Scrapyd 视图类

在 Scrapyd 源码目录结构注解中，`website.py` 代表 Web 视图，`webservice.py` 代表 JSON 视图。

视图是使用者与 Scrapyd 交互的窗口，本小节我们通过阅读源码来理解 Scrapyd 的视图设计。

Web 视图

Scrapyd 提供 Web 视图是为了便于使用者监视爬虫运行状态。`website.py` 中有 3个类，它们分别是：

```
1.Root - 根据配置文件初始化项目并设置路由映射关系。
2.Home - Scrapyd 的 Web 首页功能实现及页面渲染。
3.Jobs - Scrapyd 的 jobs 页面功能实现及页面渲染。

```

Root 类

Root 类的作用大致为根据配置文件对项目进行初始化并设置路由映射关系，其代码如下：

```
class Root(resource.Resource):

    def __init__(self, config, app):
        resource.Resource.__init__(self)
        self.debug = config.getboolean('debug', False)
        self.runner = config.get('runner')
        logsdir = config.get('logs_dir')
        itemsdir = config.get('items_dir')
        local_items = itemsdir and (urlparse(itemsdir).scheme.lower() in ['', 'file'])
        self.app = app
        self.nodename = config.get('node_name', socket.gethostname())
        self.putChild(b'', Home(self, local_items))
        if logsdir:
            self.putChild(b'logs', static.File(logsdir.encode('ascii', 'ignore'),
'text/plain'))
        if local_items:
            self.putChild(b'items', static.File(itemsdir, 'text/plain'))
        self.putChild(b'jobs', Jobs(self, local_items))
        services = config.items('services', ())
        for servName, servClsName in services:
          servCls = load_object(servClsName)
          self.putChild(servName.encode('utf-8'), servCls(self))
        self.update_projects()

    def update_projects(self):
        self.poller.update_projects()
        self.scheduler.update_projects()

    @property
```

```
    def launcher(self):
        app = IServiceCollection(self.app, self.app)
        return app.getServiceNamed('launcher')

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

    @property
    def eggstorage(self):
        return self.app.getComponent(IEggStorage)

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

```

代码释义： Root 类继承自`resource.Resource`， 在`__init__`方法中通过`config.get`读取配置文件中相关配置并赋值给变量，通过`putChild`语法将资源与 URL 进行映射（路由），在`update_projects`方法中更新项目状态。

Home 类

Home 类的作用大致为`Scrapyd`的`Web`首页功能实现及页面渲染，其代码大体如下：

```
class Home(resource.Resource):

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

    def render_GET(self, txrequest):
        vars = {
            'projects': ', '.join(self.root.scheduler.list_projects())
        }
        s = """
        <html>
        <head><title>Scrapyd</title></head>
        <body>
        <h1>Scrapyd</h1>
        <p>Available projects: <b>%(projects)s</b></p>
        <ul>
        <li><a href="/jobs">Jobs</a></li>
        """ % vars
                if self.local_items:
                    s += '<li><a href="/items/">Items</a></li>'
                s += """
        ...html ...
        """ % vars
        return s.encode('utf-8')

```

代码释义：Home 类继承自 `resource.Resource`。 在 `render_GET` 方法中，获取项目列表 `self.root.scheduler.list_projects()`,然后构造 Web 界面的 HTML 代码。最终将 utf-8 编码的 HTML 代码进行渲染。

