

后台构建

Python 3.8, Django 3.2, 测试工具Postman和浏览器。

本项目要求Python 3.7+、Django 3.0+。

安装

在Pycharm中, 创建Django项目Mammothbe, 命令行中执行以下操作

```
1 $ pip install django
2 $ pip install mysqlclient
3 $ pip install djangorestframework
```

```
1 $ django-admin startproject mammoth .
2 $ python manage.py startapp user
```

在user包目录下, 新建urls.py、serializers.py

创建utils包目录

全局配置

settings.py

本次项目是前后端分离, 不用模板, 所以不配置模板。

```
1 from pathlib import Path
2
3 # Build paths inside the project like this: BASE_DIR / 'subdir'.
4 BASE_DIR = Path(__file__).resolve().parent.parent
5
6 DEBUG = True # 生产环境一定设置为False
7
8 INSTALLED_APPS = [
9     'django.contrib.admin',
10    'django.contrib.auth',
11    'django.contrib.contenttypes',
12    'django.contrib.sessions',
13    'django.contrib.messages',
14    'django.contrib.staticfiles',
15    'user', # 注册
16 ]
17
18 # 数据库配置
19 DATABASES = {
20     'default': {
21         'ENGINE': 'django.db.backends.mysql',
22         'NAME': 't3',
23         'USER': 'wayne',
24         'PASSWORD': 'wayne',
25         'HOST': '127.0.0.1',
```

```

26         'PORT': '3306',
27     }
28 }
29
30 LANGUAGE_CODE = 'zh-Hans' #'en-us'
31 USE_TZ = True
32 TIME_ZONE = 'Asia/Shanghai' #'UTC'
33
34 USE_I18N = True
35
36 LOGGING = {
37     'version': 1,
38     'disable_existing_loggers': False,
39     'handlers': {
40         'console': {
41             'class': 'logging.StreamHandler',
42         },
43     },
44     'loggers': {
45         'django.db.backends': {
46             'handlers': ['console'],
47             'level': 'DEBUG', # 显示SQL语句
48             'propagate': False,
49         },
50     },
51 }

```

迁移Django表

```
1 $ python manage.py migrate
```

创建超级用户

```
1 $ python manage.py createsuperuser
```

admin/adminadmin, 也看到了用到系统表 `INSERT INTO auth_user`, 加入了超级用户admin

初始化Git

```

1 $ git init
2
3 创建.gitignore文件, 忽略掉一些非必要文件
4
5 $ git commit -m "Project Init"

```

Django认证

认证: 标识请求具有的合法的身份。

认证和权限是2个**不同**概念, 认证标识身份, 权限和身份相关, 表示该身份能干什么。做个比喻, 认证判断你是不是我们单位人, 权限表示你在单位能接触到什么等级的信息。

HTTP协议的无状态, 所以, 必须使用一些机制来解决无状态的问题, 例如Cookie-Session机制。

参考 <https://docs.djangoproject.com/en/3.2/topics/auth/default/>

django.contrib.auth中提供了许多方法：

1、认证

```
authenticate(**credentials)
```

提供了用户认证，即验证用户名以及密码是否正确，检查is_active字段是否为1即激活的用户。

```
1 user = authenticate(username='someone', password='somepassword')
```

本质上就是用用户名查了下数据库，如果用户存在，还需密码比对一致，且用户激活状态，才算成功，返回当前这个用户对象。

比对失败，返回None。

2、登录

```
login(request, user, backend=None)
```

- 该函数接受一个HttpRequest对象，以及一个认证了的User对象。不认证密码，只把user注入request
- 注入request.session[SESSION_KEY]
- 响应报文返回set-cookie带着SessionID，避免下一次请求后重新认证

```
1 user = authenticate(username='someone', password='somepassword')
2 if user:
3     login(request, user)
```

3、登出

```
logout(request)
```

该函数接受一个HttpRequest对象，无返回值。

当调用该函数时，当前请求的session信息会全部清除，包括清除数据库django_session记录。

该用户即使没有登录，使用该函数也不会报错。

4、身份确认

如何确认请求的身份？每一回都需要登录吗？这是就要看请求中Cookie的SessionID了。

- 有SessionID就在服务器端比对
 - 比对成功则request.user就是用户对象
 - 比对失败，去登录页
- 无SessionID，去登录页

这个身份确认的过程是在**中间件**中完成的。

中间件

参看 <https://docs.djangoproject.com/en/1.11/topics/http/middleware/#writing-your-own-middlewares>

先来看看settings.py中定义的中间件

```

1  INSTALLED_APPS = [
2      'django.contrib.admin',
3      'django.contrib.auth',
4      'django.contrib.contenttypes',
5      'django.contrib.sessions', # 基于数据库的session
6      'django.contrib.messages',
7      'django.contrib.staticfiles',
8      'user',
9  ]
10
11  MIDDLEWARE = [
12      'django.middleware.security.SecurityMiddleware',
13      'django.contrib.sessions.middleware.SessionMiddleware',
14      'django.middleware.common.CommonMiddleware',
15      'django.middleware.csrf.CsrfViewMiddleware',
16      'django.contrib.auth.middleware.AuthenticationMiddleware',
17      'django.contrib.messages.middleware.MessageMiddleware',
18      'django.middleware.clickjacking.XFrameOptionsMiddleware',
19  ]
20
21  #SESSION_COOKIE_NAME = 'sessionid' # 从cookie中提取session使用的名字

