1. **概述**
2. **实验环境**

后端使用Python的Django 的1.10.2版本作为Web Server；

前端使用Chrome等浏览器实现浏览，网址为<http://osvt.net:3000/static/>

1. **工具实现技术**

本系统在实现上，通过Python的Django提供Restful的API接口，前端通过AJAX来通过API接口来获得所需的信息（主要为JSON），并通过HTML5、JQuery、Bootstrap等技术来展示。

## 3.1 后端

后端采用Python的Django框架为前端提供API。

Django为前端提供的API接口包括（举例）：

1. Get, http://osvt.net:3000/tenants

介绍：Get方法，可以获得所有的“租户”，每个租户有一个ID和一个Name。

1. Get, http://osvt.net:3000/tenants/<TENANT\_ID>

介绍：Get方法，获得TENANT\_ID对应的metadata.json

1. Get, http://osvt.net:3000/tenants/<TENANT\_ID>/policies/<POLICY\_NAME>

介绍：Get方法，获得TENANT\_ID的某个Policy

1. Post, http://osvt.net:3000/tenants/<TENANT\_ID>

介绍：Post方法，为一个TENANT设置metadata.json

1. Post, http://osvt.net:3000/tenants/<TENANT\_ID>/policies/<POLICY\_NAME>

介绍：Post方法，为一个TENANT设置某个Policy

1. Get, http://osvt.net:3000/tenants/<TENANT\_ID>/users

介绍：Get方法，获得某个TENANT的所有User

1. Get, http://osvt.net:3000/tenants/<TENANT\_ID>/users/<USER\_NAME>/commands/<COMMAND>

介绍：Get方法，以属于<TENANT\_ID>的<USER\_NAME>的身份执行命令<COMMAND>并获得执行结果。

## 3.2 前端

在前端上，本系统主要采用了HTML5、JQuery、JQuery-Confirm来实现与API的通信与内容的生成，采用了Bootstrap来作为CSS框架，并且采用了zTree这个JS库来实现Metadata的树状策略结构的展示。

为了提升安全性，本系统支持采用登录的方式来以某个租户的身份登录。

**4. 工具的展示效果和功能测试结果**

## 4.1租户登录/用户列表页

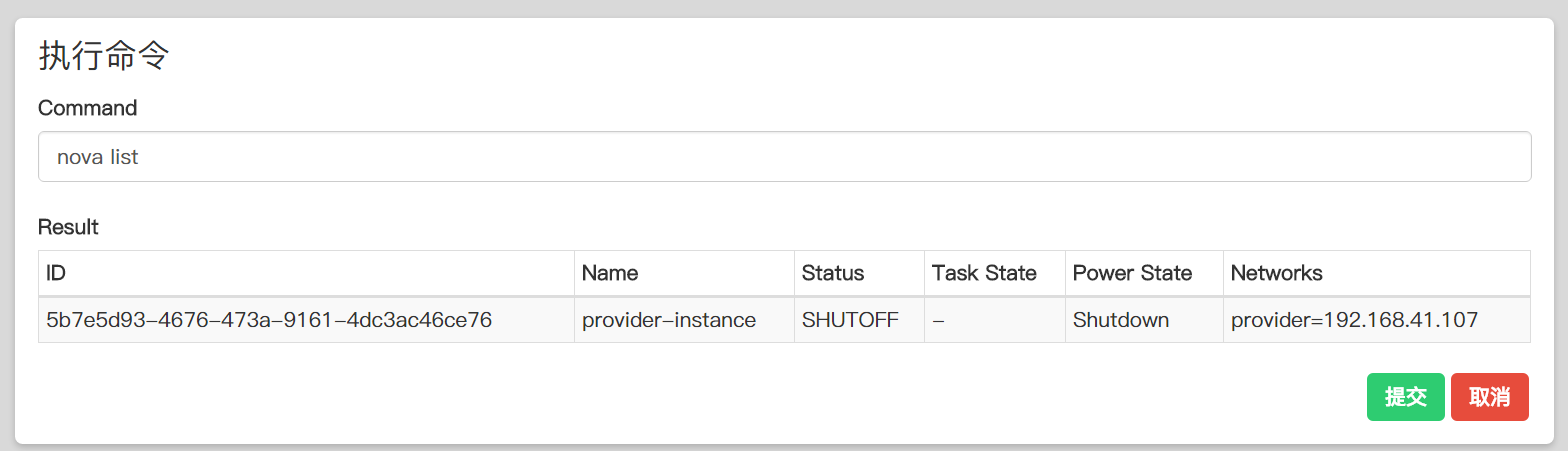
在本模块中，用户首先面对的是租户登录的页面，用户可以通过用户名密码登录，如果错误会有错误提示，成功的话，用户可以选择1）进入用户列表，或2）进入Metadata策略树编辑页面。



