用户的登陆认证

前端显示登陆页面

登录页组件

Login.vue

```
<template>
    <div class="login-box">
        <img src="https://www.luffycity.com/static/img/Loginbg.3377d0c.jpg" alt="">
        <div class="login">
            <div class="login-title">
                <img src="https://www.luffycity.com/static/img/Logotitle.1ba5466.png" alt="">
                <p>帮助有志向的年轻人通过努力学习获得体面的工作和生活!</p>
            </div>
            <div class="login_box">
                <div class="title">
                    <span @click="login_type=0">密码登录</span>
                    <span @click="login_type=1">短信登录</span>
                </div>
                <div class="inp" v-if="login_type==0">
                    <input v-model = "username" type="text" placeholder="用户名 / 手机号码"
class="user">
                    <input v-model = "password" type="password" name="" class="pwd"
placeholder="密码">
                    <div id="geetest1"></div>
                    <div class="rember">
                        <p>
                            <input type="checkbox" class="no" name="a"/>
                            <span>记住密码</span>
                        </p>
                        <p>忘记密码</p>
                    </div>
                    <button class="login_btn">登录</button>
                    <p class="go_login" >没有账号 <span>立即注册</span></p>
                </div>
                <div class="inp" v-show="login_type==1">
                    <input v-model = "username" type="text" placeholder="手机号码" class="user">
                    <input v-model = "password"  type="text" class="pwd" placeholder="短信验证
码">
          <button id="get_code">获取验证码</button>
                    <button class="login_btn">登录</button>
                    <p class="go_login" >没有账号 <span>立即注册</span></p>
                </div>
            </div>
        </div>
    </div>
</template>

<script>
export default {
  name: 'Login',
  data(){
```

```
    return {
        login_type: 0,
        username:"",
        password:"",
    }
  },

  methods:{

  },

};
</script>
<style scoped>
.login-box{
    width: 100%;
    height: 100%;
    position: relative;
    overflow: hidden;
    margin-top: -80px;
}
.login-box img{
    width: 100%;
    min-height: 100%;
}
.login-box .login {
    position: absolute;
    width: 500px;
    height: 400px;
    left: 0;
    margin: auto;
    right: 0;
    bottom: 0;
    top: -220px;
}
.login .login-title{
     width: 100%;
    text-align: center;
}
.login-title img{
    width: 190px;
    height: auto;
}
.login-title p{
    font-size: 18px;
    color: #fff;
    letter-spacing: .29px;
    padding-top: 10px;
    padding-bottom: 50px;
}
.login_box{
    width: 400px;
    height: auto;
    background: #fff;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.5);
    border-radius: 4px;
    margin: 0 auto;
    padding-bottom: 40px;
}
.login_box .title{
```

```
    font-size: 20px;
    color: #9b9b9b;
    letter-spacing: .32px;
    border-bottom: 1px solid #e6e6e6;
    display: flex;
    justify-content: space-around;
    padding: 50px 60px 0 60px;
    margin-bottom: 20px;
    cursor: pointer;
}
.login_box .title span:nth-of-type(1){
    color: #4a4a4a;
    border-bottom: 2px solid #84cc39;
}

.inp{
    width: 350px;
    margin: 0 auto;
}
.inp input{
    outline: 0;
    width: 100%;
    height: 45px;
    border-radius: 4px;
    border: 1px solid #d9d9d9;
    text-indent: 20px;
    font-size: 14px;
    background: #fff !important;
}
.inp input.user{
    margin-bottom: 16px;
}
.inp .rember{
    display: flex;
    justify-content: space-between;
    align-items: center;
    position: relative;
    margin-top: 10px;
}
.inp .rember p:first-of-type{
    font-size: 12px;
    color: #4a4a4a;
    letter-spacing: .19px;
    margin-left: 22px;
    display: -ms-flexbox;
    display: flex;
    -ms-flex-align: center;
    align-items: center;
    /*position: relative;*/
}
.inp .rember p:nth-of-type(2){
    font-size: 14px;
    color: #9b9b9b;
    letter-spacing: .19px;
    cursor: pointer;
}

.inp .rember input{
    outline: 0;
    width: 30px;
```

```
    height: 45px;
    border-radius: 4px;
    border: 1px solid #d9d9d9;
    text-indent: 20px;
    font-size: 14px;
    background: #fff !important;
}

.inp .rember p span{
    display: inline-block;
    font-size: 12px;
    width: 100px;
  /*position: absolute;*/
/*left: 20px;*/

}
#geetest{
    margin-top: 20px;
}
.login_btn{
     width: 100%;
    height: 45px;
    background: #84cc39;
    border-radius: 5px;
    font-size: 16px;
    color: #fff;
    letter-spacing: .26px;
    margin-top: 30px;
}
.inp .go_login{
    text-align: center;
    font-size: 14px;
    color: #9b9b9b;
    letter-spacing: .26px;
    padding-top: 20px;
}
.inp .go_login span{
    color: #84cc39;
    cursor: pointer;
}
</style>
```

