

中国科学技术大学学生合唱团团员管理系统

PB15020603 蔡心宇

中国科学技术大学生合唱团成立于2005年，经过13年的发展，现已成为科大诸多社团中最为优秀的社团之一，连续5年获得“五星社团”。然而社团的团务工作十分繁杂，合唱团一直致力于将团务工作标准化，有利于减轻团务同学负担、团务工作的传承。其中招新与团员管理就是非常重要的一部分。本系统致力于构建一个web应用来实现合唱团运营团员管理的工作。

需求分析

1. 招新

合唱团招新主要分为两部分，一是每学年9月面对新生的集中招新面试，二是全年任何时间的单独的招新面试。但是面试报名需要填写线下纸质表单，面试成绩和是否通过面试要手动统计、审批，十分低效。

2. 团员管理

学生合唱团虽然只有团员不足百人，但是却又较为复杂的关系：

- 团员的姓名、性别、学号、学院等基本属性
- 根据业务结构分为指挥（C）、钢琴伴奏（P）、和女高音（S）、女低音（A）、男高音（T）、男低音（B）四个声部，每个声部内又分1、2（如S1和S2）每个声部有1-2个声部长，钢琴伴奏和同时又属于4个声部
- 根据团务结构可以分为六个部门：财物部、技术部、媒体部、事务部、外联部、宣传部

一直以来团员管理的都是手动在Excel中完成，较为低效，甚至是缺失的

设计目标

终极目标：

设计一个完善的网站，所有合唱团相关的内容（宣传、活动、录音、通知等）都可以在网站上发布
通过用户（团员注册成为用户）管理的方式管理团员

本系统实现目标：

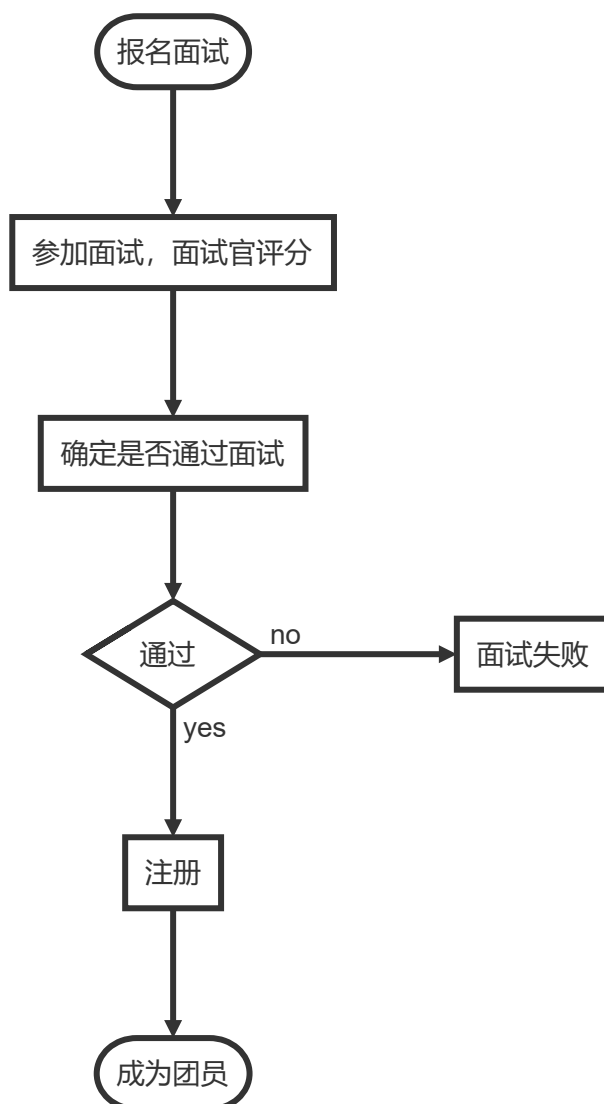
1. 面试

非团员在网站上报名面试

管理员在面试中通过本系统评分

管理员审核可以通过面试的面试者

面试者通过面试后可以在本站注册成为团员



2. 团员管理

网站中分为普通团员和管理员两种权限

团员视图：

- 查看&修改个人信息
- 查询团员信息

管理员视图：

- 团员视图
 - 修改团员信息（只能修改声部与部门不能修改个人信息）
 - 面试评分
 - 面试审核
-

关系数据库设计

概念结构设计

面试者属性 (Interviewer)

面试者学号-> (姓名、性别、院系、手机、是否通过)

面试属性 (Interview)

允许多次面试

面试者学号-> (面试时间、成绩)

团员属性 (Member)

学号-> (姓名、性别、院系、声部、部门、邮箱、手机、权限)

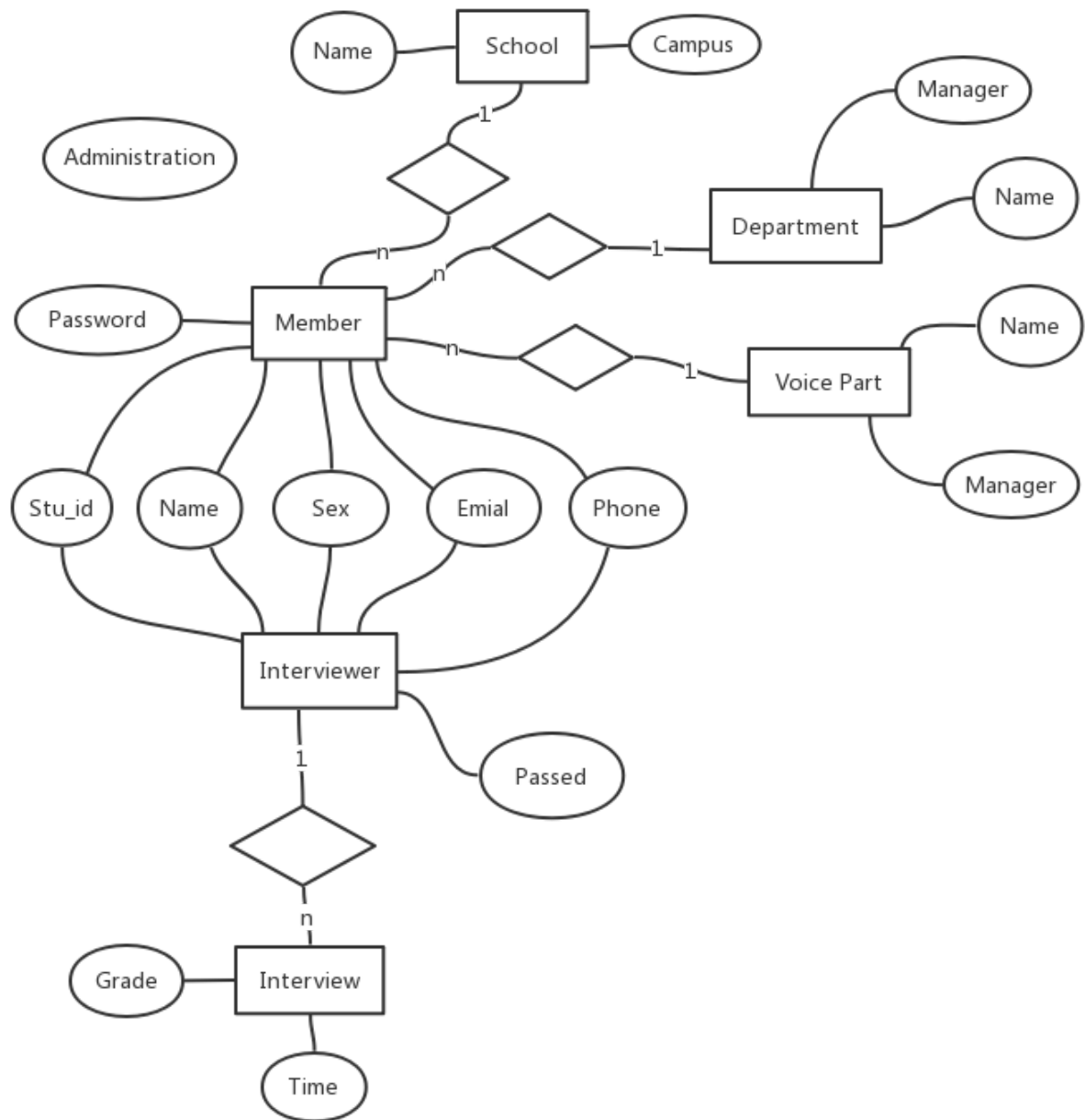
其他属性

部门->部长 (Department)

声部->声部长 (Voice Part)

学院->校区 (School)

E-R图



逻辑结构设计

建立数据库(www/static/sql/create_db.sql)

```

DROP DATABASE IF EXISTS chorus;

CREATE DATABASE chorus;

USE chorus;

GRANT SELECT, INSERT, UPDATE, DELETE ON awesome.* TO 'www-data'@'localhost' identified
BY 'www-data';

CREATE TABLE interviewers (

