第一章 项目环境

第一节 项目环境说明：

本项目是一个模仿小饭桌官网的项目实战。具有前台和CMS后台管理系统。具有以下模块：新闻、在线课程、付费资讯、搜索等模块。其中涉及到的技术要点有：`Django`、`ajax`，`Restful API`，`arttemplate.js`、在线视频播放，支付，`haystack`搜索，`UEditor`富文本编辑器，第三方分享等。其中包括前端页面布局，逻辑处理和后台逻辑代码，都会讲得非常的仔细。学完本课程后，你将成为一名能从事前后端开发的全栈开发工程师！完全可以在企业胜任一份`Python web`开发的工作！以下讲解本项目中所用到的环境！

前端方向：

1. `nvm`：用来管理`node.js`的工具。
2. `node.js`：自带有`npm`包管理工具。
3. `npm`：类似于`Python`中的`pip`。可以非常方便的管理一些前端开发的包。
4. `gulp`：用来自动化开发流程。比如`sass`转`css`，`css`和`js`压缩等。

后端方向：

1. `Python 3.6`：开发语言。
2. `Django 2.0`：开发框架。
3. `MySQL 5.7`：数据库。

第二节 前端开发环境配置：

nvm安装：

`nvm（Node Version Manager）`是一个用来管理`node`版本的工具。我们之所以需要使用`node`，是因为我们需要使用`node`中的`npm(Node Package Manager)`，使用`npm`的目的是为了能够方便的管理一些前端开发的包！`nvm`的安装非常简单，步骤如下：

1. 到这个链接下载`nvm`的安装包：`https://github.com/coreybutler/nvm-windows/releases`。
2. 然后点击一顿下一步，安装即可！
3. 安装完成后，还需要配置环境变量。在`我的电脑->属性->高级系统设置->环境变量->系统环境变量->Path`下新建一个，把`nvm`所处的路径填入进去即可！
4. 打开`cmd`，然后输入`nvm`，如果没有提示没有找不到这个命令。说明已经安装成功！
5. `Mac`或者`Linux`安装`nvm`请看这里：`https://github.com/creationix/nvm`。也要记得配置环境变量。

`nvm`常用命令：

1. `nvm install node`：安装最新版的`node.js`。nvm i == nvm install。
2. `nvm install [version]`：安装指定版本的`node.js`。

3. `nvm use [version]`：使用某个版本的`node`。
4. `nvm list`：列出当前安装了哪些版本的`node`。
5. `nvm uninstall [version]`：卸载指定版本的`node`。
6. `nvm node_mirror [url]`：设置`nvm`的镜像。
7. `nvm npm_mirror [url]`：设置`npm`的镜像。

node安装：

安装完`nvm`后，我们就可以通过`nvm`来安装`node`了。这里我们安装`6.4.0`版本的的`node.js`就可以。因为最新版的`node.js`的`npm`是`5.0`的，上面还有很多坑。安装命令如下：

```
nvm install 6.4.0
```

如果你的网络够快，那以上命令在稍等片刻之后会安装成功。如果你的网速很慢，那以上命令可能会发生超时。因为`node`的服务器地址是`https://nodejs.org/dist/`，这个域名的服务器是在国外。因此会比较慢。因此我们可以设置一下`nvm`的源。

```
nvm node_mirror https://npm.taobao.org/mirrors/node/
nvm npm_mirror https://npm.taobao.org/mirrors/npm/
```

npm安装：

`npm(Node Package Manager)`在安装`node`的时候就会自动的安装了。当时前提条件是你需要设置当前的`node`的版本：`nvm use 6.4.0`。然后就可以使用`npm`了. 关于`npm`常用命令以及用法，请看下文。

安装包：

安装包分为全局安装和本地安装。全局安装是安装在当前`node`环境中，在可以在cmd中当作命令使用。而本地安装是安装在当前项目中，只有当前这个项目能使用，并且可以通过require引用。安装的方式只有`-g`参数的区别：

```
npm install express          # 本地安装
npm install express -g   # 全局安装
```

本地安装

1. 将安装包放在`./node_modules`下（运行 npm 命令时所在的目录），如果没有`node_modules`目录，会在当前执行`npm`命令的目录下生成`node_modules`目录。
2. 可以通过`require()`来引入本地安装的包。

全局安装

1. 将安装包放在`/usr/local`下或者你`node`的安装目录。
2. 可以直接在命令行里使用。

卸载包：

```
npm uninstall [package]
```

更新包：

```
npm update [package]
```

搜索包：

```
npm search [package]
```

使用淘宝镜像：

npm install -g cnpm --registry=https://registry.npm.taobao.org 那么以后就可以使用cnpm来安装包了！

第三节 前端项目搭建

前端我们使用`gulp`来自动化开发流程。配置好`gulp`后，可以自动给我们处理好一些工作。比如写完`css`后，要压缩成`.min.css`，写完`js`后，要做混淆和压缩，图片压缩等。这些工作都可以让`gulp`帮我们完成。

安装gulp：

1. 创建本地包管理环境：

使用`npm init`命令在本地生成一个`package.json`文件，`package.json`是用来记录你当前这个项目依赖了哪些包，以后别人拿到你这个项目后，不需要你的`node_modules`文件夹（因为node_moduels中的包实在太庞大了）。只需要执行`npm install`命令，即会自动安装`package.json`下`devDependencies`中指定的依赖包。

2. 安装gulp：

`gulp`的安装非常简单，只要使用`npm`命令安装即可。但是因为`gulp`需要作为命令行的方式运行，因此需要在安装在系统级别的目录中。

```
npm install gulp -g
```

因为在本地需要使用`require`的方式`gulp`。因此也需要在本地安装一份：

```
npm install gulp --save-dev
```

以上的`--save-dev`是将安装的包的添加到`package.json`下的`devDependencies`依赖中。以后通过`npm install`即可自动安装。`devDependencies`这个是用来记录开发环境下使用的包，如果想要记录生产环境下使用的包，那么在安装包的时候使用`npm install xx --save`就会记录到`package.json`下的`dependencies`中，`dependencies`是专门用来记录生产环境下的依赖包的！

3. 创建gulp任务：

要使用`gulp`来流程化我们的开发工作。首先需要在项目的根目录下创建一个`gulpfile.js`文件。然后在`gulpfile.js`中填入以下代码：

```
var gulp = require("gulp")

gulp.task("greet",function () {
    console.log('hello world');
});
```

这里对代码进行一一解释：

1. 通过`require`语句引用已经安装的第三方依赖包。这个`require`只能是引用当前项目的，不能引用全局下的。`require`语法是`node.js`独有的，只能在`node.js`环境下使用。
2. `gulp.task`是用来创建一个任务。`gulp.task`的第一个参数是命令的名字，第二个参数是一个函数，就是执行这个命令的时候会做什么事情，都是写在这个里面的。
3. 写完以上代码后，以后如果想要执行`greet`命令，那么只需要进入到项目所在的路径，然后终端使用`gulp greet`即可执行。

