Python 爬虫 JS 处理

数据采集过程中请自觉遵守互联网界道德和法律规范

js 相关反爬虫方式

1、js 代码混淆加密

2、js 动态生成网页元素或者 css 样式

3、复杂的加密算法，参数+时间戳+sig 值

4、js 事件交互，隐藏 url

5、js 定时同步 cookie，每个界面一个 cookie

三种方法

1.js 代码用 python 实现，就是将 js 翻译为 python。

2.利用 selenium+phantomjs 模拟浏览器操作。

3.利用 pyexecjs 直接执行 js 文件

js 压缩、混淆和加密

压缩：删除 Javascript 代码中所有注释、跳格符号、换行符号及无用的空格，从而压缩 JS 文件大小，优化页面加载速度。

混淆：经过编码将变量和函数原命名改为毫无意义的命名（如 function(a,b,c,e,g)等），以防止他人窥视和窃取 Javascript 源代码，也有一定压缩效果。

加密：一般用 eval 方法加密，效果与混淆相似，也做到了压缩的效果。

美化：格式化代码，使之容易阅读

从定义中可以看出，压缩的主要目的是消除注释等无用字符，达到精简 js 代码，减小 js 文件大小的目的，这也是页面优化的一种方式；而混淆和加密的目的比较接近，都是为了防止他人直接查看源码，对代码（如重要的 api 等）起保护作用，但这也只是增加了阅读代码的代价，也就是所谓的防君子不防小人。但是当混淆和加密联合使用时，如先混淆在加密（或者先加密再混淆）时，破解时间就会增加。

演示

```
/* 这个是一个类 */

function xx(num, str) {
	//说明
	var a = num;
	this.aa = a;
	this.bb = function() {alert(str);}
	this.cc = function() {
		for (var i = 0; i < 10; i++) {
			document.title = i;
		}
	}
	this.yy = new yy();
	function xxf() {
		alert("xxf");
		if ((/\{\d+\}/).test("a\sdf{2}ab"))
			alert("{\\d} is match!");
	}
}
xx.prototype.dd = function(){
	alert("dd");
	a.yy.ll();
	var fnx = function(i) {
		this.ab = i;
```

```
            this.aa = function(){
                alert(this.ab);
            }
        }
        var f1 = new fnx(1);
        f1.aa();
}

function yy(){
        alert('yy');
}
yy.prototype.ll = function() {
        alert("yyll");
}

var a = new xx(100, "hello"), b = new xx(0, "ttyp");
eval("a.aa=20");
a.bb();
b.dd();
alert(a.aa);

var k = 9;
function kk() {
        var k = 0;
        alert(k);
}
kk();
alert(k);
//输入结果 alert:"yy"->"yy"->"hello"->"dd"->"yyll"->"12"->"20"->"0"->"9"
```

经过压缩后的代码

```
function xx(num,str){var a=num;this.aa=a;this.bb=function(){alert(str)};this.cc=function(){for(var i=0;i<10;i++){document.title=i}};this.yy=new yy();function xxf(){alert("xxf");if((/\{\d+\}/).test("a\sdf{2}ab"))alert("{\\d} is match!")}};xx.prototype.dd=function(){alert("dd");a.yy.ll();var fnx=function(i){this.ab=i;this.aa=function(){alert(this.ab)}};var f1=new fnx(1);f1.aa()};function yy(){alert('yy')};yy.prototype.ll=function(){alert("yyll")};var a=new xx(100,"hello"),b=new xx(0,"ttyp");eval("a.aa=20");a.bb();b.dd();alert(a.aa);var k=9;function kk(){var k=0;alert(k)};kk();alert(k);
```

压缩后与源码相比只是少了注释、空格、换行等。

经过混淆后的代码

function xx(d,e){var f=d;this.aa=f;this.bb=function(){alert(e)};this.cc=function(){for(var g=0;g<10;g++){document.title=g}};this.yy=new yy();function xxf(){alert("xxf");if((/\{\d+\}/).test("a\sdf{2}ab"))alert("{\\d} is match!")}};xx.prototype.dd=function(){alert("dd");a.yy.ll();var fnx=function(e){this.ab=e;this.aa=function(){alert(this.ab)}};var d=new fnx(1);d.aa()};function yy(){alert('yy')};yy.prototype.ll=function(){alert("yyll")};var a=new xx(100,"hello"),b=new xx(0,"ttyp");eval("a.aa=20");a.bb();b.dd();alert(a.aa);var c=9;function kk(){var d=0;alert(d)};kk();alert(c);

