

CMDB 项目

CMDB 是什么

大体上来说，有人说 CMDB 是自动化运维的基石，CMDB 也常常被认为是构建 ITIL 流程的基础，ITIL 项目的成败与是否成功建立 CMDB 有非常大的关系。

具体点说，CMDB 就是配置管理数据库，存储与管理企业 IT 架构中设备的各种配置信息，通过识别、控制、维护，检查企业的 IT 资源，从而高效控制与管理不断变化的 IT 基础架构与 IT 服务，并为其它流程，例如事故管理、问题管理、变更管理、发布管理等流程提供准确的配置信息。

通俗点说，CMDB 可以存储并自动发现整个 IT 网络上的各种信息，比如一个 IT 网络上有多少台服务器、多少存储、设备的品牌、资产编号、维护人员、所属部门、服务器上运营什么操作系统、操作系统的版本、操作系统上有哪些应用、每个应用的版本等等，此外，CMDB 还有一个非常重要的功能，即存储不同资源之间的依赖关系，如果网络上某个节点出现问题，通过 CMDB，可以判断因此受到影响的业务

还有一种说法，CMDB 可以将人员、应用、资源，三只之间的关系联系到一起。

CMDB 的作用

1. 信息整合：

如何将众多 IT 设备、IT 服务、甚至使用它们的部门与人员整合在一个完整的库中？这样整合的信息将使有效与高效的管理 IT 设备与服务成为可能。可自动发现各种主机、网络设备、应用。同时支持全网发现、指定子网、指定配置项三种发现方式。

2. 关系映射：

如何将硬件、软件以及 IT 服务之间的物理和逻辑关系映射可视化？使得 IT 人员可以看到其互相之间的依赖关系，并确定该 IT 组件对客户带来的潜在影响。若 IT 人员可以实时看到其对公司或客户业务的影响，将大大有助于提高 IT 服务水平。使用拓扑图形式，可视化展示 IT 资源、部门、人员之间的关联关系，并且可通过在拓扑图上直接拖拽，实现关联关系的定义与维护。CMDB 不仅仅存储 IT 资源的属性与关联关系，还自动关联 IT 资源与其发生过事故、问题、变更、发布。

3. 流程支持：

如何为其它 IT 运维流程提供准确的 IT 设备、IT 服务的配置信息（包括当前设备或服务发生过的事故、问题、变更、发布等信息）对服务台&事故管理、问题管理、变更管理、发布管理来说，准确的配置信息将极大的提高流程的运作效率。在服务台&事故、问题、变更、发布流程中，均可以快速查看当前流程涉及到的 IT 资源的全面、准确的信息。

4. 软件库与硬件库：

如何保证应用到 IT 环境的软件与硬件均是经过授权与测试的？这是保证 IT 环境质量与提供稳定 IT 服务的前提条件。通过支持 DSL（Definitive

Software Library，最终软件库）与 DHS（Definitive Hardware Store，最终硬件库），保证在发布管理中使用的软件与硬件均是通过授权与测试的。

CMDB

一：IT 运维的分类

IT 运维，指的是对已经搭建好的网络、软件、硬件进行维护。运维领域也是细分的，有硬件运维和软件运维。

硬件运维主要包括对基础设施的运维，例如机房的设备，主机的键盘，内存等物理设备的维护。

软件运维主要包括系统运维和应用运维，系统运维主要包括对 OS，数据库，中间件的监控和维护，这些系统介于设备和应用之间，应用运维主要是对线上业务系统的运维。

这里主要讨论的是软件运维的自动化，包括系统运维和应用运维的自动化。

二：传统运维的缺点

1：日常工作繁琐

日常运维工作繁琐，研发会经常需要到服务器上查看日志，重启应用，或者是上线某个产品需要部署环境等，都是运维的日常。

2：应用运行环境不统一

在部署某个应用后，应用不能访问，在开发环境运行的好，但是在测试环境后就不能继续用了，因为各类环境的类库不统一。还有一种情况是，由于运维人员的习惯不同，按照自己的习惯来安装部署环境，每种服务器上运行的软件目录不统一。

3：运维及部署效率低下

运维人员需要登录到服务器上执行命令，部署程序，不仅效率低，并且非常容易出现人为错误，人为出错之后也不容易找到问题所在。

4：无用报警信息过多

运维人员经常会收到很多的无用报警信息，常常会选择屏蔽这些报警信息，并且一旦应用的访问速度出了问题，就需要从系统、网络、应用、数据库等一步步查找信息。

5：资产管理和应用系统混乱

资产管理，服务管理经常记录在 **Excel** 或者文本里面，不便于管理。老员工由于熟悉不注重文档的维护，新员工入职时，资产才能更正一次。

三：为什么需要自动化运维？

1：项目上线

流程：产品经理调研（画出原型图）-->定需求-->三方会谈（研发，产品经理，老大们）-->定日期-->测试项目-->最终上线-->应用运维

目前：将代码打包给运维，运维解压上线

问题：随着机器数量的线性增加，运维的工作量也是线性增加，重复而且是毫无意义的劳动

解决：

1：写一个 **shell** 脚本，进行部署

2：搞一个自动化代码上线系统

必要条件：服务器的各种信息（主机名，**CPU**，硬盘大小等）

2：监控系统

监测服务器的各种信息（硬盘是否满，**CPU** 的使用率，内存使用率，网站服务运行是否正常）

问题：之前简单的脚本，监测服务器的信息，比较麻烦

解决：想将服务器的各种信息，以图表的形式展示在 **web** 界面上（可视化）

必要条件：服务器的各种信息（主机名，**CPU**，硬盘大小等）

3：自动装机系统

问题：人工装机需要一台一台去装

解决：搞一个装机系统，**cobbler** 软件

必要条件：服务器的各种信息（主机名，**CPU** 等）

4：Excel 表格审计管理资产

四：资产管理系统 (CMDB)

CMDB 是所有运维工具的数据基础，**CMDB** 全称 **Configuration Management Database**

1, CMDB 包含的功能:

- 1: 用户管理, 记录测试, 开发, 运维人员的用户表
- 2: 业务线管理, 需要记录业务的详情
- 3: 项目管理, 指定此项目需属于那条业务线, 以及项目详情
- 4: 应用管理, 指定此应用的开发人员, 属于哪个项目, 和代码地址, 部署目录, 部署集群, 依赖的应用, 软件等信息。
- 5: 主机管理, 包括云主机, 物理机, 主机属于哪个集群, 运行着哪些软件, 主机管理员, 连接着哪些网络设备, 云主机的资源地, 存储等相关信息。
- 6: 主机变更管理, 主机的一些信息变更, 例如管理员, 所属集群等信息更改, 连接的网络变更等。
- 7: 网络设备管理, 主要记录网络设备的详细信息, 及网络设备连接的上级设备
- 8: IP 管理, IP 属于哪个主机, 哪个网段, 是否被占用等

浅谈 ITIL

ITIL 即 IT 基础架构库(Information Technology Infrastructure Library, ITIL, 信息技术基础架构库)由英国政府部门 CCTA(Central Computing and Telecommunications Agency)在 20 世纪 80 年代末制订, 现由英国商务部 OGC(Office of Government Commerce)负责管理, 主要适用于 IT 服务管理 (ITSM)。ITIL 为企业的 IT 服务管理实践提供了一个客观、严谨、可量化的标准和规范。

1、事件管理 (Incident Management)

事故管理负责记录、归类和安排专家处理事故并监督整个处理过程直至事故得到解决和终止。事故管理的目的是在尽可能最小地影响客户和用户业务的情况下使 IT 系统恢复到服务级别协议所定义的服务级别。

目标是: 在不影响业务的情况下, 尽可能快速的恢复服务, 从而保证最佳的效率和服务的可持续性。事件管理流程的建立包括事件分类, 确定事件的优先级和建立事件的升级机制。

2、问题管理 (Problem Management)

问题管理是指通过调查和分析 IT 基础架构的薄弱环节、查明事故产生的潜在原因, 并制定解决事故的方案和防止事故再次发生的措施, 将由于问题和事故对业务产生的负面影响减小到最低的服务管理流程。与事故管理强调事故恢复的速度不同, 问题管理强调的是找出事故产生的根源, 从而制定恰当的解决方案或防止其再次发生的预防措施。

目标是: 调查基础设施和所有可用信息, 包括事件数据库, 来确定引起事件发生的真正潜在原因, 一起提供的服务中可能存在的故障。

3、配置管理 (Configuration Management)

配置管理是识别和确认系统的配置项, 记录和报告配置项状态和变更请求, 检验配置项的正确性和完整性等活动构成的过程, 其目的是提供 IT 基础架构的逻辑模型, 支持其它服务管理流程特别是变更管理和发布管理的运作。

目标是: 定义和控制服务与基础设施的部件, 并保持准确的配置信息。

4、变更管理 (Change Management)

变更管理是指为在最短的中断时间内完成基础架构或服务的任一方面的变更而对其进行控制的服务管理流程。变更管理的目标是确保在变更实施过程中使用标准的方法和步骤，尽快地实施变更，以将由变更所导致的业务中断对业务的影响减小到最低。

目标是：以受控的方式，确保所有变更得到评估、批准、实施和评审。

5、发布管理（Release Management）

发布管理是指对经过测试后导入实际应用的新增或修改后的配置项进行分发和宣传的管理流程。发布管理以前又称为软件控制与分发。

目标是：在实际运行环境的发布中，交付、分发并跟踪一个或多个变更。

实际工作场景中自动化工具举例：



CMDB

CMDB --Configuration Management Database 配置管理数据库, CMDB 存储与管理企业 IT 架构中设备的各种配置信息，它与所有服务支持和服务交付流程都紧密相联，支持这些流程的运转、发挥配置信息的价值，同时依赖于相关流程保证数据的准确性。

在实际的项目中，**CMDB** 常常被认为是构建其它 **ITIL** 流程的基础而优先考虑，**ITIL** 项目的成败与是否成功建立 **CMDB** 有非常大的关系。

70%~80%的 IT 相关问题与环境的变更有着直接的关系。实施变更管理的难点和重点并不是工具，而是流程。即通过一个自动化的、可重复的流程管理变更，使得当变更发生的时候，有一个标准化的流程去执行，能够预测到这个变更对整个系统管理产生的影响，并对这些影响进行评估和控制。而变更管理流程自动化的实现关键就是 CMDB。

CMDB 工具中至少包含这几种关键的功能：整合、调和、同步、映射和可视化。

- 整合是指能够充分利用来自其他数据源的信息，对 CMDB 中包含的记录源属性进行存取，将多个数据源合并至一个视图中，生成连同来自 CMDB 和其他数据源信息在内的报告；

- 调和能力是指通过对来自每个数据源的匹配字段进行对比，保证 CMDB 中的记录在多个数据源中没有重复现象，维持 CMDB 中每个配置项目数据源的完整性；自动调整流程使得初始实施、数据库管理员的手动运作和现场维护支持工作降至最低；
- 同步指确保 CMDB 中的信息能够反映联合数据源的更新情况，在联合数据源更新频率的基础上确定 CMDB 更新日程，按照经过批准的变更来更新 CMDB，找出未被批准的变更；
- 应用映射与可视化，说明应用间的关系并反应应用和其他组件之间的依存关系，了解变更造成的影响并帮助诊断问题。

CMDB 是运维自动化项目，它可以减少人工干预，降低人员成本。

- 功能：自动装机、实时监控、自动化部署软件，建立在它们的基础上是资产信息变更记录（资产管控自动进行汇报）

• 一：CMDB 是一个什么项目

- CMDB:[运维自动化的一个项目]
- -运维自动化基础
- -主要就是资产管理

- 运维自动化项目目的：减少人工干预，降低人员成本。

• 二：运维自动化的分类

- 运维自动化工具大概四类：
- 安装批量系统： linux kgstart、i z 卡
- 装环境： 初始化环境，也有一个工具
- 批量打包项目部署： ssh/crt/种子（如迅雷种子）
- 监控服务的实时状态：
- 以上的四类都要依赖 CMDB：
- 你安装系统，是不是要指定往哪一台机器安装系统？
- 装环境 一样需要执行往哪一台？
- 批量？哪几台
- 监控？监控谁？
- 常见的运维项目就是如下了：
- 我们要实现的 CMDB:主要就是针对资产管理的方面。

