**文档名称**

**摘要:** 描述文档摘要。

**知识产权申明**

此文档包含上海移宇科技有限公司知识产权信息。

未经书面授权，不得披露，复制，使用此文档的任何数据。

文档编码 :

文档版本号 :

电子文档名称 :

作者 :

创建时间 :

修改人:

最后修改日期:

**历史修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 编撰及修订者 | 修订说明 |
| 2013-1-8 | V0.1 | Jing Li | 创建文档 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目 录**

[1.【章节名称】 4](#_Toc369338149)

[1.1 4](#_Toc369338150)

[1.2 4](#_Toc369338151)

[1.3 4](#_Toc369338152)

[1.4 4](#_Toc369338153)

[2. 【章节名称】 5](#_Toc369338154)

[2.1. 5](#_Toc369338155)

[2.2 5](#_Toc369338156)

[3. 【章节名称】 6](#_Toc369338157)

[4. 【章节名称】 7](#_Toc369338158)

[5. 【章节名称】 8](#_Toc369338159)

[6.【章节名称】 9](#_Toc369338160)

[附录1 10](#_Toc369338161)

[附录2 11](#_Toc369338162)

# 项目说明

这是一个论坛项目。

## 使用指南

使用npm编译静态资源。

使用manager.py来管理项目。

## 1.2

## 1.3

## 1.4

点击按钮的JS处理代码没有找到。

# 2. 源代码

## 2.1 manager.py

### 2.1.1 运行服务器

flaskpython manager.py runserver

manager.py借助flask\_script插件提供的功能把命令行映射到某些类里面去，通过这个类来完成相应的任务。

### 2.1.2 代码

**import** os

**from** flask\_script **import** Manager, Server

**from** june.app **import** create\_app

Manager是flask\_script插件提供的一个功能。

manager = Manager(create\_app)

manager.add\_option('-c', '--config', dest='config', required=**False**)

manager.add\_command('runserver', Server())

就这么简单的几行代码就很Python.用对象来去初始化一个对象，对象之间可以互相调用。在对象增加一层包装。功能实体负责实现接口，并不关注过程。在应用层面抽象得很简单。

@manager.command

**def** **createdb**():

"""Create database for june."""

**from** june.models **import** db

db.create\_all()

@manager.command

**def** **live**(port=5000):

**from** livereload **import** Server

server = Server(manager.create\_app())

server.watch('assets/\*.js', 'make -C assets build')

server.watch('assets/page/\*.js', 'make -C assets build')

server.watch('assets/stylus', 'make -C assets stylus')

server.serve(port)

manager.command作为一个修饰器。

## 2.2 flask\_script包

**from** \_\_future\_\_ **import** absolute\_import

**from** gettext **import** gettext **as** \_

\_下划线在python里面是有效标示符

### 2.2.1 \_\_init\_\_.py

#### 2.2.1.1 类Manager

**class** **Manager**(object):

##### 2.2.1.1.1 函数command

**def** **command**(*self*, func):

"""

Decorator to add a command function to the registry.

**:param** func: command function.Arguments depend on the

options.

"""

command = Command(func)

*self*.add\_command(func.\_\_name\_\_, command)

**return** func

修饰器, 类似于一个注册函数，相当于一个定义逻辑的一个函数。相当于初始化阶段。

##### 2.2.1.1.2 函数run

软件系统基本上都是这个架构，先设置好数据或对象之间的关系。然后就开始一个无限循环处理时间，或消息，或数据。

**def** **run**(*self*, commands=**None**, default\_command=**None**):

"""

Prepares manager to receive command line input. Usually run

inside "if \_\_name\_\_ == "\_\_main\_\_" block in a Python script.

**:param** commands: optional dict of commands. Appended to any commands

added using add\_command().

**:param** default\_command: name of default command to run if no

arguments passed.

"""

**if** commands:

*self*.\_commands.update(commands)

**if** default\_command **is** **not** **None** **and** len(sys.argv) == 1:

sys.argv.append(default\_command)

**try**:

result = *self*.handle(sys.argv[0], sys.argv[1:])

**except** SystemExit **as** e:

result = e.code

sys.exit(result **or** 0)

##### 2.2.1.1.3 函数handle

**def** **handle**(*self*, prog, args=**None**):

*self*.set\_defaults()

app\_parser = *self*.create\_parser(prog)

args = list(args **or** [])

app\_namespace, remaining\_args = app\_parser.parse\_known\_args(args)

# get the handle function and remove it from parsed options

kwargs = app\_namespace.\_\_dict\_\_

func\_stack = kwargs.pop('func\_stack', **None**)