混淆后除了少了注释、空格和换行等，参数也被 a，b，c，d，e 等字符代替，提高了阅读的难度。

经过加密后的代码

eval(function(p,a,c,k,e,d){e=function(c){return(c<a?"":e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--)d[e(c)]=k[c]||e(c);k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1;};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p;}('3 e(o,l){5 a=o;6.8=a;6.p=3(){4(l)};6.B=3(){A(5 i=0;i<y;i++){x.z=i}};6.7=c 7();3 j(){4("j");t((/\\{\\d+\\}/).s("a\\u{2}g"))4("{\\\\d} w v!")}};e.r.f=3(){4("f");a.7.h();5 n=3(i){6.g=i;6.8=3(){4(6.g)}};5 m=c n(1);m.8()};37(){4(\'7\')};7.r.h=3(){4("D")};5 a=c e(F,"H"),b=c e(0,"G");E("a.8=C");a.p();b.f();4(a.8);5 k=9;3 q(){5 k=0;4(k)};q();4(k);',44,44,'|||function|alert|var|this|yy|aa||||new||xx|dd|ab|ll||xxf||str|f1|fnx|num|bb|kk|prototype|test|if|sdf|match|is|document|10|title|for|cc|20|yyll|eval|100|ttyp|hello'.split('|'),0,{}))

加密后除了有混淆的作用外，有些代码被加密了，进一步提高了阅读的难度。

经过混淆&加密后的代码

eval(function(p,a,c,k,e,d){e=function(c){return(c<a?"":e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--)d[e(c)]=k[c]||e(c);k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1;};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p;}('4 i(d,e){5 f=d;6.8=f;6.p=4(){3(e)};6.s=4(){t(5 g=0;g<y;g++){G.F=g}};6.7=h 7();4 l(){3("l");x((/\\{\\d+\\}/).v("a\\u{2}j"))3("{\\\\d} w

r!")}};i.o.k=4(){3("k");a.7.m();5 n=4(e){6.j=e;6.8=4(){3(6.j)}};5 d=h n(1);d.8()};4 7(){3(\'7\')};7.o.m=4(){3("E")};5 a=h i(A,"z"),b=h i(0,"B");D("a.8=C");a.p();b.k();3(a.8);5 c=9;4 q(){5 d=0;3(d)};q();3(c);',43,43,'|||alert|function|var|this|yy|aa|||||||||new|xx|ab|dd|xxf|ll|fnx|prototype|bb|kk|match|cc|for|sdf|test|is|if|10|hello|100|ttyp|20|eval|yyll|title|document'.split('|'),0,{}))

浏览器解析混淆和加密后的代码

其实变量名只要是 Unicode 字符就行了，对于 js 引擎来说都是一样的，只是人类觉得他们不同而已。

js 压缩混淆工具

http://tool.chinaz.com/js.aspx

JS 加密解密方法

JAVASCRIPT 代码是在浏览器中解释执行，要想绝对的保密是不可能的，开发者要做的就是尽可能的增大破解的难度

方法一

用 JAVASCRIPT 函数 escape()和 unescape()编码和解码字符串

escape()加密函数，把某些符号、汉字等变成乱码，以达到迷惑人的目的。用于转义不能用明文正确发送的任何字符。

unescape()解密函数起了还原源代码的作用。

对 0-255 以外的 unicode 值进行编码时输出%u****格式，即非 ASCII 字符的编码和解码!