4. 创建处理css文件的任务：

`gulp`只是提供一个框架给我们。如果我们想要实现一些更加复杂的功能，比如`css`压缩，那么我们还需要安装一下`gulp-cssnano`插件。`gulp`相关的插件安装也是通过`npm`命令安装，安装方式跟其他包是一模一样的（gulp插件本身就是一个普通的包）。对`css`文件的处理，需要做的事情就是压缩，然后再将压缩后的文件放到指定目录下（不要和原来css文件重合了）！这里我们使用`gulp-cssnano`来处理这个工作：

```
npm install gulp-cssnano --save-dev
```

然后在`gulpfile.js`中写入以下代码：

```
var gulp = require("gulp")
var cssnano = require("gulp-cssnano")

// 定义一个处理css文件改动的任务
gulp.task("css",function () {
    gulp.src("./css/*.css")
    .pipe(cssnano())
    .pipe(gulp.dest("./css/dist/"))
});
```

以上对代码进行详细解释：

1. `gulp.task`：创建一个`css`处理的任务。
2. `gulp.src`：找到当前`css`目录下所有以`.css`结尾的`css`文件。
3. `pipe`：管道方法。将上一个方法的返回结果传给另外一个处理器。比如以上的`cssnano`。
4. `gulp.dest`：将处理完后的文件，放到指定的目录下。不要放在和原文件相同的目录，以免产生冲突，也不方便管理。

5. 修改文件名：

像以上任务，压缩完`css`文件后，最好是给他添加一个`.min.css`的后缀，这样一眼就能知道这个是经过压缩后的文件。这时候我们就需要使用`gulp-rename`来修改了。当然首先也需要安装`npm install gulp-rename --save-dev`。示例代码如下：

```
var gulp = require("gulp")
var cssnano = require("gulp-cssnano")
var rename = require("gulp-rename")
gulp.task("css",function () {
    gulp.src("./css/*.css")
    .pipe(cssnano())
    .pipe(rename({"suffix":".min"}))
    .pipe(gulp.dest("./css/dist/"))
});
```

在上述代码中，我们增加了一行`.pipe(rename({"suffix":".min"}))`，这个我们就是使用`rename`方法，并且传递一个对象参数，指定修改名字的规则为添加一个`.min`后缀名。这个`gulp-rename`还有其他的指定文件名的方式，比如可以在文件名前加个前缀等。更多的教程可以看这个：`https://www.npmjs.com/package/gulp-rename`。

6. 创建处理js文件的任务：

处理`js`文件，我们需要使用到`gulp-uglify`插件。安装命令如下：

```
npm install gulp-uglify --save-dev
```

安装完后，我们就可以对`js`文件进行处理了。示例代码如下：

```
var gulp = require("gulp")
var rename = require("gulp-rename")
var uglify = require('gulp-uglify');
gulp.task('script',function(){
    gulp.src(path.js + '*.js')
    .pipe(uglify())
    .pipe(rename({suffix:'.min'}))
    .pipe(gulp.dest('js/'));
});
```

这里就是增加了一个`.pipe(uglify())`的处理，对`js`文件进行压缩和丑化（修改变量名）等处理。更多关于`gulp-uglify`的教程。请看：`https://github.com/mishoo/UglifyJS2#minify-options`。

7. 合并多个文件：

在网页开发中，为了加快网页的渲染速度，有时候我们会将多个文件压缩成一个文件，从而减少请求的次数。要拼接文件，我们需要用到`gulp-concat`插件。安装命令如下：

```
npm install gulp-concat --save-dev
```

比如我们现在有一个`nav.js`文件用来控制导航条的。有一个`index.js`文件用来控制首页整体内容的。那么我们可以使用以下代码将这两个文件合并成一个文件：

```
var gulp = require('gulp');
var concat = require('gulp-concat');
var uglify = require('gulp-uglify');
gulp.task('vendorjs',function(){
    gulp.src([
        './js/nav.js',
        './js/index.js'
    ])
    .pipe(concat('index.min.js'))
    .pipe(uglify())
    .pipe(gulp.dest('dist/js/'));
});
```

8. 压缩图片：

图片是限制网站加载速度的一个主要原因。图片越大，从网站上下载所花费的时间越长。因此对于一些图片，我们可以采取无损压缩，即在不改变图片质量的基础之上进行压缩。在`gulp`中我们可以通过`gulp-imagemin`来帮我们实现。安装命令如下：

```
npm install gulp-imagemin --save-dev
```

压缩图片也是一个比较大的工作量，对于一些已经压缩过的图片，我们就没必要再重复压缩了。这时候我们可以使用`gulp-cache`来缓存那些压缩过的图片。安装命令如下：

```
npm install gulp-cache --save-dev
```

两个插件结合使用的代码如下：

```
var imagemin = require('gulp-imagemin');
var cache = require('gulp-cache');
gulp.task('image',function(){
    gulp.src("./images/*.*")
    .pipe(cache(imagemin()))
    .pipe(gulp.dest('dist/images/'));
});
```

9. 检测代码修改，自动刷新浏览器：

以上所有的任务，我们都是需要手动的在终端去执行。这样很不方便我们开发。最好的方式就是我修改了代码后，`gulp`会自动的执行相应的任务。这个工作我们可以使用`gulp`内置的`watch`方法帮我们完成：

```
var gulp = require("gulp")
var cssnano = require("gulp-cssnano")
var rename = require("gulp-rename")

// 定义一个处理css文件改动的任务
gulp.task("css",function () {
    gulp.src("./css/*.css")
    .pipe(cssnano())
    .pipe(rename({"suffix":".min"}))
    .pipe(gulp.dest("./css/dist/"))
    .pipe(connect.reload())
});

// 定义一个监听的任务
gulp.task("watch",function () {
    // 监听所有的css文件，然后执行css这个任务
    gulp.watch("./css/*.css",['css'])
});
```

以后只要在终端执行`gulp watch`命令即可自动监听所有的`css`文件，然后自动执行`css`的任务，完成相应的工作。

10. 更改文件后，自动刷新浏览器：

以上我们实现了更改一些`css`文件后，可以自动执行处理`css`的任务。但是我们还是需要手动的去刷新浏览器，才能看到修改后的效果。有什么办法能在修改完代码后，自动的刷新浏览器呢。答案是使用`browser-sync`。`browser-sync`安装的命令如下：

```
npm install browser-sync --save-dev
```

