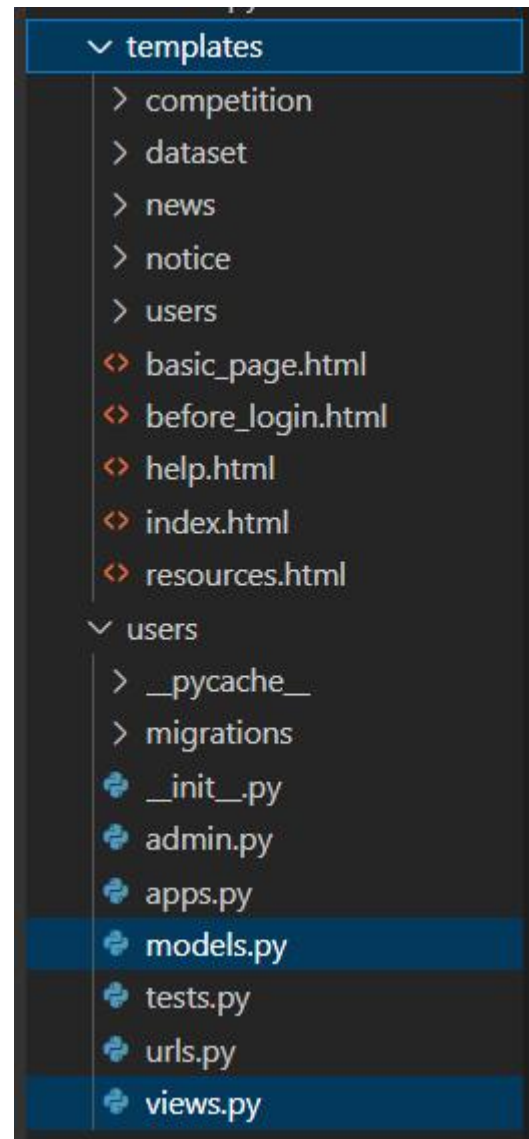
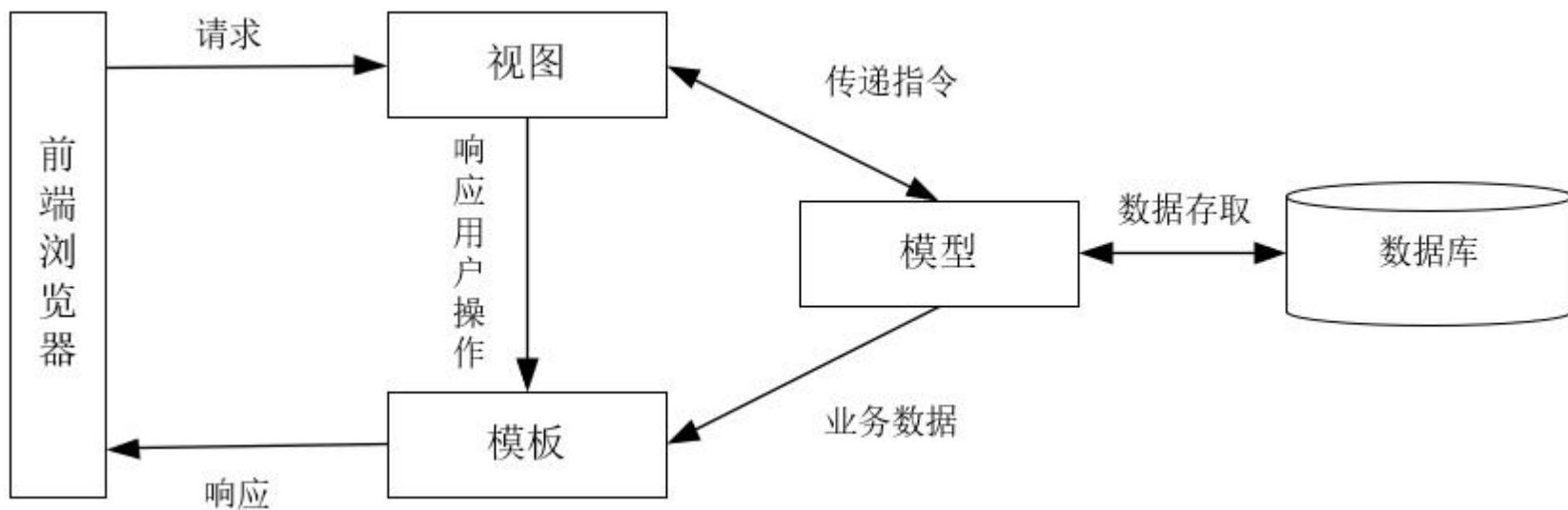
- 基本功能

- 模型（M）– 编写程序**业务功能**，负责业务对象与**数据库**的映射
- 视图（V）– 图形界面，负责与用户的**交互**
- 控制器（C）– 负责转发并**处理用户请求**



