
顶点云设备管理 开发者文档

发布

Forec

11月 01, 2017

1	用户指南	3
1.1	前言	3
1.2	部署	4
1.3	全局配置	5
1.4	快速上手	8
1.5	框架分析	11
1.6	蓝本介绍	13
1.7	模型介绍	14
1.8	视图函数说明	17
1.9	设备扩展	17
1.10	虚拟设备	19
1.11	Ajax 异步加载	20
1.12	设备管理平台测试	25
2	其他材料	29
2.1	设备管理平台设计要点	29
2.2	顶点云设备管理平台变动记录	30
3		31
3.1		31



欢迎阅读顶点云设备管理平台的开发者文档。

本文档主要分为两部分，首先通过《全局配置》和《快速上手》帮助您了解顶点云设备管理平台的使用和开发，之后详细介绍我们提供的接口。我推荐你按如下顺序阅读：

- [部署](#)
- [全局配置](#)
- [快速上手](#)
- [框架分析](#)
- [蓝本介绍](#)
- [模型介绍](#)
- [视图函数说明](#)
- [设备扩展](#)
- [虚拟设备](#)
- [Ajax 异步加载](#)
- [设备管理平台测试](#)

顶点云设备管理平台使用 [Flask](#) 框架，有关 [Flask](#) 的讨论不在本文档的范围之内，如有兴趣请移步：

- [Flask 文档](#)

此部分先介绍了顶点云设备管理平台的背景，之后通过配置介绍、快速上手分析平台框架，最后介绍平台提供的接口。

1.1 前言

在了解顶点云设备管理系统前请阅读本文。希望本文可以回答您有关顶点云设备管理系统的用途和目的，以及是否应当使用顶点云设备管理系统等问题。

1.1.1 来源

顶点云设备管理系统是我（Forec）和 non1996 在 2016 年网络编程技术的课程设计，最初是为了设计一个简单的面向虚拟设备的管理平台，现在已支持实体设备的管理、监控。

顶点（ZENITH）云设备管理系统的名称与 顶点云存储 相同，均随机取自 GRE 单词。

1.1.2 功能

顶点云设备管理系统最初的设定是提供面向虚拟设备的监控、管理功能，用户可随时掌控设备状态、发出控制指令。目前已支持对实体设备的管理，只需要根据特定的硬件扩展已有的基类即可方便的将设备信息上传到顶点云设备管理系统。

1.1.3 配置

顶点云设备管理系统提供了默认的配置文件 `config.py`，你可以在 [全局配置](#) 查看详细配置说明。

1.1.4 可持续发展

顶点云设备管理系统作为一个非常简单的设备管理平台，目前仍存在非常多的不足。在设计过程中，我们使用了很多非常幼稚的方法来投机取巧，我会在空闲时间逐步修复其中的不足之处、添加新的功能。如果你有兴趣，也欢迎你为顶点云设备管理系统添砖加瓦。你可以向我 [发送邮件](#) 或直接在顶点云设备管理系统的 [GitHub](#) 仓库中提交 [ISSUE](#) 或 [PR](#)。感谢您的支持！

接下来请阅读 [部署](#)。

1.2 部署

顶点云设备管理平台使用 Python3 编写，基于 Flask 框架，在部署前请确保您已经安装 Python3、Pip、Pyenv 等，并已正确设置环境变量。我们推荐的 Python3 版本为 Python 3.5.0。有关 Python3 和 Flask 的介绍不在本文档范围内，如果您尚不了解，请参阅以下相关链接：

- [Python3 快速上手指南](#)
- [Pip 安装指南](#)
- [Pyenv 使用指南](#)
- [Flask 文档](#)

1.2.1 获取源码

顶点云设备管理平台的源码托管在 [GitHub](#) 上，您可以使用 [Git](#) 克隆仓库或直接通过 [GitHub](#) 下载源码的压缩包。假设您熟悉 [Git](#)，请通过以下命令获取源码。

```
git clone https://github.com/Forec/zenith-monitor.git
cd zenith-monitor
```

此时您应当已经进入顶点云设备管理平台的源码目录。

1.2.2 安装第三方支持

顶点云设备管理平台使用到的所有第三方库均包含在需求文件 `requirements.txt` 中。您有两种方式部署。

注：为了减少部署困难，`requirements.txt` 中不包含 [OpenCV](#) 相关库，如果您需要电视播放等功能，请自行安装 [OpenCV](#) 以及 `opencv-python` 以完成 [OpenCV](#) 对 [Python](#) 的支持。我在其它项目中给出过 [OpenCV](#) 的配置过程，您可在 [这里](#) 查看安装过程。

一键部署脚本

顶点云设备管理平台为 [Linux](#) 提供了一键部署脚本，它位于 `settings` 下。您可以运行以下命令。

```
cd settings
./setup.sh
```

如果您使用 [Windows](#) 系统，请参考下方的手动配置，或者如果您使用 [Git Command](#)，可以在 [Git](#) 的 [Bash](#) 命令行中运行 `setup.sh`。

如果您的 [Python](#) 环境工作正常并且网络畅通，您应该可以看到终端中没有提示任何信息并且显示 部署完成 字样。

手动部署

您可以选择手动部署顶点云设备管理平台, 流程如下:

```

mkdir venv
python3 -m venv venv/
source venv/bin/activate // Windows 系统此步骤为 venv/Scripts/activate.
↪bat
pip3 install -r requirements.txt --index-url https://pypi.douban.com/simple
pip3 install gunicorn --index-url https://pypi.douban.com/simple
python3 manager.py simple_init
deactivate

```

如果您的 Python 环境工作正常并且网络畅通, 此时顶点云设备管理平台应当已经部署完毕。

请注意, 默认情况下您安装的顶点云设备管理系统是不需要 **OpenCV** 的, 因此无法提供虚拟电视的播放功能。如果您需要提供电视播放能力的设备, 请将代码 `app/devices.py` 中两处注释掉的代码恢复 (分别位于第 5 行和第 387 行)。

另: `clients` 目录下的模拟测试客户端中默认包含了 **OpenCV** 的头文件, 您需要通过 `pip3 install opencv-python` 来获取第三方支持。或者将所有与 CV 相关的代码注释掉。

1.2.3 运行测试

顶点云设备管理平台提供了一部分单元测试, 您可以运行单元测试以确保环境配置正常。

进入源码所在路径, 运行 `python3 manage.py test`。若测试结果显示通过则顶点云设备管理平台部署成功。

顶点云设备管理平台可运行在任何主流体系结构计算机以及任何操作系统上。接下来请您阅读 [全局配置](#) 根据您的系统配置顶点云设备管理平台。

1.3 全局配置

在阅读以下内容前, 请确保您已经按照 [部署](#) 正确部署了顶点云设备管理平台, 并位于源码目录下 (`/path-to/zenith-monitor/`)。

您可以根据您的环境修改源码目录下的 `config.py` 文件以配置服务器。

1.3.1 样例配置文件

您从 [GitHub](#) 仓库获取的源码中已经包含了一份默认的配置文件的, 这份配置文件中提供了一个基类 `Config`, 去掉注释后它的内容如下:

```

class Config:
    TEMP_PATH = 'temp'
    CLIENT_ADDRESS = '10.201.14.176'
    CLIENT_PORT = 50002
    SECRET_KEY = os.environ.get('SECRET_KEY') or \
        '9d0e91f3372224b3ec7afec2' \
        '4313e745efcf00ba4a5b767b' \
        '35b17834d5f26efac197fd69' \
        'd881dd92e629dbfdc2f1fbf6'
    SQLALCHEMY_COMMIT_ON_TEARDOWN = True
    SQLALCHEMY_TRACK_MODIFICATIONS = True