`browser-sync`使用的示例代码如下：

```
var gulp = require("gulp")
var cssnano = require("gulp-cssnano")
var rename = require("gulp-rename")
var bs = require("browser-sync").create()

gulp.task("bs",function () {
    bs.init({
        'server': {
            'baseDir': './'
        }
    });
});

// 定义一个处理css文件改动的任务
gulp.task("css",function () {
    gulp.src("./css/*.css")
    .pipe(cssnano())
    .pipe(rename({"suffix":".min"}))
    .pipe(gulp.dest("./css/dist/"))
    .pipe(bs.stream())
});

// 定义一个监听的任务
gulp.task("watch",function () {
    gulp.watch("./css/*.css",['css'])
});

// 执行gulp server开启服务器
gulp.task("server",['bs','watch'])
```

以上我们创建了一个`bs`的任务，这个任务会开启一个`3000`端口，以后我们在访问`html`页面的时候，就需要通过`http://127.0.0.1:3000`的方式来访问了。然后接下来我们还定义了一个`server`任务。这个任务会去执行`bs`和`watch`任务，只要修改了`css`文件，那么就会执行`css`的任务，然后就会自动刷新浏览器。`browser-sync`更多的教程请参考：`http://www.browsersync.cn/docs/gulp/`。

第四节 Sass语法：

众所周知，`css`不是一门编程语言。他没法像`js`和`python`那样拥有逻辑处理的能力，甚至导入其他的`css`文件中的样式都做不到。而`Sass`就是为了解决`css`的这些问题。他它允许你使用变量、嵌套规则、 `mixins`、导入等众多功能，并且完全兼容`css`语法。`Sass`文件不能直接被网页所识别，写完`Sass`后，还需要专门的工具转化为`css`才能使用。

Sass文件的后缀名：

`Sass`文件有两种后缀名，一个是`scss`，一个是`sass`。不同的后缀名，相应的语法也不一样。这里我们使用`scss`的后缀名。包括后面讲到的`Sass`语法，也都是`scss`的后缀名的语法。

使用`gulp`将`Sass`转换为`css`：

将`Sass`文件转换为`css`文件的工具有很多。这里我们就使用之前讲过的`gulp`来实现。这里我们需要使用`gulp-sass`插件来帮我们完成。安装方式非常简单：`npm install gulp-sass --save-dev`。那么处理`sass`的代码如下：

```
var gulp = require("gulp");
var sass = require("gulp-sass");
// 处理css的任务
gulp.task('css',function () {
    gulp.src(path.css + '*.scss')
        .pipe(sass().on("error",sass.logError))
        .pipe(cssnano())
        .pipe(rename({"suffix":".min"}))
        .pipe(gulp.dest(path.css_dist))
});
```

Sass基本语法：

注释：

支持`/* comment */`和`// 注释`两种方式。

嵌套：

`Sass`语法允许嵌套。比如`#main`下有一个类为`.header`，那么我们可以写成以下的形式：

```
#main{
    background: #ccc;
    .header{
        width: 20px;
        height: 20px;
    }
}
```

这样写起来更加的直观。一看就知道`.header`是在`#main`下的。

引用父选择器（`&`）：

有时候，在嵌套的子选择器中，需要使用父选择器，那么这时候可以通过`&`来表示。示例代码如下：

```
a{
    font-weight: bold;
    text-decoration: none;
    &:hover{
        color: #888;
    }
}
```

定义变量：

是的，你没听错。在`Sass`中可以定义变量。对于一些比较常用的值，我们可以通过变量存储起来，以后想要使用的时候就直接用就可以了。定义变量使用`$`符号。示例代码如下：

```
$mainWidth: 980px;
#main{
    width: $mainWidth;
}
```

运算：

在`Sass`中支持运算。比如现在有一个容器总宽度是`900`，要在里面平均放三个盒子，那么我们可以通过变量来设置他们的宽度。示例代码如下：

```
$mainWidth: 900px;
.box{
    width: $mainWidth/3;
}
```

@import语法：

在`css`中`@import`只能导入`css`文件，而且对网站的性能有很大的影响。而`Sass`中的`@import`则是完全实现了一套自己的机制。他可以直接将指定文件的代码拷贝到导入的地方。示例代码如下：

```
@import "init.scss";
```

@extend语法：

有时候我们一个选择器中，可能会需要另外一个选择器的样式，那么我们就可以通过`extend`来直接将指定选择器的样式加入进来。示例代码如下：

```
.error{
    background-color: #fdd;
    border: 1px solid #f00;
}
.serious-error{
    @extend .error;
    border-width: 3px;
}
```

@mixin语法：

有时候一段样式代码。我们可能要用很多地方。那么我们可以把他定义i成`mixin`。需要用的时候就直接引用就可以了。示例代码如下：

```
@mixin large-text {
  font: {
    family: Arial;
    size: 20px;
    weight: bold;
  }
  color: #ff0000;
}
```

如果其他地方想要使用这个`mixin`的时候，可以通过`@include`来包含进来。示例代码如下：

```
.page-title {
  @include large-text;
  padding: 4px;
  margin-top: 10px;
}
```

`@mixin`也可以使用参数。示例代码如下：

```
@mixin sexy-border($color, $width) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}
```

那么以后在`include`的时候，就需要传递参数了。示例代码如下：

```
p {
    @include sexy-border(blue, 1px);
}
```

更详细的教程：

更详细的教程可以参考：`http://sass.bootcss.com/docs/sass-reference/`。

第五节 flex布局

简介：

1. 概念：`CSS3`弹性盒子(`Flexible Box`或`Flexbox`)，是一种用于在页面上布置元素的布局模式，使得当页面布局必须适应不同的屏幕尺寸和不同的显示设备时，元素可预测地运行。对于许多应用程序，弹性盒子模型提供了对块模型的改进，因为它不使用浮动，`flex`容器的边缘也不会与其内容的边缘折叠。
2. 兼容性：参考`https://developer.mozilla.org/zh-CN/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes`。

相关概念：

1. 弹性容器：包含弹性项目的父元素。通过给这个元素设置`display`属性的值为`flex`或者`inline-flex`来定义弹性容器。
2. 弹性项目：弹性容器的每个子元素都称为弹性项目。
3. 轴：分为主轴和纵轴。默认横向的是主轴，纵向的为纵轴。也可以通过`flex-direction:column`来将主轴设置为纵向的。主轴其实更加恰当的解释为，元素按顺序排列的那根线。

相关css属性：

1. `flex-direction`：确定主轴的方向。默认是横向的。
    - row：横向布局。
    - row-reverse：横向反转布局。
    - column：纵向布局。
    - column-reverse：纵向反转布局。
2. `justify-content`：属性定义了项目在主轴上的对齐方式。
    - flex-start：主轴的起始点对齐（默认）。
    - flex-end：主轴的结束点对齐。
    - center：居中排列。
    - space-between：两端对齐，项目之间的间隔相等。
    - space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。
3. `align-items`：确定项目在纵轴上如何对齐。
    - flex-start：纵轴的起始点对齐。
    - flex-end：纵轴的结束点对齐。
    - center：纵轴的中点对齐。
    - stretch：默认值。如果没有设置高度。弹性元素被在侧轴方向被拉伸到与容器相同的高度或宽度。
4. `flex-wrap`：指定子元素在一行排列不下的时候，该如何表现。