绑定登陆页面路由地址

main.js

```
import Vue from "vue"
import Router from "vue-router"

// 导入需要注册路由的组件
import Home from "../components/Home"
import Login from "../components/Login"
Vue.use(Router);

// 配置路由列表
export default new Router({
```

```
  mode:"history",
  routes:[
    // 路由列表
        ...
    {
      name:"Login",
      path: "/login",
      component:Login,
    }
  ]
})
```

调整首页头部子组件中登陆按钮的链接信息

Header.vue

```
<router-link to="/login">登录</router-link>
```

后端实现登陆认证

Django默认已经提供了认证系统。认证系统包含：

- 用户管理
- 权限
- 用户组
- 密码哈希系统
- 用户登录或内容显示的表单和视图
- 一个可插拔的后台系统

Django默认用户的认证机制依赖Session机制，我们在项目中将引入JWT认证机制，将用户的身份凭据存放在Token中，然后对接Django的认证系统，帮助我们来实现：

- 用户的数据模型
- 用户密码的加密与验证
- 用户的权限系统

Django用户模型类

Django认证系统中提供了用户模型类User保存用户的数据，默认的User包含以下常见的基本字段：

字段名	字段描述
`username`	必选。150个字符以内。 用户名可能包含字母数字， `_` ， `@` ， `+` `.` 和 `-` 个字符。
`first_name`	可选（ `blank=True` ）。 少于等于30个字符。
`last_name`	可选（ `blank=True` ）。 少于等于30个字符。
`email`	可选（ `blank=True` ）。 邮箱地址。
`password`	必选。 密码的哈希加密串。 （Django 不保存原始密码）。 原始密码可以无限长而且可以包含任意字符。
`groups`	与 `Group` 之间的多对多关系。
`user_permissions`	与 `Permission` 之间的多对多关系。
`is_staff`	布尔值。 设置用户是否可以访问Admin 站点。
`is_active`	布尔值。 指示用户的账号是否激活。 它不是用来控制用户是否能够登录，而是描述一种帐号的使用状态。
`is_superuser`	是否是超级用户。超级用户具有所有权限。
`last_login`	用户最后一次登录的时间。
`date_joined`	账户创建的时间。 当账号创建时，默认设置为当前的date/time。

常用方法：

- `set_password` (*raw_password*)

  设置用户的密码为给定的原始字符串，并负责密码的。 不会保存 `User` 对象。当 `None` 为 `raw_password` 时，密码将设置为一个不可用的密码。
- `check_password` (*raw_password*)

  如果给定的raw_password是用户的真实密码，则返回True，可以在校验用户密码时使用。

管理器方法：

管理器方法即可以通过 `User.objects.` 进行调用的方法。

- `create_user` (*username*, *email=None*, *password=None*, ***extra_fields*)

  创建、保存并返回一个 `User` 对象。
- `create_superuser` (*username*, *email*, *password*, ***extra_fields*)

  与 `create_user()` 相同，但是设置 `is_staff` 和 `is_superuser` 为 `True` 。

创建用户模块的子应用

```
cd luffy/apps
python ../../manage.py startapp users
```

在settings.py文件中注册子应用。

```
INSTALLED_APPS = [
        ...
    'users',
]
```

创建自定义的用户模型类

Django认证系统中提供的用户模型类及方法很方便，我们可以使用这个模型类，但是字段有些无法满足项目需求，如本项目中需要保存用户的手机号，需要给模型类添加额外的字段。

Django提供了 `django.contrib.auth.models.AbstractUser` 用户抽象模型类允许我们继承，扩展字段来使用Django认证系统的用户模型类。

