

# 原生AJAX解析



```

详细解释代码的意思<script type="text/javascript">
//  const a=1;
// //  document.write(a); 输出网页上
// //  document.log(a); //输出到控制端
//原生ajax demo案例
var xhttp = new XMLHttpRequest();
xhttp.open("GET", "1.txt", true);
xhttp.send();
xhttp.onreadystatechange = function () {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
        console.log(xhttp.responseText);
    }
}

```

ChatGPT 说：  
ChatGPT



这段代码展示了如何使用原生 JavaScript 实现一个简单的 AJAX 请求来从服务器获取数据。下面逐步解释代码的作用：

## 1. 引入JavaScript 代码

```

javascript

<script type="text/javascript">

```

这行代码指定了 JavaScript 脚本的类型。在现代浏览器中，这是可选的，但用于确保向后兼容性。

## 2. 注释部分

```

javascript

//      const a=1;
// //      document.write(a); 输出网页上
// //      document.log(a); //输出到控制端

```

- ``const a=1;``: 定义了一个常量 ``a``，其值为 ``1``。

- `document.write(a);`: 如果取消注释, 这会将变量 `a` 的值 (即 `1`) 直接写入网页。
- `document.log(a);`: 这是错误的写法, 应该是 `console.log(a);`, 用于将 `a` 的值打印到控制台。

### 3. 创建 XMLHttpRequest 对象

```
javascript

var xhttp = new XMLHttpRequest();
```

- `XMLHttpRequest` 是原生 JavaScript 中用于与服务器进行异步通信的 API。
- 通过 `new XMLHttpRequest()` 创建一个实例对象 `xhttp`。

### 4. 初始化请求

```
javascript

xhttp.open("GET", "1.txt", true);
```

- `open`: 初始化一个请求。
  - `"GET"`: 指定 HTTP 请求方法 (GET 用于获取数据)。
  - `"1.txt"`: 指定请求的目标资源 (这里是服务器上的一个文本文件 `1.txt`)。
  - `true`: 指定请求是否为异步 (`true` 表示异步)。
- 异步请求的特点是不会阻塞脚本执行, 浏览器可以继续运行其他任务。

### 5. 发送请求

```
javascript

xhttp.send();
```

- `send`: 发送 HTTP 请求到服务器。
- 对于 `GET` 请求, 不需要传递请求体。

### 6. 监听状态变化

```
javascript

xhttp.onreadystatechange = function () {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
        console.log(xhttp.responseText);
    }
};
```

- `onreadystatechange`: 定义了一个事件处理函数，在 `XMLHttpRequest` 对象的状态发生变化时触发。
- `xhttp.readyState`: 表示请求的当前状态，可能的值有：
  1. `0`: 请求未初始化。
  2. `1`: 服务器连接已建立。
  3. `2`: 请求已接收。
  4. `3`: 请求处理中。
  5. `4`: 请求完成，响应已就绪。
- `xhttp.status`: 表示 HTTP 状态码。
  - `200`: 表示请求成功，服务器返回了正确的响应。
  - 如果状态码是其他值，比如 `404`，则表示资源未找到。

## 7. 处理响应

javascript

```
console.log(xhttp.responseText);
```

- `xhttp.responseText`: 包含服务器返回的文本响应。
- 使用 `console.log` 将响应内容打印到浏览器控制台。

## 代码完整的作用

这段代码执行以下操作：

1. 创建一个 `XMLHttpRequest` 对象。
2. 初始化一个异步 GET 请求，目标是从服务器获取 `1.txt` 文件的内容。
3. 发送请求。
4. 当请求完成且成功时（状态码 200），将服务器返回的文本内容输出到控制台。

## 注意事项

1. **跨域问题**：如果 `1.txt` 文件在不同的域名上，需要处理跨域（CORS）。
2. **替代方案**：现代开发中推荐使用 `fetch` API 或第三方库（如 Axios）进行 AJAX 请求，语法更简洁，支持 Promise。



```
<!--远程引用-->
<!-- <script src="https://code.jquery.com/jquery-3.7.1.js"></script>-->
<!--本地引用-->
<script src="jquery-3.7.1.min.js"></script>
<script>
  //采用jQuery库实现ajax技术
  $.ajax({
    method: "GET",
    url: "1.txt",
```

```

    dataType: "text",
    success: function(response){
        console.log(response);
    }

});

</script>

