

HTML JavaScript 示例解析



详细解释下面代码的意思？<!DOCTYPE html>

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>Title</title>
```

```
</head>
```

```
<body>
```

```
<!--<a id="a" href="http://baidu.com">点我</a>-->
```

```
<!--<script type="text/javascript">-->
```

```
<!--  document.getElementById("a").href = "http://www.youku.com";//百度地址跳转到优酷-->
```

```
<!--  // var a=document.getElementById("a");-->
```

```
<!--  // a.setAttribute("href","http://www.baidu.com");-->
```

```
<!--<a id="a">点我</a>-->
```

```
<!--<script type="text/javascript">-->
```

```
<!--  var a=document.getElementById("a");-->
```

```
<!--  a.setAttribute("href","http://www.baidu.com");-->
```

```
<!--<button onclick="func()">点我</button>-->
```

```
<!--<script>-->
```

```
<!--  function func() {-->
```

```
<!--    alert("1");-->
```

```
<!--  }-->
```

```
<!--</script>-->
```

```

<!--dom操作-->
<button id="bn">点我</button>
<script>
  var bn=document.getElementById("bn");
  bn.onclick=function(){
    alert("xxx");
  }
</script>

</script>
</body>
</html>

```



这段代码是一个示例 HTML 文件，展示了几个常见的 HTML 和 JavaScript 结合操作。以下是详细解释：

1. HTML 文件结构

```

html

<!DOCTYPE html>
<html lang="en">

```

- `<!DOCTYPE html>`: 声明 HTML 文档类型，表明这是 HTML5 标准文档。
- `<html lang="en">`: 表示网页的语言为英文，`lang="en"` 帮助搜索引擎或屏幕阅读器了解页面语言。

```

html

<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>

```

- `<meta charset="UTF-8">`: 声明文档使用 UTF-8 字符编码，确保可以正确显示特殊字符。
- `<title>Title</title>`: 定义网页标题，显示在浏览器标签中。

2. 示例 1: 修改链接地址

原始代码 (已注释)

html

```
<!--<a id="a" href="http://baidu.com">点我</a-->
```

- 定义了一个链接（`<a>` 标签），其 `id` 是 `a`，初始跳转地址为百度（`http://baidu.com`）。
- `点我`：超链接的显示文本。

修改链接地址

html

```
<!--
<script type="text/javascript">
    document.getElementById("a").href = "http://www.youku.com";//百度地址跳转到优酷
-->
```

- JavaScript 操作：通过 `document.getElementById("a")` 获取 `id="a"` 的 `<a>` 元素。
- `href = "http://www.youku.com"`：修改 `href` 属性，将跳转地址更改为优酷（`http://www.youku.com`）。
- 备注：这部分代码被注释，页面实际不会执行。

使用 `setAttribute`

html

```
<!--
var a=document.getElementById("a");
a.setAttribute("href","http://www.baidu.com");
-->
```

- 另一种修改属性的方式：使用 `setAttribute` 方法直接设置 `href` 值。
- 结果与前述方式类似。

3. 示例 2: 按钮与事件绑定

html

```
<button onclick="func()">点我</button>
<script>
    function func() {
        alert("1");
    }
</script>
```

- 定义了一个按钮 `<button>`，点击时会调用 JavaScript 函数 `func()`。
- `alert("1")`：弹出警告框，显示数字 `1`。

4. 示例 3: DOM 操作与事件监听

html

```
<button id="bn">点我</button>
<script>
  var bn=document.getElementById("bn");
  bn.onclick=function(){
    alert("xxx");
  }
</script>
```

代码说明

- `<button id="bn">点我</button>`: 定义了一个按钮, `id="bn"` 用于唯一标识。
- JavaScript 操作:
 1. `document.getElementById("bn")`: 获取 `id` 为 `bn` 的按钮。
 2. `bn.onclick=function(){...}`: 为按钮绑定 `onclick` 事件, 当按钮被点击时触发函数。
 3. `alert("xxx")`: 弹出一个警告框, 显示文本 "xxx"。

总结

这段代码展示了 HTML 和 JavaScript 的一些基础功能:

1. **动态修改 DOM 元素属性** (`getElementById` 和 `setAttribute`)。
2. **事件处理**, 包括直接在 HTML 中绑定事件 (`onclick="..."`) 和通过 JavaScript 动态绑定事件 (`element.onclick=function(){}`)。
3. 使用注释 (`<!-- -->`) 组织代码, 部分内容被注释, 实际页面中不会执行。

注意:

- 代码中有一些重复示例或注释掉的代码, 可以优化结构。
- 动态绑定事件更推荐使用 `addEventListener`, 而不是 `onclick`, 以避免事件覆盖问题。



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

<a id="a">点我</a>
```

```
</img>

<script type="text/javascript">

    var url=window.location.hash;

    //实现：获取用户访问的url里面的域名，将其作为点我的触发链接

    var urls=url.substring(1);
    console.log(url);
    console.log(urls);
    var a=document.getElementById("a").href=urls;
    //a.setAttribute("href","http://www.baidu.com");

    var hash = location.hash;
    console.log(hash);
    if(hash){
        var url = hash.substring(1);
        console.log(url);
        location.href = url;
    }


    var hash = location.hash;
    console.log(hash);
    if(hash){
        var url = hash.substring(1);
        console.log(url);
        document.write(decodeURI(url));
    }

</script>
```

```
</body>
</html>
```