```

是一个列表，说明处理有顺序。

基于Session的认证

SessionMiddleware

- 使用Model类django.contrib.sessions.models.Session操作表django_session
- 从请求报文中提取sessionid，构建request.session属性

AuthenticationMiddleware

- 依赖于request.session，也就是说SessionMiddleware必须在前
- auth模块的get_user中查库
 - 成功则获得user对象，is_authenticated为True
 - 失败返回匿名用户对象
 - 把返回的对象赋值给request.user

注意：authenticate不管request.user是否是匿名对象，它负责对提供的凭证（可以是用户名、密码）进行验证，验证成功返回user对象，否则返回None。

DRF也提供了基于Django Session的SessionAuthentication

- 认证成功，request.user存放user实例，request.auth是None
- 认证失败，返回403
- 不认证，request.user就是匿名用户，request.auth是None
- 浏览器发起请求时没有sessionid，不必验证csrf token，一旦浏览器端提供了sessionid，就必须验证csrf token了，否则报错

<https://www.django-rest-framework.org/api-guide/authentication/#sessionauthentication>

上面的意思是，浏览器发起的请求，在调用视图函数之前，SessionAuthentication等就已经完成了对sessionid的确认。如果查到此ID，就经确认了用户的身份，也就是认证了，request.user就是该用户实例。如果没有sessionid或者对比失败，无法确认用户身份，request.user就是匿名用户对象。

settings.py中配置

```
1 REST_FRAMEWORK = {
2     'DEFAULT_AUTHENTICATION_CLASSES': [
3         #'rest_framework.authentication.BasicAuthentication',
4         'rest_framework.authentication.SessionAuthentication',
5     ]
6 }
```

基于Token的认证

由于Session的一些固有问题，目前也有替代方法，令牌Token方案就是其中一种。

用户登录成功，发回一个Token到浏览器端，浏览器端每一次请求都发给服务器端这个令牌，有则表示登录过了，无则表示无身份，需要重新登录。

这个令牌值被签名防篡改，保护过期时间。服务器端可以校验，可以判断是否过期。

DRF提供了一个TokenAuthentication，它需要使用数据库存储Token。当下广泛用于互联网中的JWT (JSON Web Token) 也是较好的选择，它不需要数据库。DRF官网也提供了基于DRF的第三方插件

JWT官网 <https://jwt.io/>

官网 <https://github.com/jazzband/djangorestframework-simplejwt>

文档 https://django-rest-framework-simplejwt.readthedocs.io/en/latest/getting_started.html

要求：Python 3.7+、Django 2.2+、DRF 3.10+

```
1 $ pip install djangorestframework-simplejwt
```

settings.py中

```
1 INSTALLED_APPS = [
2     ...
3     'rest_framework_simplejwt', # 注册应用，国际化，可选
4     ...
5 ]
6
7 REST_FRAMEWORK = {
8     'DEFAULT_AUTHENTICATION_CLASSES': [
9         'rest_framework_simplejwt.authentication.JWTAuthentication',
10    ]
11 }
```

主路由配置

```
1 from django.contrib import admin
2 from django.urls import path
3 from rest_framework_simplejwt.views import (
4     TokenObtainPairView,
5     TokenRefreshView,
6 )
7
8 urlpatterns = [
9     path('admin/', admin.site.urls),
10    path('token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
11    # 获取
12    path('token/refresh/', TokenRefreshView.as_view(),
13        name='token_refresh'), # 刷新
14]
```

TokenObtainPairView的父类TokenViewBase继承自GenericAPIView，且只实现了post方法，所以只能使用POST请求。

登录功能

登录功能实现

测试 POST <http://127.0.0.1:8000/token/>

```
1 错误 400 Bad Request
2
3  {
4     "username": [
5         "这个字段是必填项。"
6     ],
7     "password": [
8         "这个字段是必填项。"
9     ]
10 }
```

也就是说必须提供用户名和密码，认证成功后才能返回JWT Token。

使用POST方法提交json数据（用户名、密码）到<http://127.0.0.1:8000/token/>，内部会使用用户名密码查询数据库，认证成功返回token（有过期时间、user_id），认证失败返回401。


```

9 | urlpatterns = [
10 |     path('admin/', admin.site.urls),
11 |     path('token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
12 |     # 获取
13 |     path('token/refresh/', TokenRefreshView.as_view(),
14 |     name='token_refresh'), # 刷新
15 |     path('test', test),
16 | ]