    - nowrap：不换行，被挤到一行。
    - wrap：被打断到多行中。
5. `align-content`：确定纵轴的轴线如何对齐。只有在多行的情况下才有效。
    - flex-start：纵轴的起始点对齐。
    - flex-end：纵轴的结束点对齐。
    - center：纵轴的中点对齐。
    - stretch：默认值。在没有给元素设置高度的时候（假如纵轴使用的是竖向的），弹性元素被在侧周方向被拉伸到与容器相同的高度。

第二章 短信验证码

短信验证码服务商有很多。这里我们选择一个阿里通信来作为短信服务平台。通过这个平台，中小企业及开发者可以在最短的时间内实现短信验证码发送、短信服务提醒、语音验证码、语音服务通知、IVR及呼叫中心、码号、后向流量、隐私保护相关的能力，实现互联网电信化。

第一节 官方文档：

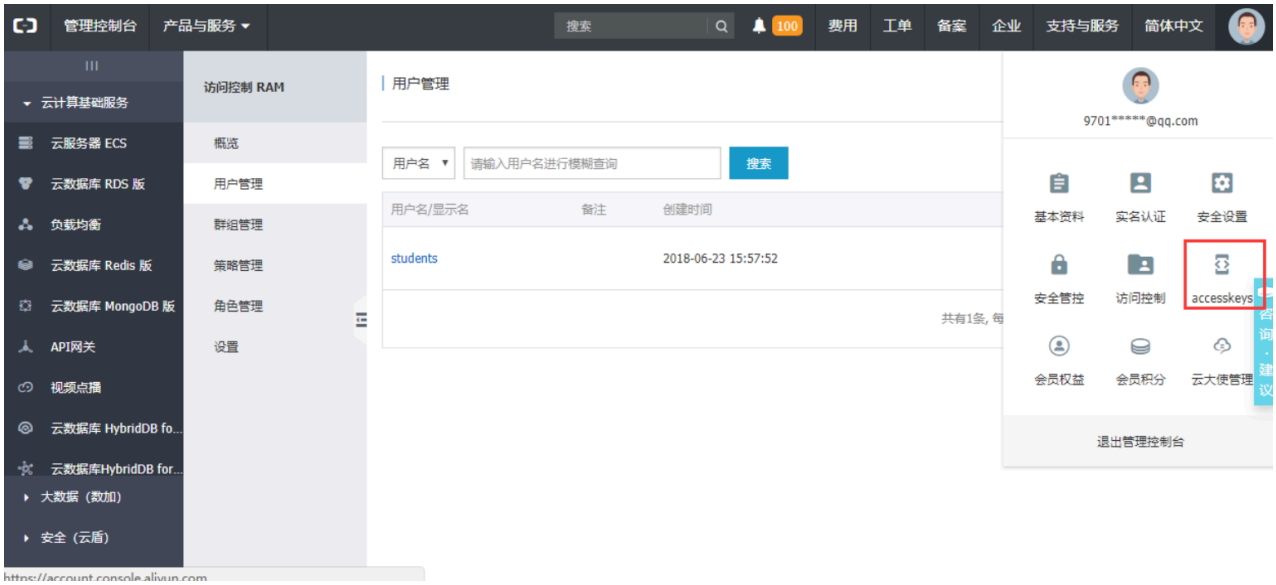
https://help.aliyun.com/document_detail/59210.html

第二节登录阿里通信：

1. 链接：https://www.aliyun.com/product/sms
2. 然后用淘宝账号进行登录。
3. 登录成功后，进入到这个界面：https://dysms.console.aliyun.com/dysms.htm#/overview

第三节 获取AccessKey和ACCESS_KEY_SECRET：

在右上角的头像下，选择AccessKey：



第四节 创建应用：

第五节 创建验证码：

创建签名：

1. 创建验证码分为两大步，先是添加签名，然后再添加短信模版。首先添加签名：

2. 然后来到添加签名的页面：

3. 签名创建后，需要几个小时的审核。

添加短信模板：

1. 点击添加模板：

2. 填入模板相关的参数：

第六节 Python发送短信验证码：

1. 在这里下载`Python`的`SDK`。
2. 在这里查看文档。

3. 下载完`SDK`后，然后进入到目录中，再进入到你的`django`开发的虚拟环境中，然后执行命令`python setup.py install`即可安装`SDK`。
4. 在下载的`SDK`文件中，有一个`test_sms_send.py`的文件，修改里面相关的参数，然后再修改`const.py`中的`AccessKey`和`SecretKey`为之前获取的参数。运行这个文件就可以发送成功了。

```
# const.py文件
# ACCESS_KEY_ID/ACCESS_KEY_SECRET 根据实际申请的账号信息进行替换
ACCESS_KEY_ID = "你的用户的AccessKey"
ACCESS_KEY_SECRET = "你的用户的Secretkey"
```

然后修改`test_sms_send.py`的代码：

```
import sys
from aliyunsdkdysmsapi.request.v20170525 import SendSmsRequest
from aliyunsdkdysmsapi.request.v20170525 import QuerySendDetailsRequest
from aliyunsdkcore.client import AcsClient
import uuid
from aliyunsdkcore.profile import region_provider
from aliyunsdkcore.http import method_type as MT
from aliyunsdkcore.http import format_type as FT
import const
import json

"""
短信业务调用接口示例，版本号：v20170525

Created on 2017-06-12

"""
# 注意：不要更改
REGION = "cn-hangzhou"
PRODUCT_NAME = "Dysmsapi"
DOMAIN = "dysmsapi.aliyuncs.com"

acs_client = AcsClient(const.ACCESS_KEY_ID, const.ACCESS_KEY_SECRET,
REGION)
region_provider.add_endpoint(PRODUCT_NAME, REGION, DOMAIN)

def send_sms(business_id, phone_numbers, sign_name, template_code,
template_param=None):
    smsRequest = SendSmsRequest.SendSmsRequest()
    # 申请的短信模板编码,必填
    smsRequest.set_TemplateCode(template_code)

    # 短信模板变量参数
    if template_param is not None:
```

```
        smsRequest.set_TemplateParam(template_param)

    # 设置业务请求流水号，必填。
    smsRequest.set_OutId(business_id)

    # 短信签名
    smsRequest.set_SignName(sign_name)

    # 数据提交方式
    # smsRequest.set_method(MT.POST)

    # 数据提交格式
    # smsRequest.set_accept_format(FT.JSON)

    # 短信发送的号码列表，必填。
    smsRequest.set_PhoneNumbers(phone_numbers)

    # 调用短信发送接口，返回json
    smsResponse = acs_client.do_action_with_exception(smsRequest)

    # TODO 业务处理
    return smsResponse

if __name__ == '__main__':
    __business_id = uuid.uuid1()
    #print(__business_id)
    params = {
        'code': 1234
    }
    #params =
u'{"name":"wqb","code":"12345678","address":"bz","phone":"13000000000"}'
    print(send_sms(__business_id, "18515287309", "小饭桌应用",
"SMS_68465012", json.dumps(params)))
```

