官方源码剖析与 API 详解

本小节我们将对 Scrapyd 原有的 API 做个详细的剖析，只有在了解它原有设计思想与代码风格后，我们才能够照猫画虎，设计出风格相似的代码模块，为之后我们自定义 API 和编写权限验证功能打下基础。

> API 相关代码在 `webservice.py` 文件中。

根据前面的小节，我们知道 Scrapyd 的视图分为 HTML 和 JSON 两种。我们所看到的数据呈现都是由视图类处理的，比如:

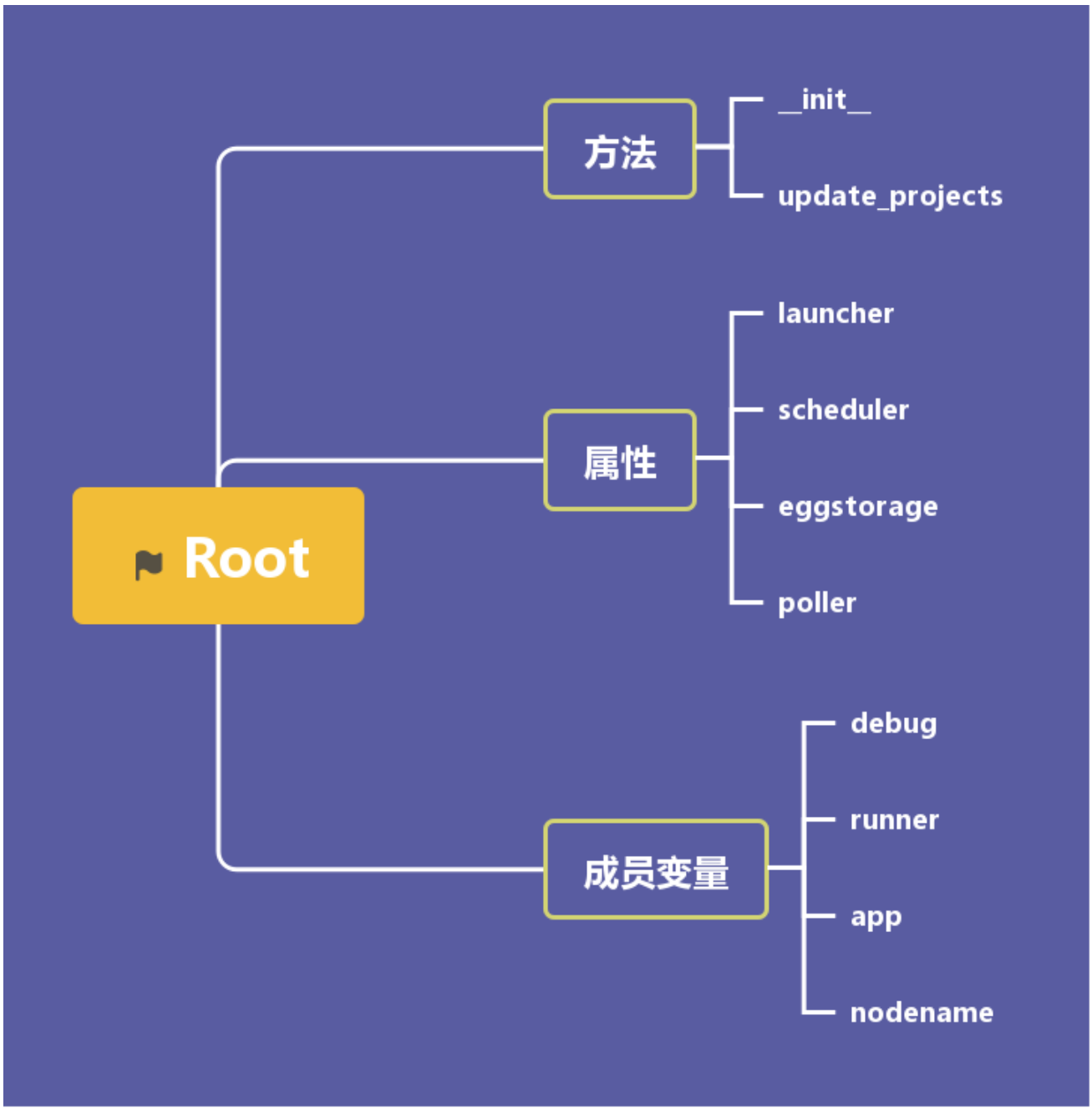
```
* 当我们访问根目录的时候，对应的的是 Home 这个类;
* 而访问 Jobs 的时候，对应的的是 Jobs 这个类;
* 而爬虫的 API 则是，如启动爬虫的 Schedule 类和查看爬虫列表的 ListSpiders 类。