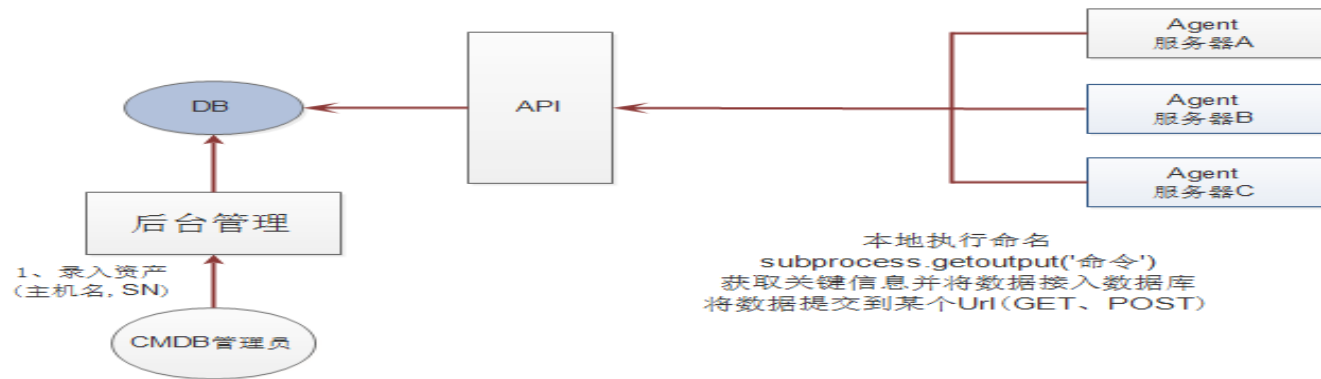


三：CMDB 资产采集的 4 种方式

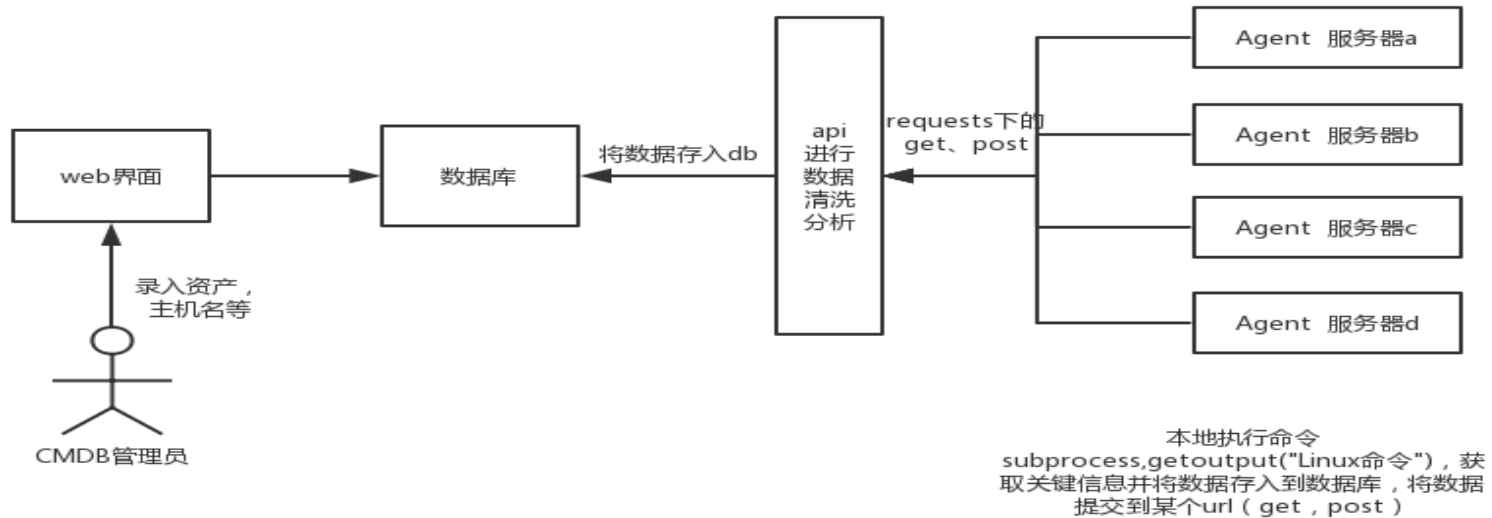
- 要做资产管理 那么就必须先获取资产的信息。
- 自动获取。

方式 1: Agent 方式 (基于 shell 命令实现)

- agent 方式: 要求在每个的服务器上安装 agent 程序,
- 在 agent 中链接上 API, 然后再有 API 去向数据库添加数据。
- API: 也是一个 django 的程序
- agent 程序: 可以通过 subprocess 模块执行命令, 再采用 requests 模块 get 或 post 发送数据。
- API: 获取数据, 执行添加数据 (比如网卡等等信息)



原理图



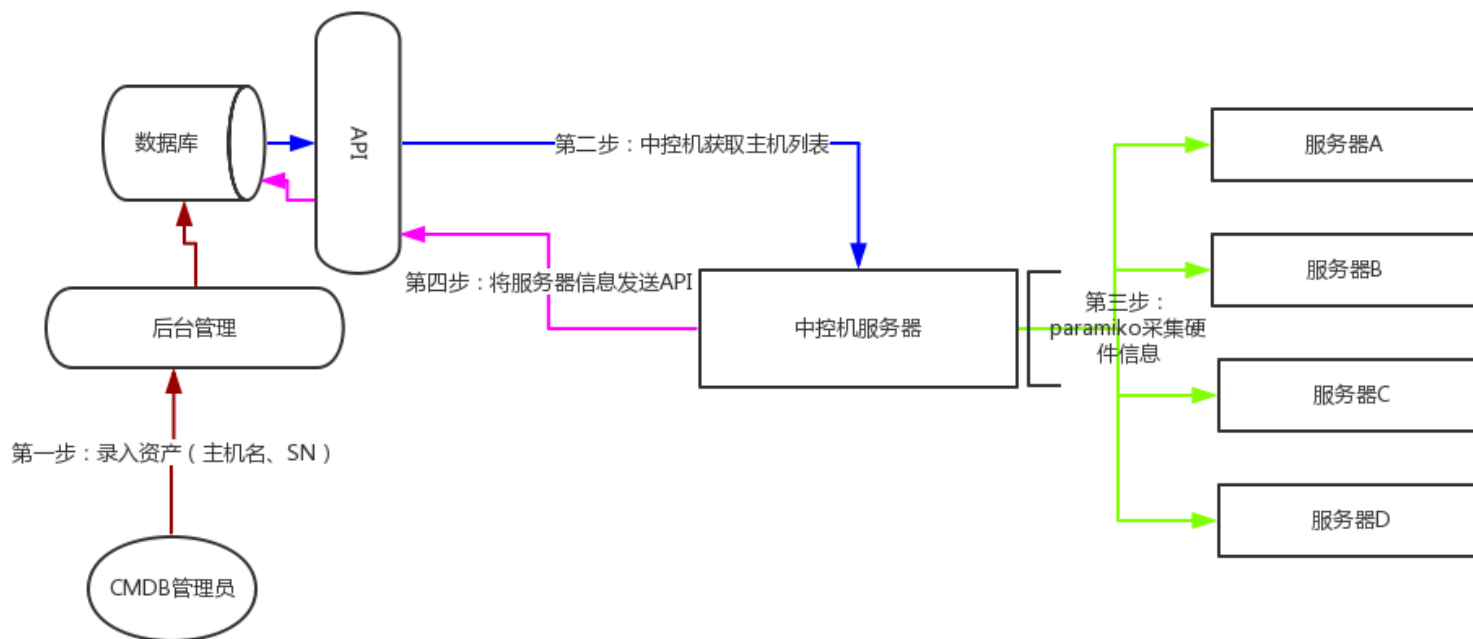
- Agent 方式，可以将服务器上面的 Agent 程序作定时任务，定时将资产信息提交到指定 API 录入数据库
- 优点：速度快 缺点：需要为每台服务器部署一个 Agent 程序

对于 Agent 的版本实现思路：

- Agent 采集硬件资产
- API 提供相关处理的接口
- 管理平台为用户提供可视化操作
 - 本质就是在各个服务器上执行 `subprocess.getoutput("命令")`, 然后将每台机器上执行的结果返回给主句 API, 然后主机 API 收到这些数据之后, 放到数据库中, 最终通过 web 界面展现给用户。
 - 优点: 速度快
 - 缺点: 需要为每台服务器部署有关 Agent 程序
 - 使用场景: 服务器比较多的时候

• 方式 2: ssh 类实现方式 (基于 paramiko 模块) (fabric,ansible)

- SSH 方式:
- 服务端没有了 agent 程序, 而是在服务器和 API 的中间加了一个 **中控服务端** [用于采集服务端信息]
- 中控服务端通过 paramiko 模块主动去链接服务器, 然后通过 subprocess 模块执行命令, 返回命令结果; 再发送到 API:
- 去链接服务器可能我们会使用别的模块: 如 ansible 或 fabric 模块, 但是他们都是对 paramiko 模块的封装。
- 注意: 最新的 ansible 不再使用 paramiko 模块封装了, 而是自己实现了新的方式。



中控机通过 **Paramiko**（**py** 模块）登录到各个服务器上，然后执行命令的方式去获取各个服务器上的信息。

优点：没有 Agent

缺点：有一个中控机，速度慢

使用场景：服务器比较少的时候

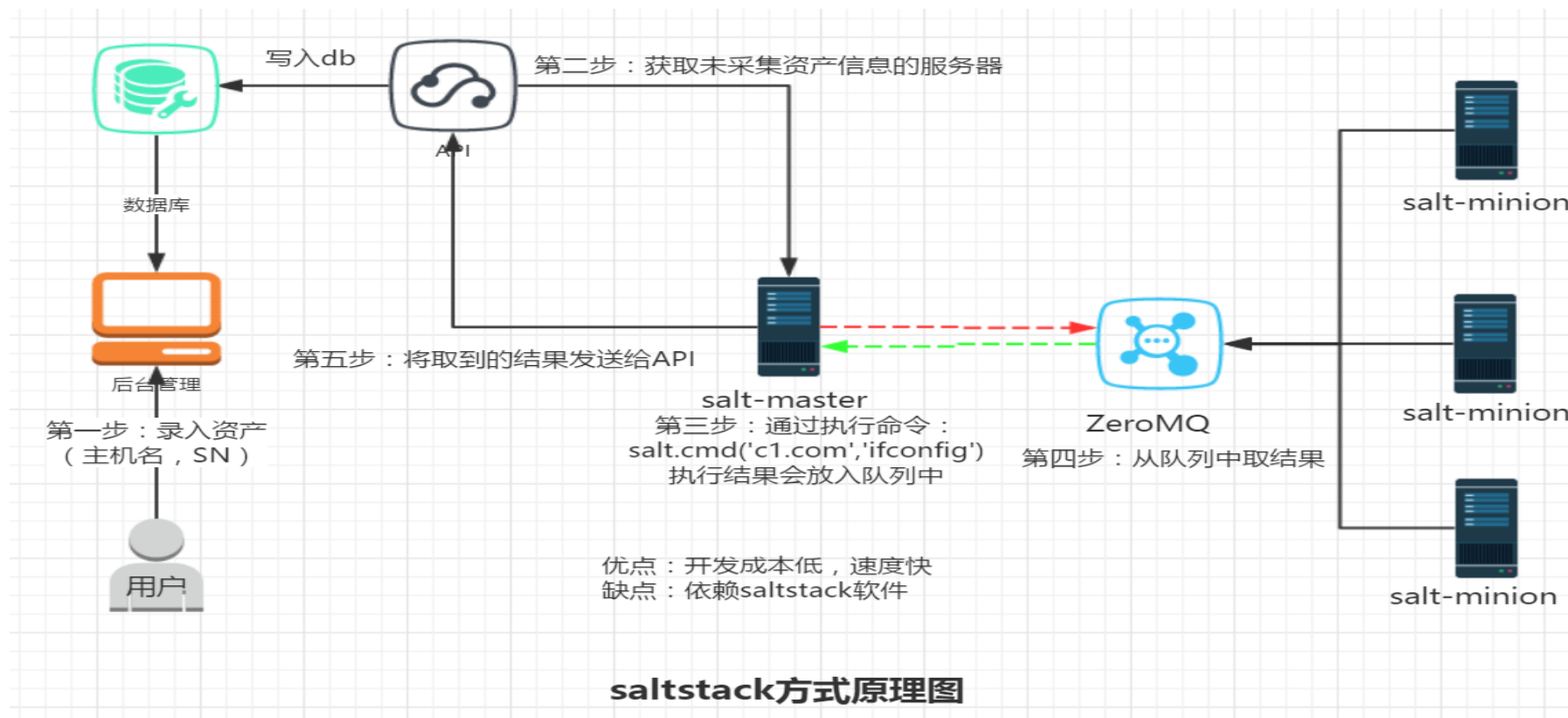
API 从数据库中获取到未采集的机器列表后发送到中控机服务器中，中控机服务器通过 **paramiko** 模块登录到服务器上，进行信息的采集，服务器采集完后再将结果返回给中控机，仍后将从服务器中得到 的信息通过 **request** 模块发送给 API，API 通过主机名和 SN 作为唯一标识，将信息录入到数据中，然后通过 web 界面将数据展现给用户