第三章 七牛云存储

第一节 介绍：

七牛云存储是一个集图片、视频对象存储为一体的网站。并且他上面集成了cdn加速服务，图片处理（加水印，图片裁剪）等功能，对于一些想要快速开发产品，不想花大量时间来构建自己资源服务器的中小型公司而言，无疑是最好的选择。

第二节 准备工作：

1. 到七牛官网：`https://www.qiniu.com/`。创建账号。然后到个人面板->秘钥管理处获取`access_key`和`secret_key`。

2. 创建空间，可以选择华东区，华南区等。

第三节 使用：

后端：

1. 下载Python SDK：

通过命令`pip install qiniu`即可下载七牛的`SDK`。

2. 创建一个获取`token`的`url`：

```
# views.py
import qiniu

@require_GET
def qntoken(request):
    access_key = '你的access_key'
    secret_key = '你的secret_key'
    q = qiniu.Auth(access_key, secret_key)

    bucket = 'xtspace'
    token = q.upload_token(bucket)
    return restful.result(kwargs={"uptoken":token})
```

然后在`urls.py`中做一个`url`与视图函数的映射。示例代码如下：

```
from django.urls import path
urlpatterns = [
    path('qntoken/',views.qntoken,name='qntoken')
]
```

前端：

1. 在模板中引入最新版(2.4.0)的`JavaScript SDK`：

   ```
   <script src="https://unpkg.com/qiniu-js@2.4.0/dist/qiniu.min.js"></script>
   ```

2. 然后监听一个`type=file`类型的按钮的`change`事件，一旦选择了文件，那么就会执行`change`事件，在`change`事件的处理函数中，我们就可以获取到当前选中的文件。然后通过七牛的`SDK`发送给服务器。示例代码如下：

   ```
   function News() {
       var self = this;
       self.progressGroup = $("#progress-group");
       self.progressBar = $(".progress-bar");
   }

   News.prototype.listenQiniuUploadFileEvent = function () {
       var self = this;
       var thumbnailBtn = $('#thumbnail-btn');
       thumbnailBtn.change(function (event) {
           var file = this.files[0];
           xfzajax.get({
               'url': '/cms/qntoken/',
               'success': function (result) {
                   var token = result['uptoken'];
                   var key = (new Date()).getTime() + '.' +
   file.name.split('.')[1];
   ```

```
                var putExtra = {
                    fname: key,
                    params: {},
                    mimeType: ['image/png','video/x-ms-wmv','image/jpeg']
                };
                var config = {
                    useCdnDomain: true,
                    retryCount: 6,
                    region: qiniu.region.z2
                };
                var observable = qiniu.upload(file, key, token,
putExtra,config);
                observable.subscribe({
                    "next":self.updateUploadProgress,
                    "error":self.uploadErrorEvent,
                    "complete": self.complateUploadEvent
                });
                self.progressGroup.show();
            }
        });
    });
};

News.prototype.updateUploadProgress = function (response) {
    var self = this;
    var total = response.total;
    var percent = total.percent;
    var percentText = percent.toFixed(0) + '%';
    var progressBar = $(".progress-bar");
    progressBar.css({"width":percentText});
    progressBar.text(percentText);
};

News.prototype.uploadErrorEvent = function (error) {
    window.messageBox.showError(error.message);
};

News.prototype.complateUploadEvent = function (response) {
    var self = this;
    var filename = response['key'];
    var domain = "http://og209xb9x.bkt.clouddn.com/";
    var thumbnailUrl = domain + filename;
    var thumbnailInput = $("#thumbnail-form");
    thumbnailInput.val(thumbnailUrl);
    var progressGroup = $("#progress-group");
    progressGroup.hide();
};
```

第四节 相关链接：

1. 七牛官网：`https://www.qiniu.com/`。
2. `Python SDK`地址：`https://developer.qiniu.com/kodo/sdk/1242/python`。
3. `JavaScript SDK`地址：`https://developer.qiniu.com/kodo/sdk/1283/javascript`。

第四章 百度云点播VOD服务

第一节 准备工作：

1. 登录百度。
2. 进入百度云：`https://cloud.baidu.com`。
3. 开通“视频点播VOD”服务。
4. 进入后台管理系统->右上角“安全认证”->获取“AccessKey”。
5. 再来到“视频点播VOD”界面->全局设置->发布设置->安全设置->获取UserKey。

第二节 自定义转码设置：

我们的视频需要经过编码加密后才能发布出去。因此要创建一个编码模板。具体模板容器的配置如下：

1. 容器：选择`HLS(Http Live Streaming)`。`HLS`是苹果公司提出的基于`HTTP`的流媒体网络传输协议。他把一整个视频切片成多段，然后生成一个`m3u8`文件，在这个文件中存储了每一段视频的真实地址，以后在实现了`m3u8`协议的播放器在播放视频的时候就从这个文件中获取视频，并且播放的时候需要设置密码。
2. 编码规格：高。
3. 分辨率：1920_1080，2180_720，800*600等。
4. 视频码率：

分辨率	视频码率
1920*1080	2500
1280*720	1024
640*360	512

1. 加密策略：Token加密。

第三节 后端：

后端唯一要做的事情就是给前端提供`token`，前端只有获取了`token`才能播放出这个视频出来。坏消息是百度云根本没有提供获取`token`的`sdk`，好消息是我们已经找到了他生成`token`的方法。方法如下：

```
token计算规则
```

名词解释：
1. userId： 百度云用户唯一标识，可在百度云管理控制台账号基本信息中得到，32位字符串。
2. userKey：用户密钥，是AES-128内容密钥，用于对MediaId和ExpirationTime组成的串进行加密。长度为16字节，用32位十六进制数表示。可在百度云VOD管理控制台中设置和更新。
3. signature：签名，是用UserKey给MediaId和ExpirationTime加密生成的字符串。长度为32字节，用64位十六进制数表示。
4. mediaId：百度云VOD中媒资的唯一标识。
5. expirationTime：Signature的过期时间，是一个unix时间戳，以秒为单位。
6. token：由Signature_UserId_ExpirationTime组成的字符串。
7. HMAC-SHA-256：签名算法，详见https://tools.ietf.org/html/rfc4868

token的生成方法：
1. 生成签名signature = HMAC-SHA-256(userKey, String.format("/%s/%s", mediaId, expirationTime), sample code见HMAC-SHA-256加密算法 Sample Code
2. 利用下划线(_) 连接signature，userId，expirationTime组合成token = String.format("%s_%s_%s", signature, userId, expirationTime)