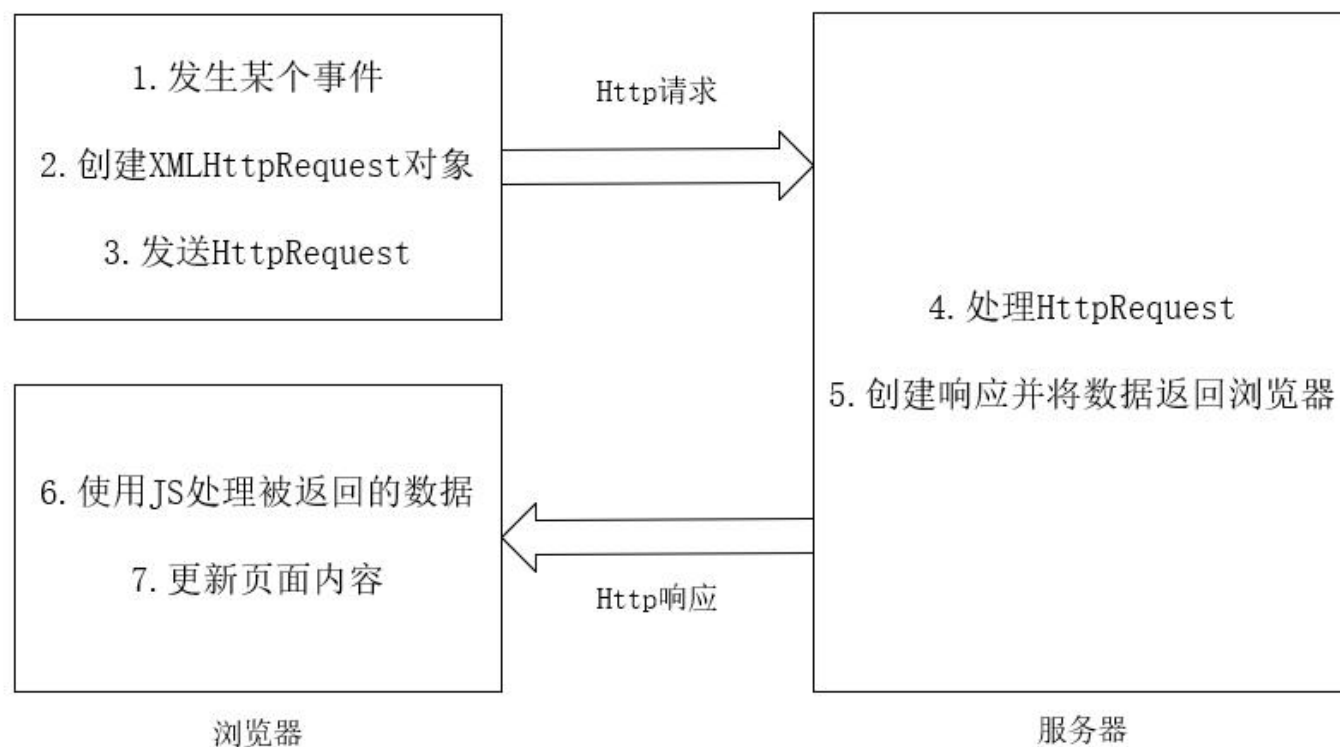
- MVT模式

- MVT 模式是Django中采用的架构模式，分别为模型（Model）、视图（View）和模板（Template）
- MVT本质上与MVC等同



- AJAX

- Asynchronous Javascript And XML (异步JavaScript和XML)
- 在**不更新**整个页面的前提下异步传输数据
- 可以传输XML、**JSON**或文件数据