```

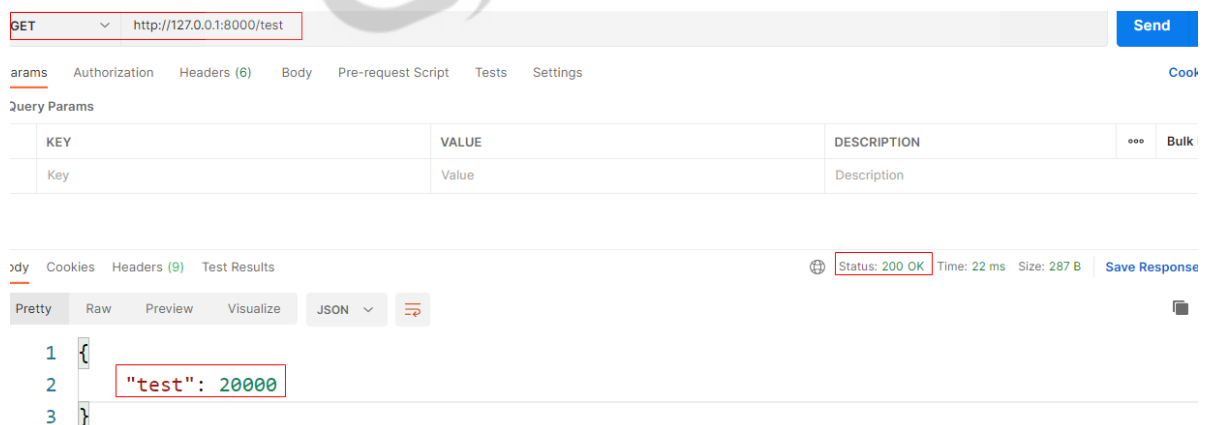
```

1 | from rest_framework.decorators import api_view
2 | from rest_framework.response import Response
3 | from rest_framework.request import Request
4 |
5 |
6 | @api_view(['POST', 'GET'])
7 | def test(request: Request):
8 |     print('~' * 30)
9 |     print(request.COOKIES, request._request.headers)
10 |    print(request.data) # 被DRF处理为字典
11 |    print(request.user) # 可能是匿名用户
12 |    print(request.auth)
13 |    print('=' * 30)
14 |    if request.auth:
15 |        return Response({'test': 10000})
16 |    else:
17 |        return Response({'test': 20000})

```

GET测试

1、无Token

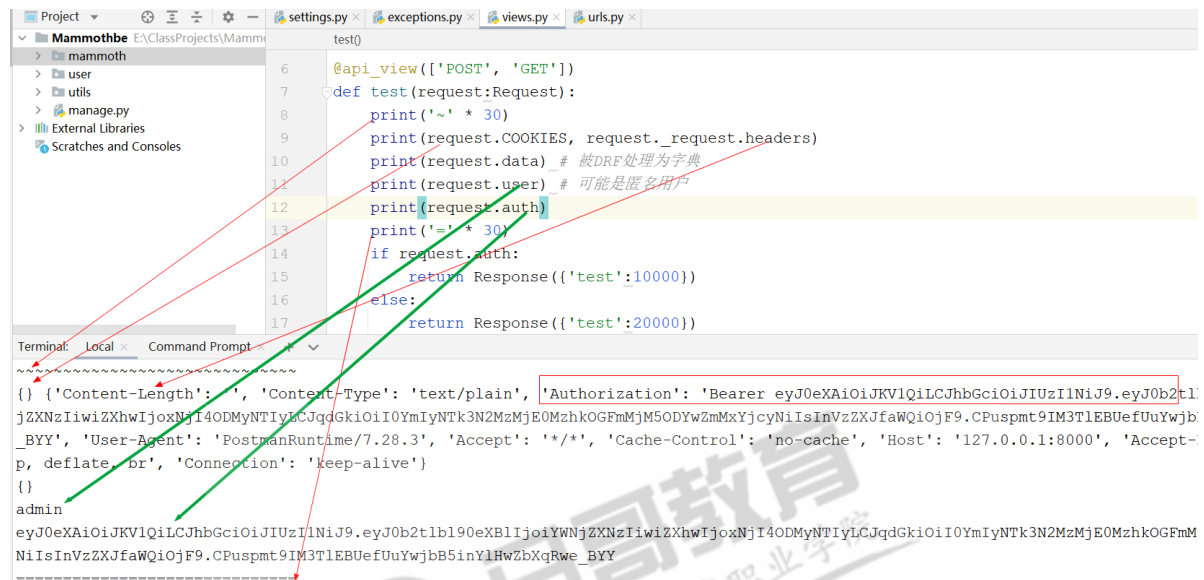
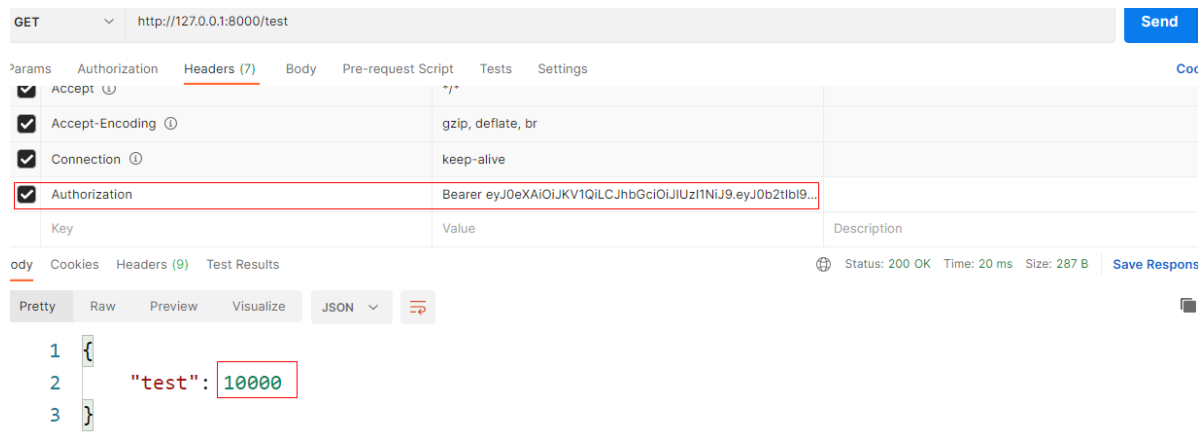


The screenshot shows a REST client interface. The top bar indicates a GET request to `http://127.0.0.1:8000/test`. Below the bar, there are tabs for `params`, `Authorization`, `Headers (6)`, `Body`, `Pre-request Script`, `Tests`, and `Settings`. The `params` tab is active, showing a table with columns `KEY`, `VALUE`, and `DESCRIPTION`. The table has one row with `Key` and `Value`. Below the table, there are tabs for `body`, `Cookies`, `Headers (9)`, and `Test Results`. The `Test Results` tab is active, showing the response status `Status: 200 OK`, `Time: 22 ms`, and `Size: 287 B`. The response body is displayed in JSON format: `{ "test": 20000 }`.