Jobs 类

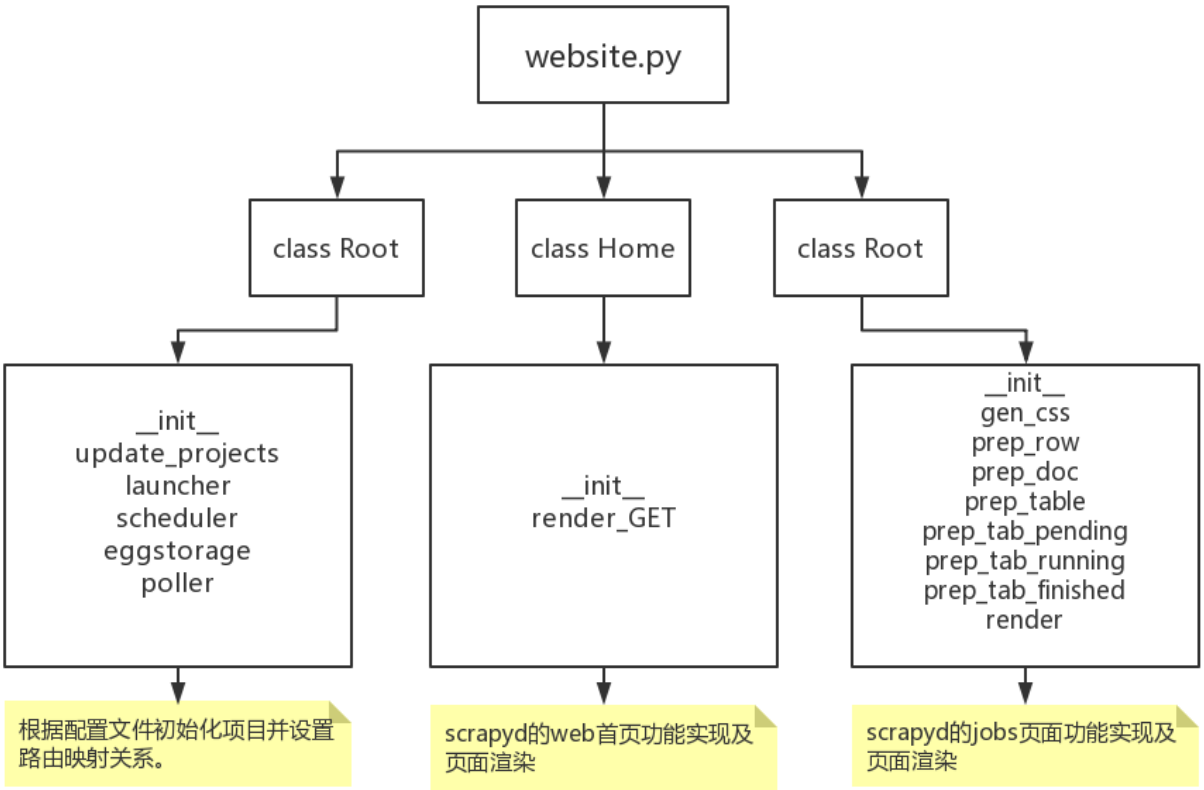
Jobs 类的作用大致为 `Scrapyd` 的 `/jobs` 页面的功能实现及页面渲染，其代码大体如下：

```
class Jobs(resource.Resource):

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

    cancel_button = """
    form...
    """.format

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

    def gen_css(self):
        css...
        return '\n'.join(css)

    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)

    def prep_doc(self):
        return (
            html...
        )

    def prep_table(self):
        return (
            html-table...
        )

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

```
            self.prep_row(dict(
                Project=project, Spider=m['name'], Job=m['_job'],
                Cancel=self.cancel_button(project=project, jobid=m['_job'])
            ))
            for project, queue in self.root.poller.queues.items()
            for m in queue.list()
        )

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

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

    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')

```

可以看到，Jobs 继承的也是 `resource.Resource`。

Jobs 中的方法是层级调用的关系，从 render 方法开始，调用`self.prep_doc()`以获取 HTML 文本，而`prep_doc`方法则通过调用`gen_css`和`prep_table`以获取 CSS 样式及不同状态的爬虫数与爬虫运行记录。所以我们可以绘制出`website.py`文件中各个类与方法的组织结构图：



resource.Resource

`resource.Resource` 是 Home 和 Jobs 的父类，它的代码是怎样的?

在 PyCharm 中，通过 `Ctrl/Commond+鼠标左键` 可以跟进代码。我们跟进 Resource，代码如下:

```
@implementer(IResource)
class Resource:
    entityType = IResource

    server = None

    def __init__(self):
        """
        Initialize.
        """
        self.children = {}

    isLeaf = 0

    ### Abstract Collection Interface

    def listStaticNames(self):
        return list(self.children.keys())

