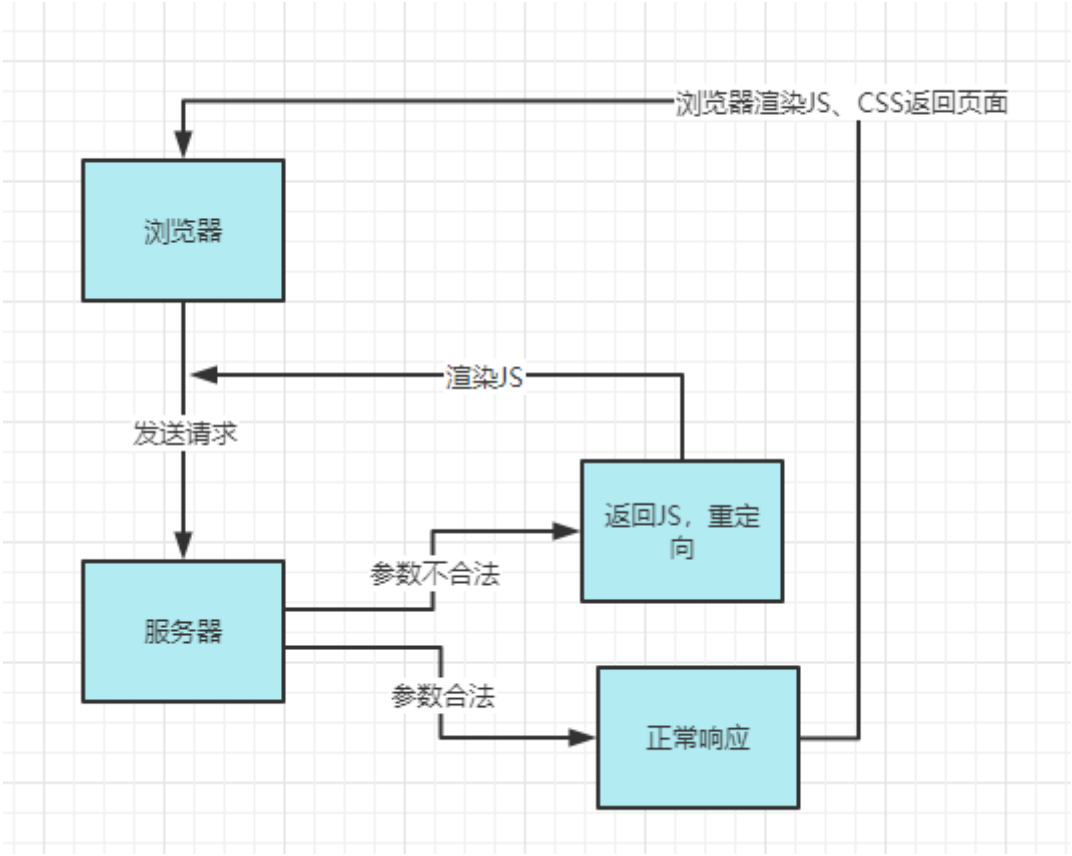
javascript调试

1 简介

JavaScript是一种可以直接被浏览器解析的直译式脚本语言，同样也是一种动态类型、弱类型、基于原型的语言



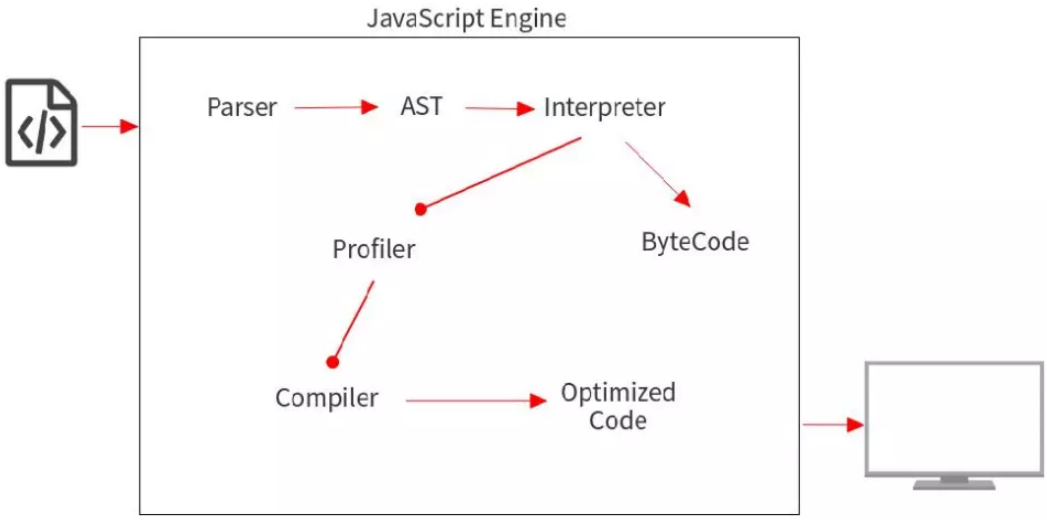
2 基本语法

```
// 变量
var xl = '夏洛';

console.log(xl);

// 函数
ff();
function ff() {
    console.log('sd')
}
// s();

s = function () {
    console.log('你好')
};


// 声明类
class Student {
```

```
    // 获取函数
    get myname() {
        return 'hh'
    }

    set name (val) {
    console.log('ff' + val)
  }
}

const b = new Student();
console.log(b.myname);
console.log(b.name = 123);