请求到达了视图函数，request.user为AnonymousUser，request.auth为None

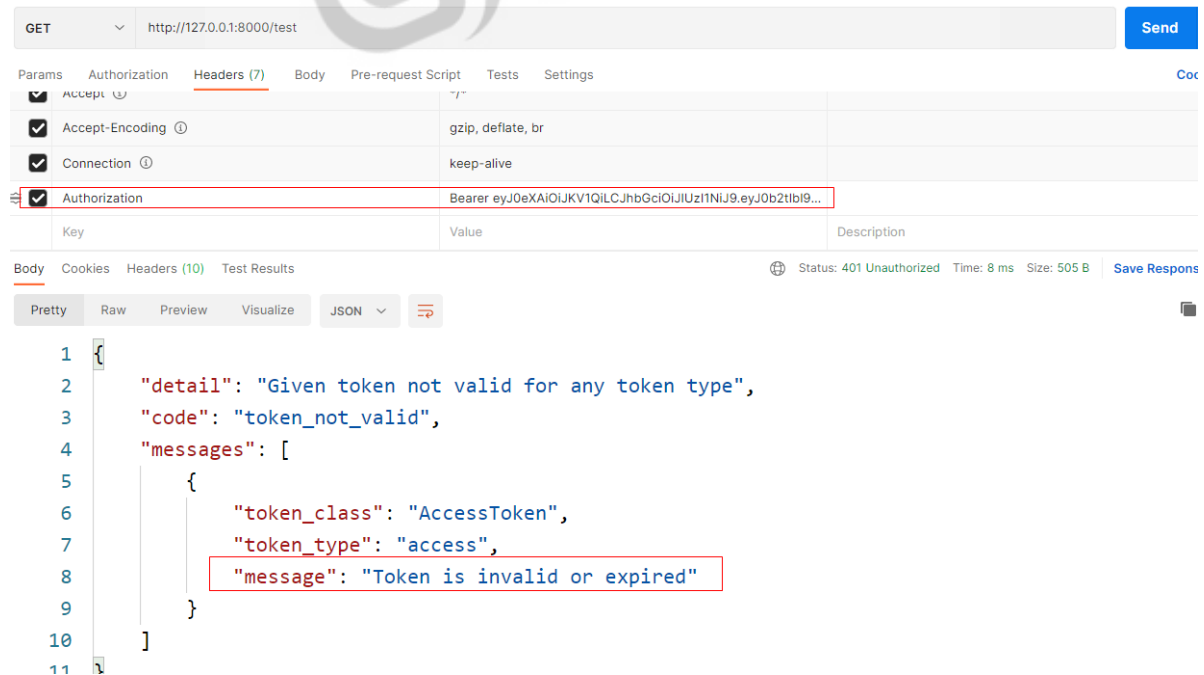
2、有Token

注意使用请求头字段Authorization



也就是访问的时候在请求报文头带上token就行。验证成功走正常流程，一旦失败走失败处理流程。

3、Token过期



注意，这个异常是在到达视图函数前就抛出的。

过期后，返回错误码，前端代码看到后需要跳转到登录页

POST测试

同GET测试，效果一样。

异常处理

如果登录或认证失败，返回4xx也可以，但是浏览器端使用的是Vue开发的，为了浏览器端处理方便，返回状态码统一采用200，那如何返回错误信息呢？

约定返回的json，如果正常code为0或者undefined，如果有异常返回非0，并且提供message提供消息。

可以每一个视图中handler中单独设置异常怎么处理，也可以全局统一处理。

DRF的异常的基类是 `rest_framework.exceptions.APIException`

在utils包下创建exceptions.py

```
1 from django.http import Http404
2 from django.core.exceptions import PermissionDenied
3 from rest_framework.views import set_rollback
4 from rest_framework import exceptions
5 from rest_framework.views import Response
6
7 class MagBaseException(exceptions.APIException):
8     """基类定义基本的异常"""
9     code = 10000 # code为0表示正常，非0都是错误
10    message = '非法请求' # 错误描述
11
12    @classmethod
13    def get_message(cls):
14        return {'code': cls.code, 'message': cls.message}
15
16 class InvalidUsernameOrPassword(MagBaseException):
17     code = 1
18     message = '用户名或密码错误'
19
20 # 内部异常暴露细节
21 exc_map = {
22     'AuthenticationFailed': InvalidUsernameOrPassword
23 }
24
25
26 def global_exception_handler(exc, context):
27     """
28     全局异常处理
29
30     照抄rest_framework.views.exception_handler，略作修改
31     不管什么异常这里统一处理。根据不同类型显示不同的
32     为了前端解析方便，这里响应的状态码采用默认的200
33     异常对应处理后返回对应的错误码和错误描述
34     异常找不到对应就返回缺省
35     """
36     if isinstance(exc, Http404):
37         exc = exceptions.NotFound()
38     elif isinstance(exc, PermissionDenied):
```

```

39         exc = exceptions.PermissionDenied()
40
41     print('异常', '=' * 30)
42     print(type(exc), exc.__dict__)
43     print('=' * 30)
44
45     if isinstance(exc, exceptions.APIException):
46         headers = {}
47         if getattr(exc, 'auth_header', None):
48             headers['WWW-Authenticate'] = exc.auth_header
49         if getattr(exc, 'wait', None):
50             headers['Retry-After'] = '%d' % exc.wait
51
52         if isinstance(exc.detail, (list, dict)):
53             data = exc.detail
54         else:
55             data = {'detail': exc.detail}
56
57         set_rollback()
58
59         if isinstance(exc, MagBaseException):
60             errmsg = exc.get_message()
61         else:
62             errmsg = exc_map.get(exc.__class__.__name__,
MagBaseException).get_message()
63         return Response(errmsg, status=200) # 状态恒为200
64         #return Response(data, status=exc.status_code, headers=headers)
65
66     return None