- **paramiko** 模块（获取主机名）
- **API**
- **web** 界面
- 场景：服务器较少
- 缺点：依赖于网络，速度慢
- 优点：没有 **Agent**，不需要为每一台服务器部署一个 **Agent** 程序
- 如果在服务器较少的情况下，可应用此方法

```
1  import paramiko
2
3  # 创建 SSH 对象
4  ssh = paramiko.SSHClient()
5  # 允许连接不在 know_hosts 文件中的主机
6  ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
7  # 连接服务器
8  ssh.connect(hostname='cl.salt.com', port=22, username='wupeiqi', password='123')
9
10 # 执行命令
11 stdin, stdout, stderr = ssh.exec_command('df')
12 # 获取命令结果
13 result = stdout.read()
14
15 # 关闭连接
16 ssh.close()
```

方式 3: saltstack 方式

- saltstack 方式:
- 基于第三方软件（去 python 写的）去实现，在中控服务端安装一个 saltstack-master；服务器安装 saltstack-slave。
- 执行过程:
- saltstack-master 指定主机发送命令（"c1.com", "ifconfig"），将命令发送到队列（RPC）中去；
- 服务器取命令，执行完又将结果发送到一个新的队列中去；
- saltstack-master 获取数据，向 API 发送数据。



此方案本质上和第二种方案大致是差不多的流程，中控机在获取到未采集资产信息的服务器（主机名），再而将命令放入一个队列中，服务器来获取。服务器将结果放入另一个队列中(中控机从 API 中获取未采集的资产信息后通过队列发送命令给服务器执行。服务器执行完后将结果放到入另一个队列中)，中控机获取将服务信息发送到 API 进而录入数据库。然后通过 web 界面将数据展现给用户。

基于 SaltStack 的 master 上的 pillar 以及远程执行命令实现

```
import salt.client
local = salt.client.LocalClient()
local.cmd('*', 'cmd.run', ['whoami'])
```

优点：速度快，开发成本低

缺点：依赖于第三方工具

使用场景：公司已经使用 salt-stack 软件

• 方式 4: puppet 方式 (ruby 语言开发)：基于 Puppet 的 factor 和 report 功能实现

- puppet 方式：
- 这是相对比较古老的一种方式，上面的几种都是由中控服务端去主动获取数据；
- 而 puppet 方式是由 puppet-master 和 puppet-salve 组成；puppet-salve 每隔 30 分钟发送数据给 puppet-master；
- puppet-master 内维护了一个报表：获取这个报表发送数据给 API
- puppet:是用 ruby 写的。所以我们在通过报表采集资产的时候，需要使用 ruby 语言。

saltstack 的安装和配置

master 端：

"""

1. 安装 salt-master

```
yum install salt-master
```

2. 修改配置文件： vim /etc/salt/master

interface:10.0.0128 表示 Master 的 ip

3. 启动

```
service salt-master start
```

"""

slave 端:

"""

1、安装 salt-minion

```
yum install salt-minion
```

2、修改配置文件 : vim /etc/salt/minion

```
master: 10.0.0.128      #master 的地址
```

3、启动: service salt-minion start

"""

```
salt-key -L      #查看已授权和未授权的 slave
salt-key -A salve_id  #接受指定的 id 的 salve
salt-key -r salve_id  #拒绝指定 id 的 salve
salt-key -d salve_id  #删除指定 id 的 salve
```

在 **master** 服务器上对 **salve** 进行 远程操作

```
salt 'c2.salt.com' cmd.run 'ifconfig'
salt "*" cmd.run 'ifconfig'
```

基于 **API** 的方式

```
import salt.client
local=salt.client.localClient()
result=local.cmd('c2.salt.com', 'cmd.run' ['ifconfig'])
```

收集服务器信息的代码：

代码出现的问题：

代码出现冗余：a. 可以写一个公共的方法；b. 可以写一个父类方法

代码高内聚：指一个方法就干一件事，剩下的不管，将相关的功能都聚集在一起，不相关的都不要

解耦合：

收集到的信息：

- 主板信息 (hostname,sn 号)
- cpu 信息 (型号, 几个 cpu 等)
- disk 磁盘信息 (大小, 几块)
- 内存 memory 信息
- 网卡信息

可插拔式的插件 收集上述信息：

配置信息

```
PLUGINS_DICT = {  
    'basic': 'src.plugins.basic.Basic',  
    'cpu': 'src.plugins.cpu.Cpu',  
    'disk': 'src.plugins.disk.Disk',  
    'memory': 'src.plugins.memory.Memory',  
    'nic': 'src.plugins.nic.Nic',  
}
```

插件的两种解决方案：

- 1、写一个公共类，让其他的所有类取继承 **Base** 这个基类
- 2、高精度 进行抽象封装

唯一标识问题

问题：实体机的 **SN** 号 and 我们的虚拟机的 **SN** 号公用一个

解决：如果公司不采用虚拟机的信息，可以用 **SN** 作唯一标识，来进行更新

否则如果公司要采集虚拟机的信息，SN 号此时不能使用
使用 进程池和线程池 解决并发的問題：

```
from concurrent.futures import ThreadPoolExecutor
    p = ThreadPoolExecutor(10)
    for hostname in hostnames:
        p.submit(self.run, hostname)
```

[Top](#)

下载 PyCrypto

<https://github.com/sfbahr/PyCrypto-Wheels>

pip3 install wheel

进入目录：

pip3 install pycrypto-2.6.1-cp35-none-win32.whl

```
from Crypto.Cipher import AES
```

```
def encrypt(message):
    key = b'dfdsdfsasdfsdfs'
    cipher = AES.new(key, AES.MODE_CBC, key)
    ba_data = bytearray(message, encoding='utf-8')
    v1 = len(ba_data)
    v2 = v1 % 16
    if v2 == 0:
        v3 = 16
    else:
        v3 = 16 - v2
    for i in range(v3):
        ba_data.append(v3)
    final_data = ba_data
    msg = cipher.encrypt(final_data) # 要加密的字符串，必须是 16 个字节或 16 个字节的倍数
    return msg
```

[illegible]

[Top](#)

```
class UserProfile(models.Model):
    """
    用户信息
    """
    name = models.CharField(u'姓名', max_length=32)
    email = models.EmailField(u'邮箱')
    phone = models.CharField(u'座机', max_length=32)
    mobile = models.CharField(u'手机', max_length=32)
    password = models.CharField(u'密码', max_length=64)

    class Meta:
        verbose_name_plural = "用户表"
```



```
def __str__(self):  
    return self.name
```

```
# class AdminInfo(models.Model):  
#     """  
#     用户登陆相关信息  
#     """  
#     user_info = models.OneToOneField("UserProfile")  
#     username = models.CharField(u'用户名', max_length=64)  
#     password = models.CharField(u'密码', max_length=64)  
#  
#     class Meta:  
#         verbose_name_plural = "管理员表"  
#  
#     def __str__(self):  
#         return self.user_info.name
```

```
class UserGroup(models.Model):  
    """  
    用户组  
    """  
    name = models.CharField(max_length=32, unique=True)  
    users = models.ManyToManyField('UserProfile')  
  
    class Meta:  
        verbose_name_plural = "用户组表"  
  
    def __str__(self):  
        return self.name
```

```
class BusinessUnit(models.Model):
```

```
"""
```

```
业务线
```

```
"""
```

```
name = models.CharField('业务线', max_length=64, unique=True)
contact = models.ForeignKey('UserGroup', verbose_name='业务联系人', related_name='c')
manager = models.ForeignKey('UserGroup', verbose_name='系统管理员', related_name='m')
```

```
class Meta:
```

```
    verbose_name_plural = "业务线表"
```

```
def __str__(self):
```

```
    return self.name
```

```
class IDC(models.Model):
```

```
"""
```

```
机房信息
```

```
"""
```

```
name = models.CharField('机房', max_length=32)
floor = models.IntegerField('楼层', default=1)
```

```
class Meta:
```

```
    verbose_name_plural = "机房表"
```

```
def __str__(self):
```

```
    return self.name
```

```
class Tag(models.Model):
```

```
"""
```

```
资产标签
```

```
"""
```

```
name = models.CharField('标签', max_length=32, unique=True)
```

```
class Meta:
```

```
verbose_name_plural = "标签表"
```

```
def __str__(self):  
    return self.name
```

```
class Asset(models.Model):
```

```
    """
```

```
    资产信息表，所有资产公共信息（交换机，服务器，防火墙等）
```

```
    """
```

```
    device_type_choices = (  
        (1, '服务器'),  
        (2, '交换机'),  
        (3, '防火墙'),  
    )
```

```
    device_status_choices = (  
        (1, '上架'),  
        (2, '在线'),  
        (3, '离线'),  
        (4, '下架'),  
    )
```

```
    device_type_id = models.IntegerField(choices=device_type_choices, default=1)
```

```
    device_status_id = models.IntegerField(choices=device_status_choices, default=1)
```

```
    cabinet_num = models.CharField('机柜号', max_length=30, null=True, blank=True)
```

```
    cabinet_order = models.CharField('机柜中序号', max_length=30, null=True, blank=True)
```

```
    idc = models.ForeignKey('IDC', verbose_name='IDC 机房', null=True, blank=True)
```

```
    business_unit = models.ForeignKey('BusinessUnit', verbose_name='属于的业务线', null=True, blank=True)
```

```
    tag = models.ManyToManyField('Tag')
```

```
    latest_date = models.DateField(null=True)
```

```
    create_at = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
    verbose_name_plural = "资产表"

def __str__(self):
    return "%s-%s-%s" % (self.idc.name, self.cabinet_num, self.cabinet_order)
```

```
class NetworkDevice(models.Model):
    """
    网络设备信息表
    """

    asset = models.OneToOneField('Asset')
    management_ip = models.CharField('管理 IP', max_length=64, blank=True, null=True)
    vlan_ip = models.CharField('VlanIP', max_length=64, blank=True, null=True)
    intranet_ip = models.CharField('内网 IP', max_length=128, blank=True, null=True)
    sn = models.CharField('SN 号', max_length=64, unique=True)
    manufacture = models.CharField(verbose_name='制造商', max_length=128, null=True, blank=True)
    model = models.CharField('型号', max_length=128, null=True, blank=True)
    port_num = models.SmallIntegerField('端口个数', null=True, blank=True)
    device_detail = models.CharField('设置详细配置', max_length=255, null=True, blank=True)

    class Meta:
        verbose_name_plural = "网络设备"
```

```
class Server(models.Model):
    """
    服务器信息
    """

    asset = models.OneToOneField('Asset')

    hostname = models.CharField(max_length=128, unique=True)
    sn = models.CharField('SN 号', max_length=64, db_index=True)
    manufacturer = models.CharField(verbose_name='制造商', max_length=64, null=True, blank=True)
```

```
model = models.CharField('型号', max_length=64, null=True, blank=True)

manage_ip = models.GenericIPAddressField('管理 IP', null=True, blank=True)

os_platform = models.CharField('系统', max_length=16, null=True, blank=True)
os_version = models.CharField('系统版本', max_length=16, null=True, blank=True)

cpu_count = models.IntegerField('CPU 个数', null=True, blank=True)
cpu_physical_count = models.IntegerField('CPU 物理个数', null=True, blank=True)
cpu_model = models.CharField('CPU 型号', max_length=128, null=True, blank=True)

create_at = models.DateTimeField(auto_now_add=True, blank=True)
```

```
class Meta:
    verbose_name_plural = "服务器表"
```

```
def __str__(self):
    return self.hostname
```

```
class Disk(models.Model):
```

```
    """
```

```
    硬盘信息
```

```
    """
```

```
    slot = models.CharField('槽位', max_length=8)
    model = models.CharField('磁盘型号', max_length=32)
    capacity = models.CharField('磁盘容量 GB', max_length=32)
    pd_type = models.CharField('磁盘类型', max_length=32)
    server_obj = models.ForeignKey('Server', related_name='disk')
```

```
class Meta:
    verbose_name_plural = "硬盘表"
```

```
def __str__(self):
    return self.slot
```

```
class NIC(models.Model):
    """
    网卡信息
    """
    name = models.CharField('网卡名称', max_length=128)
    hwaddr = models.CharField('网卡 mac 地址', max_length=64)
    netmask = models.CharField(max_length=64)
    ipaddrs = models.CharField('ip 地址', max_length=256)
    up = models.BooleanField(default=False)
    server_obj = models.ForeignKey('Server', related_name='nic')

    class Meta:
        verbose_name_plural = "网卡表"

    def __str__(self):
        return self.name


class Memory(models.Model):
    """
    内存信息
    """
    slot = models.CharField('插槽位', max_length=32)
    manufacturer = models.CharField('制造商', max_length=32, null=True, blank=True)
    model = models.CharField('型号', max_length=64)
    capacity = models.FloatField('容量', null=True, blank=True)
    sn = models.CharField('内存 SN 号', max_length=64, null=True, blank=True)
    speed = models.CharField('速度', max_length=16, null=True, blank=True)

    server_obj = models.ForeignKey('Server', related_name='memory')

    class Meta:
        verbose_name_plural = "内存表"
```

```
def __str__(self):  
    return self.slot
```

```
class AssetRecord(models.Model):
```

```
    """
```

```
    资产变更记录, creator 为空时, 表示是资产汇报的数据。
```

```
    """
```

```
    asset_obj = models.ForeignKey('Asset', related_name='ar')  
    content = models.TextField(null=True) # 新增硬盘  
    creator = models.ForeignKey('UserProfile', null=True, blank=True) #  
    create_at = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
    verbose_name_plural = "资产记录表"
```

```
def __str__(self):  
    return "%s-%s-%s" % (self.asset_obj.idc.name, self.asset_obj.cabinet_num, self.asset_obj.cabinet_order)
```

```
class ErrorLog(models.Model):
```

```
    """
```

```
    错误日志, 如: agent 采集数据错误 或 运行错误
```

```
    """
```

```
    asset_obj = models.ForeignKey('Asset', null=True, blank=True)  
    title = models.CharField(max_length=16)  
    content = models.TextField()  
    create_at = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
    verbose_name_plural = "错误日志表"
```

```
def __str__(self):  
    return self.title
```

highcharts (图表库)