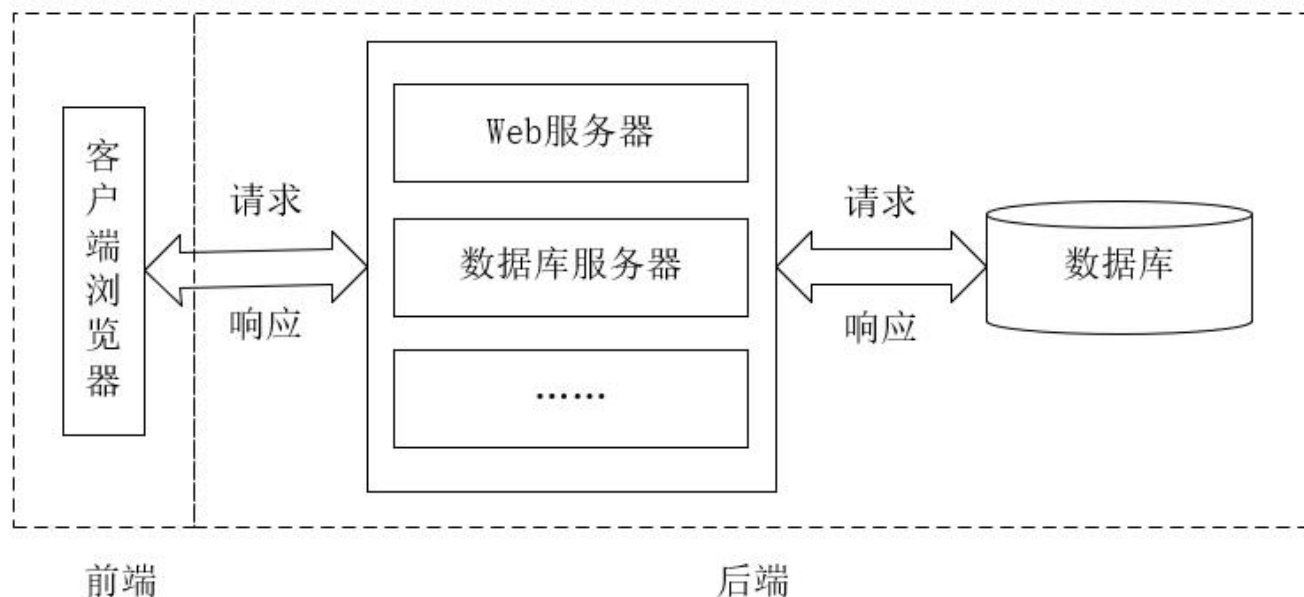
Web开发模式

- Web结构

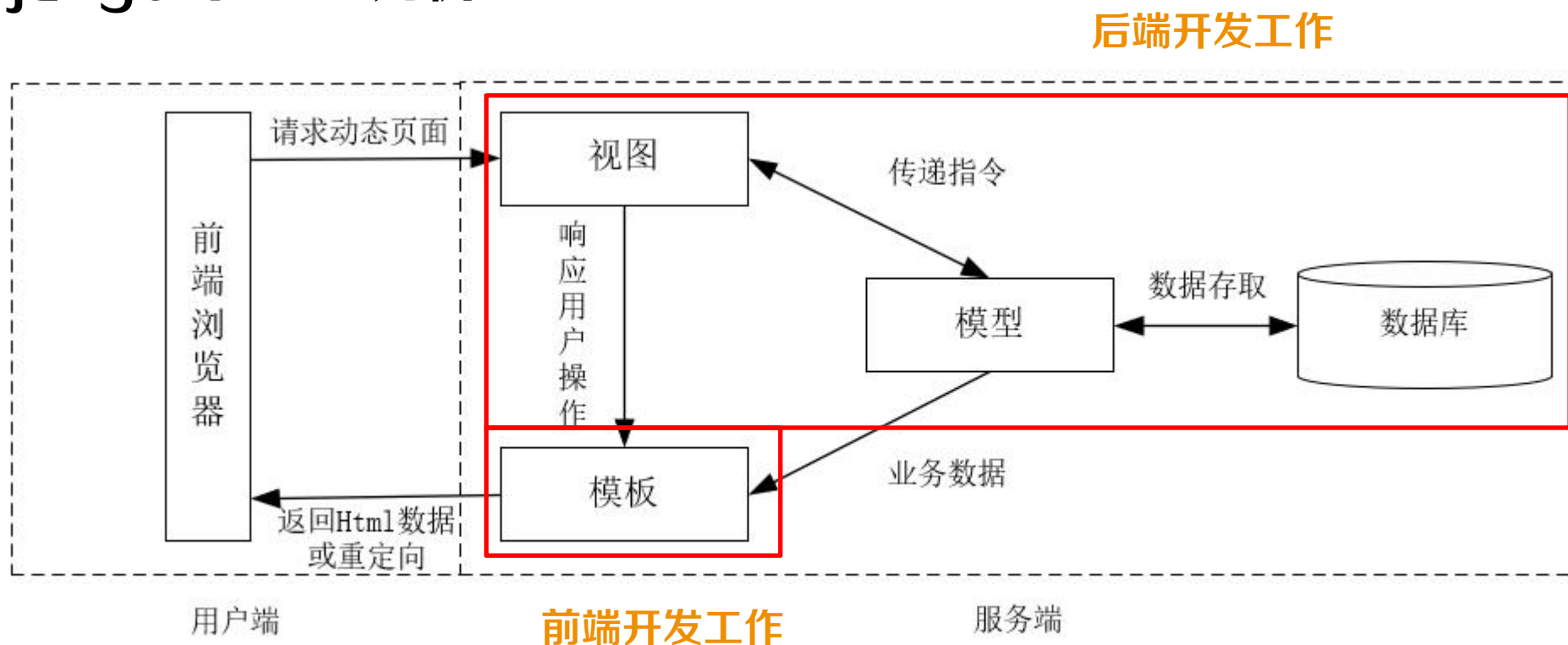
- 前端：用户肉眼看到的网站布局、内容以及一切可以**直接接触**与操作的部分
 - 静态功能：不和后端服务器进行交互，仅在前端处理并响应用户
 - 动态功能：前端发送给后端的指令，后端接到指令并处理后响应给前端
- 后端：业务逻辑，数据库IO，用户**不可直接接触**的部分

- 开发模式

- 前后端不分离
- 前后端分离



- 前后端不分离
 - 前端页面呈现效果全部由**后端控制**，后端负责渲染页面及路由跳转
- 以Django的MVT为例



- 前后端交互

- 通过render函数渲染HTML页面

- url绑定view函数

```
urlpatterns = [  
    path('datasets_list_query/', views.dataset_query),  
    path('datasets_list/', views.dataset_list),
```

- View处理请求
返回前端页面

```
def dataset_list(request):  
    fields = Field.objects.all()  
    fields_show = {f.field_name: f.dataset_set.count() for f in fields}  
  
    return render(request, 'dataset/dataset_list.html', locals())
```

- 界面编写

```
<div style="width: available; margin-left: auto; margin-right: auto">  
    <div class="classification" style="position: relative; width:100%;">  
        <span class="select-span control-label">分类: </span>  
        <ul id="field" class="nav field-packup" style="width: 680px;" role="tablist">  
            <li role="presentation" class="active select-button" name="全部"><a href="#_a">全部</a></li>  
            {% for name, count in fields_show.items %}  
                <li role="presentation" class="select-button" name="{{name}}"><a href="#">{{name}} ({{count}})</a></li>  
            {% endfor %}  
        </ul>
```

直接使用view中的数据

- 优势：
 - 小项目开发速度快
 - 开发人员把控全局，没有沟通问题
- 劣势：
 - 前后端职责不清，交流成本高
 - 路由跳转、业务逻辑
 - 代码可维护性差
 - 代码修改需要前后端同时改动调整
 - 难以满足前端需求
 - 移动端适应

```
urlpatterns = [  
    path('datasets_list_query/', views.dataset_query),  
    path('datasets_list/', views.dataset_list),
```

```
def dataset_list(request):  
    fields = Field.objects.all()  
    fields_show = {f.field_name: f.dataset_set.count() for f in fields}  
  
    return render(request, 'dataset/dataset_list.html', locals())
```

```
<div style="width: available; margin-left: auto; margin-right: auto">  
    <div class="classification" style="position: relative; width:100%;">  
        <span class="select-span control-label">分类: </span>  
        <ul id="field" class="nav field-packup" style="width: 680px;" role="tablist">  
            <li role="presentation" class="active select-button" name="全部"><a href="#_a">全  
                {% for name, count in fields_show.items %}  
                    <li role="presentation" class="select-button" name="{{name}}"><a href="#">{{nam  
                        {% endfor %}  
        </ul>
```