```



这段代码使用了 jQuery 库来实现 AJAX 请求，相较于原生 JavaScript，代码更简洁且更易于书写和理解。以下是代码的逐步解释：

## 1. jQuery 引用

```

html

<!-- 远程引用 -->
<!-- <script src="https://code.jquery.com/jquery-3.7.1.js"></script> -->
<!-- 本地引用 -->
<script src="jquery-3.7.1.min.js"></script>

```

- **远程引用:** 使用 jQuery 官方提供的远程链接（CDN）加载 jQuery 库（已注释）。
- **本地引用:** 直接引用本地下载的 jQuery 文件 `jquery-3.7.1.min.js`。

jQuery 是一个广泛使用的 JavaScript 库，提供了许多工具函数，简化了 DOM 操作、事件处理、动画效果，以及 AJAX 请求的实现。

## 2. 使用 jQuery 实现 AJAX

```

javascript

$.ajax({
    method: "GET",
    url: "1.txt",
    dataType: "text",
    success: function(response){
        console.log(response);
    }
});

```

- `$.ajax`: jQuery 提供的函数，用于发起 AJAX 请求。它封装了底层的 `XMLHttpRequest`，语法更直观。

配置参数详解：

### 1. `method`:

- 指定 HTTP 请求的方法，这里使用 `GET` 方法从服务器获取数据。

### 2. `url`:

- 请求的目标地址（这里是 `1.txt` 文件）。

### 3. `dataType`:

- 指定服务器返回数据的类型，这里是 `"text"`，表示预期返回纯文本内容。
- 其他常见值：
  - `"json"`: 返回 JSON 格式数据。
  - `"xml"`: 返回 XML 格式数据。

### 4. `success`:

- 请求成功时执行的回调函数。
- `response`: 回调函数的参数，包含服务器返回的内容。

## 3. 代码功能

### 1. 发起异步 `GET` 请求:

- 目标资源是 `1.txt` 文件。

### 2. 处理成功响应:

- 如果请求成功，服务器返回的文本内容会通过参数 `response` 传递给 `success` 回调函数。
- 通过 `console.log(response)` 将返回的文本输出到浏览器控制台。

## 4. jQuery AJAX 的优势

- 更简洁的语法，无需手动处理事件绑定（如 `onreadystatechange`）。
- 自动处理跨浏览器的兼容性问题。
- 可以轻松添加更多选项（如错误处理、超时设置等）。

## 5. 如何扩展

- 添加错误处理：

```
javascript
```

```
error: function(xhr, status, error) {
    console.error("请求失败:", error);
}
```

- 设置超时时间：

```
javascript
```

```
timeout: 5000 // 设置超时时间为 5 秒
```

完整扩展后的代码：

javascript

```
$.ajax({
  method: "GET",
  url: "1.txt",
  dataType: "text",
  timeout: 5000,
  success: function(response) {
    console.log("请求成功:", response);
  },
  error: function(xhr, status, error) {
    console.error("请求失败:", error);
  }
});
```

## 总结

这段代码通过 jQuery 的 `\$.ajax` 方法实现了一个简洁的 AJAX 请求，用于从服务器获取 `1.txt` 文件内容并打印到控制台。它的语法简洁且功能强大，尤其适合需要快速实现 AJAX 的场景。



```
<script src="axios.min.js"></script>
<script>
  // axios({
  //   method: 'GET',
  //   url: '1.txt',
  // }).then(function (response) {
  //   console.log(response.data);
  // })

  axios.get('1.txt').then(function (response) {console.log(response.data);})
</script>
```



这段代码展示了如何使用 `Axios` 库实现 AJAX 请求。`Axios` 是一个基于 Promise 的 HTTP 客户端，提供了现代化的语法和更丰富的功能，深受开发者欢迎。

以下是代码的详细解析：

## 1. Axios 引用

html

```
<script src="axios.min.js"></script>
```

- `axios.min.js` 是 Axios 的压缩版本，用于高效加载。

- 需要确保 `axios.min.js` 文件已经正确下载到项目的路径中，或者可以直接使用 [Axios 官方的 CDN 地址](#) 进行远程引用：

html

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

## 2. 使用 Axios 发起 GET 请求

javascript

```
axios({
  method: 'GET',
  url: '1.txt',
}).then(function (response) {
  console.log(response.data);
});
```