<https://www.hcharts.cn/demo/highcharts/dark-unica>

echarts (图表库)

<https://echarts.baidu.com/>

Datatables(表格插件)

<http://www.datatables.club/>

layui-经典模块化前端 **UI** 框架

<https://www.layui.com/demo/admin.html>

```
<link rel="stylesheet" href="/static/bs/dist/css/bootstrap.css">
<link rel="stylesheet" href="/static/bstable/src/extensions/editable/bootstrap-editable.css">
<link rel="stylesheet" href="/static/bstable/dist/bootstrap-table.css">
```

```
<script src="/static/jquery-3.3.1.js"></script>
<script src="/static/bs/dist/js/bootstrap.js"></script>
<script src="/static/bstable/dist/bootstrap-table.js"></script>
<script src="/static/bstable/dist/locale/bootstrap-table-zh-CN.js"></script>
<script src="/static/bstable/dist/extensions/editable/bootstrap-table-editable.js"></script>
<script src="/static/bootstrap-editable.min.js"></script>
```

```
<body>
  <div class="panel-body" style="padding-bottom:0px;">
    <div class="panel panel-default">
```



```
<div class="panel-heading">查询条件</div>
<div class="panel-body">
  <form id="formSearch" class="form-horizontal">
    <div class="form-group" style="margin-top:15px">
      <label class="control-label col-sm-1" for="txt_search_departmentname">部门名称</label>
      <div class="col-sm-3">
        <input type="text" class="form-control" id="txt_search_departmentname">
      </div>
      <label class="control-label col-sm-1" for="txt_search_statu">状态</label>
      <div class="col-sm-3">
        <input type="text" class="form-control" id="txt_search_statu">
      </div>
      <div class="col-sm-4" style="text-align:left;">
        <button type="button" style="margin-left:50px" id="btn_query" class="btn btn-primary">查询</button>
      </div>
    </div>
  </form>
</div>
<div id="toolbar" class="btn-group">
  <button id="btn_add" type="button" class="btn btn-default">
    <span class="glyphicon glyphicon-plus" aria-hidden="true"></span>新增
  </button>
  <button id="btn_edit" type="button" class="btn btn-default">
    <span class="glyphicon glyphicon-pencil" aria-hidden="true"></span>修改
  </button>
  <button id="btn_delete" type="button" class="btn btn-default">
    <span class="glyphicon glyphicon-remove" aria-hidden="true"></span>删除
  </button>
</div>
<table id="idc"></table>
</div>
</body>
```

```

$.fn.editable.defaults.mode = 'inline';
$('#'+tableid).bootstrapTable({
    url: url,                //请求后台的 URL (*)
    method: 'get',           //请求方式 (*)
    toolbar: '#toolbar',     //工具按钮用哪个容器
    striped: true,           //是否显示行间隔色
    cache: false,            //是否使用缓存，默认为 true，所以一般情况下需要设置一下这个属性 (*)
    pagination: true,        //是否显示分页 (*)
    sortable: false,         //是否启用排序
    sortOrder: "asc",        //排序方式
    sidePagination: "client", //分页方式：client 客户端分页，server 服务端分页 (*)
    pageNumber: 1,           //初始化加载第一页，默认第一页
    pageSize: 10,            //每页的记录行数 (*)
    pageList: [10, 25, 50, 100], //可供选择的每页的行数 (*)
    //search: true,          //是否显示表格搜索，此搜索是客户端搜索，不会进服务端，所以，个人感觉意义不大
    strictSearch: true,
    showPaginationSwitch: true,
    showColumns: true,        //是否显示所有的列
    showRefresh: true,        //是否显示刷新按钮
    clickToSelect: true,      //是否启用点击选中行
    uniqueId: "id",           //每一行的唯一标识，一般为主键列
    showToggle: true,         //是否显示详细视图和列表视图的切换按钮
    cardView: false,          //是否显示详细视图
    detailView: false,        //是否显示父子表
    showExport: true,         //是否显示导出
    exportDataType: "basic",  //basic', 'all', 'selected'.
    onEditableSave: function (field, row, oldValue, $el) {
        // delete row[0];
        updata = {};
        updata[field] = row[field];
        updata['id'] = row['id'];
        $.ajax({
            type: "POST",
            url: "/backend/modify/",
            data: { postdata: JSON.stringify(updata), 'action': 'edit' },

```

```
        success: function (data, status) {
            if (status == "success") {
                alert("编辑成功");
            }
        },
        error: function () {
            alert("Error");
        },
        complete: function () {
        }
    });
},
columns: [{
    checkbox: true
}, {
    field: 'one',
    title: '列1',
    editable: {
        type: 'text',
        title: '用户名',
        validate: function (v) {
            if (!v) return '用户名不能为空';
        }
    }
},
//验证数字
//editable: {
//    type: 'text',
//    title: '年龄',
//    validate: function (v) {
//        if (isNaN(v)) return '年龄必须是数字';
//        var age = parseInt(v);
//        if (age <= 0) return '年龄必须是正整数';
//    }
//}
//}
//时间框
```

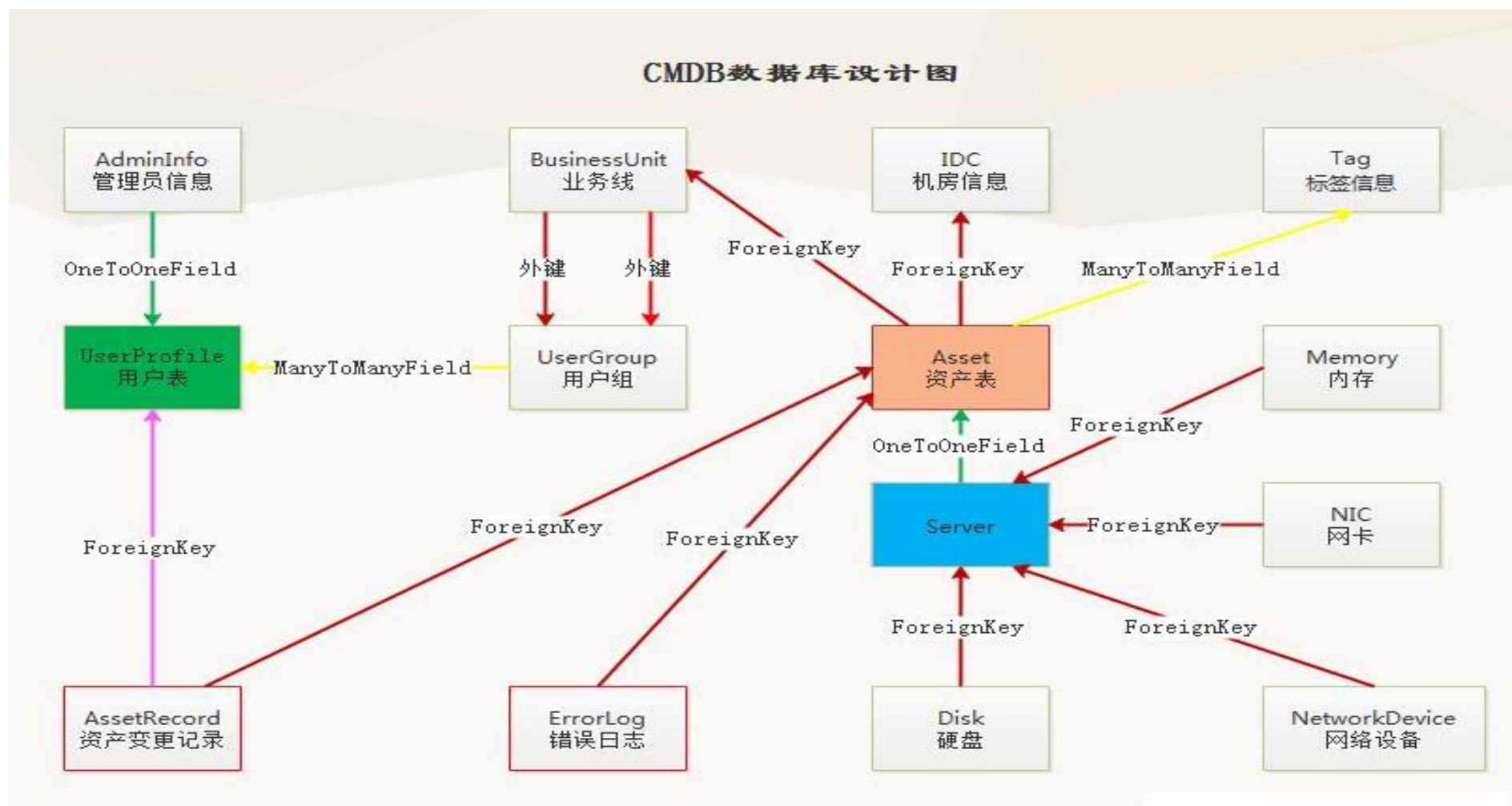
```
//editable: {
//    type: 'datetime',
//    title: '时间'
//}
//选择框
//editable: {
//    type: 'select',
//    title: '部门',
//    source: [{ value: "1", text: "研发部" }, { value: "2", text: "销售部" }, { value: "3", text: "行政部" }]
//}
//复选框
//editable: {
//type: "checkboxlist",
//separator:",",
//source: [{ value: 'bsb', text: '篮球' },
//    { value: 'ftb', text: '足球' },
//    { value: 'wsm', text: '游泳' }],
//}
//select2
//暂未使用到
//取后台数据
//editable: {
//    type: 'select',
//    title: '部门',
//    source: function () {
//        var result = [];
//        $.ajax({
//            url: '/Editable/GetDepartments',
//            async: false,
//            type: "get",
//            data: {},
//            success: function (data, status) {
//                $.each(data, function (key, value) {
//                    result.push({ value: value.ID, text: value.Name });
//                });
//            }
//        });
//    }
//}
```

```

//      }
//      });
//      return result;
//      }
//}
}]
});

```

三、CMDB 数据库设计



四、CMDB 总结

1. 三种采集资产方式

唯一标识

2. API

API 验证（tornado 源码，加密 cookie+时间限制+访问记录）

数据库表结构

3. 后台管理

告别 CURD，公共组件（前端+后端配置）

分类: [Django 框架](#)

10 分钟为你搭建一个超好用的 cmdb 系统

CMDB 是什么，作为 IT 工程师的想必已经听说过了，或者已经烂熟了，容我再介绍一下，以防有读者还不知道。CMDB 的全称是 Configuration Management Data Base，翻译下就是配置管理数据库，它存储与管理企业 IT 架构中设备的各种配置信息，它支撑服务流程的运转、发挥着配置信息的价值。在今天，无论是自动化运维、标准化运维、DevOps、甚至是时髦的智能运维，其实都离不开 CMDB，可以说 CMDB 是运维体系的基石，有了配置信息数据库，后面各种标准、流程都可以建立在 CMDB 基础之上，从而实现真正的标准化、自动化、智能化运维，节约运维成本的同时，也降低运维流程混乱带来的操作风险。

今天分享一个开源的 cmdb 系统的搭建过程，通过这一系列搭建的过程你不仅可以获得一个支持全文检索、自带 restful api 的 cmdb 系统，而且还可以学到不少时髦的技术。