- 前后端分离

- 前端负责展示效果及数据处理

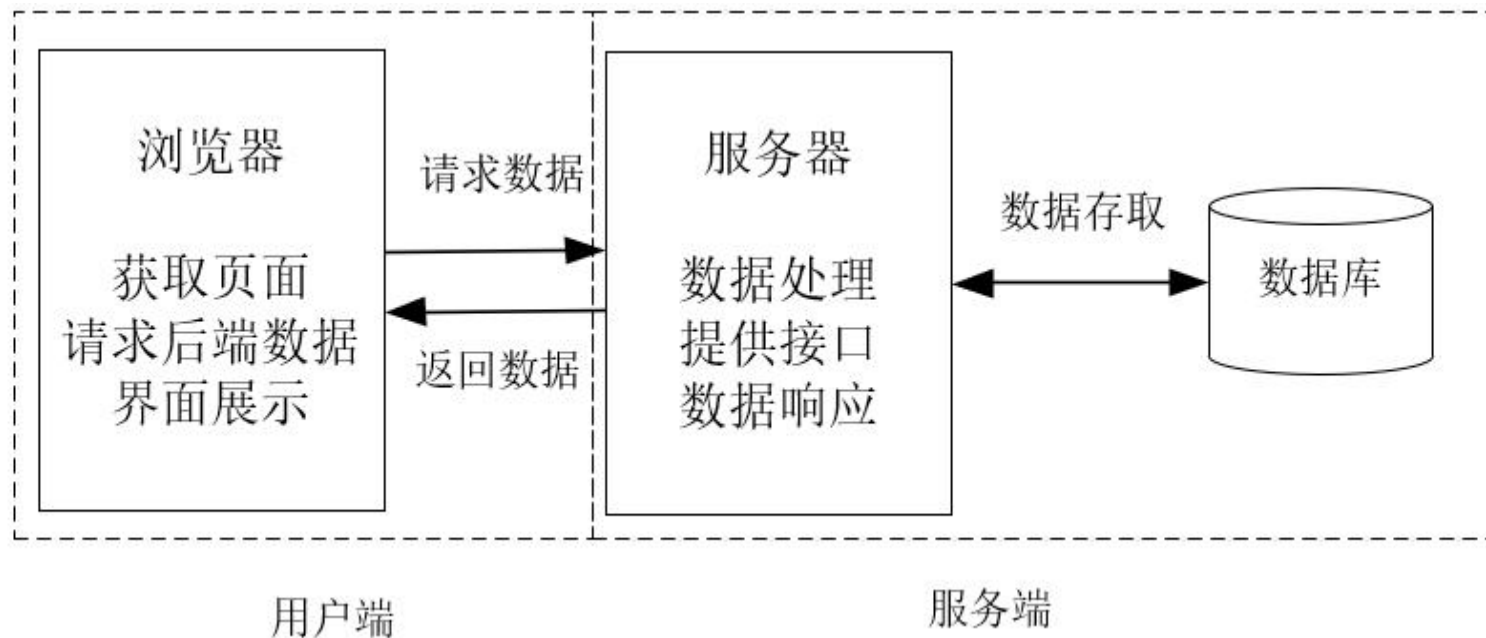
- 交互逻辑分配、界面展示

- 后端返回前端所需的数据，不再渲染前端界面

- 提供数据接口、数据库交互

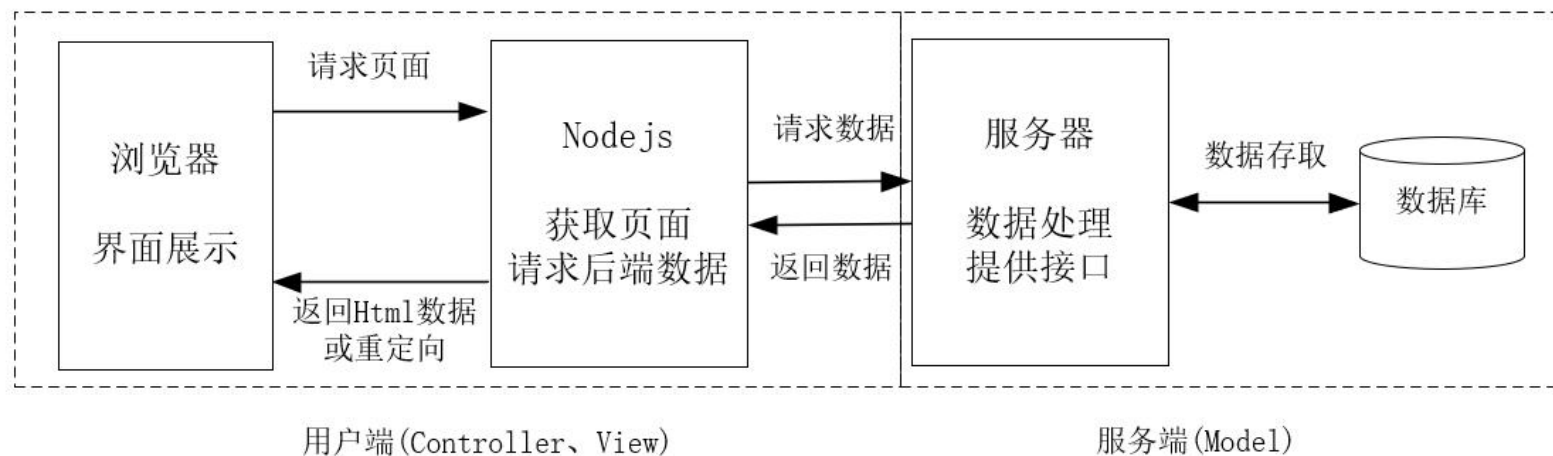
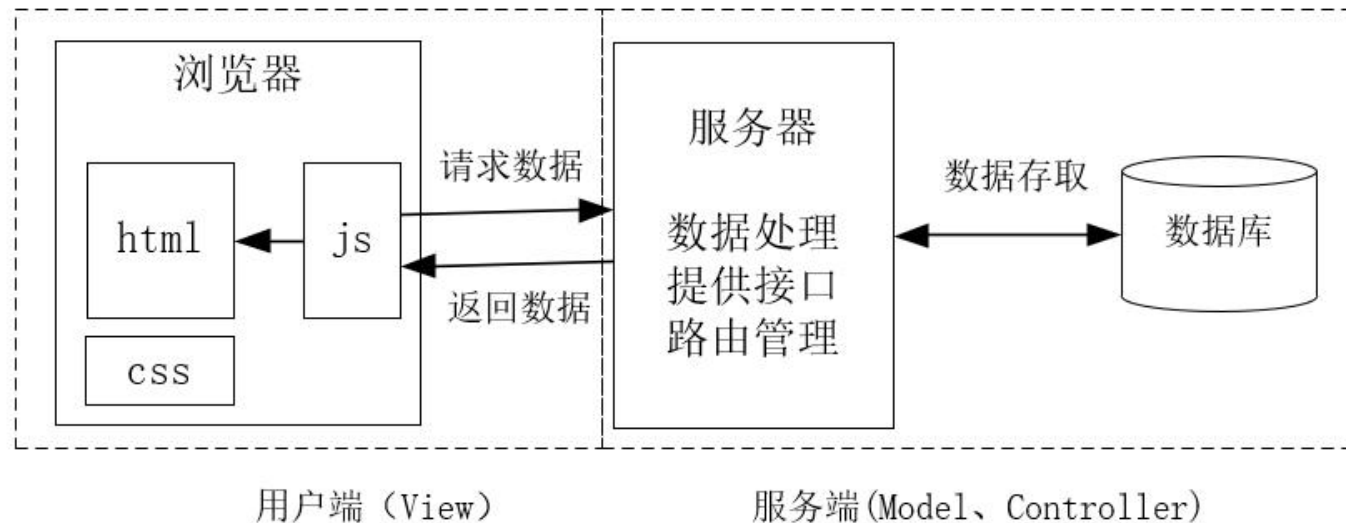
- 主要区别：