```

HTML 视图

HTML 视图类都写在 Scrapyd 目录下的`website.py`文件中，里面有三个类：`Root`、`Home`、`Jobs`。

Root 类

Root 完成了`Web`路由设置，Scrapyd 一些基础配置的读取设置，日志目录和`Item`目录与路由配置、项目信息更新等任务，是 Scrapyd 的重要组成部分，其代码结构如下图所示:



Web 路由设置

在 Root 类的`__init__`方法中 Home 类以及 Jobs 类的路由是通过 Twisted 的 putChild 进行配置的：

```
self.putChild(b'jobs', Jobs(self, local_items))
self.putChild(b'', Home(self, local_items))

```

日志与 Item 目录配置

同样在`__init__`方法中，先读取日志与 Item 的目录：

```
logsdir = config.get('logs_dir')
itemsdir = config.get('items_dir')

```

然后为它们设置路由：

```
self.putChild(b'items', static.File(itemsdir, 'text/plain'))
self.putChild(b'logs', static.File(logsdir.encode('ascii', 'ignore'), 'text/plain'))

```

项目信息更新

当项目变动，比如增删 projects，就会通过 `update_projects()` 方法进行更新：

```
def update_projects(self):
    self.poller.update_projects()
    self.scheduler.update_projects()
@property
def poller(self):
    return self.app.getComponent(IPoller)

@property
def scheduler(self):
    return self.app.getComponent(ISpiderScheduler)


```

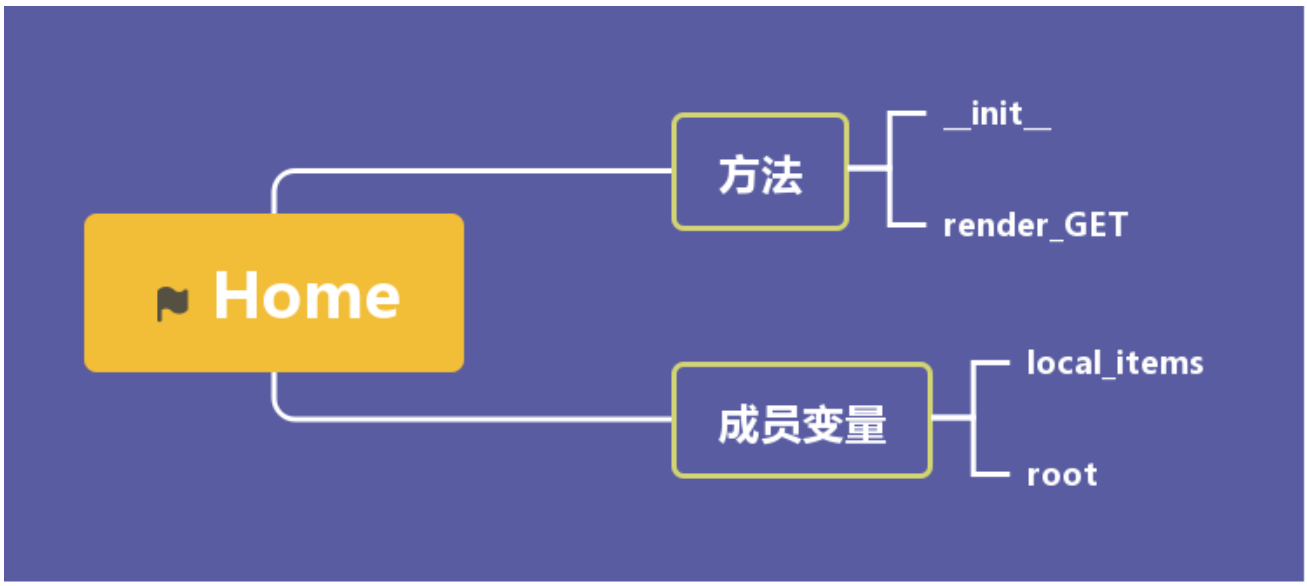
Scrapyd 的其他一些基础配置等

比如读取配置文件并且将其赋值给变量：

```
self.debug = config.getboolean('debug', False)
self.runner = config.get('runner')
services = config.items('services', ())