更多请见 `https://cloud.baidu.com/doc/VOD/BestPractise.html#token.E8.AE.A1.E7.AE.97.E8.A7.84.E5.88.99`。

示例代码如下：

```
import os,hmac,hashlib,time
from django.conf import settings

def course_token(request):
    file = request.GET.get('video')

    expiration_time = int(time.time()) + 2 * 60 * 60

    USER_ID = settings.BAIDU_CLOUD_USER_ID
    USER_KEY = settings.BAIDU_CLOUD_USER_KEY

    extension = os.path.splitext(file)[1]
    media_id = file.split('/')[-1].replace(extension, '')

    key = USER_KEY.encode('utf-8')
    message = '/{0}/{1}'.format(media_id, expiration_time).encode('utf-8')
    signature = hmac.new(key, message, digestmod=hashlib.sha256).hexdigest()
    token = '{0}_{1}_{2}'.format(signature, USER_ID, expiration_time)
    return restful.result(data={'token': token})
```

第四节 前端：

1. 用这个链接下载`videojs`文件：`http://sdk.bce.baidu.com/media-sdk/Baidu-T5Player-SDK-Web-v3.4.0.zip`。

2. 加载这三个

```
js
```

文件：

```
 <script src="{% static 'videojs/video.min.js' %}"></script>
 <script src="{% static 'videojs/videojs-contrib-hls.min.js' %}"></script>
 <script src="{% static 'videojs/videojs-contrib-quality-levels.min.js'
%}"></script>
 <script type="text/javascript"
src="https://cdn.bdstatic.com/jwplayer/latest/cyberplayer.js"></script>
```

3. 创建一个容器，用来装视频播放的。示例代码如下：

```
 <div id="playercontainer"></div>
```

4. 初始化播放器。并且将视频地址的参数传递进去。示例代码如下：

```
 var videourl = "视频地址";
 var cover = "视频封面图";
 var player = cyberplayer("playercontainer").setup({
     width: '100%',
     height: '100%',
     file: videourl,
     image: cover,
     autostart: false,
     stretching: "uniform",
     repeat: false,
     volume: 100,
     controls: true,
     primary: "flash",
     tokenEncrypt: "true",
     // AccessKey
     ak: '42455a8c985649aeaa4ca86b50482d78'
 });
 player.on('beforePlay',function (e) {
     if(!/m3u8/.test(e.file)){
         return;
     }
     xfzajax.get({
         // 获取token的url
         'url': '/course/course_token/',
         'data': {
             'video': videourl
         },
         'success': function (result) {
```

```
            if(result['code'] === 200){
                var token = result['data']['token'];
                player.setToken(e.file,token);
            }else{
                alert('token错误! ');
            }
        },
        'fail': function (error) {
            console.log(error);
        }
    });
});
```

第五章 搜索

搜索可以使用最原始的`like`的方式进行搜索。当然这种搜索方式对于一些小量的数据是非常合适的。但是随着数据量越来越大。这时候我们就需要使用搜索引擎了。搜索引擎会将所有需要搜索的数据使用算法做一个索引，以后搜索的时候就只需要根据这个索引即可找到相应的数据。搜索引擎做索引的过程会比较慢，但是一旦索引建立完成，那么以后再搜索的时候就会很快了。

第一节 django-haystack插件：

这个插件是专门给`django`提供搜索功能的。`django-haystack`提供了一个搜索的接口，底层可以根据自己的需求更换搜索引擎。他其实有点类似于`Django`中的`ORM`插件，提供了一个操作数据库的接口，但是底层具体使用哪个数据库是可以自己设置的。安装方式非常简单，通过`pip install django-haystack`即可安装。

第二节 搜索引擎：

`django-haystack`支持的搜索引擎有`Solr`、`Elasticsearch`、`Whoosh`、`Xapian`等。`Whoosh`是基于纯`Python`的搜索引擎，检索速度快，集成方便。这里我们就选择`Whoosh`来作为`haystack`的搜索引擎。安装方式同样也是通过`pip`安装的：`pip install whoosh`。

第三节 集成步骤：

1、在项目中安装`django-haystack`。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',

    # 添加
    'haystack',
]
```

2、设置搜索引擎：

在`settings.py`中添加以下配置：

```
HAYSTACK_CONNECTIONS = {
    'default': {
        # 设置haystack的搜索引擎
        'ENGINE': 'haystack.backends.whoosh_backend.WhooshEngine',
        # 设置索引文件的位置
        'PATH': os.path.join(BASE_DIR, 'whoosh_index'),
    }
}
```

3、创建索引类：

在模型所属的`app`下创建一个`search_indexes.py`文件，然后创建索引类。比如要给`News`创建索引，代码如下：

```
class NewsIndex(indexes.SearchIndex,indexes.Indexable):
    text = indexes.CharField(document=True,use_template=True)

    def get_model(self):
        return News

    def index_queryset(self, using=None):
        return self.get_model().objects.all()
```

4、添加url映射

在主`urls.py`中，添加以下代码：

```
urlpatterns = [
    path('',views.index,name='index'),
    # 添加search的url映射
    path('search/',include('haystack.urls')),
    path('news/', include("apps.news.urls")),
]
```

5、添加模板：

在`templates`文件夹下创建以下结构的目录：

```
templates
    search
        indexes
            news(app的名字)
                news(模型的名字)_text.txt
```

然后在`news_text.txt`中添加需要被索引的字段。示例代码如下：

```
{{ object.title }}
{{ object.content }}
```

接着在`templates`文件夹下创建`search.html`模板文件，`haystack`会自动的在`templates`文件夹下寻找这个模板文件渲染，并且会给这个模板文件传入`page`、`paginator`、`query`等参数。其中`page`和`paginator`分别是`django`内置的`Page`类和`Paginator`类的对象，`query`是查询的关键字。我们可以通过`page.object_list`获取到查找出来的数据。示例代码如下：

```
<ul class="recommend-list">
    {% for result in page.object_list %}
        {% with result.object as news %}
            <li>
                <div class="thumbnail-group">
                    <a href="#">
                        <img src="{{ news.thumbnail }}" alt="">
                    </a>
                </div>
                <div class="news-group">
                    <p class="title">
                        <a href="#">{{ news.title }}</a>
                    </p>
                    <p class="desc">
                        {{ news.desc }}
                    </p>
                    <p class="more">
                       <span class="category">{{ news.caetgory.name }}</span>
                        <span class="pub-time">{{ news.pub_time }}</span>
```