后端技术：

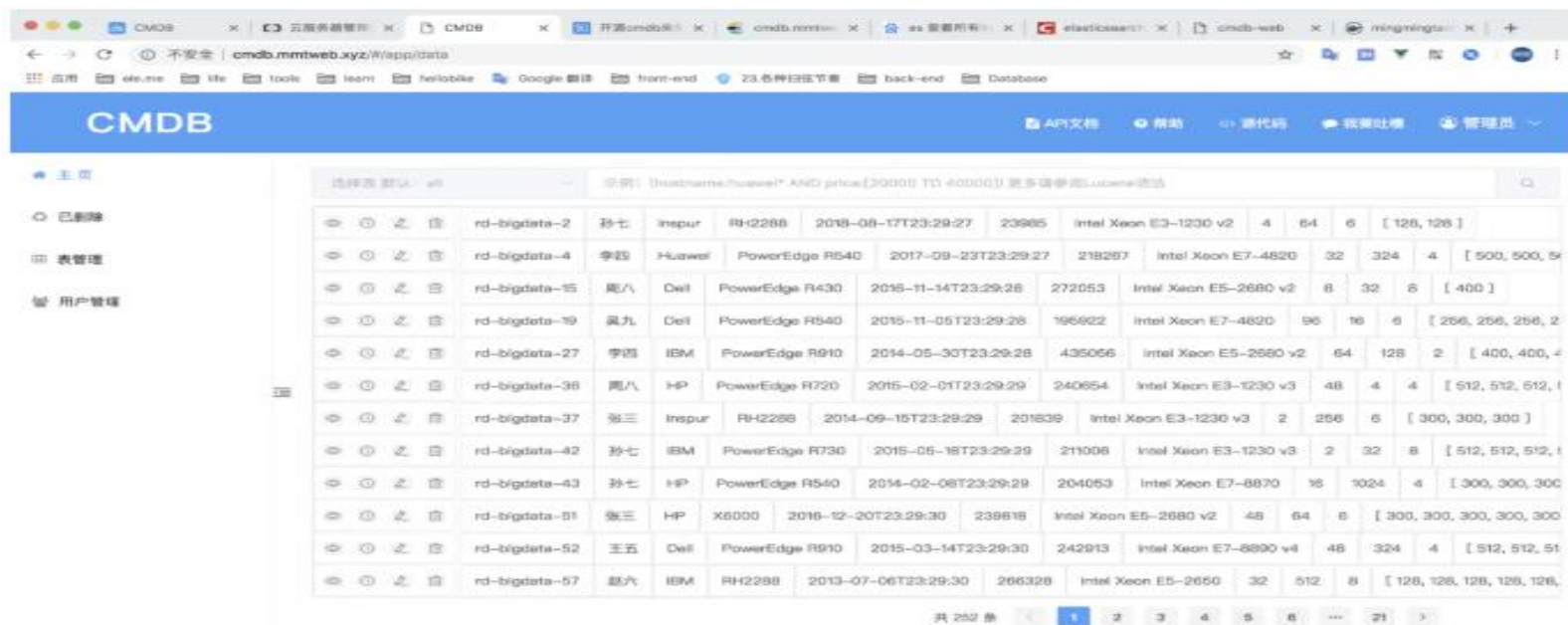
- Python3
- Django
- Django REST framework
- Elasticsearch
- uwsgi
- Nginx
- Docker

前端技术：

- Vue
- Element-ui
- Vue-Router
- Vuex

- Axios

先看一下这个 CMDB 系统的容颜，设计上参考了饿了么内部的 cmdb 系统:



The screenshot shows the CMDB web interface. The top navigation bar includes links for API文档, 帮助, 源代码, 我要吐槽, and 管理员. The left sidebar has links for 主页, 已删除, 表管理, and 用户管理. The main content area displays a table of server data with columns for ID, Name, Vendor, Model, Purchase Date, Serial Number, CPU, Memory, and Storage. The table is filtered by the query: hostname:huawei* AND price:[20000 TO 40000]. The table contains 10 rows of data.

ID	Name	Vendor	Model	Purchase Date	Serial Number	CPU	Memory	Storage
rd-bigdata-2	孙七	Inspur	RH2288	2018-08-17T23:29:27	23985	Intel Xeon E3-1230 v2	4	64
rd-bigdata-4	李四	Huawei	PowerEdge R540	2017-09-23T23:29:27	218267	Intel Xeon E7-4820	32	324
rd-bigdata-15	周八	Dell	PowerEdge R430	2015-11-14T23:29:28	272053	Intel Xeon E5-2680 v2	8	32
rd-bigdata-19	吴九	Dell	PowerEdge R540	2015-11-05T23:29:28	195922	Intel Xeon E7-4820	96	16
rd-bigdata-27	李四	IBM	PowerEdge R910	2014-05-30T23:29:28	435066	Intel Xeon E5-2680 v2	64	128
rd-bigdata-36	周八	HP	PowerEdge R720	2015-02-01T23:29:29	240654	Intel Xeon E3-1230 v3	48	4
rd-bigdata-37	张三	Inspur	RH2288	2014-09-15T23:29:29	201839	Intel Xeon E3-1230 v3	2	256
rd-bigdata-42	孙七	IBM	PowerEdge R730	2015-05-18T23:29:29	211006	Intel Xeon E3-1230 v3	2	32
rd-bigdata-43	孙七	HP	PowerEdge R540	2014-02-08T23:29:29	204053	Intel Xeon E7-8870	16	1024
rd-bigdata-51	张三	HP	X8000	2016-12-20T23:29:30	239818	Intel Xeon E5-2680 v2	48	64
rd-bigdata-52	王五	Dell	PowerEdge R910	2015-03-14T23:29:30	242913	Intel Xeon E7-8890 v4	48	324
rd-bigdata-57	赵六	IBM	RH2288	2013-07-06T23:29:30	266328	Intel Xeon E5-2650	32	512

open-cmdb

open-cmdb

基本功能有：热添加删除表、自定义字段类型，方便增删改查的前端界面，强大的搜索查找能力（后端使用 elasticsearch 存储数据）可以配合 kibana 使用，查看数据的删除修改记录、历史版本等，还带有表级权限管理，开放所有 API。

github 仓库

后端: <https://github.com/open-cmdb/cmdb>

前端: <https://github.com/open-cmdb/cmdb-web>

Popular repositories

cmdb

CMDB 配置管理系统 资产管理系统

Python ★ 124 🍴 49

cmdb-web

CMDB 配置管理系统 资产管理系统

JavaScript ★ 31 🍴 17

下面介绍两种方法搭建此开源 cmdb 系统，一个是使用 Docker，适用于 linux 操作系统，另一个是不使用 Docker，适用于 windows 和 linux。最后介绍下 vue 环境的搭建。

1. 使用 Docker

如果你熟悉容器技术，推荐使用此方法，不过最新的 Docker 目前还不支持大多数的 windows 版本，因此如果使用容器，请使用 ubuntu 或 centos 等 Linux 操作系统。首先要安装 Docker，安装 Docker 的方法请参考我之前的一篇文章 [docker 容器从入门到痴迷](#)，或直接网上搜索对应操作系统的安装方法对着做即可，没有难度。

环境准备：

- 1、一台可以访问互联网的 linux 服务器 内存最好 $\geq 4G$ ，并创建一个具有 sudo 权限的普通用户，注意要有 yum 命令，如果没有可以安装下。
- 2、一个 cmdb 专用的邮箱，用于发送密码和验证码，如果使用 163、qq 等第三方邮箱请在设置里面打开 POP3/SMTP/IMAP 服务并生成授权码。如果不使用注册和忘记密码功能，也可以不准备

一键安装

将下述代码保存到 install_cmdb.py 并执行 `sudo python3 install_cmdb.py` 即可一键安装。

```
# -*- coding: utf-8 -*-
import os
import subprocess
import argparse
import time

def base(cmd):
    if subprocess.call(cmd, shell=True):
        raise Exception("{} 执行失败".format(cmd))
```



```
def install_docker():
    base("sudo yum install -y yum-utils device-mapper-persistent-data lvm2")
    base("sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo")
    base("sudo yum makecache fast")
    base("sudo yum -y install docker-ce")
    if(not os.path.exists("/etc/docker")):
        base("mkdir -p /etc/docker")
    with open("/etc/docker/daemon.json", "w") as f:
        f.write('{\n    "registry-mirrors": ["https://9f4w4icn.mirror.aliyuncs.com"] \n}\n')
    base("sudo systemctl daemon-reload")
    base("sudo systemctl start docker")

def create_dir():
    if (not os.path.exists("/var/cmdb/db")):
        base("sudo mkdir -p /var/cmdb/db")
    if (not os.path.exists("/var/cmdb/es")):
        base("sudo mkdir -p /var/cmdb/es")

def run_db_container():
    base("sudo docker run --name cmdb-db -d -e MYSQL_ROOT_PASSWORD=cmdbcmdb -v /var/cmdb/db:/var/lib/mysql
mysql:5.7.21")

def run_es_container():
    base("sudo docker run --name cmdb-es -d -v /var/cmdb/es:/usr/share/elasticsearch/data
elasticsearch:5.6.8")

def init_db():
    base("sudo docker run -it --rm --link cmdb-db -e ENV=PRO -e DB_HOST=cmdb-db -e DB_PORT=3306 -e
DB_USERNAME=root -e DB_PASSWORD=cmdbcmdb -e DB_NAME=cmdb mingmingtang/cmdb init-db")
```

```
def run_cmdb_container(site_url, email_host, email_port, email_username, email_password):
    base("sudo docker run -d --name cmdb --link cmdb-db --link cmdb-es -p 80:80 -e ENV=PRO -e SITE_URL={} -e
DB_HOST=cmdb-db -e DB_PORT=3306 -e DB_USERNAME=root -e DB_PASSWORD=cmdbcmdb -e DB_NAME=cmdb -e
ELASTICSEARCH_HOSTS=cmdb-es -e EMAIL_HOST={} -e EMAIL_PORT={} -e EMAIL_USERNAME={} -e EMAIL_PASSWORD={}
mingmingtang/cmdb start".format(site_url, email_host, email_port, email_username, email_password))

def input_para(help):
    value = ""
    while(not value):
        value = raw_input(help)
    return value

if __name__ == '__main__':
    if(os.geteuid() != 0):
        raise("请以 root 权限运行")
    site_url = input_para("请输入网站域名或 IP (http://cmdb.xxx.com) : ")
    email_host = input_para("网站邮箱服务器 (smtp.163.com) : ")
    email_port = input_para("邮箱服务器端口 (25) : ")
    email_username = input_para("邮箱用户名 (cmdb@163.com) : ")
    email_password = input_para("邮箱密码|独立授权码 (P@ssw0rd) : ")

    print("开始安装 docker")
    install_docker()
    print("开始创建目录")
    create_dir()
    print("开始运行 mysql 容器")
    run_db_container()
    print("开始运行 elasticsearch 容器")
    run_es_container()
    print("等待数据库启动完成(10s)")
    time.sleep(10)
```

```
print("开始初始化数据库")
init_db()
print("开始运行 cmdb")
run_cmdb_container(site_url, email_host, email_port, email_username, email_password)
print("完成!")
```

输入网站地址和邮箱信息开始安装，如下图所示：

```
[ops@cmdb-2 ~]$ sudo python install-cmdb.py
请输入网站域名或IP (http://cmdb.xxx.com) : http://106.14.207.210
网站邮箱服务器 (smtp.163.com) : smtp.163.com
邮箱服务器端口 (25) : 25
邮箱用户名 (cmdb@163.com) : mmt_cmdb@163.com
邮箱密码|独立授权码 (P@ssw0rd) : mmt12345678
开始安装docker
已加载插件：fastestmirror
Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
base | 3.6 kB 00:00:00
epel | 4.7 kB 00:00:00
extras | 3.4 kB 00:00:00
updates | 3.4 kB 00:00:00
(1/5): epel/x86_64/group_gz | 266 kB 00:00:00
(2/5): epel/x86_64/updateinfo | 895 kB 00:00:00
(3/5): extras/7/x86_64/primary_db | 166 kB 00:00:00
(4/5): epel/x86_64/primary_db | 6.3 MB 00:00:00
(5/5): updates/7/x86_64/primary_db | 6.0 MB 00:00:00
Determining fastest mirrors
正在解决依赖关系
--> 正在检查事务
--> 软件包 device-mapper-persistent-data.x86_64.0.7.0-0.1.rc6.el7_4.1 将被 安装
--> 正在处理依赖关系 libaio.so.1(LIBAIO_0.4)(64bit)，它被软件包 device-mapper-persistent-data-0.7.0-0.1.rc6.el7_4.1.x86_64 需要
--> 正在处理依赖关系 libaio.so.1(LIBAIO_0.1)(64bit)，它被软件包 device-mapper-persistent-data-0.7.0-0.1.rc6.el7_4.1.x86_64 需要
--> 正在处理依赖关系 libaio.so.1()(64bit)，它被软件包 device-mapper-persistent-data-0.7.0-0.1.rc6.el7_4.1.x86_64 需要
```

如果一切顺利一会儿后您将看到安装完成，如果失败了可能就要调整一些系统参数并删除已运行的容器重新执行了，不过根据我的安装经验，基本不会出错，容器还是非常方便部署的。

```

47cafa6a79d0: Pull complete
79fcf5a213c7: Pull complete
fd532571c5d3: Pull complete
31600c9f9b48: Pull complete
78e8e9b5d10e: Pull complete
0710e619e883: Pull complete
e511da65ffab: Pull complete
49a4f0d2a48f: Pull complete
09e316d84cd5: Pull complete
9ca31a47f424: Pull complete
8e7e35190d7f: Pull complete
413588054f4d: Pull complete
35f5766cfb0d: Pull complete
3b2bbd1879e7: Pull complete
Digest: sha256:cc88067855a5ab7c7a51f5064fadb92161ad597323a222f8088beead4896ecff
Status: Downloaded newer image for elasticsearch:5.6.8
4fa0658a76abfad9f56b68351cf575b3e986108ac065080508b26593d862a53f
等待数据库启动完成(10s)
开始初始化数据库
Unable to find image 'mingmingtang/cmdb:latest' locally
latest: Pulling from mingmingtang/cmdb
d9aaf4d82f24: Pull complete
bc4a6c097929: Pull complete
Digest: sha256:9d2b02886f26c692a2836e387b7cfeb1fcdac6ebdfffb1a2ab57cf8325975a4ef
Status: Downloaded newer image for mingmingtang/cmdb:latest
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql: [Warning] Using a password on the command line interface can be insecure.
开始运行cmdb
21c6a3fc43af6ef652df2ef3fdb6faf12bcb0bca9f53d0a8200760889715ab55
完成!