    def listStaticEntities(self):
        return list(self.children.items())
```

```
22
23      def listNames(self):
24          return list(self.listStaticNames()) + self.listDynamicNames()
25
26      def listEntities(self):
27          return list(self.listStaticEntities()) + self.listDynamicEntities()
28
29      def listDynamicNames(self):
30          return []
31
32      def listDynamicEntities(self, request=None):
33          return []
34
35      def getStaticEntity(self, name):
36          return self.children.get(name)
37
38      def getDynamicEntity(self, name, request):
39          if name not in self.children:
40              return self.getChild(name, request)
41          else:
42              return None
43
44      def delEntity(self, name):
45          del self.children[name]
46
47      def reallyPutEntity(self, name, entity):
48          self.children[name] = entity
49
50      # Concrete HTTP interface
51
52      def getChild(self, path, request):
53
54          return NoResource("No such child resource.")
55
56
57      def getChildWithDefault(self, path, request):
58
59          if path in self.children:
60              return self.children[path]
61          return self.getChild(path, request)
62
63
64      def getChildForRequest(self, request):
65          warnings.warn("Please use module level getChildForRequest.", DeprecationWarning, 2)
66          return getChildForRequest(self, request)
67
68
69      def putChild(self, path, child):
70
71          self.children[path] = child
72          child.server = self.server
73
74
```

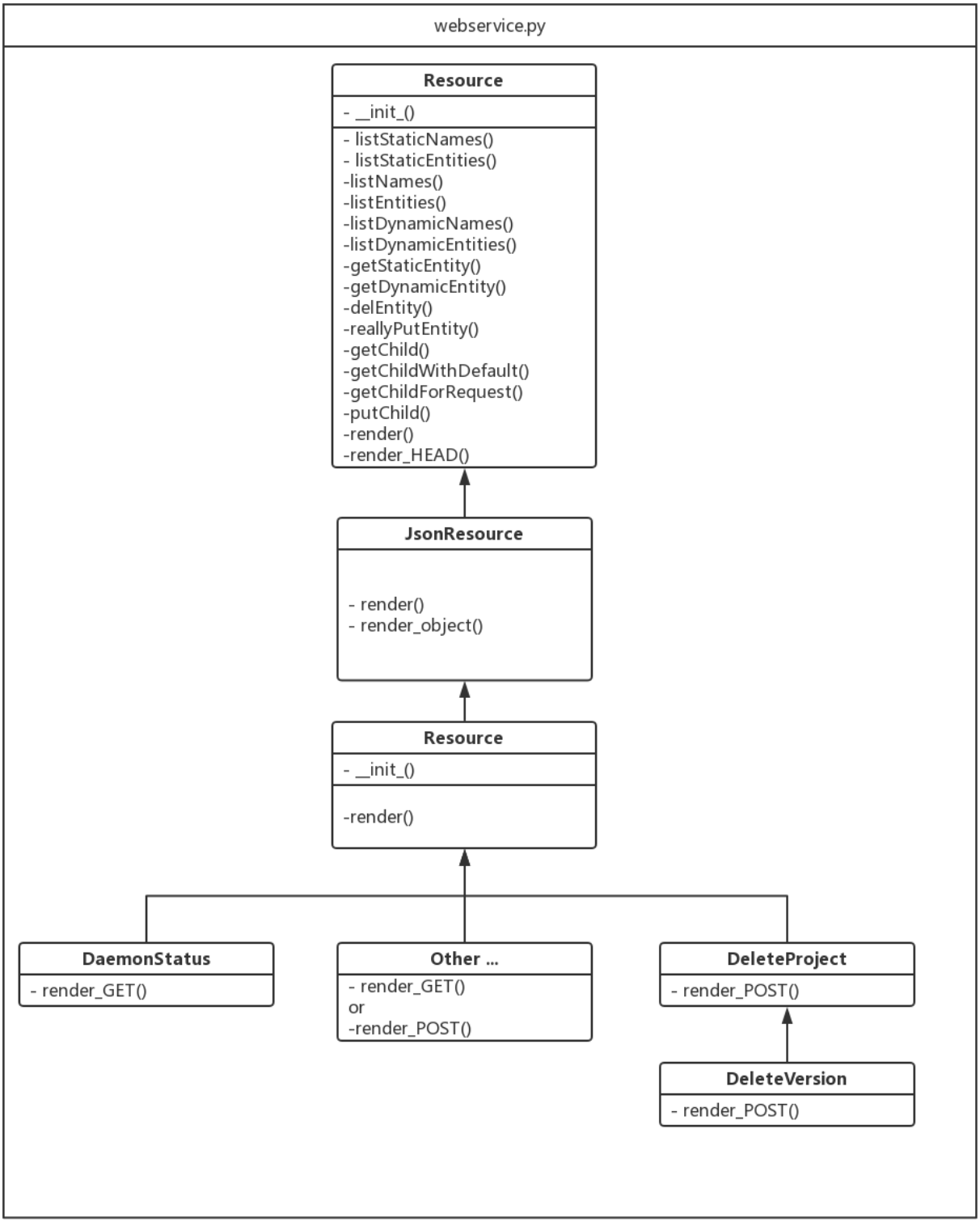
```
    def render(self, request):

        m = getattr(self, 'render_' + nativeString(request.method), None)
        if not m:
            try:
                allowedMethods = self.allowedMethods
            except AttributeError:
                allowedMethods = _computeAllowedMethods(self)
            raise UnsupportedMethod(allowedMethods)
        return m(request)