- `axios`:
  - 主函数，用于发送 HTTP 请求。
  - 接受一个配置对象作为参数。

### 配置对象说明：

1. `method`:
  - 指定 HTTP 请求的方法（`GET` 方法，用于获取数据）。
2. `url`:
  - 请求的目标地址（`1.txt` 文件）。
3. `then`:
  - 请求成功时调用的回调函数。
  - `response`: 包含服务器返回的数据和相关信息。

## 3. 简写形式

javascript

```
axios.get('1.txt').then(function (response) {
  console.log(response.data);
});
```

- `axios.get(url)` 是简化语法，等同于完整的配置形式。
- 返回的 `response` 对象包含以下内容：
  - `response.data`: 服务器返回的具体内容（这里是 `1.txt` 文件的内容）。
  - `response.status`: HTTP 状态码（如 `200` 表示成功）。
  - `response.statusText`: HTTP 状态描述（如 `OK`）。

- `response.headers`: 响应头信息。
- `response.config`: 请求的配置信息。

## 4. 代码功能

1. 使用 `axios.get` 向服务器发起一个 `GET` 请求。
2. 请求成功后，通过回调函数访问服务器返回的数据。
3. 使用 `console.log(response.data)` 将返回的数据打印到浏览器控制台。

## 5. 扩展功能

### 添加错误处理

可以通过 `catch` 方法处理请求失败的情况，例如网络错误或服务器返回错误状态码：

```
javascript

axios.get('1.txt')
  .then(function (response) {
    console.log("请求成功:", response.data);
  })
  .catch(function (error) {
    console.error("请求失败:", error);
  });
```

### 异步/等待 (Async/Await) 写法

如果在现代 JavaScript 项目中使用 `async/await`，代码更为直观：

```
javascript

(async function fetchData() {
  try {
    const response = await axios.get('1.txt');
    console.log("请求成功:", response.data);
  } catch (error) {
    console.error("请求失败:", error);
  }
})();
```

### 传递查询参数

可以通过 `params` 属性传递 URL 查询参数：

```
javascript

axios.get('1.txt', {
  params: {
    key1: 'value1',
    key2: 'value2'
  }
});
```



```
    }  
  }).then(function (response) {  
    console.log(response.data);  
  });  
});
```

## 6. Axios 的优势

1. **Promise 支持**: 使异步操作更易于处理。
2. **支持更多功能**:
  - 请求/响应拦截器。
  - 自动转换 JSON 数据。
  - 超时控制。
3. **更好的浏览器兼容性**:
  - 支持现代浏览器和旧版浏览器。
4. **简洁的语法**: 简化了 HTTP 请求的操作。

## 7. 总结

这段代码通过 `Axios` 库实现了一个简单的 `GET` 请求，用于从服务器加载 `1.txt` 文件的内容，并在成功后将其打印到控制台。相比于传统的 AJAX 实现，`Axios` 提供了更现代的语法和功能扩展，是推荐的 HTTP 请求工具。



```
<style>  
  body {  
    background-color: #f1f1f1;  
  }  
  .login {  
    width: 400px;  
    margin: 100px auto;  
    background-color: #fff;  
    border-radius: 5px;  
    box-shadow: 0 0 10px rgba(0,0,0,0.3);  
    padding: 30px;  
  }  
  .login h2 {  
    text-align: center;  
    font-size: 2em;  
    margin-bottom: 30px;  
  }  
  .login label {  
    display: block;  
    margin-bottom: 20px;  
    font-size: 1.2em;  
  }  
  .login input[type="text"], .login input[type="password"] {  
    width: 100%;
```

```
padding: 10px;
border: 1px solid #ccc;
border-radius: 5px;
font-size: 1.2em;
margin-bottom: 20px;
}
.login input[type="submit"] {
background-color: #2ecc71;
color: #fff;
border: none;
padding: 10px 20px;
border-radius: 5px;
font-size: 1.2em;
cursor: pointer;
}
.login input[type="submit"]:hover {
background-color: #27ae60;
}
</style>
</head>
<body>
<div class="login">
<h2>后台登录</h2>
<label for="username">用户名:</label>
<input type="text" name="username" id="username" class="user" >
<label for="password">密码:</label>
<input type="password" name="password" id="password" class="pass" >
<button>登录</button>
</div>

<script src="jquery-3.7.1.min.js"></script>
<script>
$("button").click(function (){
$.ajax({
type: 'POST',
url: 'login.php',
data: {
username:$(".user").val(),
password:$(".pass").val()
},
dataType: 'json',
success: function (data) {
console.log(data);
if (data['infoCode']==1){
alert('登录成功!');
window.open('admin.html');
//正确登录的业务逻辑代码在下面
}else{
alert('登录失败!');