```

全局异常配置如下

```

1  REST_FRAMEWORK = {
2      'EXCEPTION_HANDLER': 'utils.exceptions.global_exception_handler',
3      'DEFAULT_AUTHENTICATION_CLASSES': [
4          'rest_framework_simplejwt.authentication.JWTAuthentication'
5      ]
6  }

```

注意：global_exception_handler捕获的是异常，需要raise。

前后端联调

后台主路由增加登录

```

1  from django.contrib import admin
2  from django.urls import path
3  from rest_framework_simplejwt.views import (
4      TokenObtainPairView,
5      TokenRefreshView,
6  )
7  from user.views import test
8
9  tobview = TokenObtainPairView.as_view() # 这个视图函数生成一次就可以了，可以调用n次
10

```

```

11 | urlpatterns = [
12 |     path('admin/', admin.site.urls),
13 |     path('login/', tobview, name='login'),
14 |     path('token/', tobview, name='token_obtain_pair'), # 获取
15 |     path('token/refresh/', TokenRefreshView.as_view(),
16 |         name='token_refresh'), # 刷新
17 |     path('test', test),
18 | ]

```

前端main.js

```

1 | axios.defaults.baseURL = 'http://127.0.0.1:8000/'
2 | vue.prototype.$http = axios

```

前端Login.vue中的login函数

```

1 | const res = await this.$http.post('login/', this.loginForm)

```

启动前端项目，输入用户名密码后点击登录，立即发现跨域请求问题（端口不一样）。解决方法，就是发起对后台的请求，用代理。

前端main.js

```

1 | // axios全局设置，baseURL指向后台服务
2 | axios.defaults.baseURL = '/api/v1/' // 代理到 'http://127.0.0.1:8000/'
3 | // 为vue类增加全局属性$http，这样所有组件实例都可以使用该属性了
4 | vue.prototype.$http = axios

```

代理

前端项目根目录下，配置vue.config.js(和 package.json 同级的)。

参考 <https://cli.vuejs.org/zh/config/#devserver-proxy>.

```

1 | module.exports = {
2 |   devServer: {
3 |     proxy: {
4 |       '/api/v1': {
5 |         target: 'http://localhost:8000',
6 |         changeOrigin: true
7 |       },
8 |     },
9 |   },
10 | }

```

登录发现竟然返回404，为什么？浏览器中可以看到访问的是

`http://localhost:8080/api/v1/login`。也就是说，确实代理了，但是URL不对，多了/api/v1。拿掉它，就要用rewrites了。

去webpack官网 <https://www.webpackjs.com/configuration/dev-server/#devserver-proxy>.

```

1  module.exports = {
2    devServer: {
3      proxy: {
4        '/api/v1': {
5          target: 'http://localhost:8000',
6          changeOrigin: true,
7          pathRewrite: {'^/api/v1' : ''}
8        },
9      }
10   }
11 }

```

终于，联调成功，用户名、密码正确后，状态码200，token返回，里面包含着过期时间和user_id。

Home组件

登录成功后，将看见Home组件。

src/components/Home.vue

```

1  <template>
2    <div>Home组件</div>
3  </template>
4
5  <script>
6    export default {}
7  </script>
8
9  <style>
10 </style>

```

router/index.js

```

1  import Vue from 'vue'
2  import VueRouter from 'vue-router'
3  import Login from '../components/Login.vue'
4  import Home from '../components/Home.vue'
5
6  Vue.use(VueRouter)
7
8  const routes = [
9    { path: '/', redirect: '/login' },
10   { path: '/login', component: Login },
11   { path: '/home', component: Home }
12 ]
13
14 const router = new VueRouter({
15   routes
16 })
17
18 export default router

```

登陆成功跳转，使用 `this.$router`，还要用到编程式导航 <https://router.vuejs.org/zh/guide/essentials/navigation.html>

Login.vue如下

```
1 <script>
2 export default {
3   methods: {
4     login() {
5       this.$refs.loginFormRef.validate(async (valid) => {
6         if (valid) {
7           const res = await this.$http.post('login/', this.loginForm) //
// post返回一个Promise
8           console.log(res) // status状态码, data返回的数据
9           const { data: response } = res // data解构出来
10          // 返回的对象
11          console.log(response)
12          console.log(response.code)
13          if (!response.code) {
14            // 如果返回的对象没有code属性或不为0, 说明成功了
15            console.log('登录成功')
16            this.$router.push('/home') // 实现跳转
17          } else {
18            console.log('登录失败')
19          }
20        }
21      })
22    }
23  }
24 }
25 </script>
```