- 数据交互与路由管理的分离



- 前后端半分离
 - 基于AJAX的SPA
 - 仅分离数据交互
- 前后端完全分离
 - 基于Nodejs的前端框架
 - Vue
 - Angular
 - React

将JS从浏览器转移到服务器



- 数据交互
 - 基于异步接口Ajax / Jsonp交互数据
 - Axios是Ajax的二次封装，应用更加方便
 - 面向接口编程

```
getdata(){
  this.loading = true;
  this.$axios.post('/select_lb_data',{
    select_model: 0,
    select_data: 0,
  }).then((response) => {
    if(response.data.msg === 'success'){
      this.RawTableData = response.data.select_results;
      this.RawScatterData = response.data.scatter_data;
    }
    this.loading = false;
  }).catch((error) => {
    console.log(error);
  })
},
```

请求数据

返回数据

```
def select_lb_data(request):
    response = {}
    json_data = json.loads(request.body)
    try:
```

数据处理

```
response['msg'] = 'success'
response['error_num'] = 0
response['acc'] = response1.text
response['select_results'] = tableData
return JsonResponse(response)
```

- 路由管理
 - 路由完全由前端掌握设计
 - 功能分配更加合理
- 其他部分
 - 静态资源管理
 - 在前端服务器调用静态资源
 - 代码调试
 - 单独运行前端进行调试
 -

```
urlpatterns = [  
    path('datasets_list_query/', views.dataset_query),  
    path('datasets_list/', views.dataset_list),  
]
```

+

```
def dataset_list(request):  
    fields = Field.objects.all()  
    fields_show = {f.field_name: f.dataset_set.count() for f in fields}  
  
    return render(request, 'dataset/dataset_list.html', locals())
```



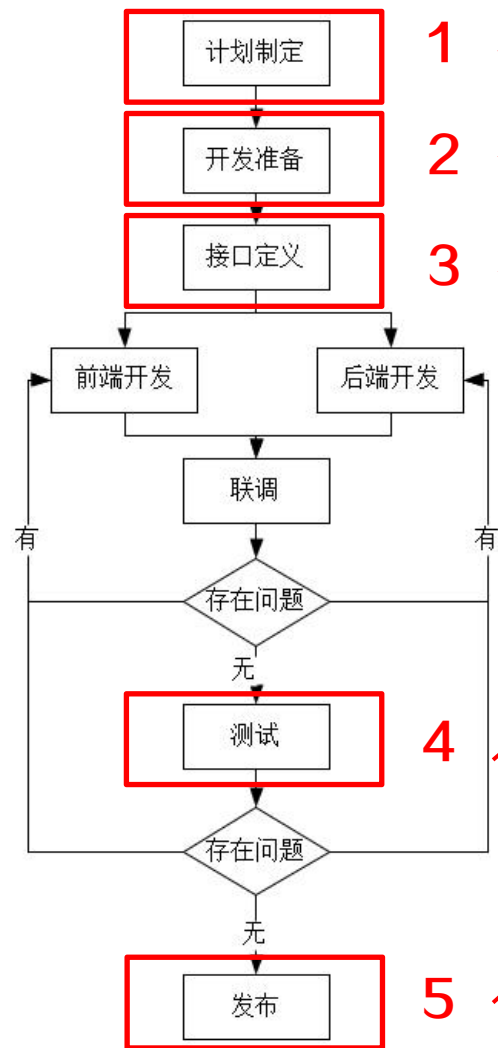
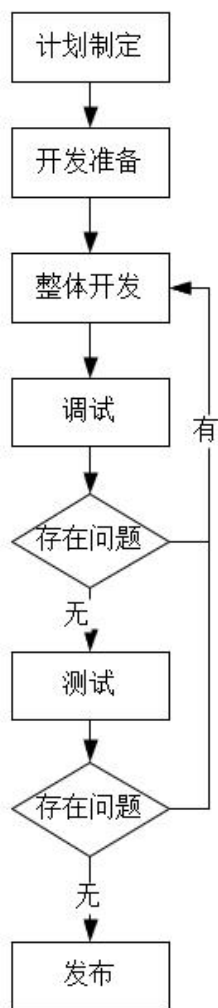
```
import Index from '@components/Index'  
export default new Router({  
  mode: 'history',  
  routes: [  
    {  
      path: '/index',  
      name: 'Index',  
      component: Index,  
      meta: {  
        requireAuth: false  
      }  
    },  
  ],  
})
```