function add(a, b) {
    console.log(a+b)
}
for (var i=0; i<100;i++) {
    add(i,i+1)
}
```

3 js 执行原理

JavaScript 引擎是一种用于将我们的代码转换为机器可读语言的引擎



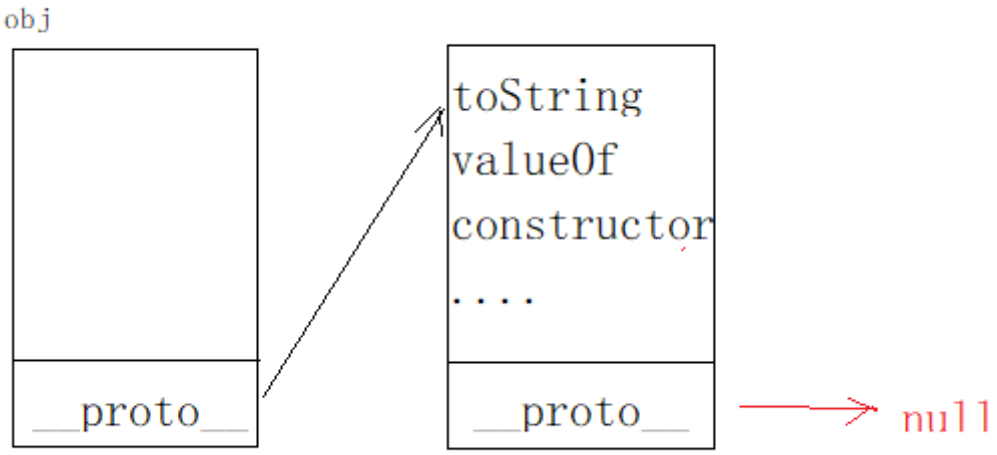
3.1 javascript原型链

原型链是原型对象创建过程的历史记录，当访问一个对象的某个属性时，会先在这个对象本身属性上查找

- JS 代码还没运行的时候，JS 环境里已经有一个 window 对象了。函数是对象
- window 对象有一个 Object 属性，window.Object 是一个函数对象
- window.Object 这个函数对象有一个重要属性是 prototype
- window.Object.prototype 里面有一堆属性

- 所有的实例函数**proto**都会 指向构造函数的 prototype
- constructor 是反向的 prototype

```
var obj = {}
obj.toString()
```