```

```

        `stu_id` VARCHAR(10) NOT NULL,
        `email` VARCHAR(50) NOT NULL,
        `name` VARCHAR(10) NOT NULL,
        `sex` VARCHAR(10) NOT NULL,
        `school` VARCHAR(30) NOT NULL,
        `phone` VARCHAR(20) NOT NULL,
        `passed` BOOL NOT NULL,
        `image` VARCHAR(500),
        `created_at` REAL NOT NULL,
        UNIQUE KEY `idx_email` (`email`),
        KEY `idx_created_at` (`created_at`),
        PRIMARY KEY (`stu_id`)
    ) ENGINE=innodb DEFAULT CHARSET=utf8;

```

```

CREATE TABLE interviews (
    `stu_id` VARCHAR(50) NOT NULL,
    `grade_1` INT,
    `grade_2` INT,
    `grade_3` INT,
    `grade_4` INT,
    `grade_5` INT,
    `extra` VARCHAR(500),
    `created_at` REAL NOT NULL,
    KEY `idx_created_at` (`created_at`),
    PRIMARY KEY (`stu_id`)
) ENGINE=innodb DEFAULT CHARSET=utf8;

```

```

CREATE TABLE members (
    `stu_id` VARCHAR(10) NOT NULL,
    `email` VARCHAR(50) NOT NULL,
    `passwd` VARCHAR(50) NOT NULL,
    `name` VARCHAR(10) NOT NULL,
    `sex` VARCHAR(10) NOT NULL,
    `school` VARCHAR(30) NOT NULL,
    `voice_part` VARCHAR(2) NOT NULL,
    `department` VARCHAR(10) NOT NULL,
    `phone` VARCHAR(20) NOT NULL,
    `admin` BOOL NOT NULL,
    `image` VARCHAR(500) NOT NULL,
    `created_at` REAL NOT NULL,
    UNIQUE KEY `idx_email` (`email`),
    KEY `idx_created_at` (`created_at`),
    PRIMARY KEY (`stu_id`)
) ENGINE=innodb DEFAULT CHARSET=utf8;

```

```

CREATE TABLE schools (
    `school` VARCHAR(30) NOT NULL,
    `campus` VARCHAR(10) NOT NULL,
    PRIMARY KEY (`school`)
) ENGINE=innodb DEFAULT CHARSET=utf8;

```

```

CREATE TABLE voice_parts (
    `voice_part` VARCHAR(2) NOT NULL,

```

```

    `vp_lead1` VARCHAR(10) NOT NULL,
    `vp_lead2` VARCHAR(10),
    PRIMARY KEY (`voice_part`)
) ENGINE=innodb DEFAULT CHARSET=utf8;

CREATE TABLE departments (
    `department` VARCHAR(10) NOT NULL,
    `dep_lead` VARCHAR(10) NOT NULL,
    PRIMARY KEY (`department`)
) ENGINE=innodb DEFAULT CHARSET=utf8;

```

数据库初始化(www/static/sql/init.sql)

```

INSERT INTO chorus.departments (department, dep_lead) VALUES ('外联部','蔡心宇'), ('财物部','吴宁谦'), ('宣传部','叶百家'), ('媒体部','白子逸'), ('技术部','白子逸'), ('事务部','林雪森')

INSERT INTO chorus.schools (school, campus) VALUES ('少年班学院','东校区'), ('数学科学学院','东校区'), ('物理学院','东校区'), ('管理学院','东校区'), ('化学与材料科学学院','东校区'), ('地球和空间科学学院','东校区'), ('人文与社会科学学院','东校区'), ('工程科学学院','西校区'), ('信息科学技术学院','西校区'), ('计算机科学与技术学院','西校区'), ('网络空间安全学院','西校区'), ('软件学院','西校区')

INSERT INTO chorus.voice_parts (voice_part, vp_lead1, vp_lead2) VALUES ('S', '李居龄', '曾嘉忻'), ('A', '时瑞', '韩江萍'), ('T', '白子逸', '陈淦斌'), ('B', '齐燕处', '叶百家')

INSERT INTO chorus.voice_parts (voice_part, vp_lead1) VALUES ('C', '钱泽华'), ('P', '钱泽华')

```

实现

本应用后端使用Python作为服务器后端语言、Mysql作为关系数据库管理系统搭建

前端使用Html、Javascript，其中用到了Js框架[Vue](#)

主要参考了[廖雪峰的python教程](#)、[w3school](#)、[老马的Vue教程](#)

搭建开发环境

首先，确认系统安装的Python版本是3.6.x：

```

$ python3 --version
Python 3.6.1

```

然后，用 `pip` 安装开发Web App需要的第三方库：

异步框架aiohttp：

```

$ pip3 install aiohttp

```

前端模板引擎jinja2：

```
$ pip3 install jinja2
```

MySQL数据库，从老师课程主页下载，设置root口令。

MySQL的Python异步驱动程序aiomysql:

```
$ pip3 install aiomysql
```

项目结构

选择一工作目录，然后，我们建立如下的目录结构：

```
USTChorusServer/ <-- 根目录
|
+- www/           <-- web目录，存放.py文件
| |
| +- static/      <-- 存放静态文件
| | |
| | +- css/       <-- 存放css库
| | |
| | +- fonts/     <-- 存放字体文件
| | |
| | +- img/       <-- 存放图片文件
| | |
| | +- js/        <-- 存放js库
| | |
| | +- sql/       <-- 存放建立数据库及测试用的sql文件
| |
| +- templates/   <-- 存放模板文件 (html)
```

封装DML(www/orm.py)

访问数据库需要创建数据库连接、游标对象，然后执行SQL语句，最后处理异常，清理资源。这些访问数据库的代码如果分散到各个函数中，势必无法维护，也不利于代码复用。

为了使用的方便性，首先把常用的SELECT、INSERT、UPDATE和DELETE操作封装起来。

Select

要执行SELECT语句，我们用 `select` 函数执行，需要传入SQL语句和SQL参数：

```
@asyncio.coroutine
def select(sql, args, size=None):
    log(sql, args)
    global __pool
```

```

with (yield from __pool) as conn:
    cur = yield from conn.cursor(aiomysql.DictCursor)
    yield from cur.execute(sql.replace('?', '%s'), args or ())
    if size:
        rs = yield from cur.fetchmany(size)
    else:
        rs = yield from cur.fetchall()
    yield from cur.close()
    logging.info('rows returned: %s' % len(rs))
    return rs

```

SQL语句的占位符是 `?`，而MySQL的占位符是 `%s`，`select()` 函数在内部自动替换。注意要始终坚持使用带参数的SQL，而不是自己拼接SQL字符串，这样可以防止SQL注入攻击。

注意到 `yield from` 将调用一个子协程（也就是在一个协程中调用另一个协程）并直接获得子协程的返回结果。

如果传入 `size` 参数，就通过 `fetchmany()` 获取最多指定数量的记录，否则，通过 `fetchall()` 获取所有记录。

Insert, Update, Delete

要执行INSERT、UPDATE、DELETE语句，可以定义一个通用的 `execute()` 函数，因为这3种SQL的执行都需要相同的参数，以及返回一个整数表示影响的行数：

```

@asyncio.coroutine
def execute(sql, args):
    log(sql)
    with (yield from __pool) as conn:
        try:
            cur = yield from conn.cursor()
            yield from cur.execute(sql.replace('?', '%s'), args)
            affected = cur.rowcount
            yield from cur.close()
        except BaseException as e:
            raise
    return affected

```

`execute()` 函数和 `select()` 函数所不同的是，`cursor`对象不返回结果集，而是通过 `rowcount` 返回结果数。

编写ORM

[ORM](#)对象关系映射(Object Relational Mapping)，是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。从效果上说，它其实是创建了一个可在编程语言里使用的--“虚拟对象数据库”。

定义用于映射数据库内数据类型的Field(www/orm.py)

例如 `StringField` 对应 `varchar`

```

class Field(object):

```



```

def __init__(self, name, column_type, primary_key, default):
    self.name = name
    self.column_type = column_type
    self.primary_key = primary_key
    self.default = default

def __str__(self):
    return '<%s, %s:%s>' % (self.__class__.__name__, self.column_type, self.name)

class StringField(Field):

    def __init__(self, name=None, primary_key=False, default=None, ddl='varchar(100)'):
        super().__init__(name, ddl, primary_key, default)

class BooleanField(Field):

    def __init__(self, name=None, default=False):
        super().__init__(name, 'boolean', False, default)

class IntegerField(Field):

    def __init__(self, name=None, primary_key=False, default=0):
        super().__init__(name, 'bigint', primary_key, default)

class FloatField(Field):

    def __init__(self, name=None, primary_key=False, default=0.0):
        super().__init__(name, 'real', primary_key, default)

class TextField(Field):

    def __init__(self, name=None, default=None):
        super().__init__(name, 'text', False, default)

```

定义与数据库中各个表对应的Models(www/models.py))