```

```

ZENITH_MAIL_SUBJECT_PREFIX = '[顶点云设备管理]'
ZENITH_MAIL_SENDER = os.environ.get('ZENITH_MAIL_SENDER') or \
    'cloud-storage@forec.cn'
ZENITH_TEMPFOLDER_LENGTH = 12
ZENITH_INVALID_INFFIX = ['//', '\\', '/', '.', '%', '^', '&',
    '*', '$', '!', '+', '#']
EMAIL_ADMIN = 'forec@bupt.edu.cn'
ZENITH_RANDOM_PATH_ELEMENTS = ['a', 'b', 'c', 'd', 'e', 'f', 'g',
    'h', 'i', 'j', 'k', 'l', 'm', 'n',
    'o', 'p', 'q', 'r', 's', 't', 'u',
    'v', 'w', 'x', 'y', 'z', '1', '2',
    '3', '4', '5', '6', '7', '8', '9',
    '0', 'A', 'B', 'C', 'D', 'E', 'F',
    'G', 'H', 'I', 'J', 'K', 'L', 'M',
    'N', 'O', 'P', 'Q', 'R', 'S', 'T',
    'U', 'V', 'W', 'X', 'Y', 'Z']
ZENITH_VALID_THUMBNAIL = ['.jpg', '.png', '.ico', '.jpeg']
ZENITH_VALID_THUMBNAIL_SIZE = 512 * 1024

```

样例配置文件中，每项均有对应注释，您也可以查看下面的 [样例文件详细解释](#) 了解每一项的具体功能。

1.3.2 样例文件详细解释

配置文件中 *Config* 类的每一项对应配置如下：

1. TEMP_PATH：服务器临时文件存放地址
2. CLIENT_ADDRESS：模拟客户端 IP 地址/域名
3. CLIENT_PORT：模拟客户端开放监听端口
4. SECRET_KEY：用于服务器生成 token 使用的密钥，不可泄露，应当设置在服务器部署系统的环境变量中
5. SQLALCHEMY_COMMIT_ON_TEARDOWN：为 True 时服务器终止运行时向数据库提交变动
6. SQLALCHEMY_TRACK_MODIFICATIONS：为 True 时数据库将追踪服务器对数据的改动
7. ZENITH_MAIL_SUBJECT_PREFIX：服务器向用户发送的验证邮件的主题前缀
8. ZENITH_MAIL_SENDER：服务器向用户发送验证邮件使用的邮箱
9. ZENITH_TEMPFOLDER_LENGTH：服务器在临时存储文件时生成的随机目录名长度
10. ZENITH_INVALID_INFFIX：此列表中的字符不能出现在用户文件名中，否则视为不合法
11. ZENITH_RANDOM_PATH_ELEMENTS：服务器生成的随机路径包含的元素
12. ZENITH_VALID_THUMBNAIL：服务器允许用户上传的头像后缀名
13. ZENITH_VALID_THUMBNAIL_SIZE：服务器允许用户上传的最大头像大小

Config 类仅仅是配置类的基类，你需要扩展此类才可完成配置。在 *config.py* 文件中，存在一些默认的 *Config* 子类，如 *LinuxConfig*、*WindowsConfig*。

下面简单介绍 *LinuxConfig* 并以一个环境为例讲解如何配置。

LinuxConfig 类的内容如下，它是我（Forec）在我的云主机部署设备管理平台时使用的配置类：

```

class LinuxConfig(Config):
    ZENITH_SERVER_ADDRESS = 'cloud-monitor.forec.cn' # 服务器部署的域名/IP地址
    SERVER_NAME = ZENITH_SERVER_ADDRESS

```

```
SQLALCHEMY_DATABASE_URI = 'sqlite:/// ' + os.path.join(basedir, 'work.db')
MAIL_SERVER = 'smtp.exmail.qq.com'
MAIL_PORT = 25 # SSL is 465
MAIL_USE_TLS = True
MAIL_USERNAME = "cloud-storage@forec.cn"
MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
```

可以看出, `LinuxConfig` 类添加了几个新的元素。新元素介绍如下:

1. `ZENITH_SERVER_ADDRESS`: 服务器部署使用的域名/IP地址
2. `SERVER_NAME`: `Flask` 中的 `url_for` 函数使用的服务器名, 通常保持和 `SERVER_NAME` 一致
3. `SQLALCHEMY_DATABASE_URI`: 服务器使用的数据库所在的路径
4. `MAIL_SERVER`: 服务器发送邮件使用的邮箱服务器
5. `MAIL_PORT`: 服务器使用 `smtp` 协议的端口号, 通常为 25。使用 `SSL` 时设置为 465 但这取决于 `MAIL_SERVER` 是否支持 `SSL`
6. `MAIL_USE_TLS`: 是否启用安全连接发送邮件, 通常设置为 `True`
7. `MAIL_USERNAME`: 服务器发送邮件使用的邮箱帐号, 通常和 `ZENITH_MAIL_SENDER` 保持一致
8. `MAIL_PASSWORD`: 服务器发送邮件使用的邮箱帐号的密码, 通常保存在环境变量中

下面通过一个实例环境解释如何配置。

例如, 在安装 `Ubuntu 16.04` 的主机上部署顶点云设备管理平台, 可参考的配置文件如下 (使用扩展类):

```
class MyConfig(Config):
    ZENITH_SERVER_ADDRESS = 'myaddress.my.io' # 自定义的域名, 你需要先购买此域名并且映射到部署主机上
    SERVER_NAME = ZENITH_SERVER_ADDRESS
    SQLALCHEMY_DATABASE_URI = '/usr/local/cloud-monitor/mydb.db'
        # 设置数据库为 /usr/local/cloud-monitor/mydb.db
    MAIL_SERVER = 'smtp.163.com' # 使用 163 邮箱
    MAIL_PORT = 25
    MAIL_USE_TLS = True
    MAIL_USERNAME = "mycloud-monitor@163.com"
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD') or '123456'
        # 在环境变量中添加密码, 若环境变量未找到对应值则使用 123456
```

1.3.3 添加自定义配置类到表

在已经定义了自定义配置类后, 你需要将自定义配置类添加到表驱动中以使 `工厂方法` 能根据我的配置类生成服务器实例。

在 `config.py` 中, 有一个名为 `config` 的字典如下:

```
config = {
    'development': DevelopmentConfig, # 开发环境
    'linux': LinuxConfig, # 提供的 Linux 模板环境
    'windows': WindowsConfig, # 提供的 Windows模板环境
    'testing': TestingConfig, # 测试环境
    'default': DevelopmentConfig # 默认为开发环境
}
```

你需要添加自己的自定义配置类到此表中, 如添加 `'myconfig': MyConfig`。之后, 修改 `manage.py` 中的第 21 行 `app = create_app('default')` 为 `app = create_app('myconfig')` 即可。

接下来请您阅读 [快速上手](#)。

1.4 快速上手

此部分文档将带领您在一个假想的纯净环境中部署、配置、扩展自定义功能并启动顶点云设备管理平台。

1.4.1 假想环境

有一天我意外获得了一台免费的 CVM，而我恰巧获得了一次得到顶点云设备管理平台源码的机会。这台云主机使用的操作系统为 CentOS 7.2，公网 IP 地址为 123.123.123.123，已安装好 Python3、Pip 以及 Pvenv 并配置了环境变量。下面的指令均通过 SSH 远程操作。

```
git clone https://github.com/Forec/zenith-monitor.git
cd zenith-monitor/
```