这段代码展示了多个功能，包括通过 `window.location.hash` 获取 URL 中的哈希值、动态修改链接，以及简单的 XSS 演示。下面是详细解析：

1. HTML 文件结构

```
html

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

- 声明文档类型、语言和字符编码，与标准 HTML5 一致。

2. 链接与图片

链接标签

```
html

<a id="a">点我</a>
```

- 一个没有定义 `href` 属性的超链接，`id="a"` 用于 JavaScript 获取和操作该元素。

图片标签

```
html

</img>
```

- 图片加载失败时触发 `onerror` 事件，弹出警告框。
- `src="dasdsad"` 指向一个不存在的路径，因此图片无法加载，会触发 `onerror`。

3. JavaScript 动态操作

获取哈希值并修改链接

```
javascript
```

```
var url = window.location.hash;
var urls = url.substring(1);
console.log(url);
console.log(urls);
var a = document.getElementById("a").href = urls;
```

- `window.location.hash`：获取当前 URL 中的哈希部分（从 `#` 开始的内容）。
 - 例如，若 URL 为 `http://example.com/#http://baidu.com`，`hash` 值为 `#http://baidu.com`。
- `url.substring(1)`：去掉哈希符号 `#`，提取实际的内容。
 - 结果为 `http://baidu.com`。
- 将提取的 URL 动态赋值给超链接 `<a>` 的 `href` 属性，链接地址变为 `http://baidu.com`。

重定向功能

```
javascript
```

```
var hash = location.hash;
console.log(hash);
if (hash) {
  var url = hash.substring(1);
  console.log(url);
  location.href = url;
}
```

- 再次获取哈希值并检查是否存在。
- 若存在，则提取哈希值内容并将其赋值给 `location.href`。
 - `location.href = url`：浏览器直接跳转到新的 URL。

显示哈希值内容

```
javascript
```

```
var hash = location.hash;
console.log(hash);
if (hash) {
  var url = hash.substring(1);
  console.log(url);
  document.write(decodeURI(url));
}
```

- 再次获取哈希值并解码为正常字符串。
- 使用 `document.write` 将解码后的 URL 内容直接写入页面。

4. 整体功能分析

功能 1: 动态修改超链接

- 根据 `window.location.hash` 中的值，将链接动态改为用户指定的地址。

功能 2: 页面重定向

- 如果 URL 中存在哈希值，页面会直接跳转到指定的地址。

功能 3: 页面内容展示

- 如果存在哈希值，页面会显示哈希值内容的解码版本。

功能 4: `onerror` 示例

- `img` 标签的 `onerror` 属性用来展示错误处理方法，证明事件机制有效。

5. 安全问题分析

(1) XSS 漏洞

- 用户可以在 URL 中插入恶意代码，例如：
 - `http://example.com/#javascript:alert('XSS')`。
 - 在功能 1 中，超链接的 `href` 会被设置为 `javascript:alert('XSS')`。
 - 如果用户点击该链接，恶意代码会执行。
- `document.write(decodeURI(url))` 直接将用户输入输出到页面，可能导致代码注入。

(2) URL 重定向攻击

- 功能 2 中，若攻击者构造恶意链接如：
 - `http://example.com/#http://phishingsite.com`，用户访问时会被直接重定向到钓鱼网站。

(3) 图片 `onerror`

- 图片的 `onerror` 属性用于测试事件机制，但实际中可能被攻击者利用引入恶意代码。

6. 改进建议

(1) 防止 XSS 漏洞

- 检查并过滤 `window.location.hash` 的内容，确保只允许合法的 URL：

```
javascript

var hash = location.hash;
if (hash) {
  var url = hash.substring(1);
  if (url.startsWith("http://") || url.startsWith("https://")) {
    document.getElementById("a").href = url;
  } else {
    console.error("Invalid URL");
  }
}
```



```
}
}
```

(2) 禁用危险功能

- 避免使用 `document.write` 和直接修改 `location.href`，改用更安全的显示和跳转方式。

(3) 使用图片 `onerror` 的替代方法

- 移除 `onerror` 或添加验证逻辑以确保仅处理受信任的内容。

7. 总结

这段代码功能丰富但存在较大的安全隐患，尤其是在处理用户输入的 URL 时。通过添加严格的验证机制和限制动态执行代码，可以提升代码的安全性和可靠性。



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

<script>
  //window.open("http://www.baidu.com");
  // window.document.getElementById = document.getElementById()
  // window.screen.width = screen.width


  console.log(screen.height);
  console.log(screen.width);
  console.log(screen);
  console.log(location);
  console.log(location.host);
  console.log(location.hostname);
  console.log(location.protocol);
  console.log(location.port);
  console.log(location.pathname);


  console.log(navigator);
  console.log(navigator.appName);
  console.log(navigator.appVersion);
  console.log(navigator.userAgent);
```

```
// history.back();  
// history.forward();  
console.log(window);  
</script>
```

```
</body>  
</html>
```