```

```

    }
  }
});
});
</script>

```



这段代码实现了一个基于 HTML、CSS、jQuery 和 AJAX 的简单后台登录界面，以下是详细分析：

## 1. 页面样式 (CSS)

CSS

```

body {
  background-color: #f1f1f1;
}

```

- 设置页面背景颜色为浅灰色，营造简洁的背景效果。

CSS

```

.login {
  width: 400px;
  margin: 100px auto;
  background-color: #fff;
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0,0,0,0.3);
  padding: 30px;
}

```

- `.login` 类定义了登录框样式：
  - 宽度 `400px`，居中显示（通过 `margin: 100px auto` 实现）。
  - 白色背景、圆角边框和阴影效果，提升视觉效果。
  - 内部间距 `30px` 增强布局美观。

CSS

```

.login h2 {
  text-align: center;
  font-size: 2em;
  margin-bottom: 30px;
}

```

- 设置标题为居中对齐，字体大小为 2 倍标准大小，增加下边距。

CSS

```

.login input[type="text"], .login input[type="password"] {
  width: 100%;
}

```

```
padding: 10px;
border: 1px solid #ccc;
border-radius: 5px;
font-size: 1.2em;
margin-bottom: 20px;
}
```

#### • 输入框样式：

- 宽度 `100%`，确保与登录框对齐。
- 内边距 `10px` 和圆角边框。
- 下边距 `20px` 为控件提供间隔。

CSS

```
.login input[type="submit"], .login button {
  background-color: #2ecc71;
  color: #fff;
  border: none;
  padding: 10px 20px;
  border-radius: 5px;
  font-size: 1.2em;
  cursor: pointer;
}
.login input[type="submit"]:hover {
  background-color: #27ae60;
}
```

#### • 按钮样式：

- 默认绿色背景，鼠标悬停时更深的绿色。
- 设置白色字体和圆角边框，提供视觉美感。

## 2. HTML 结构

html

```
<div class="login">
  <h2>后台登录</h2>
  <label for="username">用户名:</label>
  <input type="text" name="username" id="username" class="user">
  <label for="password">密码:</label>
  <input type="password" name="password" id="password" class="pass">
  <button>登录</button>
</div>
```

- 定义一个 `.login` 容器，用于承载登录框。
- 包含以下控件：
  - 用户名输入框: `  - 密码输入框: `  - 登录按钮: `

## 3. JavaScript 交互逻辑 (jQuery + AJAX)

javascript

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $("button").click(function () {
        $.ajax({
            type: 'POST',
            url: 'login.php',
            data: {
                username: $('.user').val(),
                password: $('.pass').val()
            },
            dataType: 'json',
            success: function (data) {
                console.log(data);
                if (data['infoCode'] == 1) {
                    alert('登录成功!');
                    window.open('admin.html');
                } else {
                    alert('登录失败!');
                }
            }
        });
    });
</script>
```

## 逻辑解析

### 1. 监听按钮点击事件:

javascript

```
$("button").click(function () { ... });
```

- 使用 jQuery 选择器绑定按钮的 `click` 事件。

### 2. 发送 AJAX 请求:

javascript

```
$.ajax({
    type: 'POST',
    url: 'login.php',
    data: {
        username: $('.user').val(),
        password: $('.pass').val()
    },
    dataType: 'json',
    success: function (data) { ... }
});
```

- `type: 'POST'`: 使用 `POST` 方法向服务器发送请求。
- `url: 'login.php'`: 登录请求发送到 `login.php`。
- `data`: 将用户名和密码数据以键值对形式传递：
  - `username`: 从输入框 `user` 中获取的值。
  - `password`: 从输入框 `pass` 中获取的值。
- `dataType: 'json'`: 指定服务器返回的数据格式为 JSON。

### 3. 处理服务器响应:

javascript

```
success: function (data) {
    console.log(data);
    if (data['infoCode'] == 1) {
        alert('登录成功!');
        window.open('admin.html');
    } else {
        alert('登录失败!');
    }
}
```

- `data`: 服务器返回的 JSON 数据。
- 根据返回的 `infoCode` 值判断登录是否成功：
  - `1`: 登录成功，弹出提示，并打开 `admin.html` 页面。
  - 其他值: 登录失败，弹出失败提示。

### 4. 后端接口 (login.php) 示例

php