我已经获得了顶点云设备管理平台的源码，接下来使用一键配置脚本部署环境。

```
cd settings
./setup.sh
```

很高兴看到配置脚本通知我部署完成。接下来测试一下代码是否能够在本地机器通过测试。

```
cd ..
source venv/bin/activate # Windows 下请执行 venv/Scripts/activate.bat
python manage.py test
```

非常顺利！测试脚本告诉我所有测试均已完成，顶点云设备管理平台各个基础模块能够在这台服务器上运转正常。

1.4.2 针对假想环境修改配置文件

鉴于顶点云设备管理平台提供的默认配置仅适用于 Forec 的史诗级笔记本，下面根据这台服务器的情况修改配置文件。编辑 `config.py`：

```
nano config.py
```

我拷贝了一份 LinuxConfig 并重命名该子类为 MyConfig，然后根据如下考虑对 MyConfig 做了一定修改：

- 我觉得顶点云设备管理平台默认使用的 **SQLITE** 数据库很方便，并且放置在源码根目录下也没什么问题，因此我决定保留默认配置中的 `SQLALCHEMY_DATABASE_URI`
- 我想让世界上任何一个角落均能访问我的顶点云设备管理平台，因此我修改 `ZENITH_SERVER_ADDRESS = '123.123.123.123'`
- 我要让顶点云设备管理平台用我的邮箱发送认证邮件。我的邮箱是 `mymail@gmail.com`，因此我修改如下部分：
 - `MAIL_SERVER = 'smtp.gmail.com'`
 - `MAIL_PORT = 25`
 - `MAIL_USE_TLS = True`
 - `MAIL_USERNAME = "mymail@gmail.com"`

- 我觉得邮箱的密码还是不要放在代码中比较好, 因此我向环境变量添加了 `MAIL_PASSWORD` 值并保留了 `MAIL_PASSWORD` 的设置

看起来配置文件没什么值得修改的了, 我决定按下 `CTRL+X` 保存配置文件, 顺便检查一下新定义的配置类:

```
class MyConfig(Config):
    ZENITH_SERVER_ADDRESS = '123.123.123.123' # 服务器部署的域名/IP地址
    SERVER_NAME = ZENITH_SERVER_ADDRESS
    SQLALCHEMY_DATABASE_URI = 'sqlite:/// ' + os.path.join(basedir, 'work.db')
    MAIL_SERVER = 'smtp.gmail.com'
    MAIL_PORT = 25 # SSL is 465
    MAIL_USE_TLS = True
    MAIL_USERNAME = "mymail@gmail.com"
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
```

1.4.3 添加自定义类到表驱动

我决定按照 添加自定义配置类到表 中的说法将我的自定义配置类添加到表驱动中。

向 `config.py` 的 `config` 字典中添加 `'myconfig': MyConfig` 后如下:

```
config = {
    'development' : DevelopmentConfig,      # 开发环境
    'linux': LinuxConfig,                  # 提供的 Linux 模板环境
    'windows': WindowsConfig,              # 提供的 Windows模板环境
    'testing' : TestingConfig,              # 测试环境
    'default' : DevelopmentConfig,          # 默认为开发环境
    'myconfig' : MyConfig                   # 自定义添加的配置类
}
```

之后修改 `manage.py` 的第 21 行为:

```
app = create_app('myconfig')
```

1.4.4 启动服务器

顶点云设备管理平台可通过两种方式启动。我们推荐使用 `settings` 目录下的启动脚本, 启动脚本使用 `unicorn` 能够提高服务器的并发能力。

一键启动

`settings` 目录提供了顶点云设备管理平台的启动脚本, 您可以运行 `run.sh` (Linux 系统) 或 `run.bat` (Windows 系统) 来启动服务器。默认会开启在本机 (127.0.0.1) 的 5001 端口。您可以修改启动脚本中的 IP 地址和端口号。

手动启动

您也可以选择手动控制服务器的启动。通常在 `Debug` 情况下使用此方式, 因为 `Flask` 对并发请求的原生支持并不很令人满意。

```
source venv/bin/activate      # Windows 下请执行 venv/Scripts/activate.bat
python manage.py runserver    # 您可以指定 -h 和 -p 参数, 分别代表开放服务器的IP 地址和端口号
```

现在您可以从本机的浏览器访问您的服务器了。

1.4.5 启动模拟设备客户端

顶点云设备管理平台的模拟设备客户端位于 `clients` 目录下, 您可以通过 `python simulator.py` 启动。

默认情况下, 虚拟设备客户端引用了 `OpenCV`, 但我们的安装教程中未给出 `OpenCV` 相关相关库, 您可以参考 [这里](#) 来安装、配置 `OpenCV`。

如果您不需要电视播放等功能, 可以将 `clients/models.py` 中与 `OpenCV` 有关的代码注释掉 (注释头文件中的 `cv2` 和整个 `TV` 类), 那么您无需安装 `OpenCV`。

更详细的模拟设备文档可参考 [虚拟设备](#), 此处不过多介绍。

1.4.6 扩展自定义功能

不得不说 `Forec` 的设计实在是太简陋了, 为什么用户无法注册! 幸好我学习过 `Flask` 框架, 也许我应该自己添加这个功能?

在阅读了 [框架分析](#) 后, 我了解了整个顶点云设备管理平台的结构, 下面我准备添加这个简单的功能。

进入 `app/auth` 目录并编辑 `views.py`:

```
cd app/auth
nano views.py
```

我在源码的 79 行发现了一句注释, 原来默认的顶点云设备管理平台提供了注册接口, 但将注册部分屏蔽掉了, 反馈给用户的仅仅是展示界面。注册的视图函数如下所示。

```
@auth.route('/register', methods = ['GET', 'POST'])
def register():
    # 展示状态, 禁止注册
    return render_template('auth/test.html')
#
# if request.method == 'GET':
#     if current_user.is_authenticated:
#         flash('您已经登录, 无需注册! ')
#         return redirect(url_for('main.home'))
#     return render_template('auth/register.html')
#
# else:
#     req = request.form.get('request')
#     # .....
#     # .....
```

我决定开放注册接口, 因此我将被注释的部分取消注释, 将视图函数中的第一句 `return` 删除。

```
@auth.route('/register', methods = ['GET', 'POST'])
def register():
    # 展示状态, 禁止注册
    # return render_template('auth/test.html')
    if request.method == 'GET':
        if current_user.is_authenticated:
            flash('您已经登录, 无需注册! ')
            return redirect(url_for('main.home'))
        return render_template('auth/register.html')
    else:
        req = request.form.get('request')
        if req is None:
```

```

        return jsonify({
            'code': 0    # 没有请求
        })
    req = json.loads(req)
    email = req.get('email')
    password = req.get('passwd')
    password2 = req.get('passwd2')
    nickname = req.get('nickname')
    if email is None or password is None or \
        password2 is None or nickname is None or \
        not verify_email(email) or not verify_nickname(nickname) or \
        password != password2:
        return jsonify({
            'code': 1    # 填写格式不对
        })
    user1 = User.query.filter_by(email = email).first()
    if user1 is not None:
        return jsonify({
            'code': 2    # 邮箱已被注册
        })

    user2 = User.query.filter_by(nickname = nickname).first()
    if user2 is not None:
        return jsonify({
            'code': 3    # 此昵称已被注册已被注册
        })
    user = User(email = email,
                nickname = nickname,
                password = password)
    db.session.add(user)
    db.session.commit()
    token = user.generate_confirmation_token()
    send_email(user.email,
               '确认您的帐户',
               'auth/email/confirm',
               user=user,
               token=token)
    flash('一封确认邮件已经发送到您填写的邮箱, '
          '请查看以激活您的帐号')
    login_user(user)
    return jsonify({
        'code': 4
    })