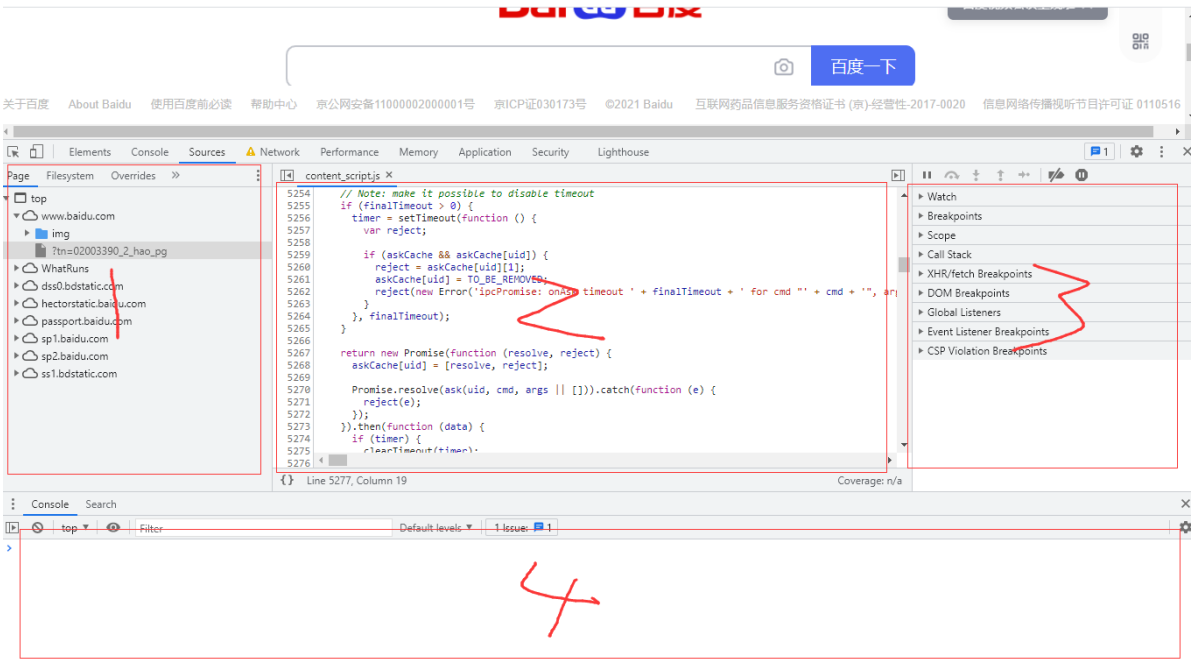
4 改文件断点

1，Source资源面板中显示加载当前页面需要的所有资源

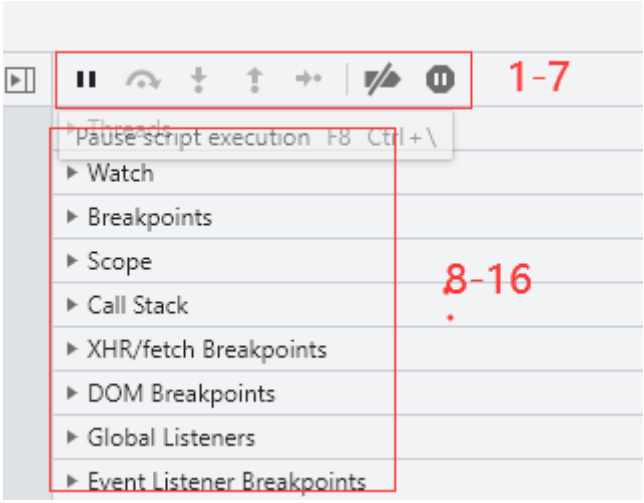
2，调试界面

3，断点策略和方法

4，调试窗口



5，组件介绍



1.Pause script excution 【单步执行，在断点处暂停，等待调试–不是直译】

2.Step over next function call【单步跳过】： 会跳到下一个断点

3.Step into next function call【单步进入】：会进入函数内部调试

4.Step out of current function 【单步跳出】: 会跳出当前这个断点的函数，

5.step： 一步步执行

6.Deactivate breakpoints : 使所有断点临时失效

7.Don’t Pause on exceptions: 不要在异常处暂停，

8.Pause On Caught Exceptions– 若抛出异常则需要暂停在那里

9.Watch： 监听表达式 不需要一次又一次地输入一个变量名或者表达式，你只需将他们添加到监视列表中就可以时时观察它们的变化：

9.Call stack：在一行代码里暂停时，可以在 Call Stack 面板查看是哪些栈将你带到了当前断点（到达当前函数调用了哪些函数）。如果不是在一行代码里暂停， Call Stack 面板是没有内容的。点击函数会跳到此函数调用的地方。蓝色箭头是当前查看的函数。在 Call Stack 面板里右键，选择 Copy stack trace 可以将面板里的 stack 信息复制到 clipboard。复制格式如下（函数名称、在代码里的行数）：

fix (jquery.min.js:formatted:2005)

dispatch (jquery.min.js:formatted:1961)

r.handle (jquery.min.js:formatted:1853)