例如 `class Interviewers` 对应 `TABLE interviewers`

```

class Interviewers(Model):
    __table__ = 'interviewers'

    stu_id = StringField(primary_key=True, ddl='varchar(10)')
    email = StringField(ddl='varchar(50)')
    name = StringField(ddl='varchar(10)')
    sex = StringField(ddl='varchar(10)')
    school = StringField(ddl='varchar(30)')
    phone = StringField(ddl='varchar(20)')
    passed = BooleanField()
    image = StringField(ddl='varchar(500)')
    created_at = FloatField(default=time.time)

class Interviews(Model):
    __table__ = 'interviews'

```

```

stu_id = StringField(primary_key=True, ddl='varchar(10)')
created_at = FloatField(default=time.time)
grade_1 = IntegerField()
grade_2 = IntegerField()
grade_3 = IntegerField()
grade_4 = IntegerField()
grade_5 = IntegerField()
extra = StringField(ddl='varchar(500)')

class Members(Model):
    __table__ = 'members'

    stu_id = StringField(primary_key=True, ddl='varchar(50)')
    email = StringField(ddl='varchar(50)')
    passwd = StringField(ddl='varchar(50)')
    admin = BooleanField()
    name = StringField(ddl='varchar(50)')
    sex = StringField(ddl='varchar(10)')
    school = StringField(ddl='varchar(30)')
    voice_part = StringField(ddl='varchar(2)')
    department = StringField(ddl='varchar(10)')
    phone = StringField(ddl='varchar(20)')
    image = StringField(ddl='varchar(500)')
    created_at = FloatField(default=time.time)

class Schools(Model):
    __table__ = 'schools'

    school = StringField(primary_key=True, ddl='varchar(30)')
    campus = StringField(ddl='varchar(10)')

class Voice_parts(Model):
    __table__ = 'voice_parts'

    voice_part = StringField(primary_key=True, ddl='varchar(2)')
    vp_lead1 = StringField(ddl='varchar(10)')
    vp_lead2 = StringField(ddl='varchar(10)')

class Departments(Model):
    __table__ = 'departments'

    department = StringField(primary_key=True, ddl='varchar(10)')
    dep_lead = StringField(ddl='varchar(10)')

```

这些对象都继承自 `Model`，其定义在 `orm.py` 中

```

class ModelMetaclass(type):

    def __new__(cls, name, bases, attrs):
        if name == 'Model':
            return type.__new__(cls, name, bases, attrs)
        tableName = attrs.get('__table__', None) or name

```

```

logging.info('found model: %s (table: %s)' % (name, tableName))
mappings = dict()
fields = []
primaryKey = None
for k, v in attrs.items():
    if isinstance(v, Field):
        logging.info('  found mapping: %s ==> %s' % (k, v))
        mappings[k] = v
        if v.primary_key:
            # 找到主键:
            if primaryKey:
                raise StandardError('Duplicate primary key for field: %s' % k)
            primaryKey = k
        else:
            fields.append(k)
if not primaryKey:
    raise StandardError('Primary key not found.')
for k in mappings.keys():
    attrs.pop(k)
escaped_fields = list(map(lambda f: '`%s`' % f, fields))
attrs['__mappings__'] = mappings # 保存属性和列的映射关系
attrs['__table__'] = tableName
attrs['__primary_key__'] = primaryKey # 主键属性名
attrs['__fields__'] = fields # 除主键外的属性名
attrs['__select__'] = 'select `%s`, %s from `%s`' % (primaryKey, ',
'.join(escaped_fields), tableName)
attrs['__insert__'] = 'insert into `%s` (%s, `%s`) values (%s)' % (tableName,
', '.join(escaped_fields), primaryKey, create_args_string(len(escaped_fields) + 1))
attrs['__update__'] = 'update `%s` set %s where `%s`=?' % (tableName, ',
'.join(map(lambda f: '`%s`=?' % (mappings.get(f).name or f), fields)), primaryKey)
attrs['__delete__'] = 'delete from `%s` where `%s`=?' % (tableName, primaryKey)
return type.__new__(cls, name, bases, attrs)

class Model(dict, metaclass=ModelMetaclass):

    def __init__(self, **kw):
        super(Model, self).__init__(**kw)

    def __getattr__(self, key):
        try:
            return self[key]
        except KeyError:
            raise AttributeError(r"'Model' object has no attribute '%s'" % key)

    def __setattr__(self, key, value):
        self[key] = value

    def getValue(self, key):
        return getattr(self, key, None)

    def getValueOrDefault(self, key):
        value = getattr(self, key, None)
        if value is None:

```

```

        field = self.__mappings__[key]
        if field.default is not None:
            value = field.default() if callable(field.default) else field.default
            logging.debug('using default value for %s: %s' % (key, str(value)))
            setattr(self, key, value)
        return value

    @classmethod
    @asyncio.coroutine
    def findAll(cls, where=None, args=None, **kw):
        ' find objects by where clause. '
        sql = [cls.__select__]
        if where:
            sql.append('where')
            sql.append(where)
        if args is None:
            args = []
        orderBy = kw.get('orderBy', None)
        if orderBy:
            sql.append('order by')
            sql.append(orderBy)
        limit = kw.get('limit', None)
        if limit is not None:
            sql.append('limit')
            if isinstance(limit, int):
                sql.append('?')
                args.append(limit)
            elif isinstance(limit, tuple) and len(limit) == 2:
                sql.append('?, ?')
                args.extend(limit)
            else:
                raise ValueError('Invalid limit value: %s' % str(limit))
        rs = yield from select(' '.join(sql), args)
        return [cls(**r) for r in rs]

    @classmethod
    @asyncio.coroutine
    def findNumber(cls, selectField, where=None, args=None):
        ' find number by select and where. '
        sql = ['select %s _num_ from `%s`' % (selectField, cls.__table__)]
        if where:
            sql.append('where')
            sql.append(where)
        rs = yield from select(' '.join(sql), args, 1)
        if len(rs) == 0:
            return None
        return rs[0]['_num_']

    @classmethod
    @asyncio.coroutine
    def find(cls, pk):
        ' find object by primary key. '

```

```

        rs = yield from select('%s where `%s`=?' % (cls.__select__,
cls.__primary_key__), [pk], 1)
        if len(rs) == 0:
            return None
        return cls(**rs[0])

    @asyncio.coroutine
    def save(self):
        args = list(map(self.getValueOrDefault, self.__fields__))
        args.append(self.getValueOrDefault(self.__primary_key__))
        rows = yield from execute(self.__insert__, args)
        if rows != 1:
            logging.warn('failed to insert record: affected rows: %s' % rows)

    @asyncio.coroutine
    def update(self):
        args = list(map(self.getValue, self.__fields__))
        args.append(self.getValue(self.__primary_key__))
        rows = yield from execute(self.__update__, args)
        if rows != 1:
            logging.warn('failed to update by primary key: affected rows: %s' % rows)

    @asyncio.coroutine
    def remove(self):
        args = [self.getValue(self.__primary_key__)]
        rows = yield from execute(self.__delete__, args)
        if rows != 1:
            logging.warn('failed to remove by primary key: affected rows: %s' % rows)

```

Model 对象中定义了用于在本表中定义了 findAll findNumber find 用于查询, save 用于插入, update 用于更新, remove 用于删除

在本系统中, 并没用直接用SQL语言进行连接查询或者子查询; 而是在主语言Python中使用上述函数实现相同功能, 在后文中可以看到清晰的例子

编写server骨架

使用 aiohttp 构建一个web app

使用 asyncio 异步IO模块建立一个事件循环, 并把web app放进去, 异步处理http请求的后端骨架就搭好了 (www/app.py)

```

import asyncio, os, json, time
from datetime import datetime

from aiohttp import web

@asyncio.coroutine
def init(loop):
    yield from orm.create_pool(loop=loop, **configs.db)
    app = web.Application(loop=loop, middlewares=[

```

```

        logger_factory, auth_factory, response_factory
    ])
    init_jinja2(app, filters=dict(datetime=datetime_filter))
    add_routes(app, 'handlers')
    add_static(app)
    srv = yield from loop.create_server(app.make_handler(), '127.0.0.1', 9000)
    logging.info('server started at http://127.0.0.1:9000...')
    return srv

loop = asyncio.get_event_loop()
loop.run_until_complete(init(loop))
loop.run_forever()

```

编写web框架

因为 `aiohttp` 相对比较底层，所以还需要将其封装一下，目的是为了在后期实现各种后端处理函数时可以减少所编写的代码

`aiohttp` 处理一个URL需要：

1. 从 `request` 中提取参数
2. 使用URL处理函数处理
3. 构造 `web.Response` 返回

首先说明第二部分的实现

定义@get和@post装饰器

Http请求分 `get` 和 `post` 两种方法，用户发来的http请求中，包含请求的URL和方法，所以定义两种方法的装饰器，之后再编写请求处理函数时用 `@get` 和 `@post` 装饰即可

```

def get(path):
    """
    Define decorator @get('/path')
    """
    def decorator(func):
        @functools.wraps(func)
        def wrapper(*args, **kw):
            return func(*args, **kw)
        wrapper.__method__ = 'GET'
        wrapper.__route__ = path
        return wrapper
    return decorator

def post(path):
    """
    Define decorator @post('/path')
    """
    def decorator(func):
        @functools.wraps(func)
        def wrapper(*args, **kw):