```

我重新启动了服务器，现在注册接口已经打开。

接下来请您阅读 [框架分析](#)。

1.5 框架分析

此部分文档主要介绍顶点云设备管理平台的框架结构。

顶点云设备管理平台源码文件结构如下：

```

- zenith-monitor
  - app

```

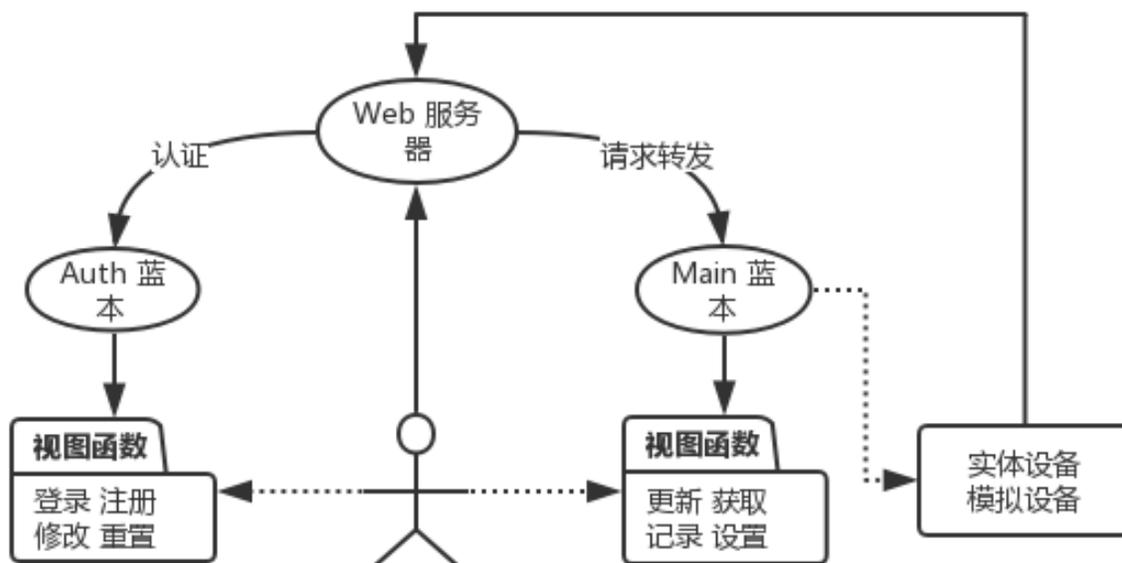
```

- auth
- main
- static
  - thumbnail
- templates
- ...
  - devices.py
- models.py
- settings
- config.py
- manage.py
- work.db
    
```

以上几个目录对应的功能如下:

- *auth*: 对应 *Auth* 蓝本, 用于处理用户注册、登录等需要特权的请求。
- *main*: 对应 *Main* 蓝本, 用于处理用户大部分不需要特权的请求, 以及主要的功能实现。
- *static*: 对应 Flask 框架中存储静态文件的目录, 此目录存储设备管理平台使用的图片、js、css等文件。
- *static/thumbnail*: 此目录存储用户自定义的头像文件。
- *template*: 此目录存储设备管理平台渲染网页使用的模板。
- *models.py*: 对应 [模型介绍](#) 中介绍的各种模型。
- *devices.py*: 对应 [设备扩展](#) 中介绍的各种设备模型。
- *config.py*: 对应 [全局配置](#) 中的配置文件。
- *manage.py*: 管理服务器的配置文件, 包含了一系列自定义命令, 你可以查看 [管理器](#) 以了解更多。

各模块之间关系如下图所示:



1.5.1 管理器

`manage.py` 管理着顶点云设备管理平台。它提供了很多操作服务器的基础命令，如启动、初始化、测试等，你可以通过 `python manage.py <command>` 来运行不同命令。默认的顶点云设备管理平台管理器提供的命令有：

- `shell`：启动交互式 Python 命令行并自动导入顶点云设备管理平台的各类模块
- `db`：数据库迁移类命令，如 `db migrate`、`db init` 等
- `test`：启动 `tests` 目录下的单元测试
- `init`：初始化数据库，随机生成用户和数据
- `simple_init`：简单初始化数据库，只加入五个特定的用户

1.5.2 工厂方法

`app/__init__.py` 中包含了设备管理平台的工厂方法，当你通过 `manage.py` 执行命令时，`manage.py` 会调用工厂方法生成一个应用实例。

工厂方法名为 `create_app`，正如我们在 [启动模拟设备客户端](#) 中介绍的那样，修改 `manage.py` 的第 21 行实际是修改了工厂方法的参数。这个工厂方法根据传入的参数从 [全局配置](#) 中寻找对应表项（配置类），并使用该配置类生成服务器实例。

接下来请您阅读 [蓝本介绍](#)。

1.6 蓝本介绍

此部分文档主要介绍顶点云设备管理平台中的两个蓝本：`main` 以及 `auth`，以及如何创建新的自定义蓝本。

1.6.1 Auth 蓝本

此蓝本主要用于授权用户，如注册、登录、重置密码等。蓝本位于 `app/auth` 目录下，结构为：

```
- auth
  - __init__.py
  - forms.py
  - views.py
```

上面三个文件中，`__init__.py` 用于初始化蓝本 `Auth`，`forms.py` 定义了 `Auth` 蓝本视图函数需要使用的表单类，`views.py` 具体处理转发给 `Auth` 蓝本的请求。

关于 `views.py` 中的视图函数，请查阅 [Auth 蓝本视图函数](#)。

1.6.2 Main 蓝本

此蓝本主要用于处理用户服务请求。蓝本位于 `app/main` 目录下，结构为：

```
- main
  - __init__.py
  - forms.py
  - views.py
  - errors.py
```

上面三个文件中, `__init__.py` 用于初始化蓝本 `Main`, `forms.py` 定义了 `Main` 蓝本视图函数需要使用的表单类, `views.py` 具体处理转发给 `Main` 蓝本的请求, `errors.py` 处理用户遇到的错误并使用服务器模板渲染错误界面。

对于 `views.py` 中的视图函数, 请查阅 [Main 蓝本视图函数](#)。

1.6.3 自定义蓝本

关于蓝本的概念不属于本文档讨论范围, 如果您尚不了解, 可以参考 [Flask 文档](#) 中关于蓝本的部分。

对于顶点云设备管理平台, 如果您需要添加自定义蓝本, 请在目录 `app` 下创建新的蓝本目录并添加您的自定义路由。

接下来请您阅读 [模型介绍](#)。

1.7 模型介绍

此部分文档主要介绍顶点云设备管理平台使用的数据库格式和模型。

1.7.1 数据库

顶点云设备管理平台默认配置使用 `SQLITE`, 您可以修改配置文件以选择适合您机器的数据库。顶点云设备管理平台的源码目录中提供了一个默认的 `work.db` 数据库文件, 如果您选择使用默认配置, 该文件足以满足您的需求。