我们可以在apps中创建Django应用users，并在配置文件中注册users应用。

在创建好的应用models.py中定义用户的用户模型类。

```
class User(AbstractUser):
    """用户模型类"""
    mobile = models.CharField(max_length=11, unique=True, verbose_name='手机号')

    class Meta:
        db_table = 'ly_users'
        verbose_name = '用户'
        verbose_name_plural = verbose_name
```

我们自定义的用户模型类还不能直接被Django的认证系统所识别，需要在配置文件中告知Django认证系统使用我们自定义的模型类。

在配置文件中进行设置

```
AUTH_USER_MODEL = 'users.User'
```

`AUTH_USER_MODEL` 参数的设置以 `点.` 来分隔，表示 `应用名.模型类名` 。

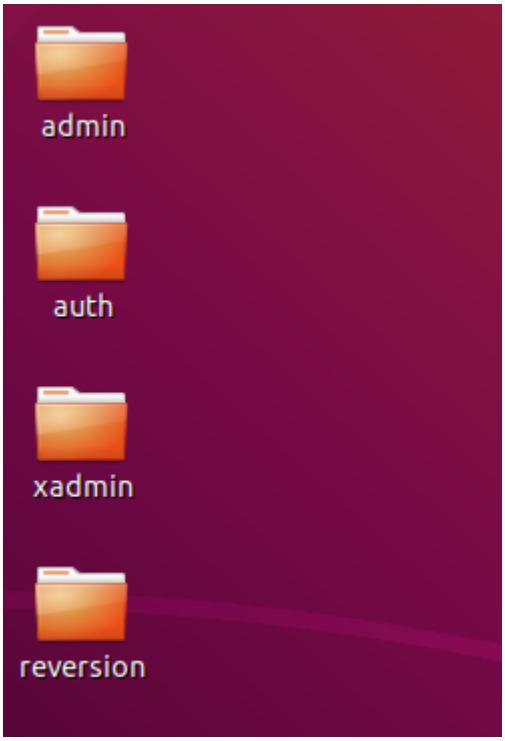
注意：Django建议我们对于AUTH_USER_MODEL参数的设置一定要在第一次数据库迁移之前就设置好，否则后续使用可能出现未知错误。

执行数据库迁移

```
python manage.py makemigrations
python manage.py migrate
```

当迁移出现问题，报错了，可以从django的解析器下找到site-package目录下的django里面的contrib里面的admin和auth，以及site-package下面的reversion以及xadmin里面的migrations目录

注意，防止手动操作误删数据，建议先备份。



Django REST framework JWT

在用户注册或登录后，我们想记录用户的登录状态，或者为用户创建身份认证的凭证。我们不再使用Session认证机制，而使用Json Web Token认证机制。

> Json web token (JWT), 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准 ((RFC 7519).该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

JWT的构成

JWT就一段字符串，由三段信息构成的，将这三段信息文本用 `.` 链接一起就构成了Jwt字符串。就像这样:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

第一部分我们称它为头部（header),第二部分我们称其为载荷（payload, 类似于飞机上承载的物品)，第三部分是签证（signature).

header

jwt的头部承载两部分信息：

- 声明类型，这里是jwt
- 声明加密的算法 通常直接使用 HMAC SHA256

完整的头部就像下面这样的JSON：

```
{
  'typ': 'JWT',
  'alg': 'HS256'
}
```

然后将头部进行base64加密（该加密是可以对称解密的),构成了第一部分.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

payload

载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分

- 标准中注册的声明
- 公共的声明
- 私有的声明

标准中注册的声明 (建议但不强制使用)：

- iss: jwt签发者
- sub: jwt所面向的用户
- aud: 接收jwt的一方
- exp: jwt的过期时间，这个过期时间必须要大于签发时间
- nbf: 定义在什么时间之前，该jwt都是不可用的.
- iat: jwt的签发时间
- jti: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

公共的声明： 公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密.