```

执行下述命令

```
sudo docker ps
```

将看到三个正在运行的容器，分别是 cmdb, cmdb-es, cmdb-db，如下图所示

```

[ops@cmdb-2 ~]$ sudo docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
21c6a3fc43af	mingmingtang/cmdb	"start"	27 seconds ago	Up 26 seconds	0.0.0.0:80->80/tcp	cmdb
4fa0658a76ab	elasticsearch:5.6.8	"/docker-entrypoint...."	About a minute ago	Up About a minute	9200/tcp, 9300/tcp	cmdb-es
3b20e2094daa	mysql:5.7.21	"docker-entrypoint.s..."	2 minutes ago	Up 2 minutes	3306/tcp	cmdb-db

其中 cmdb 运行着 web 服务器（nginx, uwsgi, django, vue.js），cmdb-es 运行着 Elasticsearch 全文检索引擎，也存储你的配置信息，cmdb-db 运行着 mysql，保存着 web 服务器的元数据（django 的知识库）。

在浏览器中输入"localhost" 尽情的开始享用吧。

2. 不使用 Docker

下面的内容主要是分享给 windows 用户的，linux 用户也可以对比操作。使用 Docker 虽然方便部署，但它屏蔽了一些细节，不利于二次开发和问题排查。在不使用 Docker 的情况下，我们不仅要装软件，还要安装依赖，配置环境，虽然麻烦，但是可以学到更多知识，出了问题也可以很快定位，更能加深对项目框架的理解，这点付出也是值得的。

(1) 安装 mysql，创建数据库，配置权限

如果你的本机已经安装 mysql，则不心再安装，直接创建数据库，配置权限即可。

1. 安装 mysql

从网官下载最新的 MySQL Community Server [<https://dev.mysql.com/downloads/mysql/>]

比如我下载的是 mysql-8.0.12-winx64.zip ，这是个免安装版本，直接解压到你想要安装的目录内，并在里面新建 my.ini 文件，位置如下图所示：



image.png

my.ini 的文件内容如下所示：

```
[mysqld]
# 设置 3306 端口
port=3306
# 设置 mysql 的安装目录
basedir=D:\program\mysql\mysql-8.0.12-winx64
# 设置 mysql 数据库的数据的存放目录
datadir=D:\program\mysql\mysql-8.0.12-winx64\data
# 允许最大连接数
max_connections=200
# 允许连接失败的次数。这是为了防止有人从该主机试图攻击数据库系统
max_connect_errors=10
# 服务端使用的字符集默认为 UTF8
```



```
character-set-server=utf8
# 创建新表时将使用的默认存储引擎
default-storage-engine=INNODB
# 默认使用“mysql_native_password”插件认证
default_authentication_plugin=mysql_native_password
[mysql]
# 设置 mysql 客户端默认字符集
default-character-set=utf8
[client]
# 设置 mysql 客户端连接服务端时默认使用的端口
port=3306
default-character-set=utf8
```

请注意下面的路径设置要正确，

```
# 设置 mysql 的安装目录
basedir=D:\program\mysql\mysql-8.0.12-winx64
# 设置 mysql 数据库的数据的存放目录
datadir=D:\program\mysql\mysql-8.0.12-winx64\data
```

如果你想从任意一个命令窗口启动 *mysql*，请把 *D:\program\mysql\mysql-8.0.12-winx64\bin* 加入环境变量。*

这个配置文件 *my.ini* 一定要保存为 **gbk** 格式，否则会报错，我费了好长时间才发现这个问题。

接下来在 MySQL 安装目录的 *bin* 目录（*D:\program\mysql\mysql-8.0.12-winx64\bin*）下以**管理员权限**执行命令：*mysqld --initialize --console*；执行完成后，在输出信息中会打印 *root* 用户的初始密码，比如

```
[Server] A temporary password is generated for root@localhost: rIafvf5f5G,a
```

表示临时密码为 *rlafvf5f5G,a*，用于 *root* 用户第一次登陆，之后再修改 *root* 用户的密码。

这一步执行后完成初始化操作，并在安装目录下生成 *data* 文件夹，用于存放数据。执行

```
mysqld --install
```

完成 *mysql* 服务的安装，安装完成之后，就可以通过命令 *net start mysql* 即启动 *mysql* 的服务了。通过命令 *net stop mysql* 停止服务。通过命令 *sc delete mysql /mysqld -remove* 卸载 *mysql* 服务。接下来就可以建库、用户、分配权限了。

修改 *root* 密码：

在 *mysql* 安装目录的 *bin* 目录下执行命令：*mysql -u root -p* 然后输入上面的密码，进入 *mysql* 环境，执行

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '新密码';
```

注意命令尾的“；”一定要有，这是 *mysql* 的语法，

管理员 *root* 的 *host* 是 *localhost*，代表仅限 *localhost* 登录访问。如果要允许开放其他 *ip* 登录，则需要添加新的 *host*。如果要允许所有 *ip* 访问，可以直接修改成“%”；

```
ALTER USER 'root'@'%' IDENTIFIED BY '远程登陆密码';
```

2. 创建数据库，并分配用户权限

使用 root 用户登陆 mysql 并执行

```
mysql>create database cmdb;
```

即可创建数据库 cmdb，但是这个数据库只能有 root 访问，如果要使用其他用户访问，则先新建用户，例如让 aaron 用户可以完全控制 cmdb

```
mysql>CREATE USER 'aaron'@'%' IDENTIFIED BY 'aaron';
```

Query OK, 0 rows affected (0.48 sec)

```
mysql> grant all on cmdb.* to 'aaron'@'%';
```

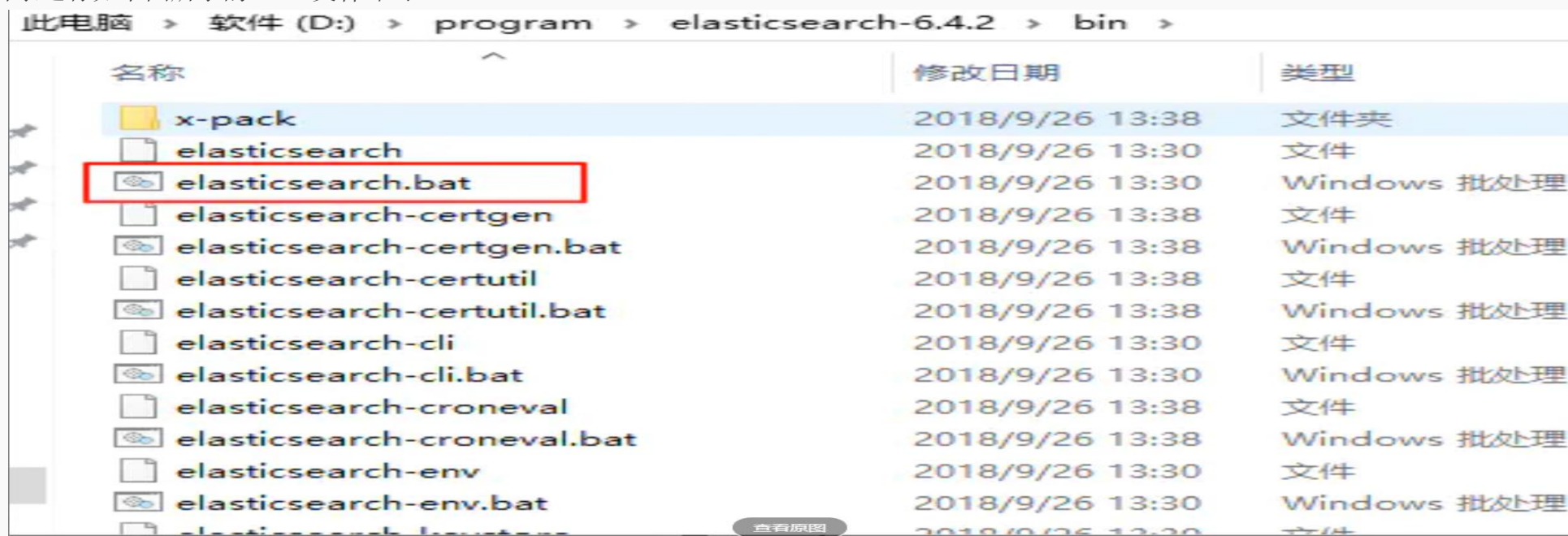
Query OK, 0 rows affected (0.23 sec)

至此 mysql 已安装配置完毕。

(2) 安装 Elasticsearch

全文搜索属于最常见的需求，开源的 Elasticsearch（以下简称 Elastic）是目前全文搜索引擎的首选。

它可以快速地储存、搜索和分析海量数据。维基百科、Stack Overflow、Github 都采用它。安装 Elasticsearch 非常简单，官网下载解压，进入其 bin 目录运行如下图所示的 bat 文件即可：



它是一个 java 编写的程序，如果要修改占用内存的大小，请找到配置文件，例如修改为 -Xmx10g -Xms10g 表示初始化 10 个 G 的内存空间给 Elasticsearch，Elasticsearch 最大使用的内存也是 10G，一般情况下，设置为内存的一半大小，但最好不超过 32 G，根据需求，生产环境适合调大，测试环境适当调小。

如果运行失败，说明本机没有安装 java，或者没有正确地配置 java 环境变量，这些操作也非常简单，网上到处都是，不在此详述。

（3）运行 cmdb 后端 api 服务、前端 ui

首先准备 Python3 的环境，这个也很简单，直接官网下载，运行即可，记得把 Python.exe 所在的路径添加到 Path 变量中。

如果你的电脑里有多个项目，为防止项目的依赖包版本冲突，建议使用 virtualenv 来为每个项目创建一个虚拟的 Python 环境，将各自的依赖包装在自己的虚拟环境里。

（1）部署后端

执行以下命令，注意命令后面的注释。

```
git clone https://github.com/open-cmdb/cmdb.git
cd cmdb
#如创建了虚拟环境，请先激活
pip install -r requirements.txt #如果这一步有包安装失败，提示缺少 microsoft visual c++ 14.0 的话，请在网站
https://www.lfd.uci.edu/~gohlke/pythonlibs/ 上查找相应的 whl 文件，直接 pip install .whl 文件即可。
```