```

Home 类

Home 负责 Scrapyd 首页的呈现，当我们访问 `http://localhost:6800` 时看到的界面，就是访问 Home 类，它完成了页面 HTML 布局以及当前已有项目 `projects` 的名称列表展示。



`__init__` 方法

Home 类继承自`resource.Resource`，并且重写了`__init__`方法，以定义一个 Web 可访问资源：

```
def __init__(self, root, local_items):
    resource.Resource.__init__(self)
    self.root = root
    self.local_items = local_items

```

对于页面呈现的建议

Rource 中 render 对于页面呈现的建议：

```
"""
I delegate to methods of self with the form 'render_METHOD'
where METHOD is the HTTP that was used to make the
request. Examples: render_GET, render_HEAD, render_POST, and
so on. Generally you should implement those methods instead of
overriding this one.
"""

```

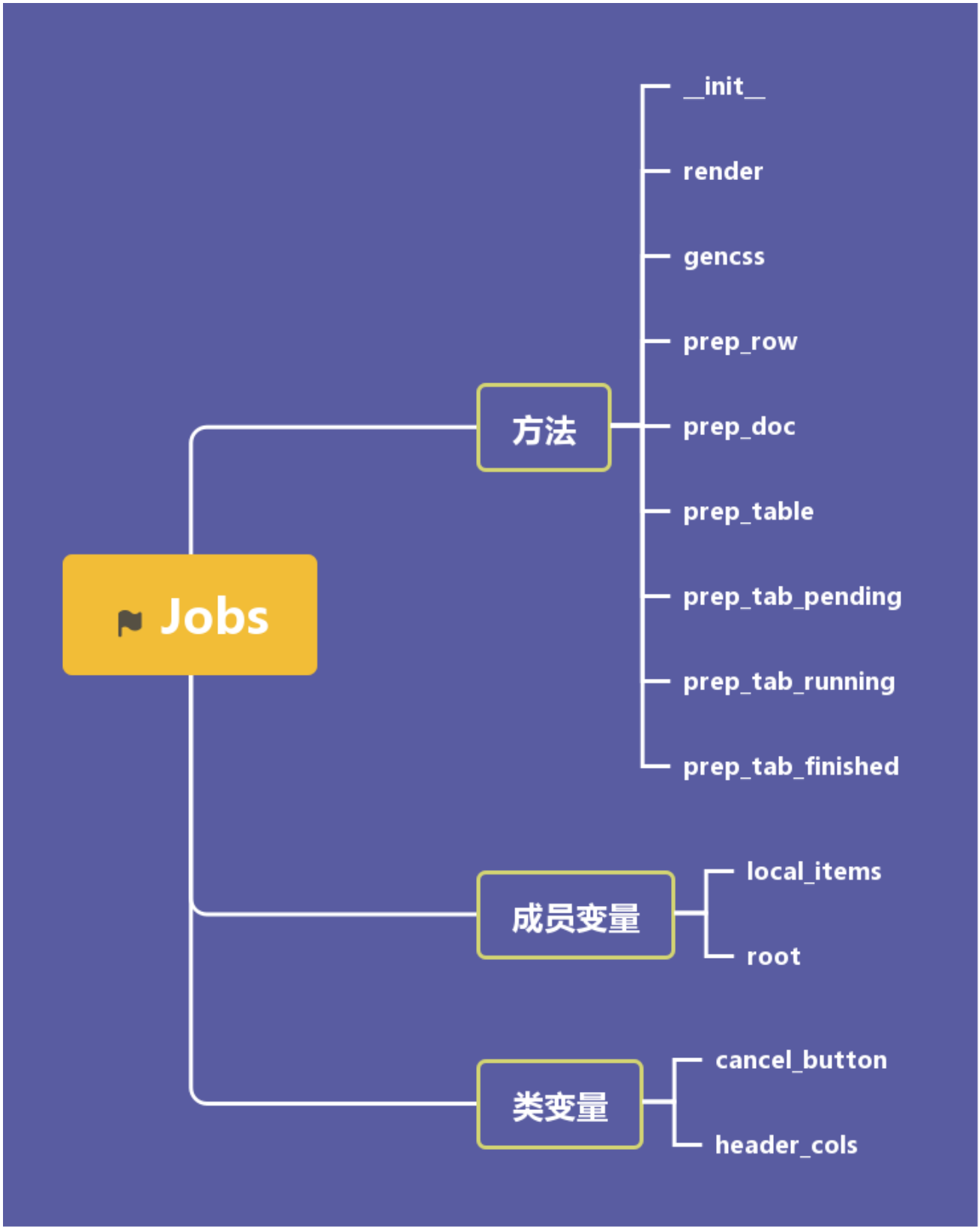
HTML 布局的实现以及已有项目列表展示

使用 render 建议的方式，用`render_GET`来完成页面的呈现：

```
def render_GET(self, txrequest):
   vars = {'projects': ', '.join(self.root.scheduler.list_projects())}
   s = "…… ……"
   return s.encode('utf-8')

```

Jobs 类

Jobs 负责 Scrapyd 首页爬虫运行状态展示、日志记录以及取消爬虫运行，当我们访问 `http://localhost:6800/jobs/` 时看到的界面，就是 Jobs 类。同样，它也是继承了 `resource.Resource`。



Cancel 与 Header

它设定取消按钮以及爬虫运行信息的列名：

```
cancel_button = """
<form method="post" action="/cancel.json">
   ……
      ……
""".format

header_cols = [
    'Project', 'Spider', 'Job', 'PID', 'Start',
    'Runtime', 'Finish', 'Log', 'Items', 'Cancel',
]

```

设定 CSS 样式

为爬虫日志表格设定 CSS 样式：

```
def gen_css(self):
    css = [
        '#jobs>thead td {text-align: center; font-weight: bold}',
        '#jobs>tbody>tr:first-child {background-color: #eee}']
    ……
    return '\n'.join(css)

```

生成表头

生成爬虫日志表格的表头数据：

```
def prep_row(self, cells):
    if not isinstance(cells, dict):
        assert len(cells) == len(self.header_cols)
    else:
        cells = [cells.get(k) for k in self.header_cols]
    cells = ['<td>%s</td>' % ('' if c is None else c) for c in cells]
    return '<tr>%s</tr>' % ''.join(cells)

```

Pending 状态爬虫列表

```
def prep_tab_pending(self):
    return '\n'.join(