比如此时登录到租户“企业A”中，可以查看此租户下的用户列表。

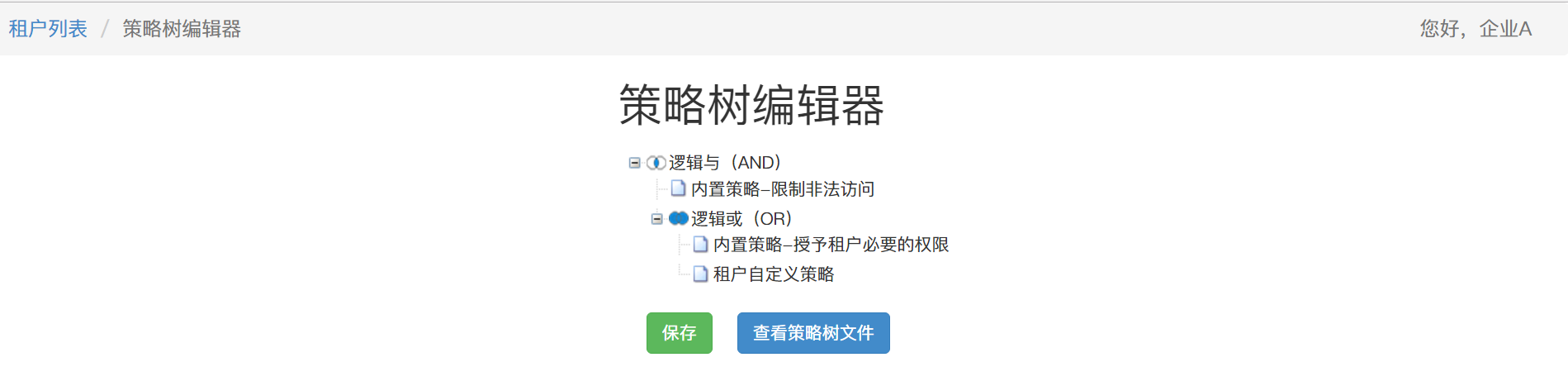


点击每个用户，可以以此用户身份执行命令并获得回显，比如下图中执行的是“Nova list”命令。



## 4.2 Metadata策略树页面

1. **用正则表达式从自身url中获取user\_id，用get()获取到Metadata.json内容**
2. **把自带的metadata.json格式递归转为zTree控件要求的格式，**
   1. 先把content(如policy-a,policy-b)按逗号分隔，拿到子结点的name。
   2. 然后设置children数组，把这些子节点放进去，把原来的子节点删掉；
   3. 然后递归子节点，重复2、3步，如果遇到type:default的叶节点就返回
   4. 最后把根节点current-policy属性删掉
3. **最后保存的时候类似，**
   1. 只不过是从zTree里获取对象，此时会增加很多控件自己增加的属性，首先需要删掉这些，然后会反过来把结点的children数组里的对象归回到节点里作为不同的属性。
   2. 额外的是，需要判断每个节点的children数组是否存在只有一个元素的情况，是的话就返回false，提醒用户修改。
   3. 因为需要转换格式，是先深复制再转换，不修改原来的树。
   4. 然后post请求，然后后退
4. **设置默认全部展开，并且屏蔽折叠事件**



1. **为每个节点增加 + 按钮，**
   1. 为新节点赋予 “policy”+6位随机数的name和content，以防重复
   2. 默认built-in为false，version为v1.0
   3. 如果它是父节点的第一个子，首先修改父节点的type为op-and，然后把父节点content的\*.json赋给这个子节点，然后把父节点的content清空并换成子节点的name
   4. 如果不是父节点的第一个子，就把子节点的name加到父节点的content末尾
2. **删除节点时，**
3. 若自己是最后的子节点，父节点的type=default，父节点content=被删节点的content
4. 若自己不是最后的子节点，就在父节点的content里删除自己节点的name
5. **双击任意节点会弹出修改的对话框，**
   1. 把节点的值填充到各个控件中
   2. 根据type(op-and / default)来确定 选择框 的选项
   3. 如果type是op-and或者op-or就禁止修改content防止造成不必要的麻烦（可以通过删除、修改、添加子节点实现修改）
   4. 如果type是default才显示“编辑此策略的按钮”，并禁止修改type（因为只有一个选项）
   5. 修改“保存”按钮的调用函参（传节点id进去），来方便接下来的函数的处理。