顶点云设备管理平台的数据库包含 3 张基本的表, 根据拓展设备的不同, 继承的子表也不同:

- `cuser`: 用户模型表
- `BasicDevices`: 用户管理设备记录表
- `records`: 服务器存储设备变动记录表

表 `cuser`

此表用于存储用户信息, 表格式如下:

uid	email	password_hash	created	confirmed	nickname	avatar_hash
INTEGER PRIMARY KEY AUTOINCREMENT	VAR-CHAR(64)	VAR-CHAR(32)	DATE	BOOLEAN	VAR-CHAR(64)	VAR-CHAR(32)
用户编号	用户邮箱	用户密钥 md5	用户创建日期	是否激活	设备列表	头像链接
about_me	last_seen	member_since	token_hash	interval	devices	monitor_url
VARCHAR(256)	DATE	DATE	VAR-CHAR(32)	INTEGER	外链	VAR-CHAR(256)
用户介绍	上次登录日期	用户注册时间	用户 TOKEN	刷新间隔	一对多表	视频监控源

表 BasicDevices

此表用于存储设备均具有的基本信息，表格式如下：

uid	ownerid	code	path	interval	created
INTEGER PRIMARY KEY AUTOINCREMENT	INTEGER	VAR-CHAR(12)	VAR-CHAR(64)	INTEGER	DATE
设备编号	资源所有者编号	设备唯一标识码	设备名称	设备刷新间隔	创建时间
name	about	temperature	volume	current	power
VARCHAR(64)	VAR-CHAR(256)	INTEGER	INTEGER	INTEGER	INTEGER
设备名称	设备介绍	设备温度	设备电流	设备电流	设备功率

表 records

此表用于存储设备变动记录，表格式如下：

id	device_id	status	created
INTEGER PRIMARY KEY AUTOINCREMENT	INTEGER	VARCHAR(256)	DATE
变动记录编号	变动设备编号	变动信息 JSON 值	变动时间

1.7.2 内置自定义类

此部分文档主要介绍顶点云设备管理平台定义的几个模型。用户模型的定义均位于 `app/models.py` 中，包括：

- *User*：用户类
- *Device*：基本设备类
- *Record*：记录类
- *AnonymousUser*：匿名用户模型，提供给 `flask_login` 作为未登录用户实例

下面将简单介绍每个模型提供的方法。

用户类

User 模型同时继承了 *UserMixin* 和 SQLAlchemy 数据库模型，它具有如下元素：

```
# 用户 id
uid = db.Column(db.Integer, primary_key=True)
email = db.Column(db.String(64), unique=True)
# 用户密码的加盐哈希值
password_hash = db.Column(db.String(32))
# 用户创建时间
created = db.Column(db.DateTime, default = datetime.utcnow)
# 用户是否已激活邮箱
confirmed = db.Column(db.Boolean, default= False)
# 用户昵称
nickname = db.Column(db.String(64))
# 用户头像链接
avatar_hash = db.Column(db.String(32))
# 用户个人介绍
```

```

about_me = db.Column(db.Text)
# 与 created 相同, adapter
member_since = db.Column(db.DateTime,
                          default = datetime.utcnow)
# 上次登录时间
last_seen = db.Column(db.DateTime,
                      default = datetime.utcnow)
# 用户拥有的设备, 外链设备表
devices = db.relationship('Device',
                          backref='owner',
                          lazy = 'dynamic')#,
                          #enable_typechecks=False)

```

User 类具有如下方法:

- *get_id*: 获取用户 id
- *verify_password*: 验证密码是否正确
- *generate_confirmation_token*: 生成用户邮箱验证 token
- *generate_email_change_token*: 生成修改邮箱 token
- *generate_reset_token*: 生成重置密码 token
- *generate_resetToken_token*: 生成重置 token 的 token
- *reset_password*: 用户验证重置密码的 token
- *confirm*: 用户验证邮箱激活的 token
- *reset_token*: 用户验证重置 token 函数
- *gravatar*: 获取用户头像链接, 若存在自定义头像则返回自定义头像链接, 否则从 gravatar 获取
- *ping*: 更新用户最近登录时间

可以看出, *User* 类的多数方法都用于处理需要鉴别用户身份的请求, 包括生成 token、验证 token 以及在验证通过后执行相应的处理。

详细方法的参数请查看 *app/models.py*, 代码中给出了详细的注释。

基础设备 *Device* 类

Device 类是所有设备的基类, 提供了如下方法:

- *setup()*: 启动设备
- *shutdown()*: 关闭设备
- *getStatus()*: 获取设备状态, DICT 格式返回
- *updateStatus(status)*: 根据参数更新设备状态
- *verify_status(jsondata)*: 根据参数判断设备是否需要更新
- *setStatus(jsondata)*: 一个闭包函数, 将用户控制指令发送给远程设备

接下来请您阅读 [视图函数说明](#)。

1.8 视图函数说明

此部分文档简要介绍顶点云设备管理平台提供的视图函数，将按 *Auth* 蓝本 和 *Main* 蓝本 分成两个部分。

1.8.1 Auth 蓝本视图函数

此部分视图函数来自 *Auth* 蓝本，位于文件 `app/auth/views.py` 中。共包含如下视图函数：

视图函数名	功能
<code>rules</code>	提供了“注册须知”界面的入口
<code>login</code>	提供了登录界面入口
<code>logout</code>	提供了登出操作，登出后默认重定向到登陆入口
<code>register</code>	提供了注册入口
<code>confirm</code>	提供了用户注册邮箱激活入口，根据激活链接尾部的 <code>token</code> 校验用户是否合法
<code>resend_confirmation</code>	用于在用户未收到激活邮件时重发
<code>change_email_request</code>	为用户重置邮箱请求入口
<code>change_email</code>	用于验证用户重置邮箱后的激活链接
<code>password_reset</code>	用于验证用户重置密码请求 <code>token</code> 的合法性
<code>secure_center</code>	返回安全中心界面
<code>before_request</code>	注册了用户未激活邮箱时的跳转接口
<code>unconfirmed</code>	提供了未验证的界面

1.8.2 Main 蓝本视图函数

此部分视图函数来自 *Main* 蓝本，位于文件 `app/main/views.py` 中。共包含如下视图函数：

视图函数名	功能
<code>index</code>	提供了顶点云设备管理平台介绍界面的入口
<code>home</code>	服务器主页入口点，也是所有设备管理列表界面
<code>user</code>	服务器用户（编辑）资料界面入口
<code>update_status</code>	为模拟设备/实体设备提供上传状态入口（仅 POST）
<code>show_status</code>	Ajax 异步获取设备状态入口（仅 POST）
<code>set_interval</code>	用户设置刷新周期入口（仅 POST）
<code>device</code>	具体设备详细界面
<code>delete_device</code>	为用户删除设备入口（仅 POST）
<code>edit_device</code>	为用户编辑设备入口（仅 POST）
<code>newdevice</code>	为用户创建新设备入口（仅 POST）
<code>reset_token</code>	用户重置 Token 入口
<code>set_device</code>	用户设置设备状态入口（仅 POST）
<code>show_history</code>	设备历史记录界面入口
<code>get_history</code>	Ajax 异步获取设备历史入口（仅 POST）