```

```

        return func(*args, **kw)
    wrapper.__method__ = 'POST'
    wrapper.__route__ = path
    return wrapper
return decorator

```

例如返回主页的URL处理函数：

```

@get('/')
def index(request):
    return {
        '__template__': 'index.html'
    }

```

对于不同URL和方法的请求，只需按照上面的形式添加即可

定义RequestHandler

接下来第一部分——如何获取参数。

用 `RequestHandler()` 来封装一个URL处理函数。

`RequestHandler` 是一个类，由于定义了 `__call__()` 方法，因此可以将其实例视为函数。

`RequestHandler` 目的就是从URL函数中分析其需要接收的参数，从 `request` 中获取必要的参数，调用URL函数，然后把结果转换为 `web.Response` 对象，这样，就完全符合 `aiohttp` 框架的要求：

```

class RequestHandler(object):

    def __init__(self, app, fn):
        self._app = app
        self._func = fn
        ...

    @asyncio.coroutine
    def __call__(self, request):
        kw = ... 获取参数
        r = yield from self._func(**kw)
        return r

```

在 `__call__` 中首先从request中获取参数，然后调用 `self._func`，即上文中讲的使用 `@get` 和 `@post` 装饰的URL处理函数，这里需要注意的是，每个由 `@get` 和 `@post` 装饰的URL处理函数对应一个 `RequestHandler` 实例。

使用middlewares

在 `app.py` 中有

```
app = web.Application(loop=loop, middlewares=[
    logger_factory, auth_factory, response_factory
])
```

`logger_factory`、`response_factory` 和 `auth_factory`，定义如下：

```
@asyncio.coroutine
def logger_factory(app, handler):
    @asyncio.coroutine
    def logger(request):
        logging.info('Request: %s %s' % (request.method, request.path))
        return (yield from handler(request))
    return logger

@asyncio.coroutine
def auth_factory(app, handler):
    @asyncio.coroutine
    def auth(request):
        logging.info('check user: %s %s' % (request.method, request.path))
        request.__user__ = None
        cookie_str = request.cookies.get(COOKIE_NAME)
        if cookie_str:
            user = yield from cookie2user(cookie_str)
            if user:
                logging.info('set current user: %s' % user.email)
                request.__user__ = user
        if request.path.startswith('/user/') and (request.__user__ is None):
            return web.HTTPFound('/signin')
        if request.path.startswith('/manage/') and (request.__user__ is None or not
request.__user__.admin):
            return web.HTTPFound('/signin')
        return (yield from handler(request))
    return auth

@asyncio.coroutine
def response_factory(app, handler):
    @asyncio.coroutine
    def response(request):
        logging.info('Response handler...')
        r = yield from handler(request)

        ...

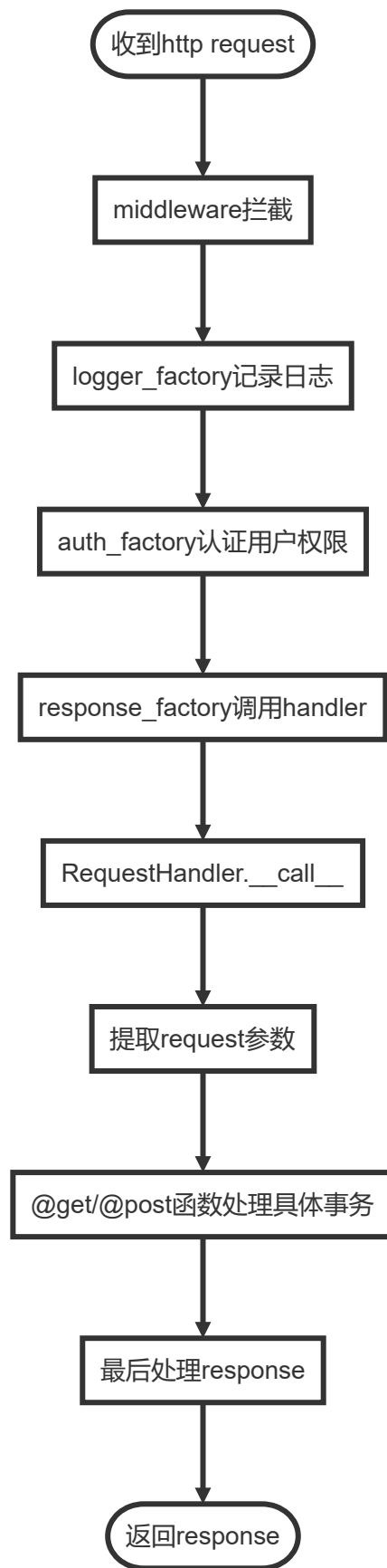
        resp = web.Response(body=str(r).encode('utf-8'))
        resp.content_type = 'text/plain; charset=utf-8'
        return resp
    return response
```

`middlewares` 是 `aiohttp` 中提供的拦截器，一个URL在被某个函数处理前，可以经过一系列的 `middleware` 的处理。

一个 `middleware` 可以改变URL的输入、输出，甚至可以决定不继续处理而直接返回。`middleware` 的用处就在于把通用的功能从每个URL处理函数中拿出来，集中放到一个地方。

`logger_factory` 是用来记录日志，`auth_factory` 会使用 `cookie` 来认证用户权限，这部分将在后面提到
`response_factory` 是用来处理后端发给浏览器的http响应。

上述处理过程比较复杂，难以理解，下边的流程图可以更形象地描绘函数调用关系



至此后端处理http request并返回response框架已经完成

使用MVC编写前端

在网站的不同页面中，顶部和底部都是相同的，如果每个 `html` 文件中都写重复的内容势必非常麻烦，而且修改起来也很繁琐。

这个过程中必不可少的就是模板引擎 [jinja2](#)

首先编写一个 `__base__.html`

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  {% block meta %}<!-- block meta -->{% endblock %}

  ...

  {% block beforehead %}<!-- before head -->{% endblock %}
</head>
<body>

  ...

  <div class="uk-container uk-container-center">
    <div class="uk-grid">
      <!-- content -->
      {% block content %}
      {% endblock %}
      <!-- // content -->
    </div>
  </div>

  ...

</body>
</html>
```

使用 `{% block xxx %}...{% endblock %}` 来标识模板中可以填充的部分，也就是不同的部分。

在其他网页中，例如 `index.html`，利用同样的语法在 `__base__.html` 中插入每个页面不同的内容

```
{% extends '__base__.html' %}

{% block title %}学生合唱团{% endblock %}

{% block beforehead %}

<script>
</script>

{% endblock %}

{% block content %}
```

```
<div class="uk-width-medium-3-4">
  <article class="uk-article">
    <h2>学生合唱团</h2>

    <p>合唱团的宗旨是弘扬合唱艺术、丰富学生课余生活，提高学生的综合素质与文化修
      养，增强集体荣誉感，以及通过对外交流弘扬学校的人文精神。</p>
  </article>
  <hr class="uk-article-divider">
</div>

...