接下来修改 3 个文件

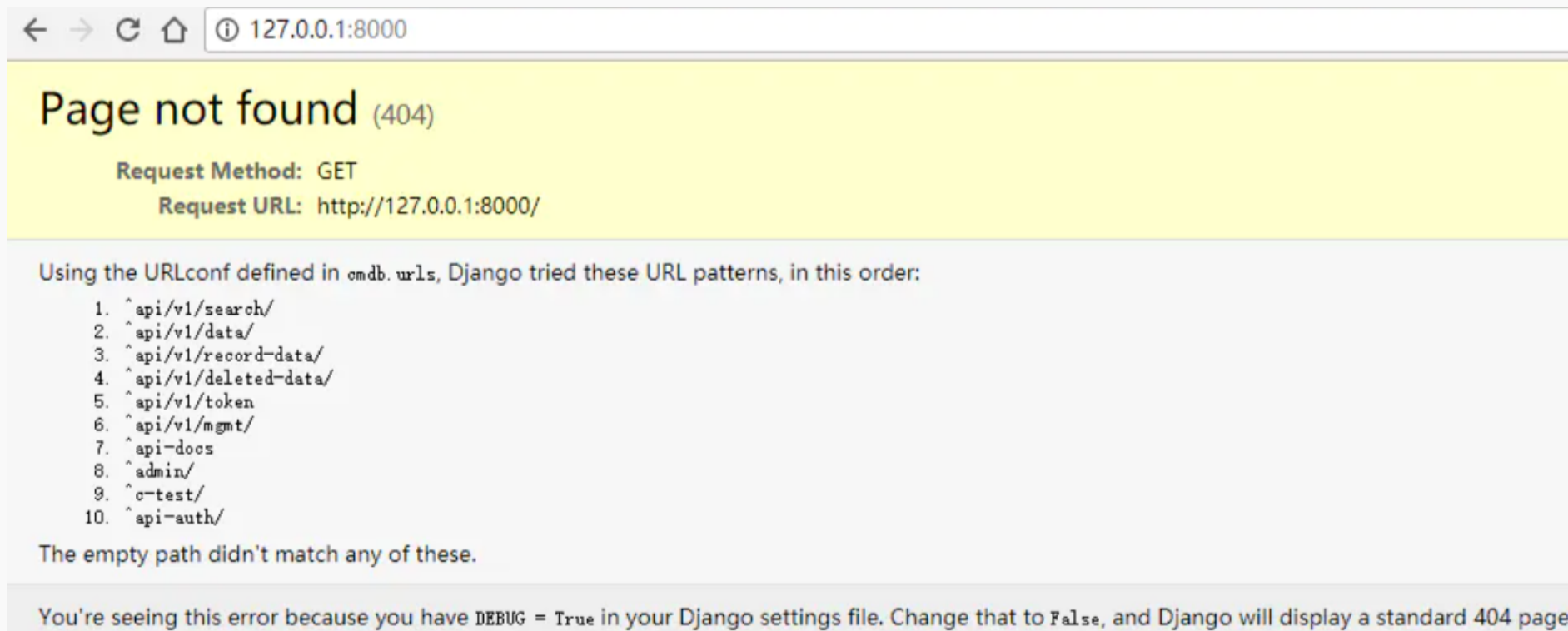
1. 修改 apps/mgmt/views.py 文件，注释掉 “from . import initialize” 这一行。
2. 修改 manage.py
将 APP_NAME = BASE_DIR.rsplitt("/", 1)[-1] 修改为
APP_NAME = BASE_DIR.rsplitt("\\", 1)[-1]，这是因为 windows 的路径\ 在 python 里会变成 \。
3. 修改 cmdb/settings.py 文件，修改 mysql 数据库的配置信息如下所示：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': "cmdb",
        "HOST": "127.0.0.1",
        "PORT": 3306,
        "USER": "aaron",
        "PASSWORD": "aaron"
    }
}
```

接着在命令窗口继续执行以下操作：请关注注释内容。

```
python manage.py makemigrations
python manage.py migrate
python manage.py cmdb_create_superuser #这一步创建一可以登陆的管理员用户
#修改 apps/mgmt/views.py 文件，取消注释“ from . import initialize ”
python manage.py runserver #这一步启动后端的 api 服务
```


此时一个后端的服务已经启动了，在浏览器中打开 “127.0.0.1:8000”就可以看到 api 的接口了。



（2）使用 nginx 部署前端并连接后端 api 服务

在命令容器执行以下命令：

```
git clone https://github.com/open-cmdb/cmdb-web.git
```

获取前端的源代码，然后下载下载 nginx 压缩包，并解压至安装目录，修改配置文件 nginx.conf，添加如下 server 配置：

```
server {  
    listen 8080;  
    server_name localhost;  
    error_page 500 502 503 504 /50x.html;  
    location = /50x.html {  
        root html;  
    }  
  
    root E:\GitHub\cmdb-web\dist;
```

```
index index.html;
location / {
    try_files $uri $uri/ @router;
    index index.html;
}

location @router {
    rewrite ^.*$ /index.html last;
}

location ~ /api{
    proxy_pass http://127.0.0.1:8000;
}

}
```

其中以配置

```
location ~ /api{
    proxy_pass http://127.0.0.1:8000;
}
```

让前台发过来中以 api 开头的 url 请求都转发至 http://127.0.0.1:8000 进行解析，即第一步部署的 django 项目，这样就连接了前端和后端。然后我们在 nginx.exe 所在的目录下启动 nginx 服务。

```
D:\program\nginx\nginx-1.14.0>nginx.exe
```

接下来在浏览器中输入 127.0.0.1:8080 即可正常访问本文开始处的 cmdb 系统，您可以尝试下强大的搜索功能及增删改功能。

← → ↺ ⌂

127.0.0.1:8080/#/app/data

☆ 🔴

CMDB

API文档 帮助 源代码 我要吐槽 郑征

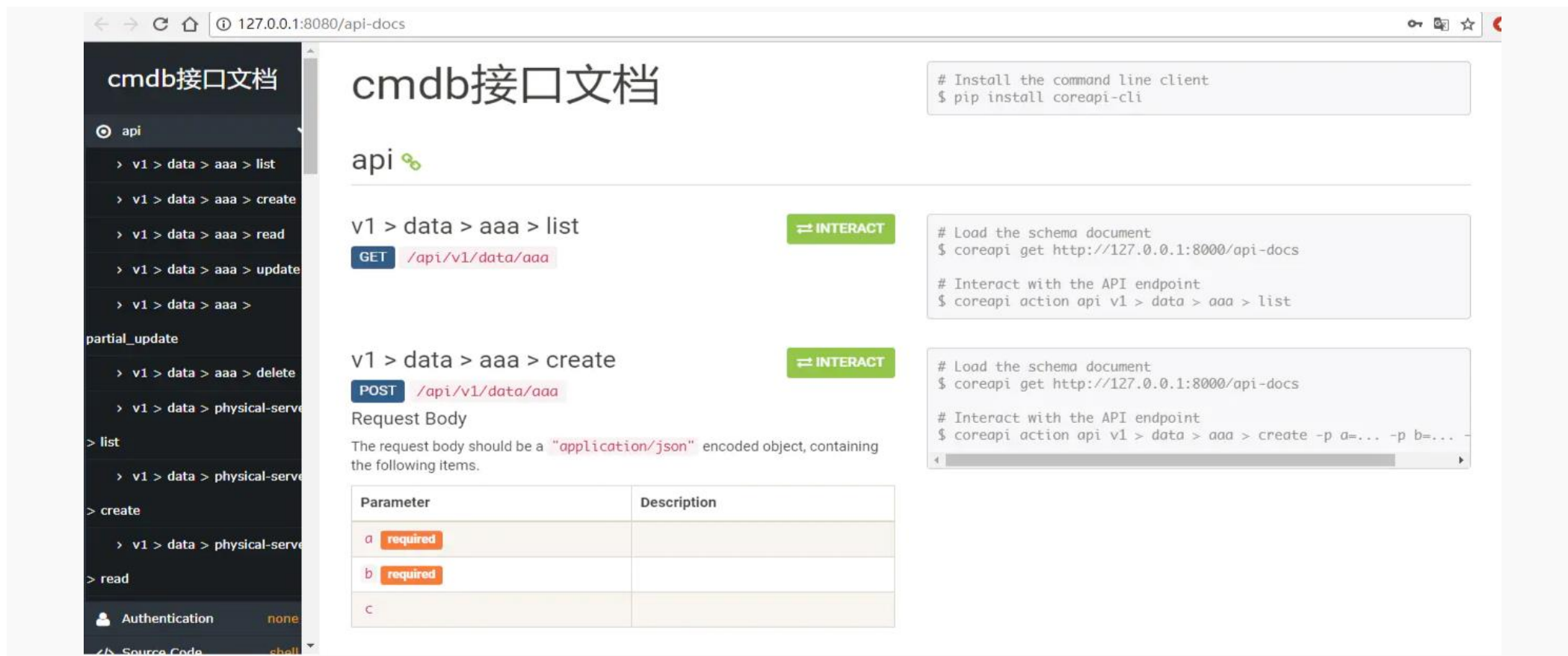
选择表 默认: all

陈二 AND IBM

🔍

👁 ⌚ 🗑	rd-bigdata-2	陈二	IBM	PowerEdge R430	2016-08-17T21:50:46	490843	Intel Xeon E7-4820	96	64	2	[400, 400, 400, 400]
👁 ⌚ 🗑	rd-bigdata-121	陈二	IBM	RH2288	2016-11-12T21:51:56	110038	Intel Xeon E5-2670	2	324	8	[256, 256, 256]
👁 ⌚ 🗑	rd-bigdata-297	陈二	IBM	PowerEdge R540	2015-02-19T21:53:26	56782	Intel Xeon E7-8890 v4	48	32	4	[500, 500, 500, 500, 500, 500]
👁 ⌚ 🗑	rd-bigdata-670	陈二	IBM	PowerEdge R730	2018-03-14T21:56:18	364245	Intel Xeon E3-1230 v2	4	32	8	[512, 512, 512, 512]
👁 ⌚ 🗑	rd-bigdata-826	陈二	IBM	PowerEdge R730	2015-03-14T21:57:09	349132	Intel Xeon E3-1230 v2	64	8	4	[256, 256, 256, 256, 256, 256]
👁 ⌚ 🗑	rd-bigdata-972	陈二	IBM	5288 V3	2017-04-01T21:58:00	214419	Intel Xeon E7-4820	64	2	6	[256, 256, 256, 256, 256, 256, 256, 256]
👁 ⌚ 🗑	rd-bigdata-13	陈二	IBM	PowerEdge R910	2016-06-15T21:50:54	175065	Intel Xeon E5-2650	96	16	4	[320, 320, 320, 320, 320, 320, 320]
👁 ⌚ 🗑	rd-bigdata-28	陈二	IBM	PowerEdge R730	2013-11-17T21:51:06	470617	Intel Xeon E3-1230 v2	8	8	8	[256, 256]
👁 ⌚ 🗑	rd-bigdata-411	陈二	IBM	PowerEdge R730	2015-09-30T21:54:20	36055	Intel Xeon E7-8890 v4	64	1024	6	[128, 128, 128, 128, 128, 128]
👁 ⌚ 🗑	rd-bigdata-502	陈二	IBM	PowerEdge R910	2015-11-08T21:54:59	38087	Intel Xeon E7-4820	96	512	6	[400, 400, 400, 400, 400, 400]
👁 ⌚ 🗑	rd-bigdata-746	陈二	IBM	X6000	2014-01-03T21:56:42	386982	Intel Xeon E3-1230 v3	32	16	4	[256, 256, 256, 256, 256, 256, 256, 256]

点击上方【API 文档】 可以访问 cmdb 的接口文档，非常方便。



cmdb接口文档

api

- > v1 > data > aaa > list
- > v1 > data > aaa > create
- > v1 > data > aaa > read
- > v1 > data > aaa > update
- > v1 > data > aaa >
- partial_update
- > v1 > data > aaa > delete
- > v1 > data > physical-server
- > list
- > v1 > data > physical-server
- > create
- > v1 > data > physical-server
- > read
- Authentication none
- Source Code shell

cmdb接口文档

api

v1 > data > aaa > list

GET /api/v1/data/aaa

INTERACT

Install the command line client
\$ pip install coreapi-cli

Load the schema document
\$ coreapi get http://127.0.0.1:8080/api-docs

Interact with the API endpoint
\$ coreapi action api v1 > data > aaa > list

v1 > data > aaa > create

POST /api/v1/data/aaa

INTERACT

Request Body

The request body should be a "application/json" encoded object, containing the following items.

Parameter	Description
a required	
b required	
c	

Load the schema document
\$ coreapi get http://127.0.0.1:8080/api-docs

Interact with the API endpoint
\$ coreapi action api v1 > data > aaa > create -p a=... -p b=...

至此系统搭建完毕。如果要用于生产环境，请使用 linux 操作系统，并使用 uwsgi 来驱动 django 项目。

3. Vue 环境搭建

我想你不会仅仅满足于将别人的项目下载下来能运行就行了，你肯定想对其进行改造来满足自己的需求。因此你可能会需要修改前端或后端，后端的修改其实上面部署的已经可以了，你可以直接阅读 django 项目的源代码进行修改调试。如果要修改前端代码进行调试，你就需要搭建 Vue 环境。你可能会问了，Vue 是个啥？Vue 是一个 javascript 框架，如果说 jQuery，你可能就知道了，使用方法是类似的，在 html 上引入一行 javascript 的文件，就可以使用框架的特性了。Vue 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

学习 vue 需要有 html、css、javascript 基础

新手可以通过 html 上引入 Vue 的 js 文件来使用 vue，如下所示：

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="utf-8">
<title>Vue 测试实例 </title>
  <script src="https://cdn.staticfile.org/vue/2.4.2/vue.min.js"></script>
</head>
<body>
<div id="app">
  <p>{{ message }}</p>
</div>

<script>
  new Vue({
    el: '#app',
    data: {
      message: 'Hello Vue.js!'
    }
  })
</script>
</body>
</html>
```

但是在较复杂的项目中，还是要使用工具来帮助我们管理项目的层级及文件之间的依赖关系，这就需要使用 `vue` 的命令行工具 `vue-cli`，`vue-cli` 需要 `npm` 工具来安装，`npm` 工具集成在 `node.js` 中，因此需要安装 `node.js`。安装 `node.js` 非常简单，直接官网下载解压即可使用：