接下来请您阅读 [设备扩展](#)。

1.9 设备扩展

此部分主要介绍顶点云设备管理系统服务器默认支持的设备，以及如何针对新的实体设备做扩展。

1.9.1 命令发送

对远程设备的控制通过和远程设备上运行的监控程序交互实现，服务器和远程设备的通信通过新建一个线程向远程设备发送自定义协议格式的数据包来完成。

线程 `RequestThread` 用于实现这一功能，它位于 `app/models.py` 中，每次执行只发送作为参数传入的 JSON 格式数据。具体发送格式如下：

```
使用 struct 打包消息长度 | 消息体, 使用 pickle 打包
```

远程设备上运行的监控程序有一个监听线程用于监听进入的连接请求，并将其转发到对应设备上。具体的远端设备如何实现请看 [虚拟设备](#)。

1.9.2 基本设备

文件 `app/models.py` 中定义了设备的基类，每个设备都拥有如 [基础设备 `Device` 类](#) 中描述的方法，以及 `dm-models-database-devices` 中描述的基本属性。

下面对其中几个方法加以说明。

- `setup()` 和 `shutdown()`：对远程设备执行开/关指令，它们会使用 `RequestThread` 发送 JSON 格式的数据包，数据包中分别包含 `setup: 1` 或者 `shutdown: 1`。
- `getStatus()`：获取设备在服务器存储的最近一次的状态，以字典形式返回。
- `updateStatus(status)`：对于基类中的此方法而言，传入的 `status` 是一个字典格式的数据，基类通过这个字典来更新所有设备都具有的信息；对于子类而言（具体设备），传入的 `status` 是一个字符序列化的 JSON 格式数据，可以将其转化为 JSON 对象，并通过父类方法设置公有的状态，再细化自身的属性。
- `verifyStatus(status)`：传入的是一个字典，设备将自身当前状态和字典中的项对比，如果相同则返回 `True`，否则返回 `False`。此函数的作用在于用户误设置设备时，如果设备状态未变更则不向远端设备发送消息，减轻远端设备负担。
- `setStatus(jsondata)`：传入的是一个字符序列化的 JSON 对象，这个 JSON 对象只包含了需要修改的项的值。`setStatus` 函数会将空缺项补全为当前状态，并通过 `RequestThread` 发送给远端设备，命令远端设备更新。

1.9.3 照明设备

下面以照明设备为例，就每个具体方法分别介绍子类设备具体如何实现。

照明设备为类 `Bulb`，定义在文件 `app/devices.py` 中。照明设备扩展了两个字段：亮度等级和灯光饱和度，并且其开/关指令的发送只需要复用父类方法。

- `getStatus()`：首先调用父类的 `getStatus` 函数获得基本属性字典，之后向字典中添加项 `lightDegree` 和 `full`，分别代表灯光亮度等级和饱和度。返回新字典。此函数位于 `app/devices.py` 的第 25 行。
- `updateStatus(status)`：首先通过 `json.loads()` 加载 JSON 对象，之后调用父类的 `updateStatus()` 方法更新基本属性。最后从 JSON 对象中提取 `lightDegree` 和 `full` 属性，检查并更新提取成功的项，忽略不存在的项或非法项。此函数位于 `app/devices.py` 的第 32 行。
- `verifyStatus(status)`：首先通过父类的 `verifyStatus` 判断基本属性是否有改动，之后再就具体的 `lightDegree` 和 `full` 提取并判断。
- `setStatus(jsondata)`：与父类方法基本相同，但发送给远端设备的 JSON 对象中增加了 `lightDegree` 和 `full` 属性。此函数位于 `app/devices.py` 的第 79 行。

1.9.4 其他设备

默认的顶点云设备管理系统支持模拟照明、模拟空调、模拟电视和实体 PC。

- 模拟照明: 类 *Bulb*
- 模拟电视: 类 *TV*
- 模拟空调: 类 *Air*
- 实体 PC: 类 *PC*

接下来请您阅读 [虚拟设备](#)。

1.10 虚拟设备

此部分主要介绍顶点云设备管理系统用于测试的虚拟设备客户端。虚拟设备客户端位于目录 *clients* 下。请注意，默认的顶点云设备

1.10.1 基础虚拟设备

虚拟设备客户端的结构为

```
- clients
  - config.py
  - encoder.py
  - models.py
  - raspberry.py
  - simulator.py
  - tvs
  - device.txt
```

其中 *config.py* 用于配置虚拟客户端和服务器交互的相关信息，*encoder.py* 是用来传输电视实时画面的编码模块，*models.py* 包含了所有虚拟设备平台提供的设备，*raspberry.py* 是一份能够监控树莓派的样例代码，*simulator.py* 包括虚拟设备管理器和模拟所需的动作。

1.10.2 配置

config.py 中包括以下内容:

- *SERVER_IP*: 服务器部署的 IP 地址或域名
- *UPLOAD_URL*: 服务器预留的用于设备上传状态的入口点
- *PORT*: 虚拟客户端监听的端口，用于接收服务器发来的控制信息

1.10.3 编码器

- *NumpyEncoder*: 将 Numpy 中的 Array 转为 JSON 对象序列化
- *json_numpy_obj_hook*: 回调钩子

1.10.4 模拟器

- 类 *WorkThread* : 用于处理服务器发送的设置请求, 每当服务器发送来一个请求, 模拟器的监听模块会创建一个新的 *WorkThread* 来处理这个请求。
- 类 *ListenThread* : 模拟客户端的监听模块, 监听 *PORT* 端口。
- 类 *Manager* : 模拟器, 从 *device.txt* 中读取配置信息并创建虚拟设备。

模拟器启动过程如下: 从配置文件读取用户的 *Token* 和室内温度, 以及每个设备的编码、类型、汇报间隔和工作状态 (是否开启)。

device.txt 的格式如下:

```
9490544C18C15B21286685B41F825684, 20
E1A9013A447E, Bulb, 5, on
BA8120601307, Bulb, 3, on
CE683231033B, TV, 3, on
EFID2141FJKD, Air, 4, on
LI29F2MV9D7Z, Bulb, 6, on
```

上面的配置文件中, 第一行为用户的 *Token* 和室温, 用逗号分隔。之后多行每行为一个设备。

每个设备需要配置四项: 设备的编码 (12位, 用户创建设备时由服务器生成), 设备类型 (参考 [设备扩展](#))、设备上传状态间隔和设备的工作状态 (*on* 或者 *off*)。

注: 对于模拟电视设备, 模拟电视画面所用的视频文件为 *.mp4* 格式, 并且放置在 *clients/tvs* 目录下, 文件名为数字 (对应的频道)。

1.10.5 扩展设备

要实现自己的虚拟设备, 您需要继承基类 *Basic*, 并参考几个已有的虚拟设备分别实现 *initialize()*、*getStatus()*、*set()* 和 *change()*。

接下来请您阅读 [Ajax 异步加载](#)。

1.11 Ajax 异步加载

此部分文档主要介绍顶点云设备管理平台的 *Ajax* 异步加载。此部分文档由 [non1996](#) 编辑。

顶点云设备管理平台通过 *DevExtreme* 框架实现网页操控设备, 定时获取设备状态, 及设备状态可视化等功能。

1.11.1 JavaScript 图表模型定义及功能

折线图模型

折线图用于显示设备电流值或 *CPU* 利用率的变化情况。以电流变化情况为例:

```
dxChart ({
  dataSource: line_chart_data_source,
  commonSeriesSettings: {
    argumentField: "id",
    point: { visible: true, size: 5, hoverStyle: {size: 7, border: 0, color:
↪ 'inherit'} },
```

```

        line: {width: 1, hoverStyle: {width: 1}}
    },
    series: [
        { valueField: "part1", name: "电流", color: "#68b828" }
    ],
    legend: {
        position: 'inside',
        paddingLeftRight: 5
    },
    commonAxisSettings: {
        label: {
            visible: false
        },
        grid: {
            visible: true,
            color: '#f9f9f9'
        }
    },
    valueAxis: {
        max: 25
    },
    argumentAxis: {
        valueMarginsEnabled: false
    },
    }).data('iCount', i);