10.scope:可以看到相当多实用的信息，比如this的指向，是否有值，断点是对象还是其他等。。

11.BreakPoints: 记录了标记的所有断点，可以点击跳转

12.XHR/FETCH BreakPoints: 针对某一个请求或者请求的关键字设置断点

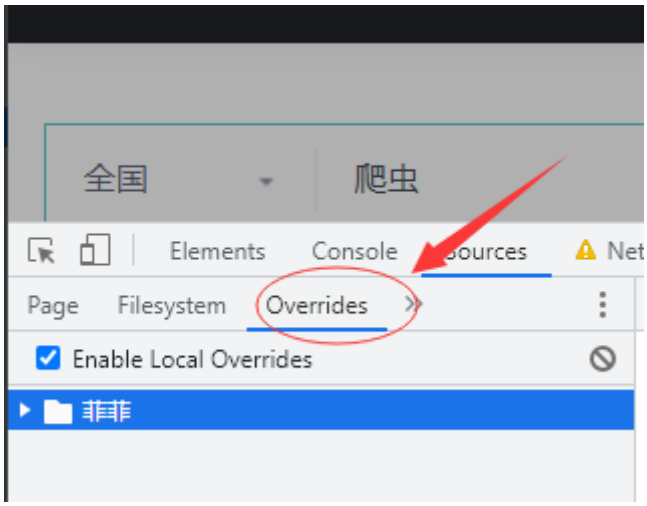
13.Dom BreakPoints:右键单击某个DOM元素，并选择Break on下的subtree modifications。这样调试器就可以在脚本遍历到该元素并且要修改它的时候自动停止，以让用户进行调试检查

14.Global listener:

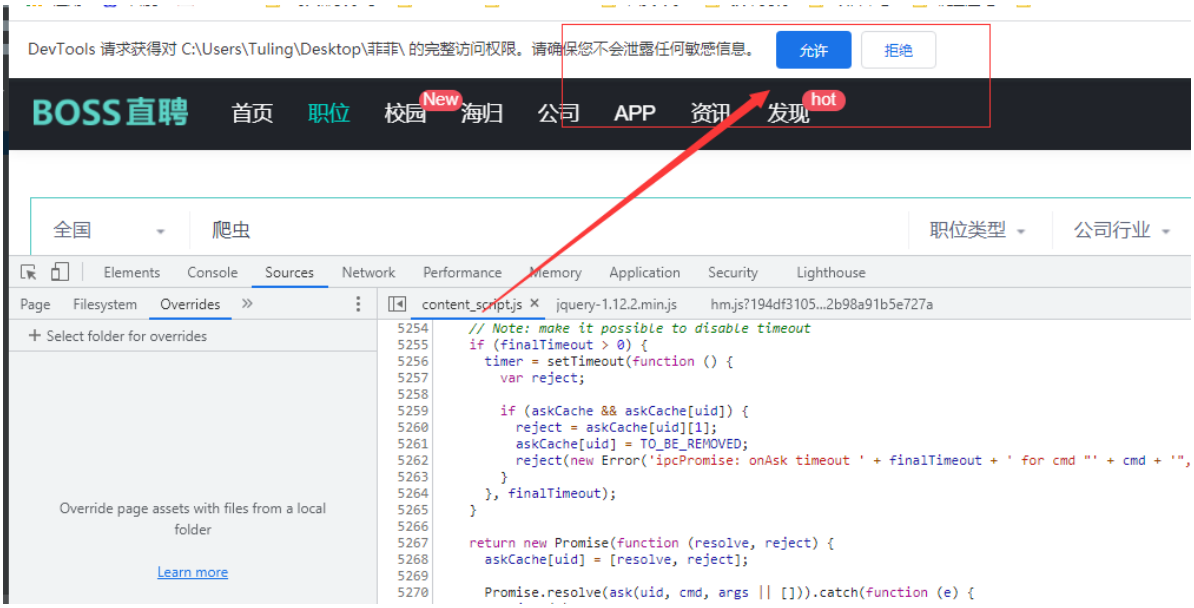
15.Event listener breakPoints:在监听器监听到某个事件发生的时候，断点暂停