不分离

分离

- 优势
 - 提升开发效率
 - 前端只关注前端的事，后端只关注后端的事，**并行**推进
 - 增强代码**可维护**性
 - 快速定位问题，无需前后端联调
 - 局部性能提升
 - 页面按需加载
 - 应对复杂多变的前端**需求**
 - 前端无需向后端提供模板或后端在前端html中嵌入后端代码
- 劣势
 - 前期规划时间成本较高

• 开发流程对比



- 0-BFS实验室系统开发规范-v1.1-2020.03.09-lslv1.0.docx
- 0-BFS项目交流材料-文档统一封面-v1.2-2020.07.30.docx
- 1-课题小组-项目任务分解与管理-v1.4-2020.03.01.xlsx
- 2-课题小组-项目详细设计说明书-v1.1-2017.08.17.doc
- 2-课题小组-项目需求分析说明书-v3.5-2020.04.26.doc
- 3-课题小组-项目接口规范说明书-v1.1-2017.08.17.doc
- 4-课题小组-项目测试-XXX-v1.1-2017.08.17-姓名.docx
- 4-课题小组-验收测试-XXX-姓名-v1.4-2010.04.10.xls
- 5-BFS项目结题材料交付手续单-XXX-v1.7-2017.08.18.docx
- 5-产品简介-XXX-v1.3-2020.11.06.docx
- 5-课题小组-项目研制报告-v1.1-2017.08.17.doc
- 5-课题小组-项目用户手册-v1.1-2017.08.17.doc
- 5-课题小组-项目资源整理核查表-XXX-v1.5-2020.08.14.xlsx
- 5-课题小组-项目总结报告-XXX - v1.5-2020.07.29.xlsx

	Vue	React	Angular
创始人	尤雨溪	FaceBook	Google
数据绑定	双向	单向	双向
程序体积	中	小	大
学习曲线	中	中	难
复杂度	中	中	高
模型	MVVM	不是完整的MV*框架 需要搭配其他组件	MVVM
优点	简单易用 轻量高效 组件化	兼容性好 可维护性高	模板功能丰富强大 框架非常完善 代码易于复用
缺陷	社区不够丰富	需要其他框架的配合	学习曲线陡峭
应用场景	中小型Web应用	移动端	大型Web应用

后端框架对比

方法	Django	Spring Boot	Iris
优势	1. 熟悉程度较高，开发速度快 2. 功能齐备 3. 自带后台管理	1. 核心竞争力强 2. Java生态好，易于集成 3. 稳定可靠 4. 前后端解耦	1. 语言简洁，易于上手 2. 性能好 3. 未来可期
劣势	1. 前后端耦合程度较高 2. 性能不及其他语言 3. 运维困难	1. 学习曲线陡峭 2. 庞大笨重	1. 市场占有率低 2. 需要学习新语言
适用场景	实验室内部项目	对外大型项目开发	中大型项目开发

- 前后端框架对比
 - 不分优劣，各取所需
 - 从学习曲线角度推荐**Django+Vue**



知识产权保护

11.5

T	目标	在软件正常运行的前提下，限制使用方对 源码、敏感数据、保密文件 等资源的获取
I	输入	工程源码
P	处理	代码混淆、代码加密、代码转换等
O	输出	受保护的工程源码