{% endblock %}
```

学生合唱团 - Awesome Python

127.0.0.1:9000

应用CSUSTC CoursesUbuntu and Free S... 微信网页版 网络通 社团活动及会议报备 Gmail cxyvlf (Xinyu Cai) pydota2/pydota2: 中国科学技术大学

中国科学技术大学学生合唱团 微信公众号 网易云音乐 登陆 报名

学生合唱团

合唱团的宗旨是弘扬合唱艺术、丰富学生课余生活，提高学生的综合素质与文化修 养，增强集体荣誉感，以及通过对外交流弘扬学校的人文精神。

了解我们

[音乐会](#)


[组织架构](#)

[合唱交流](#)

[比赛](#)

Powered by [USTC Student Chorus](#). Copyright © 2014. [\[Manage\]](#)

[github.com/cxyvlf](#). All rights reserved.



面试报名 - Awesome Python V. x

127.0.0.1:9000/signup

应用 CS USTC Courses Ubuntu and Free S... 微信网页版 网络通 社团活动及会议报备 Gmail cxyvlf (Xinyu Cai) pydota2/pydota2:1 中国科学技术大学

中国科学技术大学学生合唱团 微信公众号 网易云音乐 登陆 报名

欢迎报名学生合唱团!

姓名:

名字

学号:

PB/SA/SC/BA

性别:

男

女

电子邮件:

your-name@mail.ustc.edu.cn

手机:

18923415342

学院:

少年班学院

数学科学学院

物理学院

管理学院

化学与材料科学学院

地球和空间科学学院

人文与社会科学学院

工程科学学院

除此之外，更重要的是，后端URL事务处理函数，和前端网页显示采用MVC框架编写，将逻辑、数据、界面显示分离，最大程度减少代码的复杂性。

URL处理函数就是C：Controller，Controller负责业务逻辑，比如检查用户名是否存在，取出用户信息等等；

包含变量 `{{ name }}` 的模板就是V：View，View负责显示逻辑，通过简单地替换一些变量，View最终输出的就是用户看到的HTML。

M：Model是用来传给View的，这样View在替换变量的时候，就可以从Model中取出相应的数据。

`jinja2` 可以在后端根据参数来渲染 `html`

例如在 `index.html` 中

`{% if __user__ %}` 判断，如果用户登录后会显示相应内容；

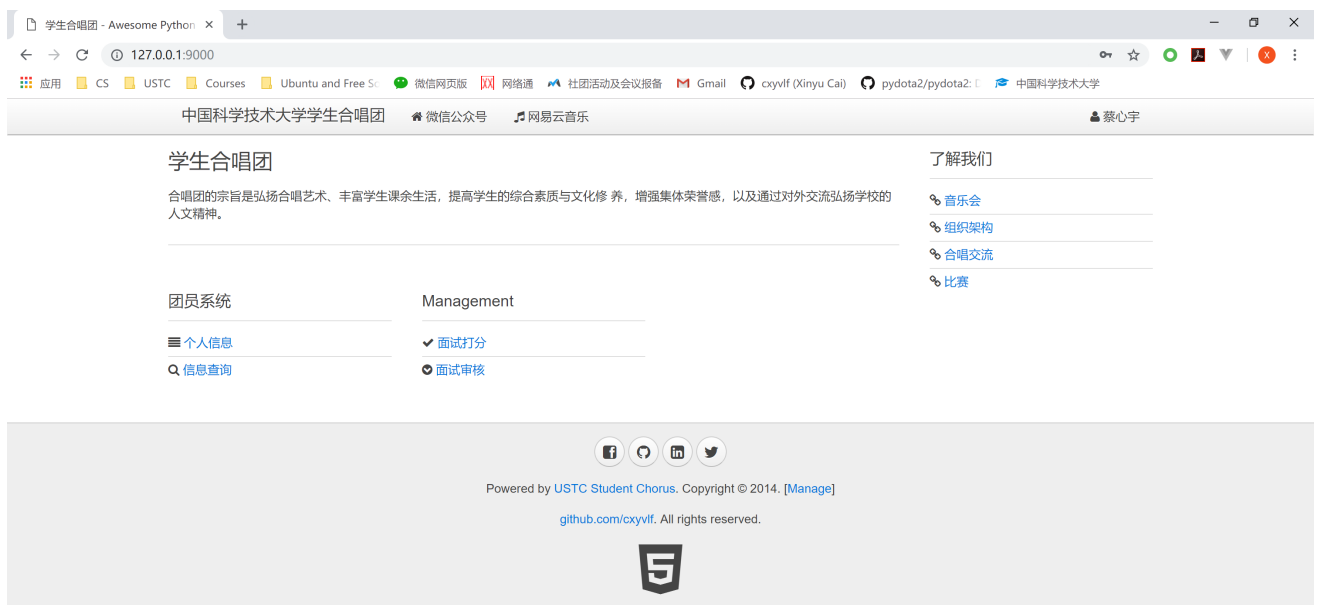
`{% if __user__.admin %}` 判断，如果用户有管理员权限则显示相应内容

```
{% if __user__ %}
<div class="uk-width-medium-1-4">
  <div class="uk-panel uk-panel-header">
    <h3 class="uk-panel-title">团员系统</h3>
    <ul class="uk-list uk-list-line">
      <li><i class="uk-icon-align-justify"></i> <a target="_blank"
href="/user/personal">个人信息</a></li>
      <li><i class="uk-icon-search"></i> <a target="_blank" href="/user/enquiry">
信息查询</a></li>
    </ul>
  </div>
</div>
```

```

</div>
{% endif %}
{% if __user__.admin %}
<div class="uk-width-medium-1-4">
  <div class="uk-panel uk-panel-header">
    <h3 class="uk-panel-title">Management</h3>
    <ul class="uk-list uk-list-line">
      <li><i class="uk-icon-check"></i> <a target="_blank"
href="/manage/interview">面试打分</a></li>
      <li><i class="uk-icon-chevron-circle-down"></i> <a target="_blank"
href="/manage/interview/select">面试审核</a></li>
    </ul>
  </div>
</div>
{% endif %}

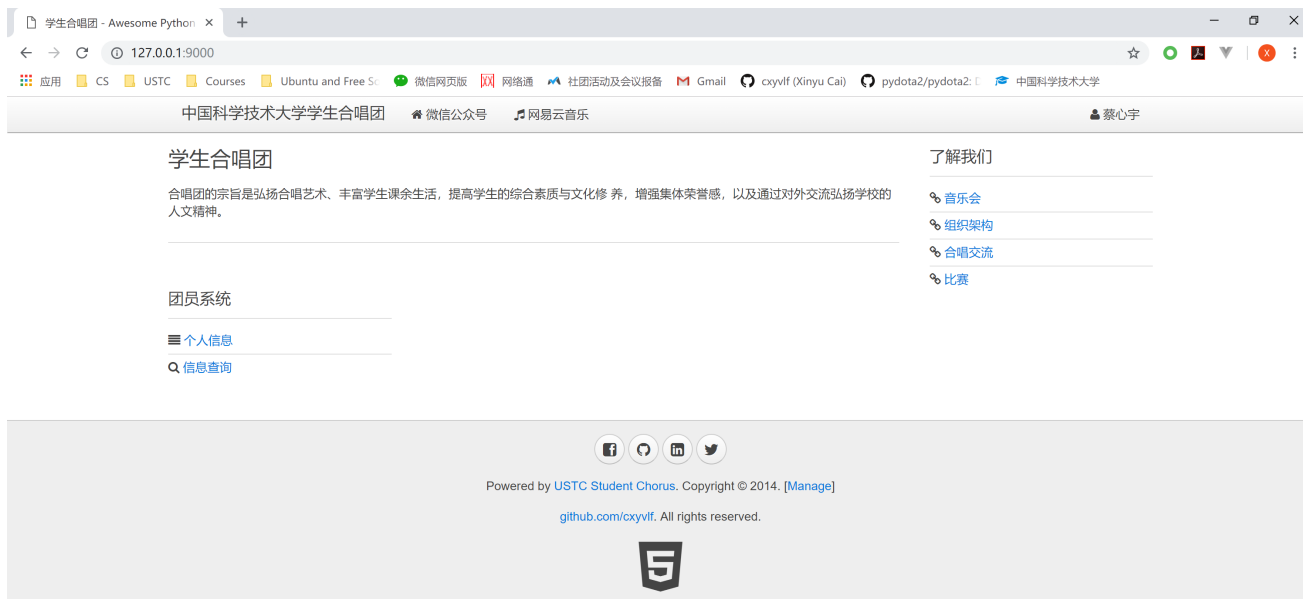
```



我的账户是有管理员权限的，所以可以看到面试相关内容，如果如果我执行SQL语句

```
update chorus.members set admin = 0 where stu_id = 'PB15020603'
```

刷新一下页面就没有面试相关内容了



在个人信息和信息查询页面也有相应的判定来取分管理员和普通用户的权限

接下来是各个具体功能的实现

各个功能往往需要从后端请求数据然后显示在网页中，这与前面提到的用 `jinja2` 做后端渲染不同。经过服务器端渲染的网页一旦被发送到客户端后是不会变的（除非你点一下刷新，这样会重新请求网页），而我们的许多功能都需要在网页不变的情况下，根据数据的变化动态地渲染网页。这就需要引入MVVM。

[MVVM](#)：Model View ViewModel

最早由微软提出来，它借鉴了桌面应用程序的MVC思想，在前端页面中，把Model用纯JavaScript对象表示：

```
<script>
  data: {
    name: '',
    email: '',
    stu_id: '',
    sex: '',
    school: '',
    phone: ''
  },
</script>
```

View是纯HTML：

```

<div class="uk-form-controls">
<input v-model="sex" type="radio" value="男" checked class="uk-width-1-1">男
<br>
<input v-model="sex" type="radio" value="女" class="uk-width-1-1">女

```

由于Model表示数据，View负责显示，两者做到了最大限度的分离。

把Model和View关联起来的的就是ViewModel。ViewModel负责把Model的数据同步到View显示出来，还负责把View的修改同步回Model。

这采用的ViewModel是前端框架[Vue](#)

所以实现一个特定功能需要做的工作有

- View：网页html的编写
- Model：网页交互所需的数据，包含在Vue实例中
- ViewModel：这是一个Vue实例，除了Model外，还包含各种对前端事件的处理函数（例如按特定按钮触发的功能）
- get或post方法从服务器请求或提交数据（例如在查询功能中，要使用post方法将查询条件提交给服务器，并显示服务器发回的查询结果）
- 在服务器端编写对应的URL处理函数，处理相应请求，返回前端所需数据（例如在查询功能中，服务器端根据前端提供的条件去数据库查询，并把查询结果返回）

报名面试

- 网页为一个表单 `form` 并于Vue实例 `vm` 绑定，提交按钮与实例中 `submit` 方法绑定

signup.html