```
<?php
header('Content-Type: application/json');

// 模拟用户名和密码
$correctUsername = 'admin';
$correctPassword = '123456';

$username = $_POST['username'];
$password = $_POST['password'];

if ($username == $correctUsername && $password == $correctPassword) {
    echo json_encode(['infoCode' => 1, 'message' => '登录成功']);
} else {
    echo json_encode(['infoCode' => 0, 'message' => '登录失败']);
}
?>
```

- 后端接收 POST 数据并校验用户名和密码。
- 返回 JSON 数据，供前端根据 `infoCode` 处理逻辑。

### 5. 总结

- 功能: 实现了登录界面与后台交互的完整流程。
- 优点:
  - 样式简洁美观。
  - 前后端分离，数据通过 AJAX 交互。
- 改进建议:
  - 增加前端表单验证（如非空验证）。
  - 使用 HTTPS 加密传输敏感数据。

- 后端增强安全性（如密码加密存储和 SQL 注入防护）。



```
<?php

$name=$_FILES['file_upload']['name'];
$type=$_FILES['file_upload']['type'];
$size=$_FILES['file_upload']['size'];
$tmp_name=$_FILES['file_upload']['tmp_name'];
$error=$_FILES['file_upload']['error'];

echo $name."<br>";
echo $type."<br>";
echo $size."<br>";
echo $tmp_name."<br>";
echo $error."<br>";
```



这段 PHP 代码用于处理通过 HTML 表单上传的文件，并提取上传文件的相关信息。以下是代码的详细解析：

## 1. 文件上传的基础知识

当用户通过 HTML 表单上传文件时，文件信息会存储在超级全局变量 `\$\_FILES` 中。`\$\_FILES` 是一个多维数组，每个文件的详细信息包括：

- `name`：文件的原始名称。
- `type`：文件的 MIME 类型。
- `size`：文件的大小（以字节为单位）。
- `tmp\_name`：文件上传到服务器时的临时存储路径。
- `error`：错误代码，指示上传过程中是否发生了问题。

## 2. 代码解析

### 定义变量

```
php

$name = $_FILES['file_upload']['name'];
$type = $_FILES['file_upload']['type'];
$size = $_FILES['file_upload']['size'];
$tmp_name = $_FILES['file_upload']['tmp_name'];
$error = $_FILES['file_upload']['error'];
```

- `\$\_FILES['file\_upload']`：
  - `file\_upload` 是文件上传表单中 `input` 的 `name` 属性值。
  - 通过 `\$\_FILES['file\_upload']` 获取该文件的所有信息。

- 每个变量的意义：
  - ``$name``: 获取文件的原始名称。
  - ``$type``: 获取文件的 MIME 类型，例如 ``image/jpeg``、``application/pdf``。
  - ``$size``: 获取文件大小，以字节为单位。
  - ``$tmp_name``: 文件存储在服务器上的临时路径。
  - ``$error``: 上传过程中产生的错误代码。

## 输出文件信息

php

```
echo $name."<br>";
echo $type."<br>";
echo $size."<br>";
echo $tmp_name."<br>";
echo $error."<br>";
```

- 使用 ``echo`` 输出文件的基本信息：
  - 文件名称 (``$name``)。
  - 文件类型 (``$type``)。
  - 文件大小 (``$size``)。
  - 文件的临时存储路径 (``$tmp_name``)。
  - 错误代码 (``$error``)。

## 3. 上传错误代码

``$error`` 的可能值和含义如下：

- ``0``: 上传成功 (``UPLOAD_ERR_OK``)。
- ``1``: 文件大小超过了 ``php.ini`` 中的 ``upload_max_filesize`` 限制 (``UPLOAD_ERR_INI_SIZE``)。
- ``2``: 文件大小超过了表单中的 ``MAX_FILE_SIZE`` 限制 (``UPLOAD_ERR_FORM_SIZE``)。
- ``3``: 文件部分上传 (``UPLOAD_ERR_PARTIAL``)。
- ``4``: 没有文件上传 (``UPLOAD_ERR_NO_FILE``)。
- ``6``: 缺少临时文件夹 (``UPLOAD_ERR_NO_TMP_DIR``)。
- ``7``: 无法写入文件到磁盘 (``UPLOAD_ERR_CANT_WRITE``)。
- ``8``: 上传被扩展程序中止 (``UPLOAD_ERR_EXTENSION``)。

## 4. 文件上传的 HTML 表单示例

要测试上述 PHP 代码，可以使用以下 HTML 表单：

html

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



