



# Flask

web development,  
one drop at a time

# 数据可视化 Python Flask框架数据库

李春芳 副教授

理工学部 计算机学院

2017-7-5



中国传媒大学

COMMUNICATION UNIVERSITY OF CHINA

# 实践课程概述

课程目标：软件开发中的数据可视化

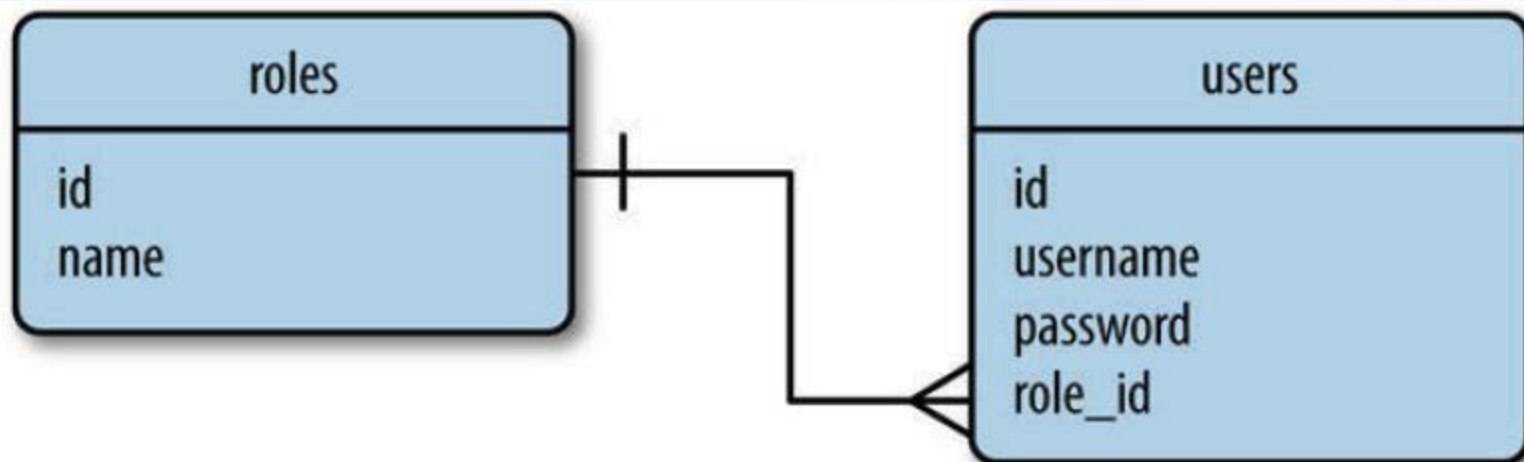


# 第三章数据库

*SQLAlchemy+PyMySQL*

# 数据库&NoSQL&NewSQL

Mysql, Postgres, SQLite, SqlServer, Oracle.....



# Python数据库框架SQLAlchemy

◆ 走别人的路，让别人无路可走……

◆ 原生SQL还是ORM

- ◆ 易用性：众说纷纭，好，还是不好……
- ◆ 性能：降低
- ◆ 可移植性：好
- ◆ 对象关系映射：Object-Relational Mapper, ORM
- ◆ 对象文档映射：Object-Document Mapper, ODM
- ◆ SQLAlchemy 的ORM支持Mysql、Postgres、SQLite，也支持使用原生SQL