特别说明：由于后面代码越来越长，只将改动部分贴出来参考

消息提示

src/plugins/element.js如下

```
1 import Vue from 'vue'
2 import { Form, FormItem, Input, Button, Message } from 'element-ui'
3 import 'element-ui/lib/theme-chalk/index.css'
4
5 Vue.use(Button)
6 Vue.use(Form)
7 Vue.use(FormItem)
8 Vue.use(Input)
9
10 // 全局导入
11 Vue.prototype.$message = Message
```

全局导入后，在每一个Vue组件上都可以通过 `this.$message` 来使用了

Login.Vue变化代码如下

```
1 <script>
2 export default {
3   methods: {
```

```

4      login() {
5          this.$refs.loginFormRef.validate(async (valid) => {
6              if (valid) {
7                  const res = await this.$http.post('login/', this.loginForm) //
post返回一个Promise
8                  // console.log(res) // status状态码, data返回的数据
9                  const { data: response } = res // data解构出来
10                 // 返回的对象
11                 // console.log(response.code, response.message)
12                 if (response.code) {
13                     // 如果返回的对象有code属性, 说明登录不成功
14                     return this.$message.error(response.message)
15                 }
16                 this.$message('登录成功')
17                 this.$router.push('/home')
18             }
19         })
20     }
21 }
22 }
23 }
24 </script>

```

Token持久化

直接在浏览器中输入<http://127.0.0.1:8080/#/home>, 不用登录也可以访问, 登录形同虚设。

应该将登录的、未登录的区分开来, 未登录跳转到登录页。浏览器如何区分? 用token, 可将token存储起来, 方便使用。

浏览器会持久化一些键值对的数据, 早期仅可以使用cookie, 现在还可以使用LocalStorage、SessionStorage等。

LocalStorage存储的数据可以长期保留, SessionStorage的数据是会话级的, 会话结束会被清除。

通过全局window对象可以使用localStorage、sessionStorage 属性设置其中的键值对。键值对总是以字符串的形式存储。

```

1 // sessionStorage方法一样
2 window.localStorage.setItem('k1', 'v1')
3 var val = localStorage.getItem('k1')
4 localStorage.removeItem('k1')
5 localStorage.clear() // 移除所有

```

导航守卫

简单的办法, 就是浏览器端登录成功后, 拿到一个不可篡改的凭证, 只要凭证在, 就认为登录过了, 这也是token。

浏览器每一次向浏览器发请求的时候, 服务器端验证这个token, 成功了服务器端返回请求的资源, 失败返回失败代码。

管理后台几乎所有组件都需要身份验证，能不能找一个公共的地方，进行判断？无token直接路由到 /login；有token，则跳转到目标路由，至于组件是否显示数据，需要把token发送到服务器端，有服务器端判断token是否有效。

全局导航守卫参考 <https://router.vuejs.org/zh/guide/advanced/navigation-guards.html>

```
1  const router = new VueRouter({ ... })
2
3  // 全局前置守卫，在每一次路由前。
4  // to要进入的目标，from当前导航要离开的路由
5  // next回调函数，next()：进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是
   confirmed（确认的）。
6  router.beforeEach((to, from, next) => {
7    // ...
8  })
```

src/router/index.js

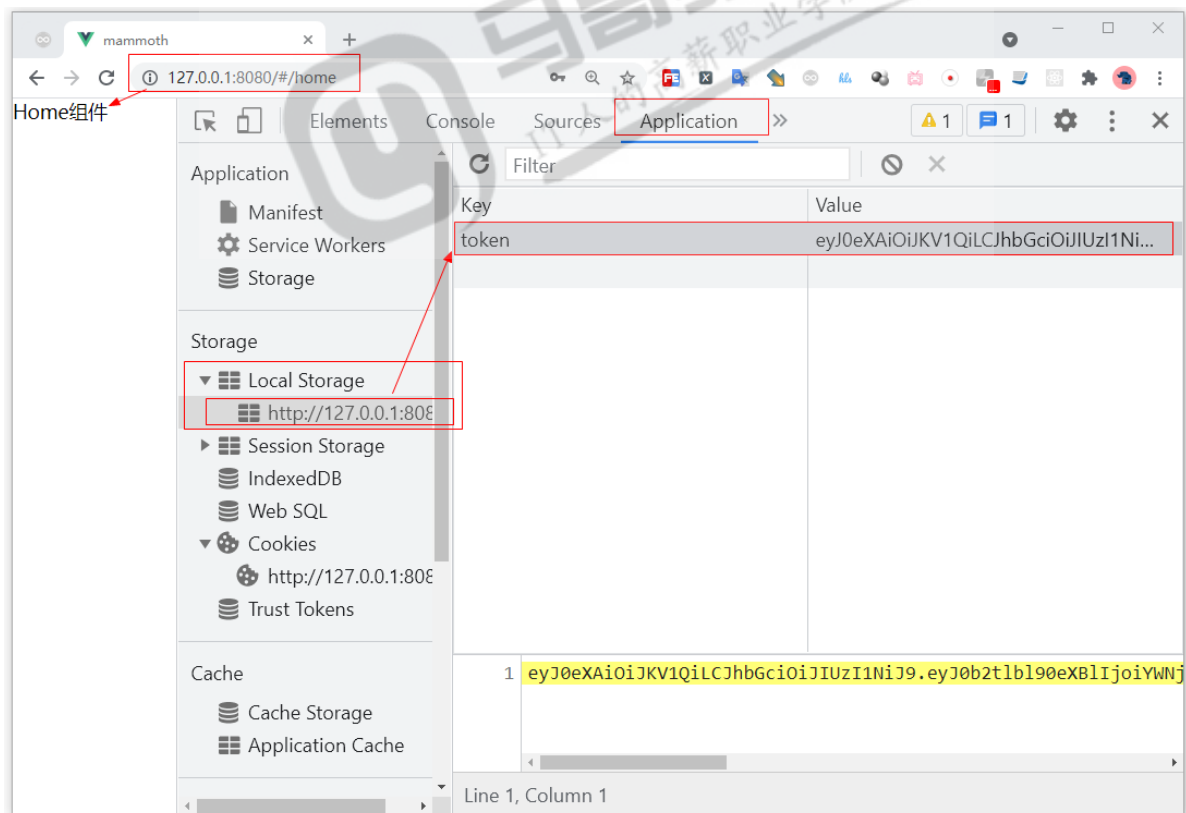
```
1  import Vue from 'vue'
2  import VueRouter from 'vue-router'
3  import Login from '../components/Login.vue'
4  import Home from '../components/Home.vue'
5
6  Vue.use(VueRouter)
7
8  const routes = [
9    { path: '/', redirect: '/login' },
10   { path: '/login', component: Login },
11   { path: '/home', component: Home }
12 ]
13
14 const router = new VueRouter({
15   routes
16 })
17
18 // 挂载全局导航守卫
19 router.beforeEach((to, from, next) => {
20   // from从哪里来，to去哪里，next函数跳转
21   if (to.path === '/login') {
22     next()
23   } else {
24     // 读取token
25     const token = window.localStorage.getItem('token')
26     if (!token) {
27       next('/login')
28     } else {
29       next()
30     }
31   }
32 })
33
34 export default router
```