        ……
    )

```

正在运行的爬虫列表

```
def prep_tab_running(self):
    return '\n'.join(
        ……
        for p in self.root.launcher.processes.values()
    )

```

运行完毕的爬虫列表及日志记录

```
def prep_tab_finished(self):
    return '\n'.join(
        ……
        for p in self.root.launcher.finished
    )

```

生成爬虫运行信息表

调用以上功能，生成爬虫运行信息表格：

```
def prep_table(self):
    return (
            '<table id="jobs" border="1">'
            self.prep_row(self.header_cols)
            self.prep_tab_pending()
            self.prep_tab_running()
            self.prep_tab_finished()
            '…… ……'
            )

```

页面渲染

使用 render 方法，将生成的爬虫运行信息呈现到`localhost:6800/jobs/`页面：

```
def render(self, txrequest):
    doc = self.prep_doc()
    txrequest.setHeader('Content-Type', 'text/html; charset=utf-8')
    txrequest.setHeader('Content-Length', len(doc))
    return doc.encode('utf-8')

```

JSON 视图

JSON 视图类都写在 Scrapyd 目录下的`webservice.py`文件中，里面除了官方文档中提到的 API 外，还有它们父类`WsResource`。 这里我挑选 3个 API 对应的类进行讲解。

ListProjects 类

ListProjects 类的作用是返回当前 Scrapyd 服务器上的爬虫项目列表。

```
class ListProjects(WsResource):

    def render_GET(self, txrequest):
        projects = list(self.root.scheduler.list_projects())
        return {"node_name": self.root.nodename, "status": "ok", "projects": projects}


```

它使用 `render_GET` 方法，所以在浏览器可以输入 `http://localhost:6800/listprojects.json` 来访问并获得 JSON 格式的结果，比如：

```
{"status": "ok", "projects": ["AlibabaProject", "JDProject"]}

```

ListSpiders 类

ListSpiders 的作用是返回当前 Scrapyd 服务器上指定项目名的爬虫列表。

```
class ListSpiders(WsResource):

    def render_GET(self, txrequest):
        args = native_stringify_dict(copy(txrequest.args), keys_only=False)
        project = args['project'][0]
        version = args.get('_version', [''])[0]
        spiders = get_spider_list(project, runner=self.root.runner, version=version)
        return {"node_name": self.root.nodename, "status": "ok", "spiders": spiders}