P	问题	使用方容易窃取未处理的核心数据资源
C	条件	程序源码已知
D	难点	攻击手段种类繁多，难以针对性防护
L	水平	

- 通用方法

- 硬件方法（USBKey）：

- 基于USB接口的身份认证智能存储设备，可以存储数字证书、密钥和其他机密信息，进行安全的用户鉴定

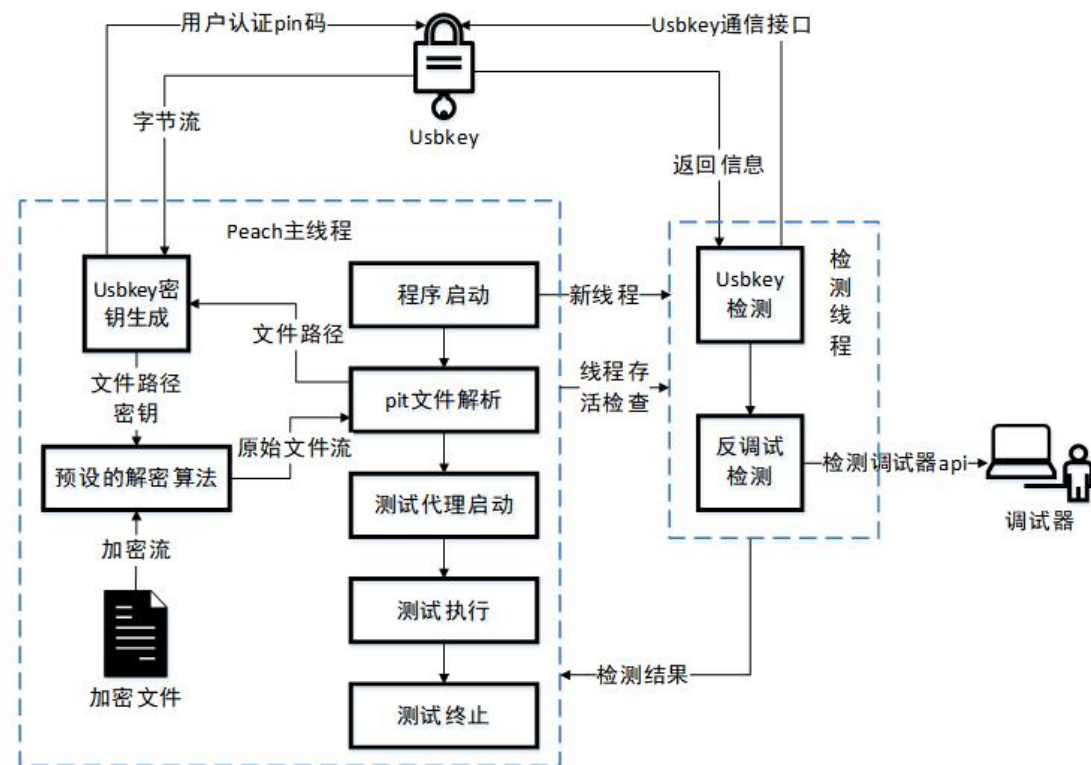
- 加解密

- 关键节点检测

- 程序加壳

- 软件方法

- 代码审计：ReviewBoard
 - 反调试：AntiNet代码
 - 授权码



- Python
 - 代码混淆

<https://pyob.oxyry.com/>

```
1 """The n queens puzzle.
2
3 https://github.com/sol-prog/N-Queens-Puzzle/blob/master/nqueens.py
4 """
5
6 __all__ = []
7
8 class NQueens:
9     """Generate all valid solutions for the n queens puzzle"""
10
11     def __init__(self, size):
12         # Store the puzzle (problem) size and the number of valid solutions
13         self.__size = size
14         self.__solutions = 0
15         self.__solve()
16
17     def __solve(self):
18         """Solve the n queens puzzle and print the number of solutions"""
19         positions = [-1] * self.__size
20         self.__put_queen(positions, 0)
21         print("Found", self.__solutions, "solutions.")
22
23     def __put_queen(self, positions, target_row):
24         """
25         Try to place a queen on target_row by checking all N possible cases.
26         If a valid place is found the function calls itself trying to place a queen
27         on the next row until all N queens are placed on the NxN board.
28         """
29         # Base (stop) case - all N rows are occupied
30         if target_row == self.__size:
31             self.__show_full_board(positions)
32             self.__solutions += 1
33         else:
```

```
1 """#line: 4
2 __all__ = []#line: 6
3 class 0000000000000000 :#line: 8
4     """#line: 9
5     def __init__ (0000000000000000 ,0000000000000000 ):#line: 11
6         0000000000000000 .__0000000000000000 =0000000000000000 #line: 13
7         0000000000000000 .__0000000000000000 =0 #line: 14
8         0000000000000000 .__0000000000000000 ()#line: 15
9     def __0000000000000000 (0000000000000000 ):#line: 17
10         """#line: 18
11         0000000000000000 =[-1 ]*0000000000000000 .__0000000000000000 #line: 19
12         0000000000000000 .__0000000000000000 (0000000000000000 ,0 )#line: 20
13         print ("Found",0000000000000000 .__0000000000000000 ,"solutions.")#line: 21
14     def __0000000000000000 (0000000000000000 ,0000000000000000 ,0000000000000000 ):#line: 23
15         """#line: 28
16         if 0000000000000000 ==0000000000000000 .__0000000000000000 :#line: 30
17             0000000000000000 .__0000000000000000 (0000000000000000 )#line: 31
18             0000000000000000 .__0000000000000000 +=1 #line: 32
19         else :#line: 33
20             for 0000000000000000 in range (0000000000000000 .__0000000000000000 ):#line: 35
21                 if 0000000000000000 .__0000000000000000 (0000000000000000 ,0000000000000000 ,0000000000000000 ):#line:
22                     0000000000000000 [0000000000000000 ]=0000000000000000 #line: 38
23                     0000000000000000 .__0000000000000000 (0000000000000000 ,0000000000000000 +1 )#line: 39
24     def __0000000000000000 (0000000000000000 ,0000000000000000 ,0000000000000000 ,0000000000000000 ):#line: 42
25         """#line: 46
26         for 0000000000000000 in range (0000000000000000 ):#line: 47
27             if 0000000000000000 [0000000000000000 ]==0000000000000000 or 0000000000000000 [0000000000000000 ]-00000000
28                 return False #line: 52
29             return True #line: 53
30     def __0000000000000000 (0000000000000000 ,0000000000000000 ):#line: 55
31         """#line: 56
32         for 0000000000000000 in range (0000000000000000 .__0000000000000000 ):#line: 57
33             0000000000000000 ="#line: 58
```

- Python
 - 代码加密
 - 对python代码直接进行加密，保护代码不被泄露
 - 常用工具
 - PyArmor、Py2exe
 - 优势
 - 操作简单方便、安全性较高
 - 劣势
 - 可能被爆破或反编译

```
if __name__ == "__main__":  
    print('hello world')
```

源码

pyarmor obfuscate test.py

```
from pytransform import pyarmor_runtime  
pyarmor_runtime()  
__pyarmor__ (__name__, __file__,  
b'\x50\x59\x41\x52\x4d\x4f\x52\x00\x00\x03\x07\x00\x18\xb8\xae\x10\xa1\xe4\x47\x6f\xa0\xa6\x4d\x4a\x11\xc2\xb5\x62\x0a\x56\x23\xcd\x59\x1d\x5b\xc\x89\xeb\x89\x0f\xdb\xa9\x56\xc5\x9a\xe8\x68\x68\xa5\x03\xc1\x61\x98\xd5\xa8\xb7\x89\x11\xe7\xe2\x2\x3e\xec\x44\xdf\x10\x72\xbb\x48\xad\xb3\xd8\xf7\xf2\x0f\x67\x55\x8b\x8f\xf6\xae\xf4\xd2\x6a\x23\xc\x85\xc1\x16\x6c\xfe\x45\x8a\x08\x0b\x5c\x80\x1c\x
```