```

<div class="uk-width-2-3">
  <h1>欢迎报名学生合唱团! </h1>
  <form id="vm" v-on="submit: submit" class="uk-form uk-form-stacked">
    <div class="uk-alert uk-alert-danger uk-hidden"></div>
    <div class="uk-form-row">
      <label class="uk-form-label">姓名:</label>
      <div class="uk-form-controls">
        <input v-model="name" type="text" maxlength="50" placeholder="名字"
class="uk-width-1-1">
      </div>
    </div>
    <div class="uk-form-row">
      <label class="uk-form-label">学号:</label>
      ...
    </div>
    <div class="uk-form-row">
      <label class="uk-form-label">性别:</label>
      ...
    </div>
    <div class="uk-form-row">
      <label class="uk-form-label">电子邮件:</label>

```



```

    ...
  </div>
  <div class="uk-form-row">
    <label class="uk-form-label">手机:</label>
    ...
  <div class="uk-form-row">
    <label class="uk-form-label">学院:</label>
    ...
    <button type="submit" class="uk-button uk-button-primary"><i class="uk-
icon-user"></i> 注册</button>
  </div>
</form>
</div>

```

- 构建Vue实例 `vm`，其中定义了数据 `data` 和 `submit` 方法
- 在 `submit` 方法中，首先对输入的格式进行检查，然后将数据通过post方法提交到 `/api/signup`

```

var vm = new Vue({
  el: '#vm',
  data: {
    name: '',
    email: '',
    stu_id: '',
    sex: '',
    school: '',
    phone: ''
  },
  methods: {
    submit: function (event) {
      event.preventDefault();
      var $form = $('#vm');
      if (! this.name.trim()) {
        return $form.showFormError('请输入名字');
      }
      if (! this.stu_id.trim()) {
        return $form.showFormError('请输入学号');
      }
      if (! validateEmail(this.email.trim().toLowerCase())) {
        return $form.showFormError('请输入正确的Email地址');
      }
      if (! this.phone.trim()) {
        return $form.showFormError('请输入学号');
      }
      var email = this.email.trim().toLowerCase();
      $form.postJSON('/api/signup', {
        name: this.name.trim(),
        email: email,
        stu_id: this.stu_id.trim(),
        sex: this.sex,
        school: this.school,
        phone: this.phone.trim()
      }, function (err, r) {

```

```

        if (err) {
            return $form.showFormError(err);
        }
        alert('报名成功!');
        return location.assign('/');
    });
}
}
});

```

- 最后编写后端一个是返回报名网页的函数，另一个是处理提交请求的函数
 - 为了安全性，防止攻击者伪造请求，后端需要再次检查数据
 - 检查是否已经报名过，是则报错
 - 使用前端传来的参数创建一个 Interviewers 实例
 - 使用 save 方法将该实例存入数据库

```

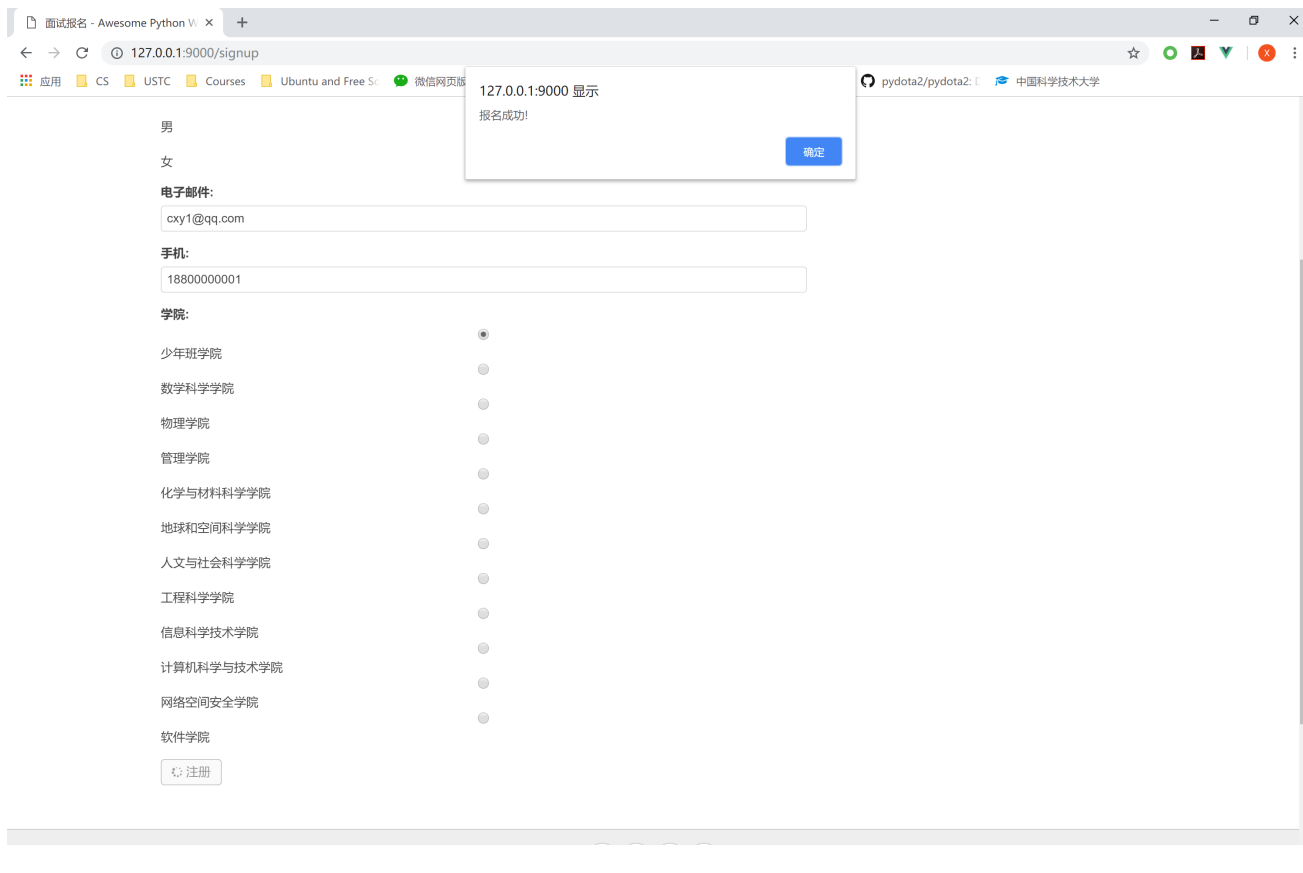
@get('/signup')
def signup():
    return {
        '__template__': 'signup.html'
    }

@post('/api/signup')
def api_signup(*, email, name, stu_id, sex, school, phone):
    if not name or not name.strip():
        raise APIValueError('name')
    if not stu_id or not stu_id.strip():
        raise APIValueError('stu_id')
    if not email or not _RE_EMAIL.match(email):
        raise APIValueError('email')
    interviewers = yield from Interviewers.findAll('stu_id=?', [stu_id])
    if len(interviewers) > 0:
        raise APIError('signup:failed', 'email', 'Email is already in use.')

    interviewer = Interviewers(stu_id=stu_id.strip(), name=name.strip(), email=email,
sex=sex, school=school, phone=phone, image='http://www.gravatar.com/avatar/%s?
d=mm&s=120' % hashlib.md5(email.encode('utf-8')).hexdigest())
    yield from interviewer.save()

    r = json.dumps(interviewer, ensure_ascii=False).encode('utf-8')
    return r

```



用户注册与登录

接下来的部分都是用户登录后才能使用的功能，首先来实现应用中最核心的用户认证功能

首先实现最为简单的用户注册

这部分与上面报名十分相似

register.html

- 一个表单和一个提交按钮
- 前端检查用户输入
- 通过post方法提交给后端（用户密码是经过SHA1计算后的40位Hash字符串，所以服务器端并不知道用户的原始口令）

```
$form.postJSON('/api/users', {
    email: email,
    passwd: CryptoJS.SHA1(email + ':' + this.password1).toString()
})
```

- 后端处理也类似，但是注意存在数据库中的密码经过了如下处理（这是为了实现用户认证，后面会提到）

```
sha1_passwd = '%s:%s' % (stu_id, passwd)
member = Members(..., passwd=hashlib.sha1(sha1_passwd.encode('utf-8')).hexdigest(),...)
yield from member.save()
```

用户登录

用户登录比用户注册复杂。由于HTTP协议是一种无状态协议，而服务器要跟踪用户状态，就只能通过cookie实现。

我们采用直接读取cookie的方式来验证用户登录，每次用户访问任意URL，都会对cookie进行验证。

由于登录成功后是由服务器生成一个cookie发送给浏览器，所以，要保证这个cookie不会被客户端伪造出来。

实现防伪造cookie的关键是通过一个单向算法（例如SHA1），举例如下：

当用户输入了正确的口令登录成功后，服务器可以从数据库取到用户的id，并按照如下方式计算出一个字符串：

```
"用户id" + "过期时间" + SHA1("用户id" + "用户口令" + "过期时间" + "SecretKey")
```

当浏览器发送cookie到服务器端后，服务器可以拿到的信息包括：

- 用户id
- 过期时间
- SHA1值

如果未到过期时间，服务器就根据用户id查找用户口令，并计算：