```
</head>
<body>
  <form action="upload.php" method="post" enctype="multipart/form-data">
    <label for="file_upload">选择文件上传:</label>
    <input type="file" name="file_upload" id="file_upload">
    <button type="submit">上传</button>
  </form>
</body>
</html>
```

## 表单细节：

- `action="upload.php"`: 指定表单提交的目标 PHP 文件路径。
- `method="post"`: 必须使用 `POST` 方法提交文件。
- `enctype="multipart/form-data"`: 指定表单的编码类型，确保能够上传文件。

## 5. 常见改进

### 1. 文件大小限制

- 在 PHP 脚本中添加对文件大小的限制：

```
php

if ($size > 1048576) { // 限制文件大小为 1MB
    echo "文件过大，请上传小于 1MB 的文件。";
    exit;
}
```

### 2. 文件类型限制

- 限制允许的文件类型：

```
php

$allowed_types = ['image/jpeg', 'image/png', 'application/pdf'];
if (!in_array($type, $allowed_types)) {
    echo "不支持的文件类型！";
    exit;
}
```

### 3. 移动文件到目标目录

- 使用 `move\_uploaded\_file()` 将文件从临时路径移动到指定目录：

```
php

$upload_dir = 'uploads/';
$destination = $upload_dir . basename($name);
if (move_uploaded_file($tmp_name, $destination)) {
    echo "文件上传成功！存储路径：" . $destination;
} else {
```

```
    }  
    echo "文件上传失败！";  
}
```

## 6. 总结

这段代码可以用来查看上传文件的基本信息，后续可以通过添加安全性检查（如限制文件类型、大小等）并将文件保存到目标路径来完成文件上传功能。务必确保上传目录的权限设置正确，并对用户上传的文件进行验证以防止安全问题（如代码注入或恶意文件）。



```
<body>  
<form action="file.php" method="post" enctype="multipart/form-data">  
    选择文件：<input type="file" name="file_upload" onchange="checkFile(this.value)">  
    <input type="submit" value="上传">  
</form>  
  
<script src="jquery-3.7.1.min.js"></script>  
<script>  
    function checkFile(filename) {  
        //x.jpg 白名单校验  
        var exts=['png','jpg','jpeg','gif','bmp','mpeg'];  
        var index=filename.lastIndexOf(".");  
        var ext = filename.substr(index+1);  
        for (var i=0;i<exts.length;i++){  
            if(ext==exts[i]){  
                alert('文件正确！');  
                break;  
                //正确上传文件逻辑代码在下面  
            }else{  
                alert('非法文件！');  
                window.location.replace("file.html")  
                break;  
            }  
        }  
    }  
</script>
```



这段代码是一个文件上传表单，包含了一个前端校验逻辑，用于检查用户选择的文件类型是否在允许的扩展名白名单内。如果文件类型合法，则提示“文件正确”；否则提示“非法文件”，并跳转到 `file.html` 页面。以下是详细解析：

## 1. HTML 部分

html

```
<form action="file.php" method="post" enctype="multipart/form-data">
  选择文件:<input type="file" name="file_upload" onchange="checkFile(this.value)">
  <input type="submit" value="上传">
</form>
```

- ``action="file.php"``:
  - 文件上传表单的目标地址, 指定为 ``file.php``, 即后端处理上传的 PHP 文件。
- ``method="post"``:
  - 使用 ``POST`` 方法提交表单数据, 因为文件上传必须通过 POST 方法。
- ``enctype="multipart/form-data"``:
  - 指定表单的编码类型, 确保可以上传文件。
- ``<input type="file">``:
  - 文件选择控件, 用户通过此控件选择文件。
- ``onchange="checkFile(this.value)"``:
  - 在文件选择后调用 ``checkFile`` 函数, 实时校验文件类型。

## 2. JavaScript 校验逻辑

javascript

```
function checkFile(filename) {
  // x.jpg 白名单校验
  var exts = ['png', 'jpg', 'jpeg', 'gif', 'bmp', 'mpeg'];
  var index = filename.lastIndexOf(".");
  var ext = filename.substr(index + 1);
  for (var i = 0; i < exts.length; i++) {
    if (ext == exts[i]) {
      alert('文件正确!');
      break;
      // 正确上传文件逻辑代码在下面
    } else {
      alert('非法文件!');
      window.location.replace("file.html");
      break;
    }
  }
}
```

### 功能和逻辑解析

#### 1. 获取文件扩展名:

javascript