父节点范例

1. **在上面的对话框中，当点击“保存”时**
   1. 首先会对名字、内容判空
   2. 检查content是否有逗号，以及一些url特定字符 比如& % = #等等
   3. 检查content是否以.json结尾
   4. 然后会从输入框里读信息更新节点的数据，并关闭对话框
   5. 另外如果name修改了，如果自己不是根节点，则会更新父节点content中的对应的name



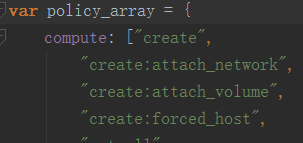
叶节点范例

1. **在点击“编辑此策略”时，会从content里取\*.json的\*部分，然后用location.href进行跳转**

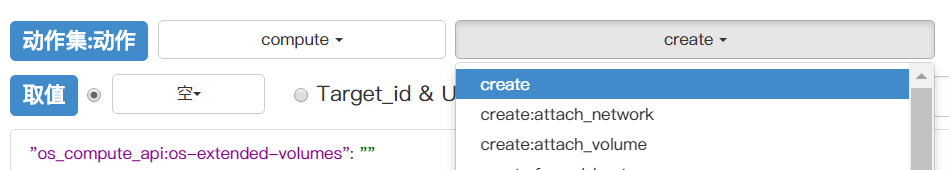
## 4.3 Policy编辑器



1. **从JS读取二维数组，第一层是name，第二层是action。**



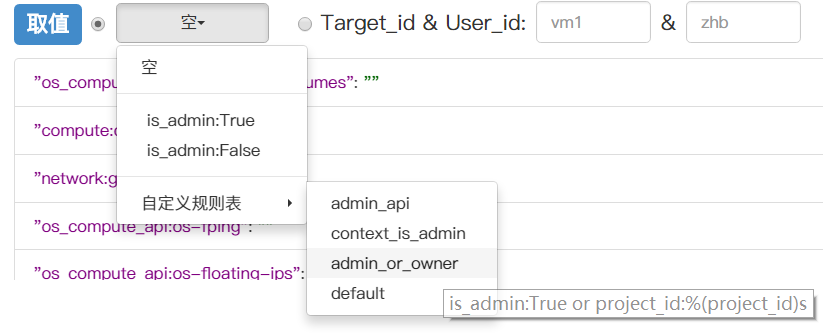
1. **获取\*\*\*.json**
   1. 然后从自身Url中获取user\_id、Policy\_name；
   2. 再调用GET接口，获取用户要编辑的\*\*\*.json的内容
2. **建立策略数组、自定义规则数组**
   1. 根据key中是否有冒号来分成 策略 和 自定义规则 两部分，比如"compute:create": ""就是策略，而"context\_is\_admin": "role:admin"就是规则
3. **为了美观，引入了第三方库，dropdown-enhancement**
4. **刷新Name和Action**
   1. Name菜单加载后，Action的下拉菜单也会清空并添加对应的Action，然后会默认选择第一项
   2. Name菜单改变后，同上



1. **添加策略时，**
   1. 添加是先在数组中增加，然后特意添加到列表项开头（默认是末尾）
   2. 添加的时候，如果有同名的策略/规则，会问是否要覆盖，覆盖的话就刷新列表
2. **删除列表项时**
   1. 加入了删除按钮方便删除，在删除的时候，先用正则表达式取出Key删除数组中的元素，再删除列表项。
   2. 鼠标指向删除按钮会变成手型
3. **添加自定义规则时，**
   1. 当用户点击自定义的文本框时，会同时选中旁边的单选框（单选框和输入框不是一体）



* 1. 添加的时候，如果有同名的策略/规则，会问是否要覆盖
  2. 然后对新增的规则，刷新“取值”菜单，再刷新自定义规则列表。此处“自定义Rule”中的每一项，用鼠标指上去都会显示具体的值。



* 1. 规则名禁止带：冒号，也会判空

1. **列表其他**
   1. 内置的列表没有滚动条，加入了限高和滚动条。
   2. 设置了鼠标指上去，列表项会变灰
   3. 给列表设置文字的同时，对Key和Value设置了不同的颜色，类似语法高亮
2. **其他**
   1. “取值”下拉菜单当点击的时候，不仅改变按钮的值，也在后面加上下三角
   2. 最后保存的时候，合并两个列表的数据，然后Post给服务器。