```

它也是使用 `render_GET` 方法，但是它需要携带项目名称作为请求参数。如：`http://localhost:6800/listspiders.json?project=AlibabaProject` 返回的结果同样是 json 格式，通过返回的 json 数据，我们知道 AlibabaProject 项目中当前有哪些爬虫。

```
{"status": "ok", "spiders": ["taobao", "1688", "tmall"]}

```

DeleteProject 类

DeleteProject 的作用是用于删除 Scrapyd 上已有的爬虫项目。

```
class DeleteProject(WsResource):

        def render_POST(self, txrequest):
            args = native_stringify_dict(copy(txrequest.args), keys_only=False)
            project = args['project'][0]
            self._delete_version(project)
            UtilsCache.invalid_cache(project)
            return {"node_name": self.root.nodename, "status": "ok"}

        def _delete_version(self, project, version=None):
            self.root.eggstorage.delete(project, version)
            self.root.update_projects()

```

它使用的则是`render_POST`方法，所以请求的时候我们必须以 post 方式进行，并且它要求携带项目名称作为参数值，如:`http://localhost:6800/delproject.json -d project=AlibabaProject`在成功删除项目后，更新爬虫项目列表，

```
self.root.update_projects()

```

如果操作全部成功，我们得到的响应为:

```
{"status": "ok"}

```

视图父类与 Resource

JSON 视图和 HTML 视图的父类是不同的。在 HTML 视图部分，Home 和 Jobs 都继承自`resource.Resource`，而 JSON 视图部分则继承自`WsResource`。

WsResource

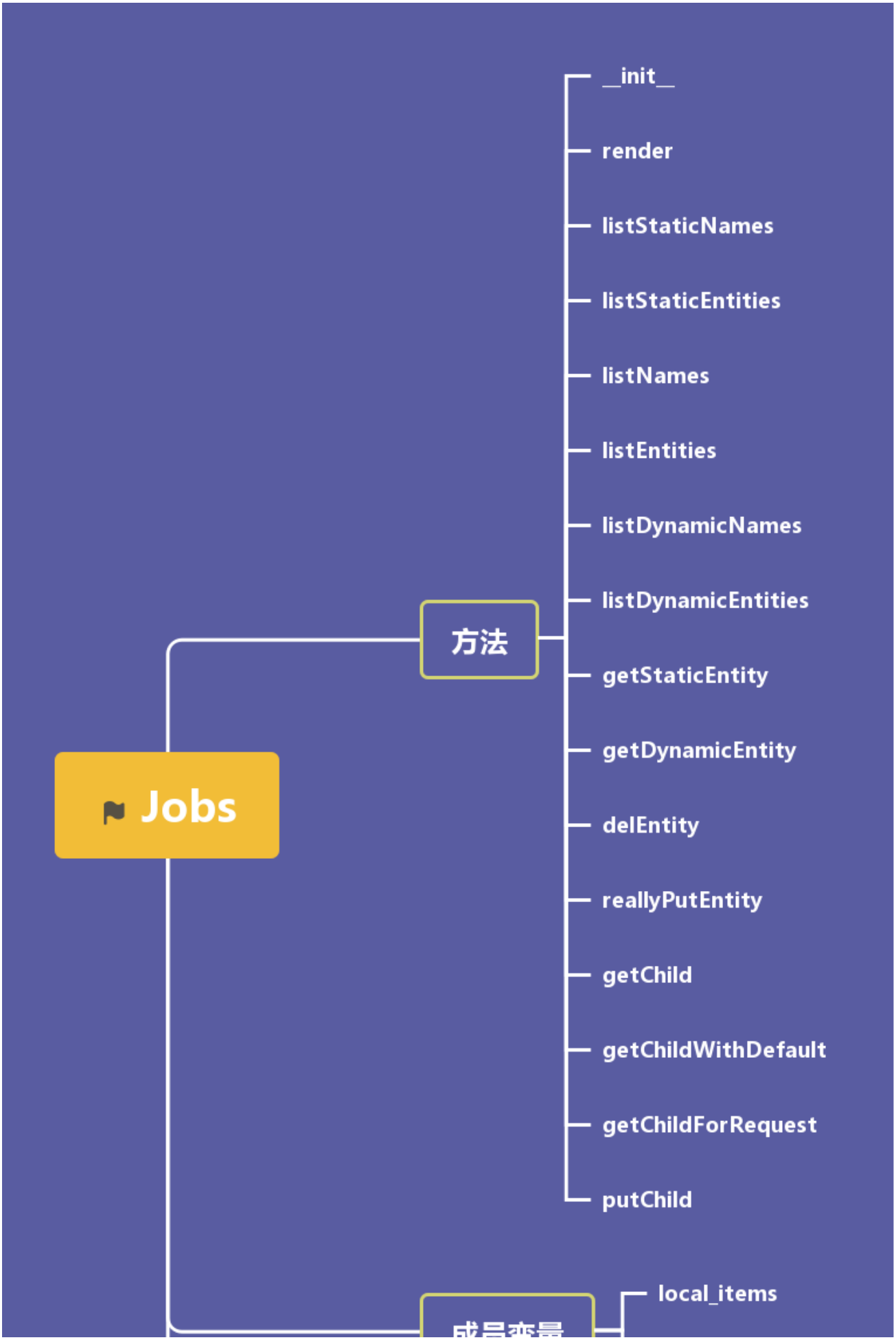
```
class WsResource(JsonResource):

        def __init__(self, root):
            JsonResource.__init__(self)
            self.root = root

        def render(self, txrequest):
            try:
                return JsonResource.render(self, txrequest).encode('utf-8')
            except Exception as e:
                if self.root.debug:
                    return traceback.format_exc().encode('utf-8')
                log.err()
                r = {"node_name": self.root.nodename, "status": "error", "message": str(e)}

                return self.render_object(r, txrequest).encode('utf-8')
```