# Flask链接Mysql库

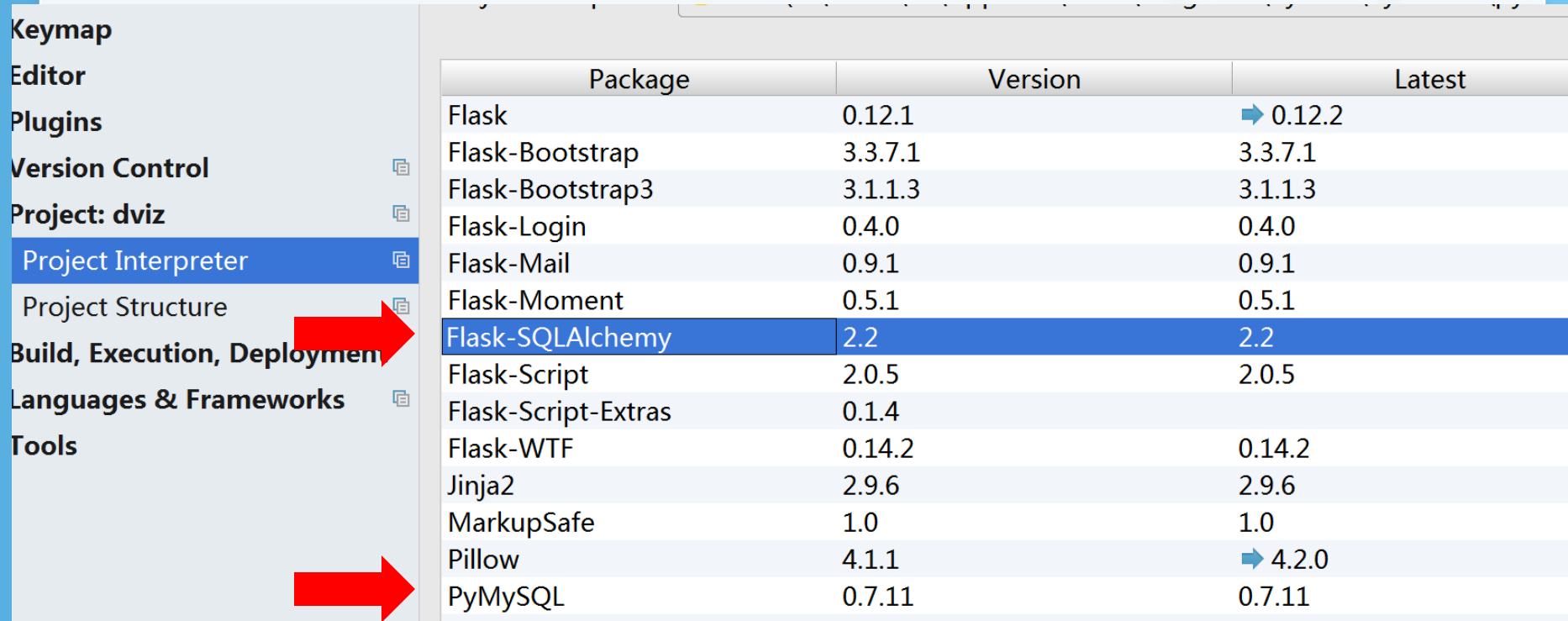
- ◆ 安装flask-sqlalchemy、pymysql模块
- ◆ `pip install flask-sqlalchemy pymysql`
- ◆ Flask-SQLAlchemy管理数据库
- ◆ Flask-SQLAlchemy是一个Flask扩展，它简化了在Flask应用程序中对SQLAlchemy的使用。
- ◆ <http://docs.sqlalchemy.org/en/latest/dialects/mysql.html>

表5-1 FLask-SQLAlchemy数据库URL

数据库引擎	URL
MySQL	<i>mysql://username:password@hostname/database</i>
Postgres	<i>postgresql://username:password@hostname/database</i>
SQLite (Unix)	<i>sqlite:///absolute/path/to/database</i>
SQLite (Windows)	<i>sqlite:///c:/absolute/path/to/database</i>

# 链接数据库——装包

› Flask-SQLAlchemy、PyMySQL



The screenshot shows the PyCharm IDE interface. On the left is the 'Keymap' sidebar with categories: Editor, Plugins, Version Control, Project: dviz, Project Interpreter, Project Structure, Build, Execution, Deployment, Languages & Frameworks, and Tools. A red arrow points from the 'Build, Execution, Deployment' category to the 'Package List' window. The 'Package List' window displays a table of installed and available packages. The 'Flask-SQLAlchemy' package is highlighted in blue, and the 'PyMySQL' package is also highlighted in blue. A second red arrow points from the 'PyMySQL' package row to the 'Tools' category in the sidebar.

Package	Version	Latest
Flask	0.12.1	➡ 0.12.2
Flask-Bootstrap	3.3.7.1	3.3.7.1
Flask-Bootstrap3	3.1.1.3	3.1.1.3
Flask-Login	0.4.0	0.4.0
Flask-Mail	0.9.1	0.9.1
Flask-Moment	0.5.1	0.5.1
<b>Flask-SQLAlchemy</b>	<b>2.2</b>	<b>2.2</b>
Flask-Script	2.0.5	2.0.5
Flask-Script-Extras	0.1.4	
Flask-WTF	0.14.2	0.14.2
Jinja2	2.9.6	2.9.6
MarkupSafe	1.0	1.0
Pillow	4.1.1	➡ 4.2.0
<b>PyMySQL</b>	<b>0.7.11</b>	<b>0.7.11</b>

# 链接数据库——代码

```
> from flask_sqlalchemy import SQLAlchemy
> from flask import Flask

> app = Flask(__name__)
> app.config['SECRET_KEY'] = 'hard to guess'
> app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymy
sql://root:123456@127.0.0.1:3306/flasky'
> app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True
> db = SQLAlchemy(app)
```