函数不会对 ASCII 字母和数字进行编码，也不会对下面这些 ASCII 标点符号进行编码：

*@-_+./ 。其他所有的字符都会被编码

```
<script type="text/javascript">
var test1='Hello 爬虫!'
test1=escape(test1)
document.write (test1 + "<br />")
test1=unescape(test1)
document.write(test1 + "<br />")

var test1="alert('Hello 爬虫!')"
test1=escape(test1)
document.write (test1 + "<br />")
eval(test1)
</script>
```

方法二

encodeURIComponent(URIstring)

encodeURIComponent() 函数可把字符串作为 URI 组件进行编码。

该方法不会对 ASCII 字母和数字进行编码，也不会对这些 ASCII 标点符号进行编码,其他字符，都是由一个或多个十六进制的转义序列替换的。

decodeURIComponent(URIstring)

decodeURIComponent() 函数对 encodeURIComponent() 函数编码的 URI 进行解码。

返回值

URIstring 的副本，其中的十六进制转义序列将被它们表示的字符替换。

方法三

"\"后面可以跟八进制或十六进制的数字表示字符

如字符"a"则可以表示为："\141"或"\x61"（注意是小写字 符"x"）

至于双字节字符如汉字"黑"则仅能用十六进制表示为"\u9ED1"（注意是小写字符"u"），其

中字符"u"表示是双字节字符

八进制转义字符串如下:

```
<SCRIPT LANGUAGE="JavaScript">
eval("\141\154\145\162\164\50\42\u9ED1\u5BA2\u9632\u7EBF\42\51\73")
</SCRIPT>
```

十六进制转义字符串如下:

```
<SCRIPT LANGUAGE="JavaScript">
eval("\x61\x6C\x65\x72\x74\x28\x22\u9ED1\u5BA2\u9632\u7EBF\x22\x29\x3B")
</SCRIPT>
```

这次没有了解码函数，因为 JavaScript 执行时会自行转换

同样解码也是很简单如下:

```
<SCRIPT LANGUAGE="JavaScript">
alert("\x61\x6C\x65\x72\x74\x28\x22\u9ED1\u5BA2\u9632\u7EBF\x22\x29\x3B")
</SCRIPT>
```

就会弹出对话框告诉你解密后的结果！

方法四

自写解密函数法

加密代码如下：

```
<SCRIPT LANGUAGE="JavaScript">
function compile(code)
{
    var c=String.fromCharCode(code.charCodeAt(0)+code.length);
    for(var i=1;i<code.length;i++){
    c+=String.fromCharCode(code.charCodeAt(i)+code.charCodeAt(i-1));
    }
    alert(escape(c));
}
compile('alert("爬虫");')
</SCRIPT>
```

相应的加密后解密的代码如下：

```
<SCRIPT LANGUAGE="JavaScript">
function uncompile(code)
{
    code=unescape(code);
    var c=String.fromCharCode(code.charCodeAt(0)-code.length);
    for(var i=1;i<code.length;i++){
    c+=String.fromCharCode(code.charCodeAt(i)-c.charCodeAt(i-1));
    }
    return c;
}
eval(uncompile("m%CD%D1%D7%E6%9CJ%u724E%uF897%u868DKd"));
</SCRIPT>
```

eval() 函数

eval() 函数可计算某个字符串，并执行其中的的 JavaScript 代码。

css3 伪元素

伪元素采用双冒号写法。

: :before,::after

::before 和::after 下特有的 content，用于在 css 渲染中向元素逻辑上的头部或尾部添加内容。**这些添加不会出现在 DOM 中，不会改变文档内容，不可复制**，仅仅在 css 渲染层加入。所以不要用:before 或:after 展示有实际意义的内容，尽量使用它们显示修饰性内容

举例：网站有些联系电话，希望在它们前加一个 icon☎，就可以使用:before 伪元素，如下：

```
<!DOCTYPE html>
<meta charset="utf-8" />
<style type="text/css">
    .phoneNumber::before {
    content:'\260E';
    font-size: 15px;
}
```

```
</style>
<p class="phoneNumber">12345645654</p>
```

execjs 模块

在网页数据提取的日常中，经常有一些有用的信息以 json 的格式存放在网页的源代码中，

这时候要规则的提取的这些数据，就需要一个能够解析 js 的包了

execjs 安装

使用 pip 安装：

```
pip install PyExecJS
```

使用 easy_install 安装：

```
easy_install PyExecJS
```

使用

模块导入

import execjs

简单使用

print(execjs.eval("new Date"))

结果：2018-09-13T03:05:20Z

调用函数

```
ctx = execjs.compile("""
            function add(x, y) {
                    return x + y;
                }
""")
result = ctx.call("add", 1, 2)
print(result)
```