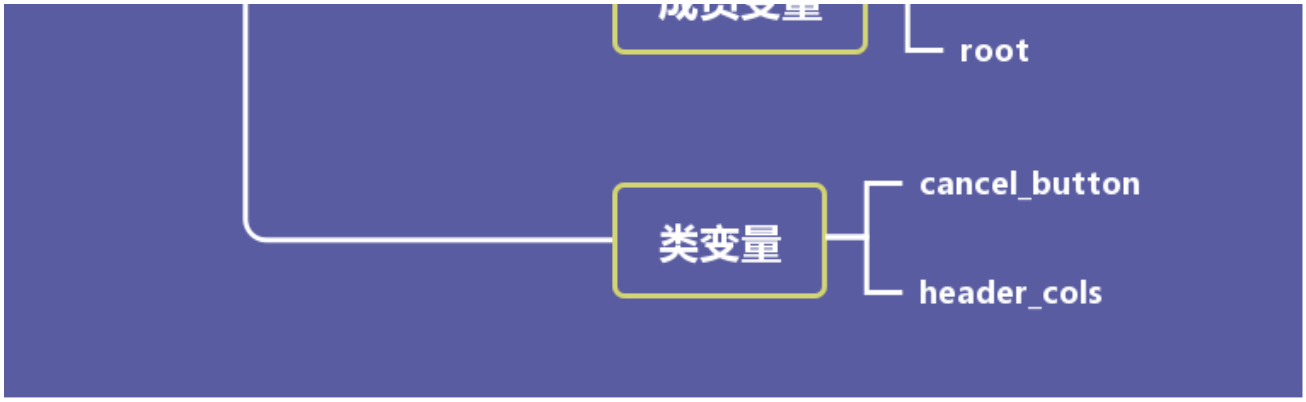
```
16
```

它定义了 render 方法，默认返回的是 json 格式，当返回 json 格式出错时返回报错信息。`JsonResource中为json格式定义了一些头信息`，并且将 return 的数据转为 json 格式数据。

resource.Resource

`Resource`中有很多方法，它的主要功能是定义一个可访问的 Web 资源，并且提供 HTTP 请求的标准、URL 路由设定标准，如`putChild`等，下图为`Resource`类的结构图：


Jobs
方法
__init__
render
listStaticNames
listStaticEntities
listNames
listEntities
listDynamicNames
listDynamicEntities
getStaticEntity
getDynamicEntity
delEntity
reallyPutEntity
getChild
getChildWithDefault
getChildForRequest
putChild
local_items



小结

本小节通过阅读 Scrapyd 中负责 HTML 视图的`Root`、`Home`、`Jobs`类和阅读`API 中的几个类`，知道了 Scrapyd 的`视图及 API 构成`，并通过阅读其`父类`加深了对 Scrapyd 视图的理解。