存储Token

登录成功后，提取access token存储起来。

Login.vue如下

```
1 <script>
2 export default {
3   methods: {
4     login() {
5       this.$refs.loginFormRef.validate(async (valid) => {
6         if (valid) {
7           const { data: response } = await this.$http.post('login/',
this.loginForm)
8           if (response.code) {
9             return this.$message.error(response.message)
10          }
11          this.$message('登录成功')
12          window.localStorage.setItem('token', response.access)
13          this.$router.push('/home')
14        }
15      })
16    }
17  }
18 }
19 </script>
```



前端提交合并代码

还在login分支

合并到主分支

```

1 $ git checkout master
2 Switched to branch 'master'
3 Your branch is up to date with 'origin/master'.
4
5 $ git branch
6   login
7  * master
8
9 $ git merge login
10 Updating e3182dc..0d55786
11 Fast-forward
12  .eslintrc.js           |    3 +-
13  .prettierrc            |    5 +
14  README.md              |    2 +-
15  babel.config.js        |   15 +-
16  package.json           |    7 +-
17  src/App.vue            |   27 +-
18  src/assets/css/main.css |    8 +
19  src/components/Home.vue |   20 +
20  src/components/Login.vue |  128 ++++
21  src/main.js            |    7 +
22  src/plugins/element.js |   11 +
23  src/router/index.js     |   34 +-
24  yarn.lock              | 1635 +-----
25
26 13 files changed, 1037 insertions(+), 865 deletions(-)
27 create mode 100644 .prettierrc
28 create mode 100644 src/assets/css/main.css
29 create mode 100644 src/components/Home.vue
30 create mode 100644 src/components/Login.vue
31 create mode 100644 src/plugins/element.js
32 缺省使用fast-forward合并，如果不需要使用--no-ff
33
34 $ git push
35 目前在master分支，所以，只把master分支推送到了远程仓库