```
                        <span class="author">{{ news.author.username }}</span>
                    </p>
                </div>
            </li>
        {% endwith %}
    {% endfor %}
</ul>
```

6、建立索引：

在项目的根目录下，使用命令`python manage.py rebuild_index`创建索引。 如果不想每次数据增删改查后都要手动的创建索引，可以在`settings.py`中配置：

```
# 增删改查后自动创建索引
HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'
```

7、使用jieba分词替换Whoosh默认的分词：

`Whoosh`默认是采用正则表达式进行分词的。这对于英文来说是足够了。但是对于中文却支持不好。因此我们要替换为`jieba`分词。`jieba`分词是中文分词中最好用的免费的分词库。要使用`jieba`分词库，需要通过`pip install jieba`进行安装。

安装完成后，拷贝`django-env/Lib/site-packages/haystack/backends/whoosh_backend.py`其中的代码，将他放在项目的其他包中，然后创建一个名叫`whoosh_cn_backend.py`文件，把刚刚复制的代码粘贴进去，然后再添加以下代码：

```
import jieba
from whoosh.analysis import Tokenizer, Token

class ChineseTokenizer(Tokenizer):
    def __call__(self, value, positions=False, chars=False,
                 keeporiginal=False, removestops=True,
                 start_pos=0, start_char=0, mode='', **kwargs):
        t = Token(positions, chars, removestops=removestops, mode=mode,
                  **kwargs)
        seglist = jieba.cut(value, cut_all=True)
        for w in seglist:
            t.original = t.text = w
            t.boost = 1.0
            if positions:
                t.pos = start_pos + value.find(w)
            if chars:
                t.startchar = start_char + value.find(w)
                t.endchar = start_char + value.find(w) + len(w)
            yield t

def ChineseAnalyzer():
```

```
    return ChineseTokenizer()
```

然后再将之前的代码中的`analyzer=StemmingAnalyzer()`替换为`analyzer=ChineseAnalyzer()`即可。

第四节 更多教程：

http://django-haystack.readthedocs.io/en/master/tutorial.html

第六章 项目部署：

这里用的是非常干净的`ubuntu 16.04`系统环境，没有使用任何云服务器，原因是因为不同的云服务器环境都不一样。我们就从零开始来完成部署。

第一节 在开发机上的准备工作：

1. 确认项目没有bug。
2. 用`pip freeze > requirements.txt`将当前环境的包导出到`requirements.txt`文件中，方便部署的时候安装。
3. 把`dysms_python`文件准备好。因为短信验证码的这个包必须通过
4. 将项目上传到服务器上的`/srv`目录下。这里以`git`为例。使用`git`比其他上传方式（比如使用pycharm）更加的安全，因为`git`有版本管理的功能，以后如果想要回退到之前的版本，`git`轻而易举就可以做到。
5. 在`https://git-scm.com/downloads`下载`Windows`版本的客户端。然后双击一顿点击下一步安装即可。

    1. 然后使用码云，在码云上创建一个项目。码云地址：https://gitee.com/
    2. 然后进入到项目中，使用以下命令做代码提交：

```
# 初始化一个残酷
* git init
# 添加远程的仓库地址
* git remote add origin xxx.git
# 添加所有的代码到缓存区
* git add .
# 将代码提交到本地仓库
* git commit -m 'first commit'
# 从码云仓库上拉数据下来
* git pull origin master --allow-unrelated-histories
# 将本地仓库中的代码提交到远程服务器的master分支上
* git push origin master
```

在服务器上的准备工作：

1. ubuntu开启root用户：

```
> sudo passwd root
> 然后输入root用户的密码
```

2. 为了方便`xshell`或者`CRT`连接服务器，建议安装`OpenSSH`（一般云服务器上都已经安装了）：

```
sudo apt install openssh-server openssh-client
service ssh restart
```

3. 安装`vim`：

```
sudo apt install vim
```

4. 修改一下`ubuntu`的`apt`源（云服务器一般都有自己的源，可以不用修改），`apt`源是用来安装软件的链接： 先拷贝`/etc/apt/sources.list`为`/etc/apt/sources.list.bak`，然后用`vi`编辑`/etc/apt/sources.list`，删除`sources.list`中的其他内容，将下面代码粘贴到文件中。然后保存：

```
deb-src http://archive.ubuntu.com/ubuntu xenial main restricted
deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial main restricted
multiverse universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main
restricted multiverse universe
deb http://mirrors.aliyun.com/ubuntu/ xenial universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates universe
deb http://mirrors.aliyun.com/ubuntu/ xenial multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates multiverse
```

```
deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted
universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-backports main
restricted universe multiverse
deb http://archive.canonical.com/ubuntu xenial partner
deb-src http://archive.canonical.com/ubuntu xenial partner
deb http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main
restricted multiverse universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-security universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-security multiverse
```

然后更新源：

```
sudo apt update
```

5. 安装`MySQL`服务器和客户端：

```
sudo apt install mysql-server mysql-client
```

6. 安装`memcached`： 通过命令`apt install memcached`即可安装。更多的`memcached`的知识点请参考`memcached`那一章节。
7. 安装好项目要用到的`Python`：

```
* sudo apt install python3
* sudo apt install python3-pip
* pip install --upgrade pip
```

如果系统上已经有`Python3`了，就无需再安装了。因为`supervisor`不支持`Python3`，所以还需要安装`Python2`，如果没有，就安装一下：

```
* sudo apt install python2.7
* sudo apt install python-pip
```

然后输入`python2.7`即可使用了。 如果在输入`pip`的时候提示以下错误：

```
Traceback (most recent call last):
  File "/usr/bin/pip", line 9, in
    from pip import main
ImportError: cannot import name main
```

这是因为`pip 10`的一个`bug`，可以零时使用以下解决方案： 将`/usr/bin/pip`中的：

```
from pip import main
if __name__ == '__main__':
    sys.exit(main())
```

改成：

```
from pip import __main__
if __name__ == '__main__':
    sys.exit(__main__._main())
```

8. 安装`virtualenvwrapper`，并创建好项目要用到的虚拟环境：

```
* pip install virtualenvwrapper
```

安装完`virtualenvwrapper`后，还需要配置`virtualenvwrapper`的环境变量。

- 首先通过`which virtualenvwrapper.sh`命令查看`virtualenvwrapper.sh`文件所在的路径。
- 在当前用户目录下创建`.virtualenv`文件夹，用来存放所有的虚拟环境目录。
- 在当前用户目录下编辑`.bashrc`文件，添加以下代码：