注入编辑JS文件，调试

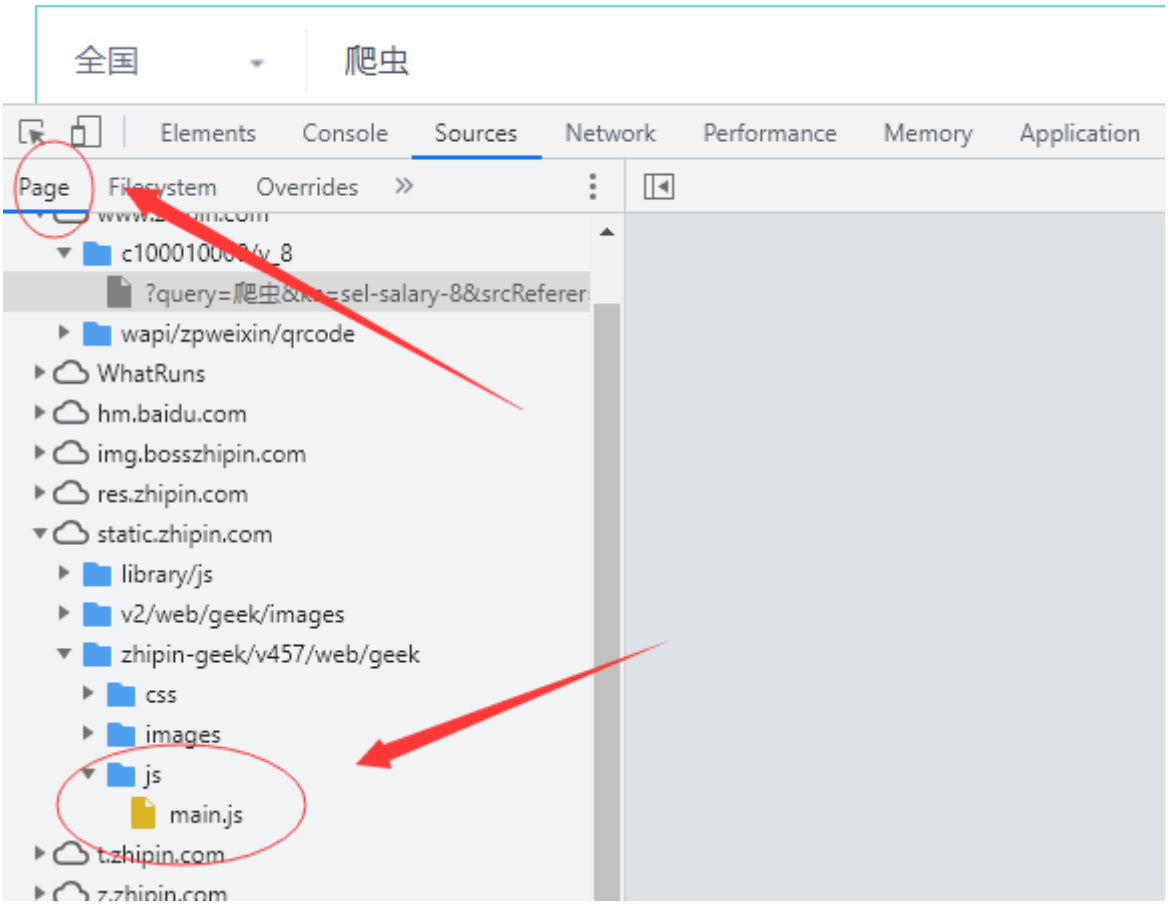
1，选择文件



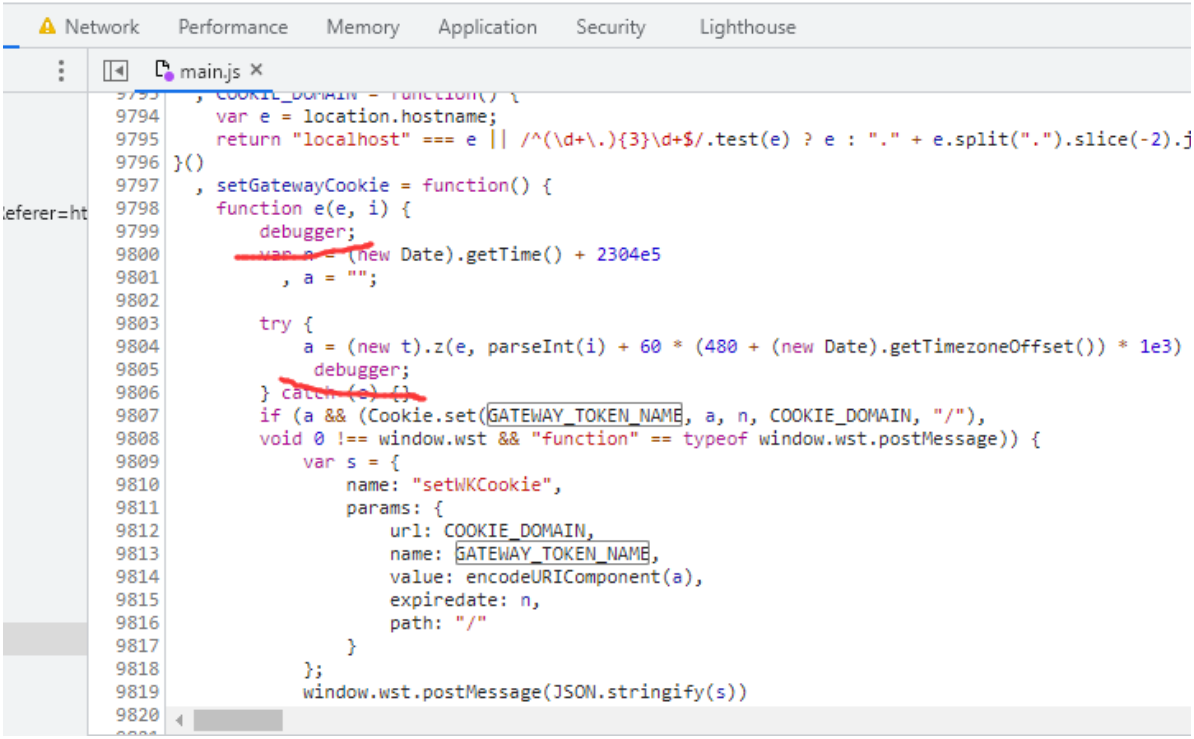
2， 导入文件

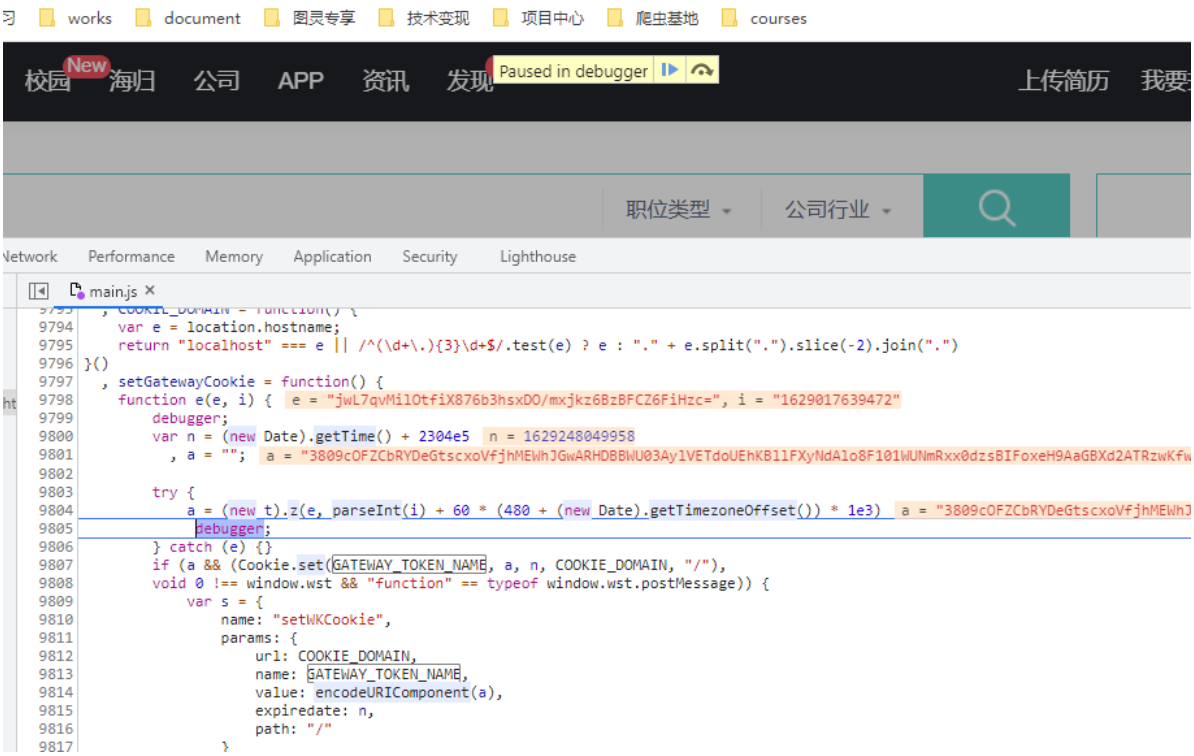


3，调试JS


全国
爬虫
Elements
Console
Sources
Network
Performance
Memory
Application
Page
Filesystem
Overrides
?query=爬虫&=sel-salary-8&srcReferer
wapi/zpweixin/qrcode
WhatRuns
hm.baidu.com
img.bosszhipin.com
res.zhipin.com
static.zhipin.com
library/js
v2/web/geek/images
zhipin-geek/v457/web/geek
css
images
js
main.js
t.zhipin.com