加密后

python test.py

hello world

输出

- Python

- 代码转换

- Python为解释性语言，无需编译即可运行
 - 破解者可以**直接获得源码**，安全性不佳
 - 将python转换为编译性语言后再进行**编译**，大幅提升代码安全性

- 常用工具

- Cython、TFLite

- 优点

- 兼具两种语言特性
 - 安全性较高

- 缺点

- 部分代码可能不支持

```
if __name__ == "__main__":  
    print('hello world')
```

源码



```
from distutils.core import setup  
from Cython.Build import cythonize  
setup(name='Hello World app',  
      ext_modules=cythonize('test.py'))
```

加密脚本



```
#ifndef PY_SSIZE_T_CLEAN  
#define PY_SSIZE_T_CLEAN  
#endif /* PY_SSIZE_T_CLEAN */  
#include "Python.h"  
#ifndef Py_PYTHON_H  
    #error Python headers needed to compile C extensions, please install  
#elif PY_VERSION_HEX < 0x02060000 || (0x03000000 <= PY_VERSION_HEX &&  
    #error Cython requires Python 2.6+ or Python 3.3+.  
#else  
#define CYTHON_ABI "0_29_32"  
#define CYTHON_HEX_VERSION 0x001D20F0
```

C语言代码



```
C test.c  
test.cp37-win_amd64.pyd  
test.py
```

.pyd动态链接库文件

- Python

- 定制解释器

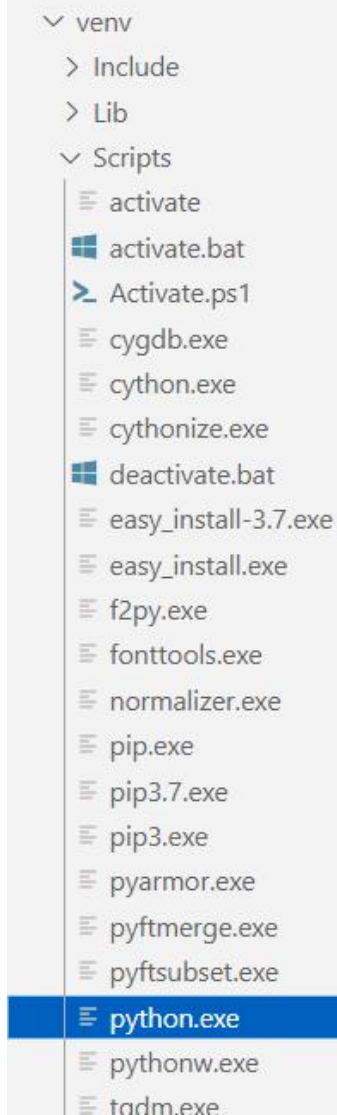
- 使用算法加密原始python代码
 - 根据自定义的python解释器解密并运行python文件

- 优点

- 安全性**最高**

- 缺点

- 需要了解各种加解密算法、解释器执行代码方式、接口修改位置等
 - 实验室**基本不需要考虑**



```
venv
├── Include
├── Lib
└── Scripts
    ├── activate
    ├── activate.bat
    ├── Activate.ps1
    ├── cygdb.exe
    ├── cython.exe
    ├── cythonize.exe
    ├── deactivate.bat
    ├── easy_install-3.7.exe
    ├── easy_install.exe
    ├── f2py.exe
    ├── fonttools.exe
    ├── normalizer.exe
    ├── pip.exe
    ├── pip3.7.exe
    ├── pip3.exe
    ├── pyarmor.exe
    ├── pyftmerge.exe
    ├── pyftsubset.exe
    └── python.exe
        ├── pythonw.exe
        └── tadm.exe
```

- 其他语言源码保护方法

保护对象	保护方法	保护强度	实现难度	实现方式
Java源码	代码混淆	一般	易	Zelix、JODE等开源工具直接混淆
	字节码转换	一般	易	对编译后class文件进行手动字节码转换（加密）
	加密Class文件	良好	难	自定义ClassLoader加密相关class文件
	本地代码转换	良好	难	将整体程序转换为难以被反编译的本地代码
Go源码	代码混淆	一般	易	Garble、gobfuscate等开源工具直接混淆
	代码删除	一般	易	删除部分代码块，例如删除调试符号，删除trace文件
C++源码	代码混淆	一般	易	Ildasm、Dotfuscator等开源工具进行加密
	Exe/Dll加密	一般	易	Codeorder、Stunrix等开源工具进行混淆

- 关键对象保护

- 数据库保护

- 利用内置加密函数对数据库内容进行加密

- 文档保护

- 压缩包加密

- 音视频保护

- 开源工具加密
 - 定制播放器 / 定制算法加密

- 虚拟机镜像保护

- 创建虚拟机镜像时自带AES加密

- 硬盘保护

- 自带工具加密

```
/* 加密后的数据如下 */
mysql> select * from f_user_p;
+----+-----+-----+-----+
| id | name | tel   | pwd   |
+----+-----+-----+-----+
| 1  | 曹操 | 10000000000 | Q2NAMTIz |
| 2  | 关羽 | 21000000000 | R3Vhbnl1QDIx |
| 3  | 刘备 | 20000000000 | TEIjMjAwMDAw |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select name, tel, AES_DECRYPT(pwd, 'MySQL')pwd f;
+-----+-----+-----+
| name | tel   | pwd   |
+-----+-----+-----+
| 曹操 | 10000000000 | Cc@123 |
| 关羽 | 21000000000 | Guanyu@21 |
| 刘备 | 20000000000 | LB#200000 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```



总结

- Web开发流程
 - 小型项目开发推荐使用前后端不分离模式
 - 有一定运维需求，需多人合作的中大型项目使用前后端分离模式
 - 前后端分离思想拓展应用
 - 明确职责，专职专用
 - 合作开发交流
- 知识产权保护
 - 明确项目地位，确定交付组件，实施多层保护
 - 时刻留存安全意识

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！