私有的声明： 私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

定义一个payload:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

然后将其进行base64加密，得到JWT的第二部分。

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9
```

signature

JWT的第三部分是一个签证信息，这个签证信息由三部分组成：

- header (base64后的)
- payload (base64后的)
- secret

这个部分需要base64加密后的header和base64加密后的payload使用 `.` 连接组成的字符串，然后通过header中声明的加密方式进行加盐 `secret` 组合加密，然后就构成了jwt的第三部分。

```
// javascript如果要模拟生成你的jwttoken，可能可以采用以下代码生成[注意：伪代码]
var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);

var signature = HMACSHA256(encodedString, 'secret'); //
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

将这三部分用 `.` 连接成一个完整的字符串,构成了最终的jwt:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4
iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

注意：secret是保存在服务器端的，jwt的签发生成也是在服务器端的，secret就是用来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了。

关于签发和核验JWT，我们可以使用Django REST framework JWT扩展来完成。

文档网站http://getblimp.github.io/django-rest-framework-jwt/

安装配置JWT

安装

```
pip install djangorestframework-jwt
```

settings/dev.py配置文件

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}

import datetime
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
}
```

- JWT_EXPIRATION_DELTA 指明token的有效期

生成jwt

Django REST framework JWT 扩展的说明文档中提供了手动签发JWT的方法

```
from rest_framework_jwt.settings import api_settings

jwt_payload_handler = api_settings.JWT_PAYLOAD_HANDLER
jwt_encode_handler = api_settings.JWT_ENCODE_HANDLER

payload = jwt_payload_handler(user)
token = jwt_encode_handler(payload)
```

在用户注册或登录成功后，在序列化器中返回用户信息以后同时返回token即可。

后端实现登陆认证接口

Django REST framework JWT提供了登录获取token的视图，可以直接使用

在子应用users路由urls.py中

```
from rest_framework_jwt.views import obtain_jwt_token

urlpatterns = [
    path(r'authorizations/', obtain_jwt_token, name='authorizations'),
]
```

在主路由中，引入当前子应用的路由文件

```
urlpatterns = [
        ...
    path('users/', include("users.urls")),
    # include 的值必须是 模块名.urls 格式,字符串中间只能出现一个圆点
]
```

接下来，我们可以通过postman来测试下功能

注意，测试之前因为我们上面自定义用户模型，所以删除了数据库信息，所以需要我们再次补充。

前端实现登陆功能

在登陆组件中src/components/Login.vue找到登陆按钮，绑定点击事件

```
<button class="login_btn" @click="loginhander">登录</button>
```

在methods中请求后端

```
export default {
  name: 'Login',
  data(){
    return {
        login_type: 0,
        remember:false, // 记住密码
        username:"",
        password:"",
    }
  },

  methods:{
    // 登录
    loginhander(){
      this.$axios.post(this.$settings.Host+"/users/login/",
{"username":this.username,"password":this.password}).then(response=>{
        console.log(response.data)
      }).catch(error=>{
        console.log(error)
      })
    }
  },
};
```

2.3.7 前端保存jwt

我们可以将JWT保存在cookie中，也可以保存在浏览器的本地存储里，我们保存在浏览器本地存储中

浏览器的本地存储提供了sessionStorage 和 localStorage 两种：

- sessionStorage 浏览器关闭即失效
- localStorage 长期有效

使用方法

```
sessionStorage.变量名 = 变量值    // 保存数据
sessionStorage.变量名   // 读取数据
sessionStorage.clear()  // 清除所有sessionStorage保存的数据

localStorage.变量名 = 变量值    // 保存数据
localStorage.变量名   // 读取数据
localStorage.clear()  // 清除所有localStorage保存的数据
```

登陆组件代码Login.vue的loginhander()方法内

```
    // 使用浏览器本地存储保存token
  if (this.remember) {
    // 记住登录
    sessionStorage.clear();
    localStorage.token = response.data.token;
  } else {
    // 未记住登录
    localStorage.clear();
    sessionStorage.token = response.data.token;
  }
    // 页面跳转回到上一个页面 也可以使用 this.$router.push("/") 回到首页
    this.$router.go(-1)
```

默认的返回值仅有token，我们还需在返回值中增加username和id，方便在客户端页面中显示当前登陆用户

通过修改该视图的返回值可以完成我们的需求。

在users/utils.py 中，创建

```
def jwt_response_payload_handler(token, user=None, request=None):
    """
    自定义jwt认证成功返回数据
    """
    return {
        'token': token,
        'id': user.id,
        'username': user.username
    }
```

修改settings/dev.py配置文件