结果： 3

JS 的执行环境查看

```
print(execjs.get().name)
```

结果： JScript

在 windows 上不需要其他的依赖便可运行 execjs， 也可以调用其他的 JS 环境

源码中给出了可执行 execjs 的环境：

```
PyV8                 = "PyV8"
Node                 = "Node"
JavaScriptCore = "JavaScriptCore"
SpiderMonkey     = "SpiderMonkey"
JScript              = "JScript"
PhantomJS          = "PhantomJS"
SlimerJS           = "SlimerJS"
Nashorn            = "Nashorn"
```

测试

```
print(execjs.eval("Date.now()"))
```

出错信息：

execjs._exceptions.ProgramError: TypeError: 对象不支持此属性或方法

分析

我的电脑上默认的 JS 执行环境是 Jscript ， 不支持 Date 对象

环境切换

通过 os.environ

```
import os
os.environ["EXECJS_RUNTIME"] = " PhantomJS"
print(execjs.get().name)
print(execjs.eval("Date.now()"))
```

结果:

```
PhantomJS
1536808839291
```

扩展

在一些数据的抽取中用到了模拟浏览器，通常我们会用 selenium 或者其他的 webkit 包，但是一般的模拟包只是返回了渲染后的页面，有的时候仅仅是返回动态渲染的页面是不够的，还需要能够执行 js 并控制 js 与 dom 交互，有兴趣的同学可以看一下 PyV8 和 w3c 包

PyExecJS 的缺点之一就是性能。PyExecJS 通过文本传递 JavaScript 运行时，并且速度很慢。另一个缺点是它不完全支持运行时特定的功能。

总注：使用 execjs 的难点并不是在 execjs 这个库， 而是解析 JS 的过程， 因为没有浏览器的环境， 没有加密源码的依赖。从成千上万行的 JS 中择出想要的内容，可能是一段孤零零的 JS 函数，也可能是从几个 JS 文件去找出各自找出一段 JS 代码， 并可以通过 execjs 顺利执行， 这并非易事。 需要慢慢积累经验。 一旦掌握， 便可以提高爬虫的效率， 以及代码的健壮性， 节省资源!

Chrome 开发者工具面板

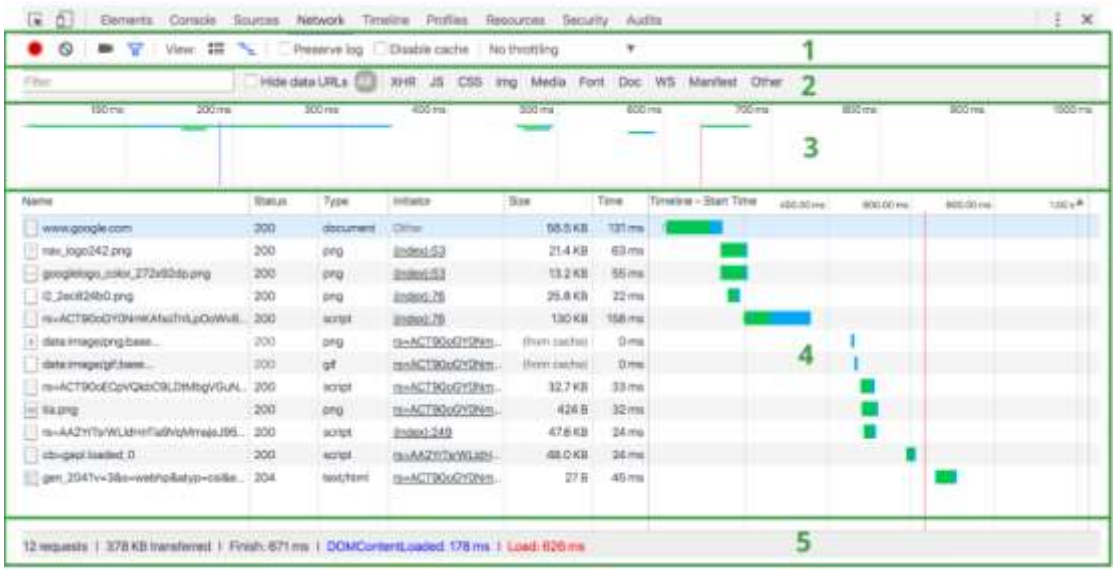
- **Elements**:查找网页源代码 HTML 中的任一元素,手动修改任一元素的属性和样式且能实时在浏览器里面得到反馈。
- **Console**:记录开发者开发过程中的日志信息,且可以作为与 JS 进行交互的命令行 Shell。
- **Sources**:断点调试 JS。
- **Network**:从发起网页页面请求 Request 后分析 HTTP 请求后得到的各个请求资源信息（包括状态、资源类型、大小、所用时间等），可以根据这个进行网络性能优化。
- **Timeline**:记录并分析在网站的生命周期内所发生的各类事件，以此可以提高网页的运行时间的性能。
- **Profiles**:如果你需要 Timeline 所能提供的更多信息时，可以尝试一下 *Profiles*,比如记录 JS CPU 执行时间细节、显示 JS 对象和相关的 DOM 节点的内存消耗、记录内存的分配细节。
- **Application**:记录网站加载的所有资源信息，包括存储数据（Local Storage、Session Storage、IndexedDB、Web SQL、Cookies）、缓存数据、字体、图片、脚本、样式表等。
- **Security**:判断当前网页是否安全。
- **Audits**:对当前网页进行网络利用情况、网页性能方面的诊断，并给出一些优化建议。比如列出所有没有用到的 CSS 文件等。