4, 可以修改JS做调试


Network
Performance
Memory
Application
Security
Lighthouse
main.js
var e = location.hostname;
return "localhost" === e || /^(\d+\.){3}\d+$/.test(e) ? e : "." + e.split(".").slice(-2).j
}()
, setGatewayCookie = function() {
function e(e, i) {
debugger;
var n = (new Date).getTime() + 2304e5
, a = "";
try {
a = (new t).z(e, parseInt(i) + 60 * (480 + (new Date).getTimezoneOffset()) * 1e3)
debugger;
} catch (e) {}
if (a && (Cookie.set(GATEWAY_TOKEN_NAME, a, n, COOKIE_DOMAIN, "/"),
void 0 !== window.wst && "function" == typeof window.wst.postMessage)) {
var s = {
name: "setWKCookie",
params: {
url: COOKIE_DOMAIN,
name: GATEWAY_TOKEN_NAME,
value: encodeURIComponent(a),
expiredate: n,
path: "/"
}
};
window.wst.postMessage(JSON.stringify(s))



5 debug原理

- 无限debbugger不会真正得死循环，而是有规律得执行逻辑，一般用定时器

```
Function("debugger;").call()
```

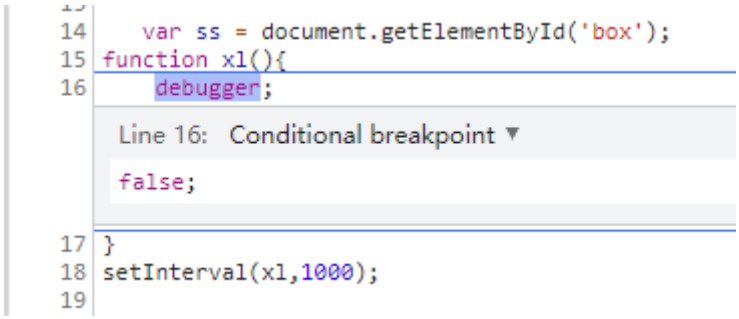
5.1 样例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>