```
SHA1("用户id" + "用户口令" + "过期时间" + "SecretKey")
```

并与浏览器cookie中的哈希进行比较，如果相等，则说明用户已登录，否则，cookie就是伪造的。

```
def user2cookie(user, max_age):
    """
    Generate cookie str by user.
    """
    # build cookie string by: id-expires-sha1
    expires = str(int(time.time() + max_age))
    s = '%s-%s-%s-%s' % (user.stu_id, user.passwd, expires, _COOKIE_KEY)
    L = [user.stu_id, expires, hashlib.sha1(s.encode('utf-8')).hexdigest()]
    return '-'.join(L)

@post('/api/authenticate')
def authenticate(*, email, passwd):
    if not email:
        raise APIValueError('email', 'Invalid email.')
    if not passwd:
        raise APIValueError('passwd', 'Invalid password.')
    users = yield from Members.findAll('email=?', [email])
    if len(users) == 0:
        raise APIValueError('email', 'Email not exist.')
    user = users[0]
    # check passwd:
    sha1 = hashlib.sha1()
    sha1.update(user.stu_id.encode('utf-8'))
    sha1.update(b':')
    sha1.update(passwd.encode('utf-8'))
```

```

if user.passwd != sha1.hexdigest():
    raise APIValueError('passwd', 'Invalid password.')
# authenticate ok, set cookie:
r = web.Response()
r.set_cookie(COOKIE_NAME, user2cookie(user, 86400), max_age=86400, httponly=True)
user.passwd = '*****'
r.content_type = 'application/json'
r.body = json.dumps(user, ensure_ascii=False).encode('utf-8')
return r

```

对于每个URL处理函数，如果我们都去写解析cookie的代码，那会导致代码重复很多次。

利用 `middleware` 在处理URL之前，把cookie解析出来，并将登录用户绑定到 `request` 对象上，这样，后续的URL处理函数就可以直接拿到登录用户。

也就是前面提到的 `auth_factory`，会对每个URL请求验证用户是否登陆和是否为管理员

以 `/user/` 开头的URL请求只能由用户发起

以 `/manage/` 开头的URL请求只能由管理员发起

```

@asyncio.coroutine
def cookie2user(cookie_str):
    """
    Parse cookie and load user if cookie is valid.
    """
    if not cookie_str:
        return None
    try:
        L = cookie_str.split('-')
        if len(L) != 3:
            return None
        stu_id, expires, sha1 = L
        if int(expires) < time.time():
            return None
        user = yield from Members.find(stu_id)
        if user is None:
            return None
        s = '%s-%s-%s-%s' % (stu_id, user.passwd, expires, _COOKIE_KEY)
        if sha1 != hashlib.sha1(s.encode('utf-8')).hexdigest():
            logging.info('invalid sha1')
            return None
        user.passwd = '*****'
        return user
    except Exception as e:
        logging.exception(e)
        return None

@asyncio.coroutine
def auth_factory(app, handler):
    @asyncio.coroutine
    def auth(request):
        logging.info('check user: %s %s' % (request.method, request.path))
        request.__user__ = None

```

```

cookie_str = request.cookies.get(COOKIE_NAME)
if cookie_str:
    user = yield from cookie2user(cookie_str)
    if user:
        logging.info('set current user: %s' % user.email)
        request.__user__ = user
if request.path.startswith('/user/') and (request.__user__ is None):
    return web.HTTPFound('/signin')
if request.path.startswith('/manage/') and (request.__user__ is None or not
request.__user__.admin):
    return web.HTTPFound('/signin')
return (yield from handler(request))
return auth

```

面试评分&审核

评分

面试评分也为一个表单，和前面类似，不再赘述，可参考 `interview.html` 和 `@get('/manage/interview')`、`@post('/manage/api/interview_grade')`

面试成绩

学号:
PB15000001

音高感1:
10

音高感2:
10

节奏感1:
10

节奏感2:
10

自由展示:
10

综合调查:
QWE

更新

Powered by USTC Student Chorus. Copyright © 2014. [Manage]
github.com/cxyvf. All rights reserved.

审核

面试审核与前面不同，实现这个功能需要将所有面试信息显示在网页中



@get('/manage/interview/select') 用于返回适合页面

```
@get('/manage/interview/select')
def interview_select():
    return {
        '__template__': 'interview_select.html',
    }
```

用HTML的 `table` 显示每一条面试信息

那么这些数据从哪里来的呢？答案是通过Vue的 `mounted` 方法，`mounted` 定义的函数会在Vue实例初始化的过程中被调用，这个函数里实现了向后端请求数据的功能

对于每条记录，可以通过勾选是/否来决定面试是否通过

最后通过提交按钮调用 `submit` 将更改的数据通过post方法提交给后端

```
<div id='app' class="uk-width-2-3">
  <h1>面试审核</h1>
  <table class="uk-table">
    <thead>
      <tr>
        <th>学号</th>
        <th>性别</th>
        <th>音高感1</th>
        <th>音高感2</th>
        <th>节奏感1</th>
        <th>节奏感2</th>
        <th>自由展示</th>
      </tr>
    </thead>
  </table>
</div>
```

```

        <th>状态</th>
        <th>是</th>
        <th>否</th>
    </tr>
</thead>
<tfoot>
    <tr v-for="item in interviews">
        <td>[[ item.stu_id ]]</td>
        <td>[[ item.sex ]]</td>
        <td>[[ item.grade_1 ]]</td>
        <td>[[ item.grade_2 ]]</td>
        <td>[[ item.grade_3 ]]</td>
        <td>[[ item.grade_4 ]]</td>
        <td>[[ item.grade_5 ]]</td>
        <td>[[ item.passed ]]</td>
        <td><input v-model="item.passed" type="radio" value=true class="uk-
width-1-1"></td>
        <td><input v-model="item.passed" type="radio" value=false
class="uk-width-1-1"></td>
    </tr>
</tfoot>
</table>
<button v-on:click="submit" class="uk-button uk-button-primary">提交</button>
</div>
<script>
var app = new Vue({
    el: '#app',
    delimiters: ['[[', ']]'],
    data: {
        interviews: [],
    },
    methods: {
        submit: function(event){
            this.$http.post('/manage/api/submit_interview_result', {'interviews':
this.interviews}).then(
                function(res){
                    this.interviews = res.body;
                    alert("更新成功! ");
                },
                function(){
                    console.log('请求失败处理');
                }
            )
        },
    },
    mounted: function () {
        this.$http.get('/manage/api/get_interviews').then(
            function(res){
                this.interviews = res.body;
            },
            function(){
                console.log('请求失败处理');
            }
        )
    }
})

```



```

    );
  },
})
</script>

```

后端的 `@get('/manage/api/get_interviews')` 和 `@post('/manage/api/submit_interview_result')` 分别用于响应 `mounted` 和 `submit` 提交的请求

```

@get('/manage/api/get_interviews')
def get_interviews():
    interviews = yield from Interviews.findAll()
    for interview in interviews:
        interviewer = yield from Interviewers.find(interview.stu_id)
        interview['sex'] = interviewer.sex
        interview['passed'] = interviewer.passed
    return interviews