这段代码展示了 JavaScript 中常用的浏览器对象模型（BOM）相关属性和方法的基本使用。以下是每一部分的详细解析：

1. HTML 文件结构

```
html

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

- 声明 HTML5 文档类型，语言为英文，字符编码为 UTF-8。

2. 脚本内容

2.1 已注释部分

```
javascript

// window.open("http://www.baidu.com");
```

- 使用 `window.open` 打开一个新窗口，跳转到指定的 URL（如百度）。
- **注意：**现代浏览器可能会限制此操作，除非通过用户交互触发。

```
javascript

// window.document.getElementById() = document.getElementById()
```

- 简单说明 `window` 是全局对象，`document.getElementById` 可直接使用，无需前缀 `window.`。

```
javascript

// window.screen.width = screen.width
```

- 说明 `screen` 对象直接挂载在 `window` 上，`window.screen` 和 `screen` 等效。

2.2 浏览器对象模型属性与方法

1. `screen` 对象

javascript

```
console.log(screen.height);
console.log(screen.width);
console.log(screen);
```

- `screen.height`：屏幕的高度（以像素为单位）。
- `screen.width`：屏幕的宽度（以像素为单位）。
- `console.log(screen)`：输出 `screen` 对象，包含更多屏幕相关信息，例如：
 - `availHeight` 和 `availWidth`：可用的屏幕高度和宽度（减去任务栏等不可用部分）。
 - `colorDepth`：屏幕的颜色深度。
 - `pixelDepth`：屏幕的像素深度。

2. `location` 对象

javascript

```
console.log(location);
console.log(location.host);
console.log(location.hostname);
console.log(location.protocol);
console.log(location.port);
console.log(location.pathname);
```

- `location`：包含当前文档 URL 的相关信息，以下是常见属性：
 - `host`：主机名和端口号（例如 `example.com:8080`）。
 - `hostname`：主机名（例如 `example.com`）。
 - `protocol`：协议（例如 `http:` 或 `https:`）。
 - `port`：端口号（如果未显式指定端口，返回空字符串）。
 - `pathname`：URL 的路径部分（例如 `/path/to/page`）。

3. `navigator` 对象

javascript

```
console.log(navigator);
console.log(navigator.appName);
console.log(navigator.appVersion);
console.log(navigator.userAgent);
```

- `navigator`：表示用户代理（浏览器）相关信息，以下是常见属性：
 - `appName`：浏览器名称（通常返回 `Netscape`，现代浏览器不可靠）。
 - `appVersion`：浏览器版本（包含大量信息，实际意义有限）。
 - `userAgent`：用户代理字符串，提供有关浏览器、操作系统的信息。
 - 例如：

SCSS

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36

4. `history` 对象

javascript

```
// history.back();
// history.forward();
```

- `history.back()`：浏览器导航到上一个历史记录（等效于用户点击返回按钮）。
- `history.forward()`：浏览器导航到下一个历史记录（等效于用户点击前进按钮）。

5. `window` 对象

javascript

```
console.log(window);
```

- `window` 是全局对象，包含浏览器窗口的所有内容和方法。输出 `window` 对象会展示其所有属性和方法，包括：
 - `alert`、`confirm`、`prompt` 等常用弹窗方法。
 - `setTimeout`、`setInterval` 等计时器方法。
 - `document`、`location`、`navigator`、`screen`、`history` 等子对象。

3. 功能与输出

执行脚本时，浏览器控制台会输出以下信息（内容视运行环境而定）：

屏幕相关信息

plaintext

```
1080 // screen.height
1920 // screen.width
Screen { ... } // 完整的 screen 对象信息
```

URL 相关信息

假设当前 URL 是 `http://example.com:8080/path/to/page`：

```
plaintext
```

```
Location { ... } // location 对象
example.com:8080 // location.host
example.com      // location.hostname
http:           // location.protocol
8080            // location.port
/path/to/page    // location.pathname
```

浏览器信息

```
plaintext
```

```
Navigator { ... } // navigator 对象
Netscape      // navigator.appName (现代浏览器通常返回固定值)
5.0 (Windows) ... // navigator.appVersion
Mozilla/5.0 ... // navigator.userAgent
```

窗口信息

```
plaintext
```

```
Window { ... } // window 对象
```

4. 总结

- 代码演示了 JavaScript 通过 BOM 操作浏览器环境，获取设备和浏览器信息。
- 输出的结果依赖于用户的操作系统、浏览器和当前访问的页面 URL。
- 实际开发中需注意以下几点：
 1. **兼容性**：不同浏览器对某些属性的支持和返回值可能不一致。
 2. **安全性**：动态跳转（如 `location.href`）需验证输入，避免 XSS 和重定向攻击。
 3. **性能优化**：避免不必要的 `console.log` 或冗余的属性访问操作。