**if** **not** func\_stack:

app\_parser.error('too few arguments')

last\_func = func\_stack[-1]

**if** remaining\_args **and** **not** getattr(last\_func, 'capture\_all\_args', **False**):

app\_parser.error('too many arguments')

args = []

**for** handle **in** func\_stack:

# get only safe config options

config\_keys = [action.dest **for** action **in** handle.parser.\_actions

**if** handle **is** last\_func **or** action.\_\_class\_\_ **in** safe\_actions]

# pass only safe app config keys

config = dict((k, v) **for** k, v **in** iteritems(kwargs)

**if** k **in** config\_keys)

# remove application config keys from handle kwargs

kwargs = dict((k, v) **for** k, v **in** iteritems(kwargs)

**if** k **not** **in** config\_keys)

**if** handle **is** last\_func **and** getattr(last\_func, 'capture\_all\_args', **False**):

args.append(remaining\_args)

**try**:

res = handle(\*args, \*\*config)

**except** TypeError **as** err:

err.args = ("{}: {}".format(handle,str(err)),)

**raise**

args = [res]

**assert** **not** kwargs

**return** res

### 2.2.2 commands.py

#### 2.2.2.1 类Server

**class** **Server**(Command):

##### 2.2.2.1.1 函数\_\_call\_\_

**def** **\_\_call\_\_**(*self*, app, host, port, use\_debugger, use\_reloader,

threaded, processes, passthrough\_errors):

# we don't need to run the server in request context

# so just run it directly

**if** use\_debugger **is** **None**:

use\_debugger = app.debug

**if** use\_debugger **is** **None**:

use\_debugger = **True**

**if** sys.stderr.isatty():

**print**("Debugging is on. DANGER: Do not allow random users to connect to this server.", file=sys.stderr)

**if** use\_reloader **is** **None**:

use\_reloader = app.debug

app.run(host=host,

port=port,

debug=use\_debugger,

use\_debugger=use\_debugger,

use\_reloader=use\_reloader,

threaded=threaded,

processes=processes,

passthrough\_errors=passthrough\_errors,

\*\**self*.server\_options)

## 2.3 app.js

app.js是node的component工具生成的代码。

### 2.3.1 函数require

[20160706193038]: require start:june,undefined,undefined

app.js?v=4b021:29 [20160706193038]: o.call

app.js?v=4b021:29 [20160706193038]: require start:june/deps/bootstrap,june/index.js,bootstrap

app.js?v=4b021:29 [20160706193038]: o.call

app.js?v=4b021:29 [20160706193038]: require start:lepture-bootstrap/deps/jquery,lepture-bootstrap/index.js,jquery

app.js?v=4b021:29 [20160706193038]: o.call

app.js?v=4b021:29 [20160706193038]: register:component-jquery/index.js:,[object Object],function i(t) {

var n = i.resolve(t);

return require(n, e, t);

},[object Object]

app.js?v=4b021:29 [20160706193038]: require end

app.js?v=4b021:29 [20160706193038]: require end

app.js?v=4b021:29 [20160706193038]: require start:june/deps/jquery,june/index.js,jquery

app.js?v=4b021:29 [20160706193038]: require end

app.js?v=4b021:29 [20160706193038]: require end

app.js?v=4b021:29 [20160706193038]: v.handle

这个函数是最上层的入口函数，从代码的trace可以看出，代码调用了4次，但是除了第一次的调用其他都是嵌套调用。

html代码里面只有这一句代码

<script>require('june')</script>

var i = require.resolve(e);

…

var o = require.modules[i];

if (!o.\_resolving && !o.exports) {

var s = {};

s.exports = {};

s.client = s.component = true;

o.\_resolving = true;

jstrace('o.call');

o.call(this, s.exports, require.relative(i), s);

delete o.\_resolving;

o.exports = s.exports;

}

#### 2.3.1.1 o.call

o.call(this, s.exports, require.relative(i), s);

调用june/index.js函数。

### 2.3.2 函数register

require.register = function(e, t) {

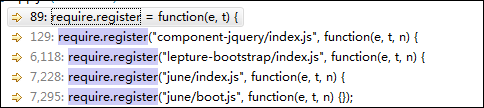
require.modules[e] = t;

};

register是注册函数。把名称和函数放进一个字典对象。

这个应该是注册june的函数。

require.register("june/index.js", function(e, t, n) { …



也就是说其实app.js是由4个js文件组成的。

#### 2.3..2.1 june/index.js

在assets/index.js的文件就是对应的june/index.js的代码。app.js就是利用nodejs提供的工具对需要用到的js文件进行合并和压缩。

前三行代码如下：

t("bootstrap");

var i = t("jquery");

e.jQuery = i;