```
# JWT
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
    'JWT_RESPONSE_PAYLOAD_HANDLER': 'users.utils.jwt_response_payload_handler',
}
```

登陆组件代码Login.vue

```
```

```
    // 使用浏览器本地存储保存token
  if (this.remember) {
    // 记住登录
    sessionStorage.clear();
    localStorage.token = response.data.token;
    localStorage.id = response.data.id;
    localStorage.username = response.data.username;
  } else {
    // 未记住登录
    localStorage.clear();
    sessionStorage.token = response.data.token;
    sessionStorage.id = response.data.id;
    sessionStorage.username = response.data.username;
  }
```

2.3.9 多条件登录

JWT扩展的登录视图，在收到用户名与密码时，也是调用Django的认证系统中提供的authenticate()来检查用户名与密码是否正确。

我们可以通过修改Django认证系统的认证后端（主要是authenticate方法）来支持登录账号既可以是用户名也可以是手机号。

修改Django认证系统的认证后端需要继承django.contrib.auth.backends.ModelBackend，并重写authenticate方法。

`authenticate(self, request, username=None, password=None, **kwargs)` 方法的参数说明：

- request 本次认证的请求对象
- username 本次认证提供的用户账号
- password 本次认证提供的密码

我们想要让用户既可以以用户名登录，也可以以手机号登录，那么对于authenticate方法而言，username参数即表示用户名或者手机号。

重写authenticate方法的思路：

1. 根据username参数查找用户User对象，username参数可能是用户名，也可能是手机号
2. 若查找到User对象，调用User对象的 check_password 方法检查密码是否正确

在users/utils.py中编写：

```
from django.contrib.auth.backends import ModelBackend
from .models import User
from django.db.models import Q
import re

def get_user_by_account(account):
    """根据账号信息获取用户模型"""
    try:
        # if re.match('^1[3-9]\d{9}$', account):
        #     # 手机号
        #     user = User.objects.get(mobile=account)
        # else:
        #     # 用户名
        #     user = User.objects.get(username=account)
```

```
        user = User.objects.get(Q(mobile=account) | Q(username=account))

    except User.DoesNotExist:
        user = None

    return user


class UsernameMobileAuthBackend(ModelBackend):
    def authenticate(self, request, username=None, password=None, **kwargs):
        # 进行登录判断
        user = get_user_by_account(username)

        # 账号通过了还要进行密码的验证,以及判断当前站好是否是激活状态
        if isinstance(user,User) and user.check_password(password) and
self.user_can_authenticate(user):
            return user
```

在配置文件settings/dev.py中告知Django使用我们自定义的认证后端

```
AUTHENTICATION_BACKENDS = [
    'users.utils.UsernameMobileAuthBackend',
]
```

前端首页实现登陆状态的判断

Home组件代码：

```
<template>
  <div class="home">
    <Header :is_login="is_login" @logout="logout" :current_page="current_page"/>
    <Banner/>
    <Footer/>
  </div>
</template>

<script>
  import Header from "./common/Header"
  import Banner from "./common/Banner"
  import Footer from "./common/Footer"
  export default{
    name:"Home",
    data(){
      return {
        id: sessionStorage.id || localStorage.id,
        username: sessionStorage.username || localStorage.username,
        token: sessionStorage.token || localStorage.token,
        current_page:0,
        is_login:false,
      };
    },
    components:{
```

```
      Header,
      Banner,
      Footer,
    },
    mounted(){
      if(this.id && this.token){
        this.is_login=true;
      }
    },
    methods:{
      logout(){
        this.is_login=false;
      }
    }

</script>
```

Header子组件显示登陆状态

Header.vue ----- ??????????????????????????????????????

```
<script>
  export default {
    name:"Header",
    props:["current_page","is_login"],
    data(){
      return {
        nav_list: [],
      }
    },
    created: function(){
      // 获取轮播图
      this.$axios.get("http://api.luffycity.cn:8000/home/nav/").then(res => {
        this.nav_list = res.data
      }).catch(error => {
        console.log(error);
      });
    },
    methods:{
      logout(){
        localStorage.clear();
        sessionStorage.clear();
        this.$emit("logout",{"is_login":false})
      }
    }
  }
</script>
```

在登录认证中接入极验验证

接下一节课 day86...