django-auth模块

本节内容

- 了解 Django中的auth模块、安装及配置
- 理解 auth中的用户模型
- 掌握 登录验证、权限认证装饰器的使用
- 掌握 用户管理命令的使用

功能介绍

- 用户模型
  - ✓ 用户认证、登录、退出等
- 后台管理
  - ✓ 用户管理、权限分配
- 命令行工具
  - ✓ 创建用户、设置密码等

安装及配置

- 第一步：INSTALLED_APPS安装应用
  - ✓ 'django.contrib.auth'
  - ✓ 'django.contrib.contenttypes'
- 第二步：MIDDLEWARE中间件配置
  - ✓ SessionMiddleware
  - ✓ AuthenticationMiddleware
- 第三步：migrate同步模型到数据库

用户模型

- 注册用户（User）

张三

- 游客(AnonymousUser)

游客

用户模型

- 常用属性

张三

属性	描述
username	用户名
password	登录密码
email	电子邮箱
is_staff	是否为内部员工
is_active	是否为激活用户
is_superuser	是否为超级管理员

用户模型

- 常用属性

张三

属性	描述
is_authenticated	用户是否已登录的只读属性
is_anonymous	用户登录是否已失效的只读属性
last_login	最后登录的时间
date_joined	注册时间
groups	用户组多对多关系
user_permissions	用户权限多对多关系

用户认证授权系统

- 用户：记录用户的登录账号密码等信息
- 权限：用户是否有权限访问某一资源（执行操作）
- 组：对多个用户进行权限管理

用户管理

- 创建普通用户

```
user = User.objects.create_user('john',
  'test@example.com', 'johnpassword')
```

- 使用命令行创建超级管理员

```
>>> python manage.py createsuperuser
```

密码管理

- 设置、修改用户的密码

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

- 检查用户的密码是否正确

```
>>> u = User.objects.get(username='john')
>>> u. check_password( ‘my password')
```

用户管理

- 验证用户名和密码是否匹配

```
user = authenticate(username, password)
```

- 在视图中获取当前用户

```
request.user
```

- 需要登录才可访问的视图

```
@login_required
def my_view(request):
    pass
```

权限管理

- 判断用户是否具备某权限

```
request.user.has_perm('foo.add_bar')
```

- 强制权限验证

```
@permission_required('foo.add_bar')
def my_view(request):
    pass
```

小结

拿来即用，不要重复发明轮子

思考

用户模型要增加一些字段怎么办？

扩展Django中的用户模型

本节内容

- 了解 扩展用户模型的使用场景
- 掌握 如何自定义用户模型

对用户进行扩展

- 思考如下场景如何实现
  - ✓ Django内置的用户模型满足不了需求怎么办？
  - ✓ 要用户模型添加头像字段怎么办？

对用户进行扩展

- 两种方式解决用户问题
  - ✓ 方式1：使用OneToOneField对用户进行扩展
  - ✓ 方式2：替换现有的用户模型

替换现有的用户模型

- 步骤一：配置用户模型，告诉django框架

  AUTH_USER_MODEL = 'accounts.User'