```
var index = filename.lastIndexOf(".");
var ext = filename.substr(index + 1);
```

- 通过 `lastIndexOf(".")` 找到最后一个点号的位置。
- 使用 `substr(index + 1)` 获取扩展名部分。

## 2. 白名单校验:

javascript

```
var exts = ['png', 'jpg', 'jpeg', 'gif', 'bmp', 'mpeg'];
```

- 定义允许上传的文件扩展名白名单。

## 3. 遍历检查:

javascript

```
for (var i = 0; i < exts.length; i++) {
    if (ext == exts[i]) {
        alert('文件正确!');
        break;
    } else {
        alert('非法文件!');
        window.location.replace("file.html");
        break;
    }
}
```

- 遍历白名单，检查用户选择的文件扩展名是否符合条件：
  - 如果扩展名在白名单中，提示“文件正确”并允许上传。
  - 如果扩展名不符合，提示“非法文件”并跳转到 `file.html` 页面。

## 存在的问题

### 1. 校验逻辑错误:

- 在循环中无论文件是否匹配，都会触发 `break`，导致校验逻辑无法正确运行。
- 修复方法：

javascript

```
function checkFile(filename) {
    var exts = ['png', 'jpg', 'jpeg', 'gif', 'bmp', 'mpeg'];
    var index = filename.lastIndexOf(".");
    var ext = filename.substr(index + 1).toLowerCase(); // 转小写，防止大小写问题

    if (exts.includes(ext)) {
        alert('文件正确!');
    } else {
        alert('非法文件!');
        window.location.replace("file.html");
    }
}
```

### 2. 跳转行为:

- 直接使用 `window.location.replace("file.html")` 强制跳转，用户体验较差。
- 可替换为清空文件输入框，避免页面跳转：

```
javascript
```

```
alert('非法文件!');
document.querySelector('input[type="file"]').value = '';
```

### 3. 改进后的代码

以下是修正后的完整代码：

```
html
```

```
<body>
<form action="file.php" method="post" enctype="multipart/form-data">
  选择文件：<input type="file" name="file_upload" onchange="checkFile(this)">
  <input type="submit" value="上传">
</form>

<script src="jquery-3.7.1.min.js"></script>
<script>
  function checkFile(input) {
    // 文件扩展名白名单
    var exts = ['png', 'jpg', 'jpeg', 'gif', 'bmp', 'mpeg'];
    var filename = input.value;
    var index = filename.lastIndexOf(".");
    var ext = filename.substr(index + 1).toLowerCase(); // 转为小写

    if (exts.includes(ext)) {
      alert('文件正确!');
    } else {
      alert('非法文件!');
      input.value = ''; // 清空选择的文件
    }
  }
</script>
```

#### 改进内容

1. 修复了文件类型校验逻辑，使用 `Array.includes()` 简化检查。
2. 文件类型不符合时，不再强制跳转页面，而是清空输入框的值。

### 4. 后端补充校验 (file.php)

尽管前端校验可以提高用户体验，但并不能确保安全性，后端需要进行二次校验：

```
php
```

```
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
  $allowed_exts = ['png', 'jpg', 'jpeg', 'gif', 'bmp', 'mpeg'];
  $filename = $_FILES['file_upload']['name'];
  $tmp_name = $_FILES['file_upload']['tmp_name'];

  // 获取文件扩展名
  $ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION));
```

```
// 校验扩展名
if (in_array($ext, $allowed_exts)) {
    $upload_dir = 'uploads/';
    $destination = $upload_dir . basename($filename);

    if (move_uploaded_file($tmp_name, $destination)) {
        echo "文件上传成功！存储路径：" . $destination;
    } else {
        echo "文件上传失败！";
    }
} else {
    echo "非法文件类型！";
}
}
?>
```

## 5. 总结

- **前端校验:** 增强用户体验，减少无效请求。
- **后端校验:** 确保安全性，避免绕过前端逻辑上传非法文件。
- **建议:** 始终验证文件类型、大小，并对上传文件进行必要的安全处理（如文件名重命名、防止代码注入等）。



```
<?php

$user=@$_POST['username'];
$pass=@$_POST['password'];
//真实情况需要在数据库获取
$success=array('msg'=>'ok');
if($user=='xiaodi' && $pass=='xiaodisec'){
    $success['infoCode']=1;
}else{
    $success['infoCode']=0;
}
echo json_encode($success);
```



这段 PHP 代码实现了一个简单的用户登录验证逻辑，并返回 JSON 格式的结果给客户端。以下是代码的详细解析：

### 1. 接收表单数据