```

推送login分支

```
1 $ git checkout login
2 Switched to branch 'login'
3
4 $ git push
5 fatal: The current branch login has no upstream branch.
6 To push the current branch and set the remote as upstream, use
7
8     git push --set-upstream origin login
9
```

```

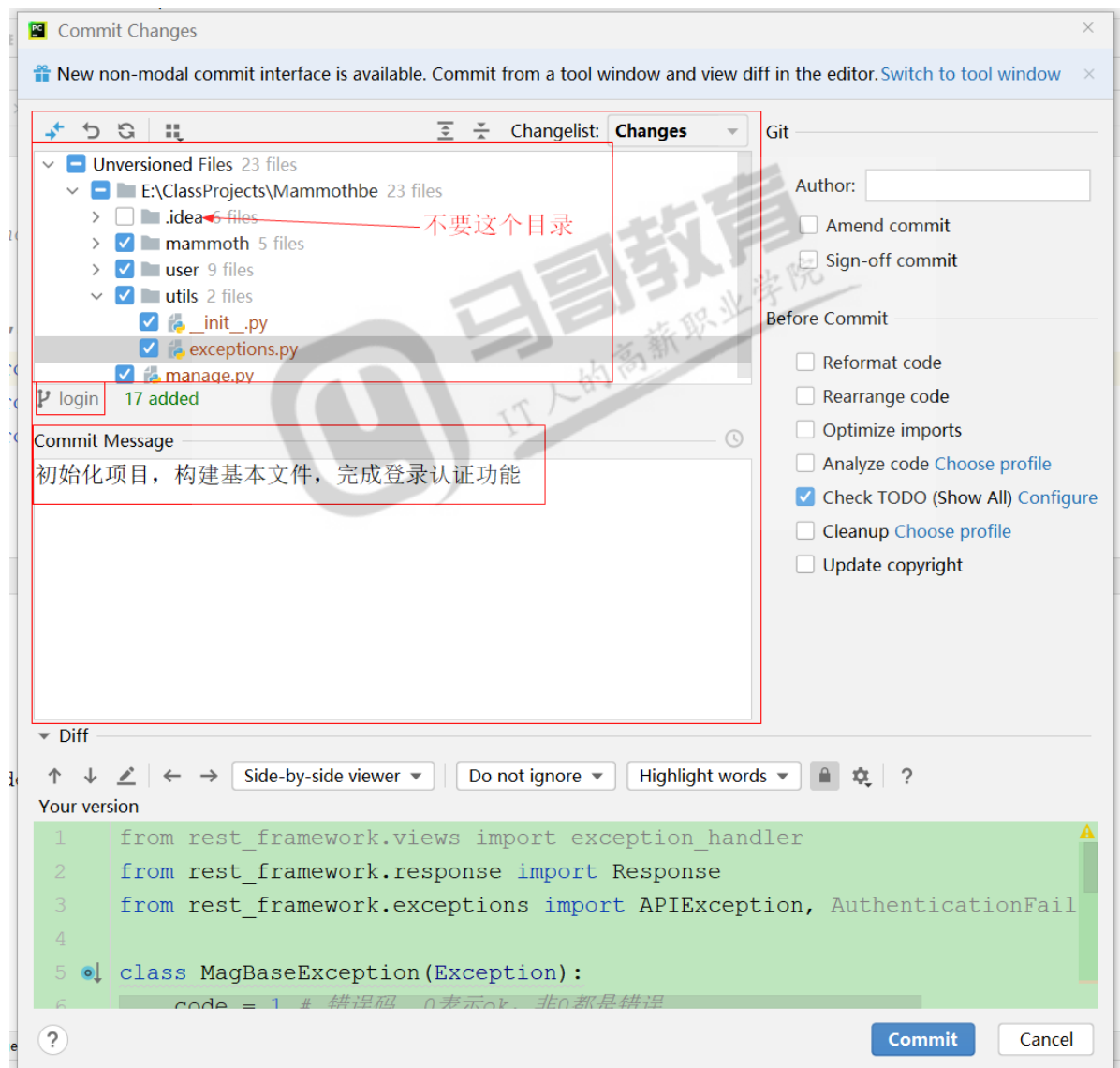
10 $ git push -u origin login
11 Total 0 (delta 0), reused 0 (delta 0)
12 remote: Powered by GITEE.COM [GNK-5.0]
13 remote: Create a pull request for 'login' on Gitee by visiting:
14 remote:
15 https://gitee.com/cloudytimes/vueadmin/pull/new/cloudytimes:login...cloudytimes:master
16 To gitee.com:cloudytimes/vueadmin.git
17 * [new branch] login -> login
18 Branch 'login' set up to track remote branch 'login' from 'origin'.

```

后端提交合并代码

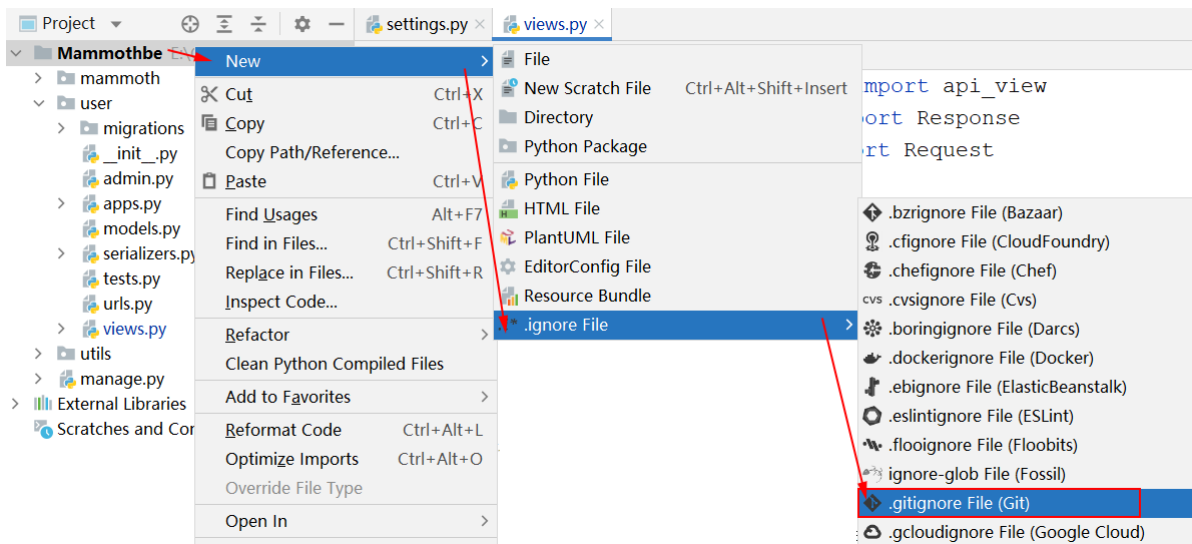
Pycharm中

```
1 $ git checkout -b login
```



git checkout master 然后在菜单中操作login合并到master，然后push远程仓库。

构建git忽略文件



.gitignore

```
1 /t*.py
2 .idea/
3 migrations/
```

migrations/, .idea/, 带斜杠结尾的只能匹配目录

/t*.py, .gitignore所在目录下的t开头的py文件