- 步骤二：继承自AbstractUser抽象模型
- 步骤三：添加字段，同步模型到数据库

小结

扩展Django的用模型是对ORM抽象模型的应用

下节预告

入门Vuex状态管理

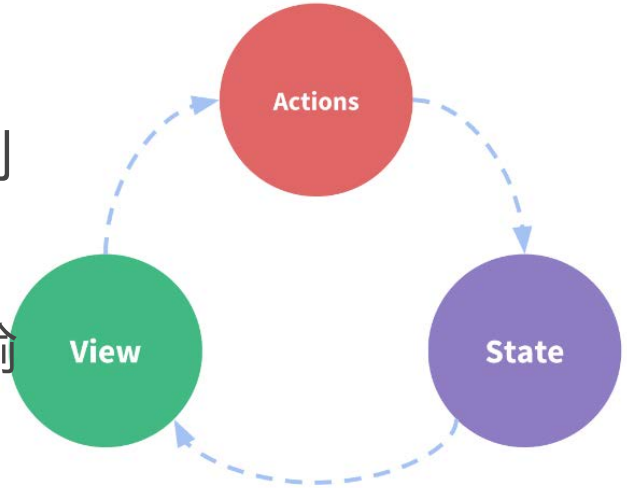
Vuex状态管理介绍

本节内容

- 了解 什么是状态管理
- 掌握 Vuex的安装和配置
- 理解 Vuex的原理
- 掌握 查询Vuex中的数据

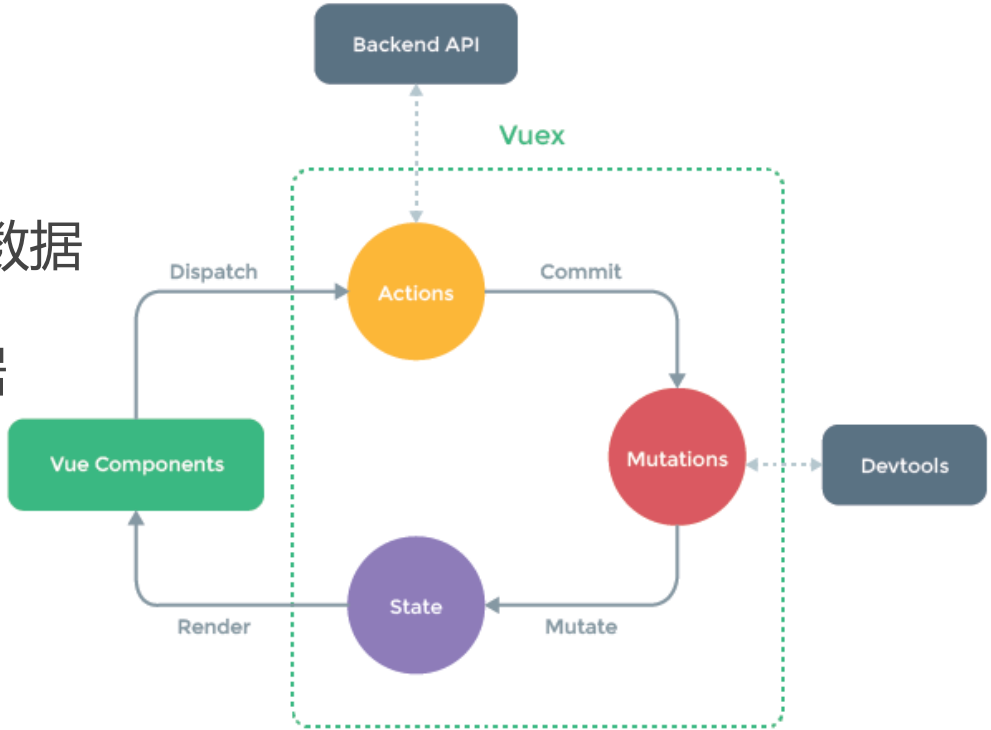
什么是Vuex

- 专为 Vue.js 应用程序开发的状态管理模式
  - ✓ state: 驱动应用的数据源
  - ✓ view: 以声明方式将 state 映射到视图
  - ✓ actions: 响应在 view 上的用户输入导致的状态变化



Vuex的工作流程

- state：存放数据
- mutation：同步提交数据
- action：异步提交数据



安装及配置

- 第一步，安装

```
cnpm install vuex -S
```

- 第二步，配置

```
Vue.use(Vuex)

export default new Vuex.Store({
  state: {},
  mutations: {},
  actions: {},
  modules: {}
})
```

```
new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

state

- 含义

  单一状态树，全局唯一数据源

查询state中的数据

- 示例代码

```
computed: {
    user () {
      return this.$store.state.user
    }
  }
```

小结

Vuex就是用来全局共享数据的

思考

如何将数据存储到Vuex中呢

使用Vuex保存用户信息

本节内容

- 了解 Vuex存储数据的原理
- 理解 同步和异步的区别
- 掌握 如何进行数据的存储

mutation

- 更改store中状态的唯一方法
- 示例代码

```
mutations: {
    increment (state) {
      // 变更状态
      state.count++
    }
  }
// 提交
store.commit('increment')
```

action

- 提交的是mutation，可异步
- 示例代码

```
actions: {
  increment (context) {
    context.commit('increment')
  }
}

store.dispatch('increment')
```

小结

注意不要通过赋值语法去修改store中的状态

思考

代码还能否再优化？

Vuex相关代码优化

本节内容

- 了解 Vuex提供的辅助函数
- 掌握 mapState的使用
- 掌握 mapMutations的使用
- 掌握 mapActions的使用

辅助函数

- mapState
- mapMutations
- mapActions

mapState

- 示例代码

```
computed: mapState({
    // 箭头函数可使代码更简练
    count: state => state.count
})
```

mapMutations

- 示例代码

```
methods: {
  ...mapMutations([
        // 将 `this.increment()` 映射为 `this.$store.commit('increment')`
    'increment'
  ])
}
```

mapActions

- 示例代码

```
methods: {
  ...mapActions([
        // 将 `this.increment()` 映射为 `this.$store.dispatch('increment')`
    'increment',
      ])
}
```

小结

辅助函数使我们的代码更加精简

用户登录页面开发

实现步骤

- 第一步，查找Vant中可以使用的组件
- 第二步，实现组件模板部分
- 第三步，模型层准备数据
- 第四步，模拟数据，实现效果

小结

用户注册页面开发

实现步骤

- 第一步，查找Vant中可以使用的组件
- 第二步，实现组件模板部分
- 第三步，实现验证码组件
- 第四步，实现登录注册页面间的跳转

小结

个人中心页面开发

实现步骤

- 第一步，查找Vant中可以使用的组件
- 第二步，实现组件模板部分
- 第三步，模型层准备数据
- 第四步，模拟数据，实现效果

小结