```

柱状图模型

柱状图用于显示电压或内存使用的变化情况:

```

dxChart({
    dataSource: [
        {id: ++i, sales: 1},
        {id: ++i, sales: 2},
        {id: ++i, sales: 3},
        {id: ++i, sales: 4},
        {id: ++i, sales: 5},
        {id: ++i, sales: 4},
        {id: ++i, sales: 5},
        {id: ++i, sales: 6},
        {id: ++i, sales: 7},
        {id: ++i, sales: 6},
        {id: ++i, sales: 5},
        {id: ++i, sales: 4},
        {id: ++i, sales: 5},
        {id: ++i, sales: 4},
        {id: ++i, sales: 4},
        {id: ++i, sales: 3},
        {id: ++i, sales: 4},
    ],
    series: {
        argumentField: "id",
        valueField: "sales",
        name: "Sales",
        type: "bar",
        color: '#7c38bc'
    }
});

```

```

    },
    commonAxisSettings: {
      label: {
        visible: false
      },
      grid: {
        visible: false
      }
    },
    legend: {
      visible: false
    },
    argumentAxis: {
      valueMarginsEnabled: true
    },
    valueAxis: {
      max: 12
    },
    equalBarWidth: {
      width: 11
    }
  })
  .data('iCount', i);

```

饼图模型

饼图根据不同设备显示如室内温度，电灯饱和度等信息。

```

dxPieChart({
  dataSource: doughnut1_data_source,
  tooltip: {
    enabled: false,
    format: "millions",
    customizeText: function() {
      return this.argumentText + "<br/>" + this.valueText;
    }
  },
  size: {
    height: 90
  },
  legend: {
    visible: false
  },
  series: [{
    type: "doughnut",
    argumentField: "region"
  }],
  palette: ['#5e9b4c', '#6ca959', '#b9f5a6'],
});

```

股价图模型

股价图用于显示设备温度变化。

```

dxChart({
  dataSource: realtime_network_stats,

```

```

        commonSeriesSettings: {
            type: "area",
            argumentField: "id"
        },
        series: [
            { valueField: "x1", name: "Packets Sent", color: '#7c38bc', ↵
↵opacity: .4 },
            { valueField: "x2", name: "Packets Received", color: '#000', ↵
↵opacity: .5},
        ],
        legend: {
            verticalAlignment: "bottom",
            horizontalAlignment: "center"
        },
        commonAxisSettings: {
            label: {
                visible: false
            },
            grid: {
                visible: true,
                color: '#f5f5f5'
            }
        },
        legend: {
            visible: false
        },
        argumentAxis: {
            valueMarginsEnabled: false
        },
        valueAxis: {
            max: 500
        },
        animation: {
            enabled: false
        }
    }).data('iCount', i);

```

仪表盘模型

```

dxCircularGauge({
    scale: {
        startValue: 0,
        endValue: 100,
        majorTick: {
            tickInterval: 25
        }
    },
    rangeContainer: {
        palette: 'pastel',
        width: 3,
        ranges: [
            { startValue: 0, endValue: 25, color: "#68b828" },
            { startValue: 25, endValue: 50, color: "#faff01" },
            { startValue: 50, endValue: 75, color: "#ffb801" },
            { startValue: 75, endValue: 100, color: "#ff3201" },
        ],
    },

```

```

    },
    value: 100,
    valueIndicator: {
      offset: 10,
      color: '#7c38bc',
      type: 'triangleNeedle',
      spindleSize: 12
    }
  });

```

旋钮模型

旋钮主要用于控制设备属性，如电视音量，空调温度等。

```

$(".knob").knob({
  max: 300,
  min: 0,
  thickness: .3,
  fgColor: '#40bbea',
  bgColor: '#f0f0f0',
  'release': function(e) {
    //action
  }
});

```

1.11.2 各设备页面主要 JavaScript 脚本

页面共有函数

- *getAjaxFromServer()*：以用户密钥，用户邮箱，设备号为参数，向服务器发送 Ajax 请求，获取对应设备的实时信息并调用设置函数改变数据和图表的显示。
- *myElectricityChange(range)*：参数为当前电流值，向设备电流图添加新的样值。
- *myVoltageGraphChange(range)*：参数为当前电压值，向设备电压图添加新的样值。
- *myEleAndVolChange(vol, ele, pow)*：参数为当前电压、电流、功率值，调用函数 *myElectricityChange* 和 *myVoltageGraphChange* 对电流和电压以及功率值进行设置。
- *myDegreeStateChange(range)*：参数为当前设备温度，向设备温度图添加新的样值。
- *closeEquipment(type)*：参数为按钮状态。响应函数，用户点击开启/关闭按钮时调用，向服务器发送ajax请求，执行对设备的开/关操作。
- *upperEquipmentInterval()*：增加设备汇报状态周期。
- *lowerEquipmentInterval()*：降低设备汇报状态周期。

照明设备

- *mySaturabilityChange(range)*：参数为当前光线饱和度值，设置饱和度及其饼图。
- *setBrightness(up)*：参数为用户点击按钮的类型，响应函数，用户点击设置电灯亮度的按钮时调用，向服务器发出ajax请求以改变电灯亮度。

空调

- *switchSpeed(up)*: 参数为按键类型。响应函数, 向服务器发出ajax请求改变空调风速, 风速范围 1-5。
- *lowerPowerConsumption()* 和 *upperPowerConsumption()*: 响应函数, 控制空调功耗等级, 功耗范围为1-3。
- *setColdHotMode()*: 响应函数, 控制空凋制冷或制热模式间切换。
- *setDegree()*: 响应函数, 用户释放旋钮时触发, 控制空调温度。

电视

- *myTelevisionChange(src)*: 参数为电视画面的 url, 函数接收服务器发来的电视截图 url, 并将 img 标签属性更新。
- *switchChannel(up)*: 参数为按键类型, 响应函数, 控制电视频道切换。
- *setVoice()*: 响应函数, 用户释放旋钮时触发, 控制电视音量大小。

接下来请您阅读 [设备管理平台测试](#)。

1.12 设备管理平台测试

1.12.1 单元测试

此部分文档主要介绍顶点云设备管理平台的单元测试。你可以在源码目录下运行 `python manage.py test` 运行全部单元测试。

单元测试主要覆盖用户登录、注册、设备创建、修改、信息更新、命令执行等, 各单元测试文件覆盖视图函数如下表所示。

单元测试文件名	测试覆盖视图函数
<code>test_basic.py</code>	测试环境是否正常, 服务器是否启动
<code>test_client1.py</code>	<code>show_status</code> 、 <code>update_status</code> 、 <code>get_history</code> 、 <code>show_history</code> 、 <code>newdevie</code> 、 <code>edit_device</code> 、 <code>delete_device</code> 、 <code>set_status</code> 、 <code>set_interval</code> 、 <code>reset_token</code> 、 <code>index</code> 、 <code>home</code> 、 <code>user</code>
<code>test_client2.py</code>	<code>login</code> 、 <code>logout</code> 、 <code>register</code> 、 <code>confirm</code> 、 <code>resend_confirmation</code> 、 <code>change_email_request</code> 、 <code>change_email</code> 、 <code>password_reset</code> 、 <code>secure_center</code> 、 <code>before_request</code> 、 <code>unconfirmed</code>