```
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

然后退出`bashrc`文件，输入命令`source ~/.bashrc`。

注意：因为我们是把`virtualenvwrapper`安装在了`python2`的环境中，所以在创建虚拟环境的时候需要使用`--python`参数指定使用哪个`Python`文件。比如我的`python3`的路径是在`/usr/bin/python3`。那么示例代码如下：

```
mkvirtualenv --python=/usr/bin/python3 xfz-env
```

9. 安装`git`：

```
sudo apt install git
```

10. 使用`git`下载项目代码：

```
* git init
* git remote add origin https://gitee.com/hynever/xfz_beike
* git pull origin master
```

11. 进入虚拟环境中，然后进入到项目所在的目录，执行命令：`pip install -r requirements.txt`，安装项目依赖的包。如果提示`OSError: mysql_config not found`，那么再安装`sudo apt install libmysqld-dev`即可。 注意短信验证码的包需要单独安装。把`dysms_python`文件夹上传到项目中，然后进入到这个文件夹中。执行命令：`python setup.py install`。
12. 进入`mysql`数据库中，创建好项目的数据库。
13. 执行`python manage.py makemigrations/migrate`将模型映射到数据库中。
14. 执行`python manage.py runserver 0.0.0.0:8000`，然后在自己电脑上访问这个网站，确保没有BUG。
15. 在`settings.py`中的`ALLOWED_HOST`添加网站的域名或者`ip`地址。
16. 设置`DEBUG=False`，避免如果你的网站产生错误，而将错误信息暴漏给用户。
17. 在`settings.py`中配置`STATIC_ROOT`，用来存储收集的静态文件。收集静态文件的命令如下：

    ```
    python manage.py collectstatic
    ```

安装uwsgi：

uwsgi是一个应用服务器，非静态文件的网络请求就必须通过他完成，他也可以充当静态文件服务器，但不是他的强项。uwsgi是使用python编写的，因此通过`pip3 install uwsgi`就可以了。(uwsgi必须安装在系统级别的Python环境中，不要安装到虚拟环境中)。然后创建一个叫做`uwsgi.ini`的配置文件：

```
[uwsgi]

# 必须全部为绝对路径
# 项目的路径
chdir           = /srv/beike/xfz
# Django的wsgi文件
module          = xfz.wsgi
# Python虚拟环境的路径
home            = /root/.virtualenvs/xfz-env

# 进程相关的设置
# 主进程
master          = true
# 最大数量的工作进程
processes       = 10

http            = :8000

# 设置socket的权限
chmod-socket    = 666
```

```
# 退出的时候是否清理环境
vacuum          = true

daemonize       = /var/log/xfz_uwsgi.log
```

然后通过命令`uwsgi --ini uwsgi.ini`运行，确保没有错误。然后在浏览器中访问`http://ip地址:8000`，如果能够访问到页面（可能没有静态文件）说明`uwsgi`配置没有问题。

安装和配置nginx：

虽然`uwsgi`可以正常的部署我们的项目了。但我们还是依然要采用`nginx`来作为**web服务器**。使用`nginx`来作为web服务器有以下好处：

1. uwsgi对静态文件资源处理并不好，包括响应速度，缓存等。
2. nginx作为专业的web服务器，暴露在公网上会比uwsgi更加安全一点。
3. 运维起来更加方便。比如要将某些IP写入黑名单，nginx可以非常方便的写进去。而uwsgi可能还要写一大段代码才能实现。

安装：

通过`apt install nginx`即可安装。

nginx简单操作命令：

- 启动：service nginx start
- 关闭：service nginx stop
- 重启：service nginx restart
- 测试配置文件：service nginx configtest

添加配置文件：

在`/etc/nginx/conf.d`目录下，新建一个文件，叫做`zhiliaoketang.conf`，然后将以下代码粘贴进去：

```
upstream zhiliaoketang {
    server unix:///srv/zhiliaoketang/zhiliaoketang.sock;
}

# 配置服务器
server {
    # 监听的端口号
    listen      80;
    # 域名
    server_name 192.168.0.101;
    charset     utf-8;

    # 最大的文件上传尺寸
    client_max_body_size 75M;
```

```
    # 静态文件访问的url
    location /static {
        # 静态文件地址
        alias /srv/zhiliaoketang/static_dist;
    }

    # 最后，发送所有非静态文件请求到django服务器
    location / {
        uwsgi_pass  zhiliaoketang;
        # uwsgi_params文件地址
        include     /etc/nginx/uwsgi_params;
    }
}
```

写完配置文件后，为了测试配置文件是否设置成功，运行命令：`service nginx configtest`，如果不报错，说明成功。 每次修改完了配置文件，都要记得运行`service nginx restart`。

使用supervisor管理`uwsgi`进程：

让supervisor管理uwsgi，可以在uwsgi发生意外的情况下，会自动的重启。

安装supervisor：

因为`supervisor`是用`python`写成的，所以通过`pip`即可安装。 并且因为`supervisor`不支持`python3`，因此需要把`supervisor`安装在`python2`的环境中。 `pip2 install supervisor`。

启动：

在项目的根目录下创建一个文件叫做`supervisor.conf`，然后将以下代码填入到配置文件中：

```
    # supervisor的程序名字
    [program:mysite]
    # supervisor执行的命令
    command=uwsgi --ini zlkt_uwsgi.ini
    # 项目的目录
    directory = /srv/zhiliaoketang
    # 开始的时候等待多少秒
    startsecs=0
    # 停止的时候等待多少秒
    stopwaitsecs=0
    # 自动开始
    autostart=true
    # 程序挂了后自动重启
    autorestart=true
    # 输出的log文件
    stdout_logfile=/srv/zhiliaoketang/log/supervisord.log
```

```
    # 输出的错误文件
    stderr_logfile=/srv/zhiliaoketang/log/supervisord.err

    [supervisord]
    # log的级别
    loglevel=debug

    [inet_http_server]
    # supervisor的服务器
    port = :9001
    # 用户名和密码
    username = admin
    password = 123

    # 使用supervisorctl的配置
    [supervisorctl]
    # 使用supervisorctl登录的地址和端口号
    serverurl = http://127.0.0.1:9001

    # 登录supervisorctl的用户名和密码
    username = admin
    password = 123

    [rpcinterface:supervisor]
    supervisor.rpcinterface_factory =
supervisor.rpcinterface:make_main_rpcinterface
```

然后使用命令`supervisord -c supervisor.conf`运行就可以了。 以后如果想要启动`uwsgi`，就可以通过命令`supervisorctl -c supervisor.conf`进入到管理控制台，然后可以执行相关的命令进行管理：

- status # 查看状态
- start program_name #启动程序
- restart program_name #重新启动程序
- stop program_name # 关闭程序
- reload # 重新加载配置文件
- quit # 退出控制台