在assets/index.js的前三行代码就是

require('bootstrap');

var $ = require('jquery');

exports.jQuery = $;

与之对应。

### 2.3.3 函数resolve

require.resolve = function(e) {

if (e.charAt(0) === "/") e = e.slice(1);

var t = [ e, e + ".js", e + ".json", e + "/index.js", e + "/index.json" ];

for (var n = 0; n < t.length; n++) {

var e = t[n];

if (require.modules.hasOwnProperty(e)) return e;

if (require.aliases.hasOwnProperty(e)) return require.aliases[e];

}

};

june 与june/index/js匹配成功。

## 2.4 lepture-bootstrap/index.js

这个是app的主要js

var jQuery = require('jquery')

这个js文件也是bootstrap的代码

## 2.5 assets/index.js

require('bootstrap');

var $ = require('jquery');

exports.jQuery = $;

index.js加载bootstrap代码，bootstrap加载jquery代码。

# 3.模板系统

## 3.1 Signup.html

<form class="panel-body" role="form" method="post" action="">

<h2>{{ \_('Sign Up') }}</h2>

{{ form.csrf\_token }}

{{ form\_field(form.username) }}

{{ form\_field(form.email) }}

{{ form\_field(form.password) }}

<div class="form-group">

<button class="btn btn-primary">{{ \_('Sign Up') }}</button>

<a class="btn btn-link" href="{{ url\_for('.signin') }}">{{ \_('Already have an account?') }}</a>

</div>

</form>

form.username是SignupForm类的username成员。

**from** wtforms **import** TextField, PasswordField, BooleanField

**class** **SignupForm**(BaseForm):

username = TextField(

\_('Username'), validators=[

DataRequired(), Length(min=3, max=20),

Regexp(r'^[a-z0-9A-Z]+$')

], description=\_('English Characters Only.'),

)

email = EmailField(

\_('Email'), validators=[DataRequired(), Email()]

)

password = PasswordField(

\_('Password'), validators=[DataRequired()]

)

**这个有一个坑，如果form的action为空时，默认提交给当前页面处理。**

# 4. 安装

## 4.1 Readme

$ npm install -g stylus component uglify-js nib

Development

-----------

You should read the Contribution Guide first.

Start a development server::

$ git checkout your\_fork\_of\_june

$ cd june

$ virtualenv --distribute venv

$ source venv/bin/activate

(venv)$ pip install -r requirements.txt

(venv)$ make static

(venv)$ python manager.py createdb

(venv)$ python manager.py runserver

先安装nodejs的四个模块。

用make构建工具命令make static生成静态文件。

## 4.2 Makefile

.PHONY: assets

staticdir = june/public/static

assets:

@$(MAKE) -C assets build

.PHONY: static

static:

@$(MAKE) -C assets compile

@uglifyjs ${staticdir}/app.js -m -o ${staticdir}/app.js

## 4.3 assets/Makefile

.PHONY: build compile components clean stylus

build: components stylus

@component build --dev -o ../june/public/static -n app

components: component.json

@component install --dev

compile: components stylus

@component-build -o ../june/public/static -n app -p .

clean:

rm -fr build components node\_modules

stylus:

@stylus -u nib index.styl -I stylus -I node\_modules

利用nodejs的插件来生成静态资源。

## 4.4 component.json

{

"name": "june",

"repo": "pythoncn/june",

"description": "Assets for june forum.",

"version": "0.0.1",

"keywords": [],

"dependencies": {

"component/jquery": "\*",

"lepture/bootstrap": "\*",

"lepture/yue.css": "\*"

},

"scripts": [

"index.js",

"boot.js"

],

"styles": [

"index.css",

"pygments-manni.css"

],

"images": [

"images/bg.png"

],

"development": {}

}

是componet工具的配置文件，npm的component工具也是一种用来管理前端库的一种工具。类似于maven, pypi, PECL等包管理工具。

# 5. URL路由

## 5.1 /account/sigup

@bp.route('/signup', methods=['GET', 'POST'])

**def** **signup**():

判断在URL地址里面有没有token

token = request.args.get('token')

# 6.后端

## 6.1 font.py

### 6.1.1 根路由 /

nodes = Node.query.order\_by(Node.id.desc()).all()

home\_nodes = filter(**lambda** o: o.on\_home, nodes)

home\_node\_ids = map(**lambda** o: o.id, home\_nodes)

# 附录1

# 附录2