Network 面板

记录页面上的网络请求的详情信息，从发起网页页面请求 Request 后分析 HTTP 请求后得到的各个请求资源信息（包括状态、资源类型、大小、所用时间、Request 和 Response 等），可以根据这个进行网络性能优化。

该面板主要包括 5 大块窗格(Pane)：

1. **Controls** 控制 **Network** 的外观和功能。
2. **Filters** 控制 **Requests Table** 具体显示哪些内容。
3. **Overview** 显示获取到资源的时间轴信息。
4. **Requests Table** 按资源获取的前后顺序显示所有获取到的资源信息，点击资源名可以查看该资源的详细信息。
5. **Summary** 显示总的请求数、数据传输量、加载时间信息。



Requests Table 显示如下信息列：

- **Name** 资源名称，点击名称可以查看资源的详情情况，包括 Headers、Preview、Response、Cookies、Timing。

- **Status** HTTP 状态码。
- **Type** 请求的资源 MIME 类型。
- **Initiator** 标记请求是由哪个对象或进程发起的（请求源）。
    - Parser： 请求由 Chrome 的 HTML 解析器时发起的。
    - Redirect：请求是由 HTTP 页面重定向发起的。
    - Script：请求是由 Script 脚本发起的。
    - Other：请求是由其他进程发起的，比如用户点击一个链接跳转到另一个页面或者在地址栏输入 URL 地址。
- **Size** 从服务器下载的文件和请求的资源大小。如果是从缓存中取得的资源则该列会显示(from cache)
- **Time** 请求或下载的时间，从发起 Request 到获取到 Response 所用的总时间。
- **Timeline** 显示所有网络请求的可视化瀑布流(时间状态轴)，点击时间轴，可以查看该请求的详细信息，点击列头则可以根据指定的字段可以排序。

查看具体资源的详情

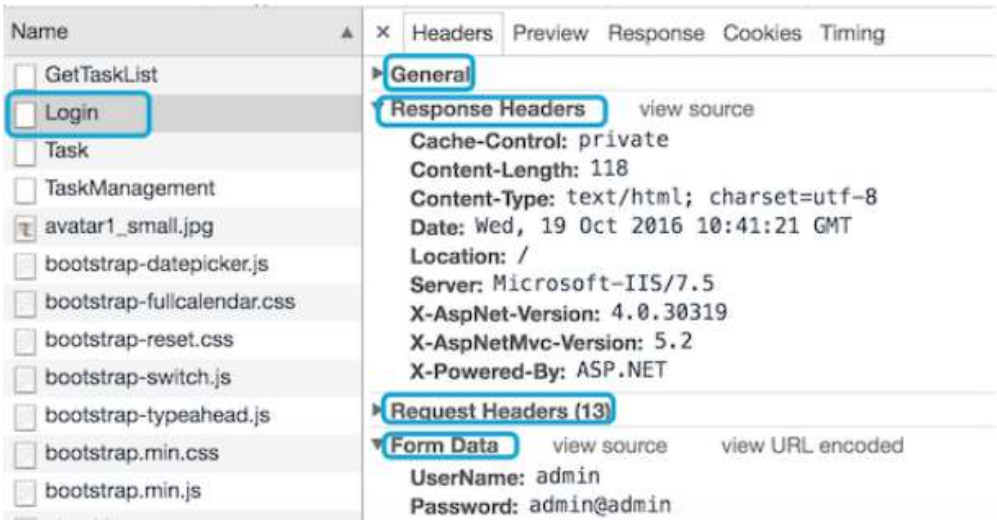
通过点击某个资源的 **Name** 可以查看该资源的详细信息，根据选择的资源类型显示的信息也不太一样，可能包括如下 Tab 信息：