此电脑 > 新加卷 (D:) > program > node-v8.12.0-win-x64

名称	修改日期	类型	大小
node_modules	2018/10/25 21:40	文件夹	
CHANGELOG.md	2018/9/4 1:13	MD 文件	55 KB
cnpm	2018/10/25 21:36	文件	1 KB
cnpm.cmd	2018/10/25 21:36	Windows 命令脚本	1 KB
LICENSE	2018/9/4 1:13	文件	59 KB
node.exe	2018/9/11 2:29	应用程序	22,475 KB
node_etw_provider.man	2018/2/11 9:08	MAN 文件	11 KB
node_perfctr_provider.man	2018/2/11 8:08	MAN 文件	5 KB
nodevars.bat	2018/2/11 9:08	Windows 批处理	1 KB
npm	2018/2/11 9:08	文件	1 KB
npm.cmd	2018/2/11 8:08	Windows 命令脚本	1 KB
npx	2018/9/4 1:13	文件	1 KB
npx.cmd	2018/9/4 1:13	Windows 命令脚本	1 KB
README.md	2018/9/4 1:13	MD 文件	28 KB

将 npm 所在路径添加到环境变量 Path 中，你就可以在任意的命令窗口使用 npm 命令了。

1、安装 vue-cli

先安装淘宝镜像，大家都知道国内直接使用 npm 的官方镜像是非常慢的，这里推荐使用淘宝 NPM 镜像。

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

执行结果如下：

```
C:\Users\xx>npm install -g cnpm --registry=https://registry.npm.taobao.org
npm WARN deprecated socks@1.1.10: If using 2.x branch, please upgrade to at least 2.1.6 to avoid a serious bug with sock
et data flow and an import issue introduced in 2.1.0
D:\program\node-v8.12.0-win-x64\cnpm -> D:\program\node-v8.12.0-win-x64\node_modules\cnpm\bin\cnpm
+ cnpm@6.0.0
updated 1 package in 65.377s
C:\Users\xx>
```

这样就可以使用 cnpm 命令来安装模块了：使用 cnpm 安装 vue-cli


```

C:\Users\xx>cnpm install --global vue-cli
Downloading vue-cli to D:\program\node-v8.12.0-win-x64\node_modules\vue-cli_tmp
Copying D:\program\node-v8.12.0-win-x64\node_modules\vue-cli_tmp\_vue-cli@2.9.6@vue-cli to D:\program\node-v8.12.0-win-x64\node_modules\vue-cli
Installing vue-cli's dependencies to D:\program\node-v8.12.0-win-x64\node_modules\vue-cli\node_modules
1/20] commander@2.9.0 installed at node_modules\_commander@2.19.0@commander
2/20] multimatch@2.1.0 installed at node_modules\_multimatch@2.1.0@multimatch
3/20] ora@1.3.0 installed at node_modules\_ora@1.4.0@ora
4/20] minimatch@3.0.0 installed at node_modules\_minimatch@3.0.4@minimatch
5/20] rimraf@2.5.0 existed at node_modules\_rimraf@2.6.2@rimraf
6/20] chalk@2.1.0 installed at node_modules\_chalk@2.4.1@chalk
7/20] semver@5.1.0 installed at node_modules\_semver@5.6.0@semver
8/20] consolidate@0.14.0 installed at node_modules\_consolidate@0.14.5@consolidate
9/20] tildify@1.2.0 installed at node_modules\_tildify@1.2.0@tildify
10/20] uid@0.0.2 installed at node_modules\_uid@0.0.2@uid
11/20] user-home@2.0.0 installed at node_modules\_user-home@2.0.0@user-home
12/20] coffee-script@1.12.7 existed at node_modules\_coffee-script@1.12.7@coffee-script
13/20] read-metadata@1.0.0 installed at node_modules\_read-metadata@1.0.0@read-metadata
14/20] validate-npm-package-name@3.0.0 installed at node_modules\_validate-npm-package-name@3.0.0@validate-npm-package-name
15/20] metalsmith@2.1.0 installed at node_modules\_metalsmith@2.3.0@metalsmith
16/20] download-git-repo@1.0.1 installed at node_modules\_download-git-repo@1.1.0@download-git-repo
17/20] handlebars@4.0.5 installed at node_modules\_handlebars@4.0.12@handlebars
18/20] request@2.67.0 installed at node_modules\_request@2.88.0@request
19/20] async@2.4.0 installed at node_modules\_async@2.6.1@async
20/20] inquirer@6.0.0 installed at node_modules\_inquirer@6.2.0@inquirer
Deprecate metalsmith@2.3.0 > gray-matter@2.1.1 > coffee-script@1.12.4 CoffeeScript on NPM has moved to "coffeescript" (no hyphen)
Recently updated (since 2018-10-18): 2 packages (detail see file D:\program\node-v8.12.0-win-x64\node_modules\vue-cli\node_modules\.recently_updates.txt)
2018-10-20
  → request@2.88.0 > mime-types@2.1.19(2.1.21) (11:37:07)
  → request@2.88.0 > mime-types@2.1.21 > mime-db@1.37.0(1.37.0) (09:39:34)
All packages installed (235 packages installed from npm registry, used 17s(network 17s), speed 388.29kB/s, json 221(1.58KB), tarball 4.85MB)
[vue-cli@2.9.6] link D:\program\node-v8.12.0-win-x64\vue@ -> D:\program\node-v8.12.0-win-x64\node_modules\vue-cli\bin\vue
[vue-cli@2.9.6] link D:\program\node-v8.12.0-win-x64\vue-init@ -> D:\program\node-v8.12.0-win-x64\node_modules\vue-cli\bin\vue-init

```

[查看原图](#)

然后我们就可以使用 `vue init webpack my-project` 命令来建一个项目 `my-project`，这里需要进行一些配置，可以直接回车来使用默认配置，如下图所示：

```
C:\Users\xx>vue init webpack my-project

? Project name my-project
? Project description A Vue.js project
? Author somenzz <somenzz@163.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests Yes
? Pick a test runner jest
? Setup e2e tests with Nightwatch? Yes
? Should we run `npm install` for you after the project has been created? (recommended) npm

vue-cli · Generated "my-project".
```

等待其初始化完毕后，进入项目，执行以下命令：

```
C:\Users\xx>cd my-project
C:\Users\xx\my-project>cnpm install
C:\Users\xx\my-project>cnpm run dev
```

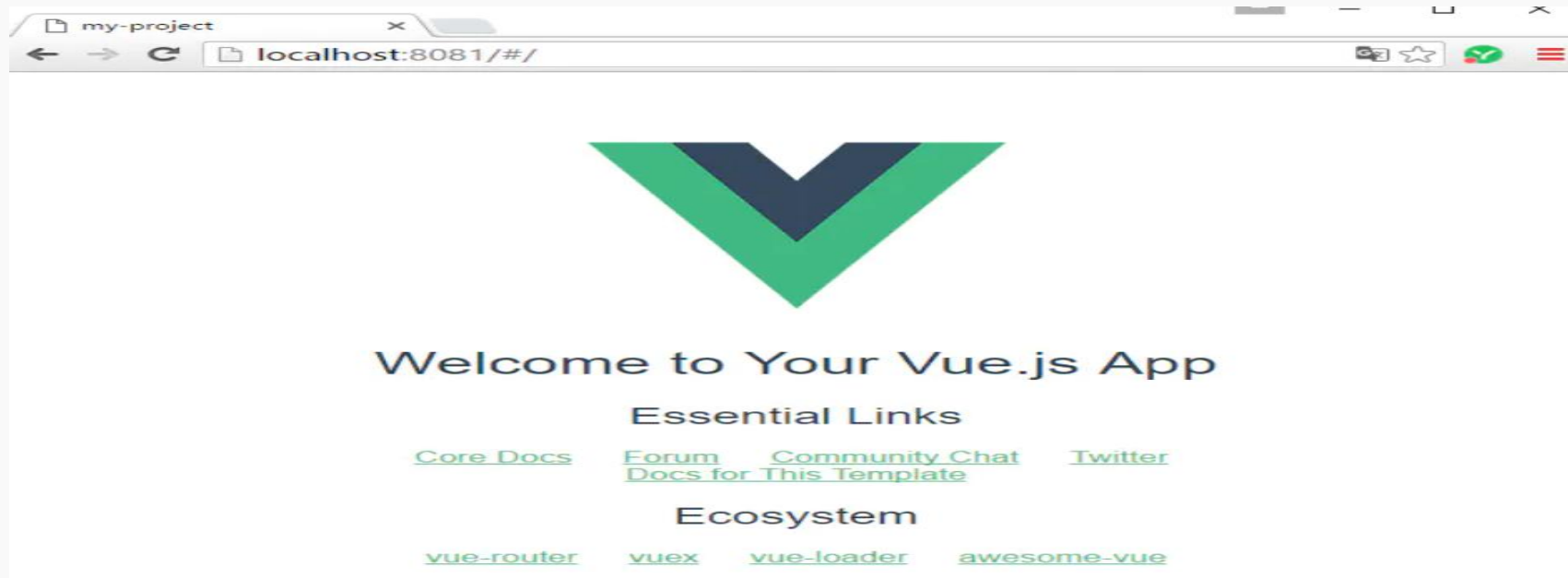
此时会自动 build ，运行成功后会打印如下信息表示服务已正常启动：

```
DONE Compiled successfully in 40009ms

Your application is running here: http://localhost:8081
```

3complicesucce.png

此时打开浏览器，输入 <http://localhost:8081>，即可看到 my-project 在浏览器中的展示。



如果要把这个页面部署在 nginx 服务器上，你还需要 build 命令来生成静态资源，如下图所示：

```

C:\Users\xx\my-project>npm run-script build

> my-project@1.0.0 build C:\Users\xx\my-project
> node build/build.js

Hash: 7celeb7bc175fec1c987
Version: webpack 3.12.0
Time: 70334ms

   Asset      Size  Chunks             Chunk Names
static/js/vendor.e2e62f45f19b9d1a1b09.js    112 kB      0  [emitted]  vendor
static/js/app.c70baf7ff544d9305be4.js     11.6 kB      1  [emitted]  app
static/js/manifest.2ae2e69a05c33dfc65f8.js    857 bytes      2  [emitted]  manifest
static/css/app.30790115300ab27614ce176899523b62.css    432 bytes      1  [emitted]  app
static/css/app.30790115300ab27614ce176899523b62.css.map    828 bytes      1  [emitted]
static/js/vendor.e2e62f45f19b9d1a1b09.js.map    553 kB      0  [emitted]  vendor
static/js/app.c70baf7ff544d9305be4.js.map    22.6 kB      1  [emitted]  app
static/js/manifest.2ae2e69a05c33dfc65f8.js.map    4.97 kB      2  [emitted]  manifest
index.html    512 bytes      2  [emitted]

Build complete.

Tip: built files are meant to be served over an HTTP server.
Opening index.html over file:/// won't work.

```

image.png

此时会在 my-project 下生成 dist 目录，相当于我们写程序编译生成的目标文件。my-project 下的内容及说明如下图所示：

/Users/xx/my-project/

~ build/ → webpack相关配置文件

- build.js
- check-versions.js
- logo.png
- utils.js
- vue-loader.conf.js
- webpack.base.conf.js
- webpack.dev.conf.js
- webpack.prod.conf.js

~ config/ → VUE基本配置文件

- dev.env.js
- index.js
- prod.env.js
- test.env.js

~ dist/

+ static/

- index.html

+ node_modules/

~ src/ → 项目核心文件，我们所写的代码都放在这个文件夹下

+ assets/ 静态资源 css,less,sass,js

+ components/ 公共组件

+ router/ 路由

- App.vue 根组件

- main.js 入口文件

+ static/ → 静态资源（图片类）

+ test/

- index.html 主页

- package.json 项目基本信息

- README.md 项目说明

需要掌握部分

[查看原图](#)

此时已经可以开启你的 `vue` 之旅了。

当学会了 `Vue` 之后，你就可以修改本项目的前端源代码来满足自己的需求了，进入 `src` 目录，查看并修改源代码之后，进入 `cmdb_web` 的项目目录，执行

```
# 安装依赖，如果慢可换成 cnpm
```

```
npm install
```

```
# 启动服务，默认端口为 8080，如果被占用会自动选取一个未被占用的端口
```

```
npm run dev
```

```
# 建立静态文件，可以放在 nginx 上运行
```

```
npm run build
```

```
# 查看建立报告
```

```
npm run build --report
```

即可将生成的 `dist` 部署到 `web` 服务器了。

也许你会认为这样实在太麻烦了，不如自己动手写一个 `cmdb` 系统，如果时间允许当然可以自己写，但是我不推荐这样做，上述这些操作其实都很简单，在熟悉之后就像打字一样容易，而且自己写一个同样效果的系统要直接拿优秀的代码来改造要慢得多，大神例外。作为一般程序员我们应该避免重复造轮子，学会站在巨人的肩膀上。因此开发一个项目最好是找 `github` 上类似的优秀开源项目，借鉴其优良设计，甚至可以直接拿来二次开发，这才是最高效的做法。

（完）