<h1 id="box"></h1>

<body>

<script>
    var ss = document.getElementById('box')
    function f() {
        debugger;
    }
    setInterval(f,100);

    ss.innerHTML = "大家晚上好";

</script>
</body>
```

```
</html>
```

5.1.1 过debugger

1，当定义器运行到这个debugger这个代码的时候，那么这个时候它为true，它肯定执行我们的debugger代码，那我们可以用浏览器的功能给他改成false



2， hook代码置空函数

```
setInterval_ba = setInterval
setInterval = function xl(){}
```

3，类似于第一种

```
setInterval_ba = setInterval
setInterval = function(a, b){
    if(a.toString().indexOf("debugger") != -1){
        return setInterval_ba(a, b)
    }
}
setInterval = function(a, b){
    if(a.toString().indexOf("debugger") == -1){
        return setInterval_ba(a, b)
    }
}
```

5.2 过debugger

- http://cpquery.cnipa.gov.cn/

```
定时器倒计时：
for (var i = 1; i < 100; i++)window.clearInterval(i);
```

6 hook技术

- 主要用来找函数入口哦

```
Object.defineProperty(document,'cookie',{
get:function(){
    debugger;
    return;
},
  set:function(val){
    debugger;
    return;
}
})
```

7 web_API

7.1 浏览器指纹

地址：https://developer.mozilla.org/zh-CN/docs/Web/API

全局相关： window，document
环境相关： navigator(包括经纬度在内都在这个接口里)，screen，history
请求相关：XMLHttpRequest fetch worker
dom相关：canvas ，所有对dom节点操作，包括 jquery等三方库以及自设导入接口
数据库相关： Storage IndexedDB cookie
其他：caches WebGL AudioContext WebRTC

7.2 补环境案例

```
document = {};
document.getElementsByTagName = function(){
    return [
        {'textContent':'夏洛'}
    ]
};

title = document.getElementsByTagName('title')[0].textContent;
console.log(title);
```

8 中国航空综合案例

地址：https://et.airchina.com.cn/www/jsp/userManager/login.jsp

接口地址：https://et.airchina.com.cn/www/servlet/com.ace.um.userLogin.servlet.B2CUserLogin

js逆向代码

补环境

```
let navigator = {
    "cookieEnabled": true,
    "language": "zh-CN",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36",
    "webdriver": false,
    "appVersion": "5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/90.0.4430.212 Safari/537.36"
};

let window = {
 "navigator": navigator,
};
window.window = window;
```

```
RSAUtils.encryptedString = function(key, s) {
    var a = [];
    var sl = s.length;
    var i = 0;
    while (i < sl) {
        a[i] = s.charCodeAt(i);
        i++;
    }

    while (a.length % key.chunkSize != 0) {
        a[i++] = 0;
    }

    var al = a.length;
    var result = "";
    var j, k, block;
    for (i = 0; i < al; i += key.chunkSize) {
        block = new BigInt();
        j = 0;
        for (k = i; k < i + key.chunkSize; ++j) {
            block.digits[j] = a[k++];
            block.digits[j] += a[k++] << 8;
        }
        var crypt = key.barrett.powMod(block, key.e);
        var text = key.radix == 16 ? RSAUtils.biToHex(crypt) :
RSAUtils.biToString(crypt, key.radix);
        result += text + " ";
    }
    return result.substring(0, result.length - 1); // Remove last space.
};

var key ;
function bodyRSA()
{
    //setMaxDigits(130);
```

```
    key =
RSAUtils.getKeyPair("010001","","00984471b9a05c816ee949b4fe93520a8681a14e65d7a050
1221136951a52a3b76cf9e2375e45aca1ad6fc9f00b401ece966a1f8fb521dd9de4215c90b7e9cd7
7b1c7d2f6e9b7aba6f94322d7375cbb321be653826d921030b6ef9fd453a7ece0ae4785a6166dd5d
1560f3992cbad493201bb18616251610890bd0ea6736c346e15");
}

function getxl(password,username) {
    bodyRSA();
    var a = RSAUtils.encryptedString(key, password);
    var b = RSAUtils.encryptedString(key, username);
    console.log(b)
    console.log(a);
    return a
}
```