1.12.2 手动测试报告

此部分报告由 non1996 测试并编写。

用户认证

测试用例名称	登陆	标识	TEST-1
测试用例综述	检验用户登陆正确性		
用例初始化	管理员权限账户用户名: test@test.com, 密码: TESTTHISPASSWORD		
测试步骤			
序号	操作	期望结果	
1	在系统登录界面中输入正确的测试用户名和正确登陆密码。	可正常登陆, 登陆后页面跳转到 设备详细信息页面。	
2	在系统登录界面中输入错误的用户名或错误登陆密码。	登陆页面弹出登陆失败提示, 告知用户用户名或密码错误	

设备管理

测试用例名称	设备信息管理	标识	TEST-2
测试用例综述	检验设备管理界面操作的正确性		
用例初始化	预制的模拟设备		
测试步骤			
序号	操作	期望结果	实测结果
1	点击设备操作中的“详细”按钮。	进入对应设备的详细信息界面。	与预期结果一致。
2	点击设备操作中的“历史”按钮。	进入对应设备的历史信息界面。	与预期结果一致。
3	点击设备操作中的“删除”按钮。	弹出确认框, 点击确认后, 网页向服务器发出请求, 设备从模拟程序上删除, 网页不再显示设备信息。	与预期结果一致。
4	点击“创建新设备”按钮。	弹出模态对话框, 在输入设备名称、设备简介、监控间隔并选择设备后, 网页向服务器发出添加请求, 在模拟程序上添加相应类型的设备, 网页开始接收该设备信息并显示。	与预期结果一致。
5	点击侧边栏页面索引按钮。	跳转到相应页面。	与预期结果一致。
6	点击用户头像, 并在弹出的下拉框中点击“账户资料”和“安全中心”。	点击“账户资料”后, 跳转到账户资料页面, 用户可设置用户头像, 昵称, 及用户简介。点击“安全中心”后, 跳转到安全中心页面, 用户可在此页面更换登陆邮箱和修改登陆密码。	与预期结果一致。
7	点击设置按钮。	弹出对话框, 输入刷新间隔后, 改变页面向服务器发送ajax刷新界面请求的间隔。	与预期结果一致。

详细界面

测试用例名称	设备详细信息页面测试	标识	TEST-3
测试用例综述	检验各设备详细信息界面显示和操作的正确性		
用例初始化	预制的模拟设备		
界面显示测试			
能够显示模拟设备的电流、电话、功率、设备温度，并能以动态的图表将其变化过程实时展现给用户。除了上述基本属性，针对不同设备，能显示其特有的属性变化情况，如电灯光线饱和度和电视画面、空调所处室内温度等。对于实体设备，如计算机，能显示CPU、内存、磁盘的使用情况和CPU、GPU温度情况。			
测试步骤			
序号	操作	期望结果	实测结果
1	点击设备开关按钮。	当设备运行时，单击按钮能停止模拟程序对该设备的模拟，设备状态信息不再变化。当设备未开启时，单击按钮能使程序继续对设备状态模拟，设备状态开始变化。	与预期结果一致。
2	点击设备监控间隔的按钮。	点击“+”能增加网页向服务器请求数据的间隔，上限为10，点击“-”能降低网页向服务器请求的速率，下限为1。点击后设备各数据刷新频率改变。	与预期结果一致。
3	点击电灯页面设置亮度按钮。	单击左边按钮能降低电灯亮度，下限为1，单击右边按钮能提高电灯亮度，上限为5，点击后改变模拟程序中电灯亮度值。	与预期结果一致。
4	点击电视界面设置频道按钮。	电视频道能在1到9之间切换，切换频道后，网页显示的节目画面能进行切换。	与预期结果一致。
5	转动电视页面音量设置旋钮。	鼠标松开后，将当前音量值发送给模拟程序，改变模拟程序中电视的音量值。	与预期结果一致。
6	转动空调页面温度设置旋钮下拉框中点击“账户资料”和“安全中心”。	鼠标松开后，将当前温度值发送给模拟程序，改变模拟程序中空调温度值。	与预期结果一致。
7	点击空调页面制冷/制热按钮。	当按钮为红色时，点击后设置模拟程序中空调的模式为制冷，当按钮为白色时，点击后设置空调的模式为制热。	与预期结果一致。

历史记录

测试用例名称	设备历史信息页面测试	标识	TEST-4
测试用例综述	检验各设备历史信息界面显示和操作的正确性		
用例初始化	预制的模拟设备		
界面显示测试			
能够显示模拟设备的电流、电话、功率、设备温度在某个小时内的历史记录。能显示实体计算机设备的内存使用率等信息。			
测试步骤			
序号	操作	期望结果	实测结果
1	选择要查询的设备历史时间区间后, 点击查询按钮。	历史信息能正常显示	与预期结果一致。

到这里顶点云设备管理平台的开发者文档部分就结束了, 您可以继续阅读 [设备管理平台设计要点](#) 来查看我们对设计者提出的几点建议。

此部分文档包括顶点云设备管理平台的设计要点和变动记录

2.1 设备管理平台设计要点

顶点云设备管理平台的设计主要考虑了如下几点：

我们建议，如果您想使用顶点云设备管理平台的代码做一些自定义开发工作，最好能够维护如下几条准则，这是我们根据编写过程中遇到过的问题总结出的经验：

1. 先设计蓝本，以及蓝本的职责，再决定蓝本提供的视图函数。
2. 视图函数应尽量避免重复代码，应尽量将共同的代码部分抽象，如使用一个表驱动函数池。
3. 将需要根据环境改变的变量写入到配置文件中，如果是扩展功能的配置项，应创建新的子类并包含此项。用户可以选择多重继承来实现扩展功能的组合。

感谢您的阅读！到这里顶点云设备管理平台文档就结束了，您可以阅读 [顶点云设备管理平台变动记录](#) 以了解顶点云设备管理平台开发过程的变动历史。

2.2 顶点云设备管理平台变动记录

版本	日期	变动
v0.01	2016-12-24	创建项目
v0.10	2016-12-25	完成用户管理 (注册、登录、密码重置、修改、邮箱 修改等
v0.20	2016-12-26	完成虚拟设备模拟平台
v0.30	2016-12-27	完成设备状态获取入口、虚拟设备控制入口
v0.40	2016-12-28	HTML 模板迁移
v0.60	2016-12-29	增加真实 PC 设备监控
v0.80	2016-12-30	调试、完成 Ajax 异步调用
v0.90	2016-12-31	细化模板细节, 完成设备告警提醒、侧边栏异步更新
v0.98	2017-01-01	完成 PC 设备远程控制、监控视频查看等细节
v1.0	2017-01-02	编写测试代码、文档。上线测试。

此部分标明顶点云设备管理平台使用的开源许可证以及授权声明。

3.1 许可证信息

顶点云使用的许可证可在 [此处](#) 查看，内容如下：

```
Copyright (c) 2015-2018, Forec <forec@bupt.edu.cn>

Permission to use, copy, modify, and/or distribute this code for
any purpose with or without fee is hereby granted, provided that
the above copyright notice and this permission notice appear in
all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL
THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

无论是否用于商业目的，你均可以根据自己的需要使用、修改、拷贝顶点云设备管理平台的代码，但必须保证以上版权声明和许可证信息出现在所有顶点云设备管理平台代码的副本中。