- **Headers** 该资源的 HTTP 头信息。
- **Preview** 根据你所选择的资源类型（JSON、图片、文本）显示相应的预览。
- **Response** 显示 HTTP 的 Response 信息。
- **Cookies** 显示资源 HTTP 的 Request 和 Response 过程中的 Cookies 信息。
- **Timing** 显示资源在整个请求生命周期过程中各部分花费的时间。

针对上面 4 个 Tab 进行详细讲解一下各个功能：

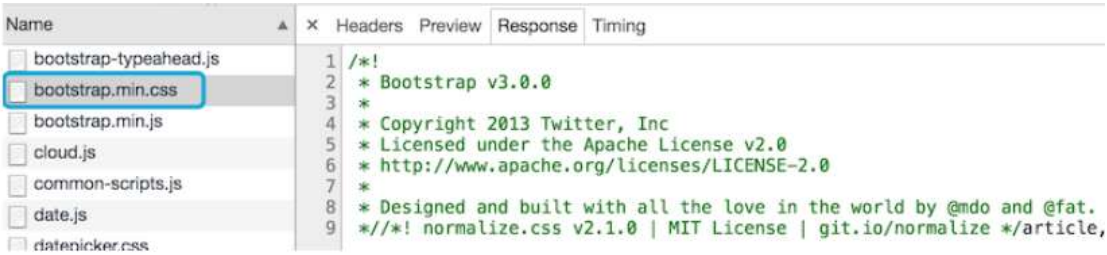
① 查看资源 HTTP 头信息

在 **Headers** 标签里面可以看到 HTTP Request URL、HTTP Method、Status Code、Remote Address 等基本信息和详细的 **Response Headers**、**Request Headers** 以及 **Query String Parameters** 或者 **Form Data** 等信息。



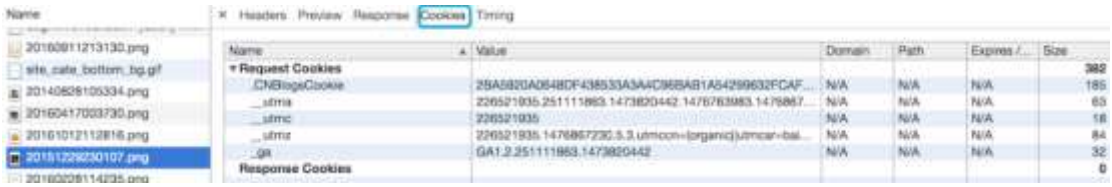
② 查看资源预览信息

在 **Preview** 标签里面可根据选择的资源类型（JSON、图片、文本、JS、CSS）显示相应的预览信息。下图显示的是当选择的资源是 JSON 格式时的预览信息。



④ 查看资源 Cookies 信息

如果选择的资源在 Request 和 Response 过程中存在 Cookies 信息，则 **Cookies** 标签会自动显示出来，在里面可以查看所有的 Cookies 信息。



查看资源的发起者(请求源)和依赖项

通过按住 Shift 并且把光标移到资源名称上，可以查看该资源是由哪个对象或进程发起的（请求源）和对该资源的请求过程中引发了哪些资源（依赖资源）。

在该资源的上方第一个标记为**绿色**的资源就是该资源的发起者（请求源），有可能会有第二个标记为**绿色**的资源是该资源的发起者的发起者，以此类推。

在该资源的下方标记为**红色**的资源是该资源的依赖资源。