@post('/manage/api/submit_interview_result')
def submit_interview_result(*, interviews):
    for interview in interviews:
        interviewer = yield from Interviewers.find(interview['stu_id'])
        interviewer.passed = True if interview['passed'] == 'true' or
interview['passed'] == 1 else False
        yield from interviewer.update()
    interviews_ = yield from Interviews.findAll()
    for interview in interviews_:
        interviewer = yield from Interviewers.find(interview.stu_id)
        interview['sex'] = interviewer.sex
        interview['passed'] = interviewer.passed
    return interviews_

```

这里需要注意 `jinja2` 和 `vue` 默认的模板语言都是 `{{ }}`，两者在渲染的时候会起冲突，所以需要在 `Vue` 实例中加入 `delimiters:['[', '']` 更改 `Vue` 的模板语法

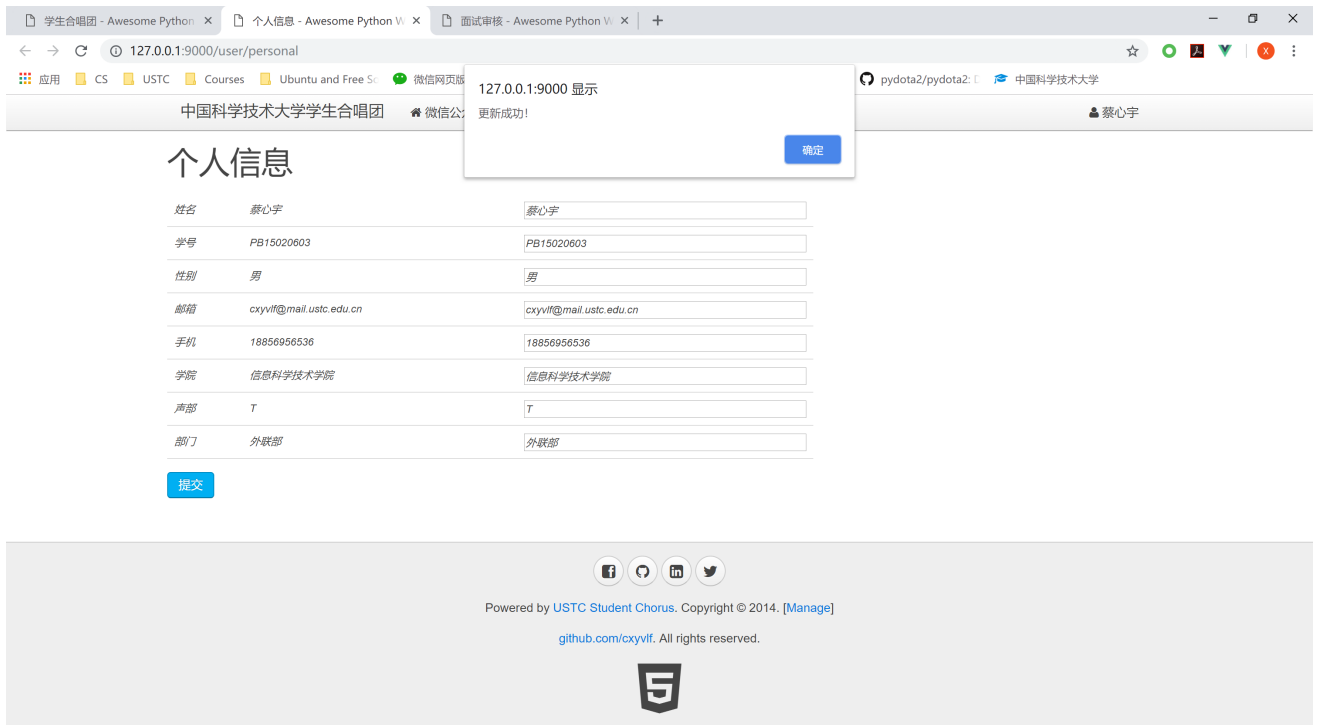
个人信息

个人信息功能与面试审核非常类似，需要先从数据库中请求登录用户的信息（`mounted`），显示在列表中，修改后提交到服务器（`submit`）

具体细节可查看 `member.html` 和 `@get('/user/personal')`、`@get('/user/api/personal')`、`@post('/user/api/edit_personal')`

其中有一个细节需要注意的是，用户本人可以修改个人信息，而管理员可以修改声部和部门信息

通过 `jinja2` 服务器端渲染实现，参照 `{% if __user__.email != user.email %}`、`{% if __user__.admin %}` 处，例如我的账户是有管理员权限的，所以我把我的声部修改为 `τ` 部门修改为 `外联部`



查询页面

查询页面实现通过不同条件对团员进行查询，显示在网页上，而且管理员可以对团员的声部、部门进行修改信息的显示、修改与提交和之前的面试审核基本一致，不同的是如何实现条件查询

```
<table class="uk-table">
  <thead>
    <tr>
      <td>性别</td>
      <td>学院</td>
      <td>声部</td>
      <td>部门</td>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>
        <select v-model="option.sex">
          <option>all</option>
          <option>男</option>
          <option>女</option>
        </select>
      </td>
      <td>
        <select v-model="option.school">
          <option>all</option>
          <option>少年班学院</option>
        </select>
      </td>
    </tr>
  </tfoot>
</table>
```

```

        <option>数学科学学院</option>
        <option>物理学院</option>
        <option>管理学院</option>
        <option>化学与材料科学学院</option>
        <option>地球和空间科学学院</option>
        <option>人文与社会科学学院</option>
        <option>工程科学学院</option>
        <option>信息科学技术学院</option>
        <option>计算机科学与技术学院</option>
        <option>网络空间安全学院</option>
        <option>软件学院</option>
    </select>
</td>
<td>
    <select v-model="option.voice_part">
        <option>all</option>
        <option>S</option>
        <option>A</option>
        <option>T</option>
        <option>B</option>
    </select>
</td>
<td>
    <select v-model="option.department">
        <option>all</option>
        <option>财物部</option>
        <option>技术部</option>
        <option>媒体部</option>
        <option>事务部</option>
        <option>外联部</option>
        <option>宣传部</option>
    </select>
</td>
<td>
    <button v-on:click="enquiry" class="uk-button uk-button-primary">查询
</button>
</td>
</tr>
</tfoot>
</table>

```

将可供选择的4个条件放在一个列表中，每个条件用HTML的 `select` 进行选择，最后将查询按钮和Vue实例中的 `enquiry` 方法绑定

```

enquiry: function(event){
    this.$http.post('/user/api/optional_enquiry', {option: this.option}).then(
        function(res){
            this.members = res.body;
        },
        function(){
            console.log('请求失败处理');
        }
    )
},

```

下面这个URL处理函数根据条件的不同构造不同的 `where` 条件，查询后返回结果

```

@post('/user/api/optional_enquiry')
def optional_enquiry(*, option):
    if not option:
        raise APIValueError('option', 'No option')
    where = ''
    args = []
    if option['sex'] != 'all':
        args.append(option['sex'])
        where += 'sex=?'
    if option['school'] != 'all':
        args.append(option['school'])
        if where:
            where += 'and school=?'
        else:
            where += 'school=?'
    if option['voice_part'] != 'all':
        args.append(option['voice_part'])
        if where:
            where += 'and voice_part=?'
        else:
            where += 'voice_part=?'
    if option['department'] != 'all':
        args.append(option['department'])
        if where:
            where += 'and department=?'
        else:
            where += 'department=?'
    members = yield from Members.findAll(where, args)
    return members

```

例如，首先为蔡心宇123号添加声部和部门

学生合唱团 - Awesome Python x 信息查询 - Awesome Python V x +

127.0.0.1:9000/user/enquiry

应用 CS USTC Courses Ubuntu and Free S 微信网页版 网络通 社团活动及会议设备 Gmail cxyvlf (Xinyu Cai) pydota2/pydota2: 中国科学技术大学

中国科学技术大学学生合唱团 微信公众号 网易云音乐 蔡心宇

127.0.0.1:9000 显示 更新成功! 确定

信息查询

性别

学院

声部

部门

all ▼

all ▼

all ▼

all ▼


查询

姓名	学号	性别	邮箱	手机	学院	声部	部门	声部	部门
蔡心宇1号	PB15000001	男	cxy1@qq.com	18800000001	少年班学院	B	宣传部	<input type="text" value="B"/>	<input type="text" value="宣传部"/>
蔡心宇2号	PB15000002	女	cxy2@qq.com	18800000002	物理学院	A	媒体部	<input type="text" value="A"/>	<input type="text" value="媒体部"/>
蔡心宇3号	PB15000003	男	cxy3@qq.com	18800000003	数学科学学院	B	技术部	<input type="text" value="B"/>	<input type="text" value="技术部"/>
蔡心宇	PB15020603	男	cxyvlf@mail.ustc.edu.cn	18856956536	信息科学技术学院	T	外联部	<input type="text" value="T"/>	<input type="text" value="外联部"/>

提交

Powered by [USTC Student Chorus](#). Copyright © 2014. [\[Manage\]](#)

[github.com/cxyvlf](#). All rights reserved.



查询性别为 女 的团员

学生合唱团 - Awesome Python x 信息查询 - Awesome Python V x +

127.0.0.1:9000/user/enquiry

应用 CS USTC Courses Ubuntu and Free S 微信网页版 网络通 社团活动及会议设备 Gmail cxyvlf (Xinyu Cai) pydota2/pydota2: 中国科学技术大学

中国科学技术大学学生合唱团 微信公众号 网易云音乐 蔡心宇

信息查询

性别

学院

声部

部门

女 ▼

all ▼

all ▼

all ▼


查询

姓名	学号	性别	邮箱	手机	学院	声部	部门	声部	部门
蔡心宇2号	PB15000002	女	cxy2@qq.com	18800000002	物理学院	A	媒体部	<input type="text" value="A"/>	<input type="text" value="媒体部"/>

提交

Powered by [USTC Student Chorus](#). Copyright © 2014. [\[Manage\]](#)

[github.com/cxyvlf](#). All rights reserved.



查询性别为 男 且声部为 B 的团员

- 执行 `\www\static\sql\` 中的 `create_db.py` 和 `init.py` 初始化数据库
- 将 `config_default.py` 中的 `password` 改为本机MySQL server的密码
- 使用 `pip` 安装好需要的库后, 运行 `app.py` ;
- 然后就可以在浏览器输入 `127.0.0.1:9000` 访问

如何添加第一个用户

在一开始时, 数据库里没有任何的用户, 需要在后台通过命令行手动添加

- 在报名面试页面报名
- 执行 `update chorus.interviewers set passed = 1`
- 通过用户注册页面注册
- 执行 `update chorus.members set admin = 1`

这样就向系统中添加了第一个管理员用户, 接下来的用户都可以由这个用户审核, 但是授予管理员权限只能在后台命令行中进行