# 链接数据库——代码

- ◆ `app.config['SQLALCHEMY_DATABASE_URI']='mysql+pymysql://root:123456@127.0.0.1:3306/flasky'`
- ◆ `Mysql+pymysql://root:123456@127.0.0.1:3306/flasky`
- ◆ 用户名: root
- ◆ 密码: 123456
- ◆ IP地址: 127.0.0.1
- ◆ 端口: 3306
- ◆ 数据库名: flasky

# 创建数据表: roles

- ◆ `class Role(db.Model):`
- ◆ `__tablename__ = 'roles'`
- ◆ `id = db.Column(db.Integer, primary_key=True)`
- ◆ `name = db.Column(db.String(64), unique=True)`
- ◆ `user = db.relationship('User', backref='role')`
- ◆ `#repr()方法显示一个可读字符串`
- ◆ `def __repr__(self):`
- ◆ `return '<Role {}> '.format(self.name)`
- ◆ 关系: 用户属于一个角色, 一个角色对应多个用户

# 创建数据表: users

- ◆ `class User(db.Model):`
- ◆ `__tablename__ = 'users'`
- ◆ `id = db.Column(db.Integer, primary_key=True)`
- ◆ `username = db.Column(db.String(64),`  
`unique=True, index=True)`
- ◆ `role_id = db.Column(db.Integer,`  
`db.ForeignKey('roles.id'))`
- ◆ `def __repr__(self):`
- ◆ `return '<User {}>'.format(self.username)`

0	100	过滤
id	username	role_id
	<NULL>	<NULL>

# SQLAlchemy类型

表5-2 最常用的SQLAlchemy列类型

类型名	Python类型	说 明
Integer	int	普通整数，一般是 32 位
SmallInteger	int	取值范围小的整数，一般是 16 位
BigInteger	int 或 long	不限制精度的整数
Float	float	浮点数
Numeric	decimal.Decimal	定点数
String	str	变长字符串
Text	str	变长字符串，对较长或不限长度的字符串做了优化
Unicode	unicode	变长 Unicode 字符串
UnicodeText	unicode	变长 Unicode 字符串，对较长或不限长度的字符串做了优化
Boolean	bool	布尔值
Date	datetime.date	日期
Time	datetime.time	时间
DateTime	datetime.datetime	日期和时间
Interval	datetime.timedelta	时间间隔

# SQLAlchemy列选项

- › Flask-SQLAlchemy 要求每个模型都要定义主键，常命名为id

表5-3 最常使用的SQLAlchemy列选项

选项名	说 明
primary_key	如果设为 True，这列就是表的主键
unique	如果设为 True，这列不允许出现重复的值
index	如果设为 True，为这列创建索引，提升查询效率
nullable	如果设为 True，这列允许使用空值；如果设为 False，这列不允许使用空值
default	为这列定义默认值

# 执行数据库

```
◆ if __name__ == '__main__':  
◆     db.drop_all()  
◆     db.create_all()  
◆     app.run(debug=True)
```



# 增删改查：CRUD

描述软件系统中数据库或者持久层的基本操作功能。

Operation	SQL	HTTP	DDS
Create	INSERT	<a href="#">PUT</a> / <a href="#">POST</a>	write
Read (Retrieve)	SELECT	<a href="#">GET</a>	read / take
Update (Modify)	UPDATE	<a href="#">PUT</a> / <a href="#">POST</a> / <a href="#">PATCH</a>	write
Delete (Destroy)	DELETE	<a href="#">DELETE</a>	dispose

# 增删改查之增

create@CRUD

- ◆ `admin_role = Role(name='Admin')`
- ◆ `user_role = Role(name='User')`
- ◆ `user_cuc = User(username='cuc', role=admin_role)`
- ◆ `user_dviz = User(username='dviz', role=user_role)`
- ◆ `user_bigdata = User(username='bigdata', role=user_role)`
- ◆ `db.session.add_all([admin_role, user_role, user_cuc, user_dviz, user_bigdata])`
- ◆ `# 提交会话到数据库`
- ◆ `db.session.commit()`





- ◆ `admin_role.name = 'Administrator'`
- ◆ `db.session.add(admin_role)`
- ◆ `db.session.commit()`

id	name	
1	Administratc	
2	User	

# 增删改查之删

## Delete@CRUD

- ◆ # 删除数据库会话，从数据库中删除 “user”角色
- ◆ `db.session.delete(user_role)`
- ◆ `db.session.commit()`

0	100	过滤
id	name	
1	Administratc	

id	username	role_id	
1	cuc	1	
2	dviz	<NULL>	
3	bigdata	<NULL>	

```
> # 查询  
> print(user_role)  
> print(admin_role)  
> print(User.query.filter_by(role=user_role).all())
```

<Role User>

<Role Administrator>

[<User dviz>, <User bigdata>]

# 谢谢



## 期待您的佳作