    def render_HEAD(self, request):

        return self.render_GET(request)


```

其中我们需要关注的方法有以下几个：

```
* putchild - twisted 语法，通过此方法设置路由。
* render - 负责页面渲染。
* render_HEAD - 将请求转发给 `render_GET`
```

JSON 视图

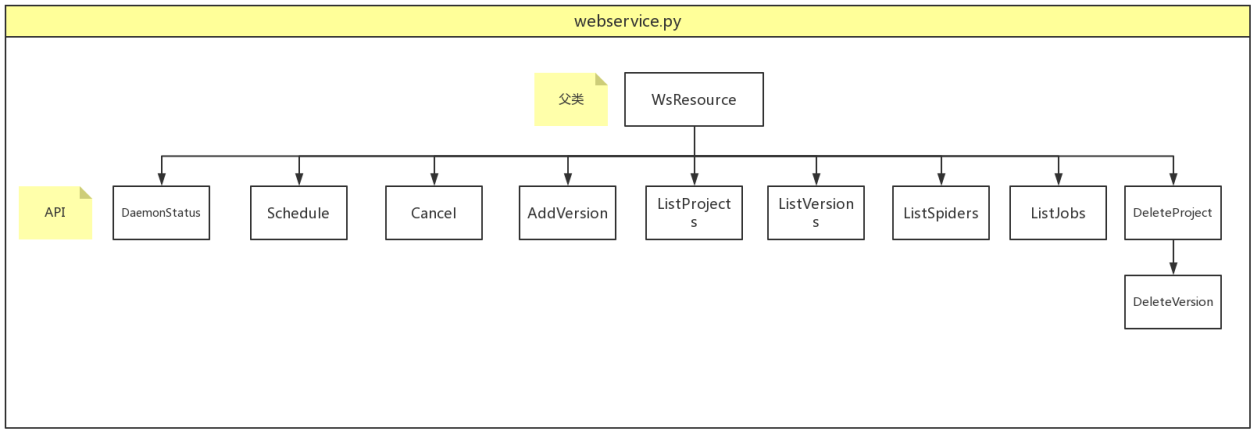
Scrapyd 提供了很多 API，通过 API 的返回信息能够知晓爬虫的运行状态或结果等信息，`webservice.py`中视图类与父类的继承关系 UML 类图如下：



以下是各个类的大致功能释义：

```
* WsResource - 继承自 JsonResource，是大多数 API 的父类，它默认返回 json 数据。
* DaemonStatus - 对应 API 中的daemonstatus.json，用以检查 Scrapyd 服务的爬虫状态及对应数量。
* Schedule - 对应 API 中的 schedule.json，用以启动指定的爬虫。
* Cancel - 对应 API 中的 cancel.json，用以取消指定的爬虫。
* AddVersion - 对应 API 中的 addversion.json，用以为项目添加版本，如果项目不存在则创建项目。
* ListProjects - 对应 API 中的 listprojects.json，用以查看当前已部署的项目名称。
* ListVersions - 对应 API 中的 listversions.json，用以查看指定项目的版本号。
* ListSpiders - 对应 API 中的 listspiders.json，用以查看指定项目的爬虫名称。
* ListJobs - 对应 API 中的 listjobs.json，用以查看指定项目下爬虫的运行状态。
* DeleteProject - 对应 API 中的 delversion.json，用以删除指定项目的指定版本，版本不存在则删除项目。
* DeleteVersion - 对应 API 中的 delproject.json，用以删除指定项目及其已上传的所有版本。

```

以及对应的**类组织结构图**：



可以看到，大多数 API 类继承的是`WsResource`，并且通过重写 render 方法将数据渲染。

在 PyCharm 中，通过`Ctrl/Commond+鼠标左键`可以跟进代码，我们跟进`WsResource`：

代码如下：

```
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')

```

它继承自 `JsonResource` 并且通过 JsonResource 的 `render` 返回 `json` 格式数据，如果报错则返回报错信息，而跟进 `JsonResource` 发现它其实也是继承自 `resource.Resource` 类。所以这里可以得出几个结论：

- Resource 是 Scrapyd 中视图部分最底层的类，且其 render 默认返回 bytes 类型数据。
- JsonResource 以及 WsResource 都是通过重写 render 以实现返回 JSON 类型数据。
- 或许可以通过自定义视图类，继承自 Resource，也通过重写 render 方法实现兼容 bytes 与 JSON 的视图类。