```
php

$user = @$_POST['username'];
$pass = @$_POST['password'];
```

- 使用 `\$\_POST` 超全局数组接收通过 POST 请求传递的 `username` 和 `password`。
- `@` 前缀用于忽略可能出现的警告，例如当 `\$\_POST['username']` 或 `\$\_POST['password']` 未设置时。

## 2. 定义响应结构

```
php

$success = array('msg' => 'ok');
```

- 初始化一个数组 ` \$success `，表示响应的基础结构，包含：
  - `msg`：响应消息，固定值为 ` "ok" `。

## 3. 验证用户名和密码

```
php

if ($user == 'xiaodi' && $pass == 'xiaodisec') {
    $success['infoCode'] = 1;
} else {
    $success['infoCode'] = 0;
}
```

- 模拟验证逻辑：
  - 如果用户名为 ` "xiaodi" ` 且密码为 ` "xiaodisec" `，设置 ` \$success['infoCode'] ` 为 ` 1 `，表示登录成功。
  - 否则，将 ` \$success['infoCode'] ` 设置为 ` 0 `，表示登录失败。

### 真实场景：

- 用户名和密码通常存储在数据库中。验证过程需要从数据库检索用户信息，并对密码进行校验（通常采用加密验证，如哈希）。

## 4. 返回 JSON 响应

```
php

echo json_encode($success);
```

- 使用 ` json\_encode() ` 函数将数组 ` \$success ` 转换为 JSON 格式的字符串，并返回给客户端。
- 示例返回：
  - 登录成功时： ` {"msg":"ok","infoCode":1} `
  - 登录失败时： ` {"msg":"ok","infoCode":0} `

## 5. 示例客户端代码

假设前端使用 AJAX 请求登录，代码示例如下：

```
html

<form id="loginForm">
  用户名: <input type="text" id="username" name="username">
  密码: <input type="password" id="password" name="password">
  <button type="button" id="loginBtn">登录</button>
</form>

<script src="jquery-3.7.1.min.js"></script>
<script>
  $("#loginBtn").click(function () {
    var username = $("#username").val();
    var password = $("#password").val();

    $.ajax({
      type: "POST",
      url: "login.php", // PHP 文件路径
      data: { username: username, password: password },
      dataType: "json",
      success: function (response) {
        if (response.infoCode === 1) {
          alert("登录成功!");
        } else {
          alert("登录失败, 用户名或密码错误!");
        }
      },
      error: function () {
        alert("登录请求失败!");
      }
    });
  });
</script>
```

## 6. 代码改进建议

### 1. 输入验证:

- 在接收用户输入时，应验证输入是否为空并进行安全过滤，防止 SQL 注入或 XSS 攻击。

php

```
$user = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
$pass = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_STRING);
```

### 2. 密码加密:

- 在真实场景中，密码应使用哈希算法存储（如 `password\_hash()`）。
- 验证时使用 `password\_verify()`。

php

```
$stored_hash = '$2y$10$abcdefghijklmnopqrstuvwx...'; // 从数据库中获取
if (password_verify($pass, $stored_hash)) {
```



```
$success['infoCode'] = 1;
} else {
    $success['infoCode'] = 0;
}
```

### 3. 数据库查询:

- 使用 PDO 或 MySQLi 安全地查询数据库：

php

```
$pdo = new PDO('mysql:host=localhost;dbname=test', 'root', '');
$stmt = $pdo->prepare("SELECT password FROM users WHERE username = :username");
$stmt->execute(['username' => $user]);
$result = $stmt->fetch(PDO::FETCH_ASSOC);

if ($result && password_verify($pass, $result['password'])) {
    $success['infoCode'] = 1;
} else {
    $success['infoCode'] = 0;
}
```

### 4. 添加跨域保护:

- 如果前端和后端不在同一个域名下，需处理跨域问题：

php

```
header('Access-Control-Allow-Origin: *');
header('Content-Type: application/json');
```

## 7. 完整改进后的代码

php

```
<?php
header('Content-Type: application/json');

$user = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
$pass = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_STRING);

$success = array('msg' => 'ok');

try {
    $pdo = new PDO('mysql:host=localhost;dbname=test', 'root', '');
    $stmt = $pdo->prepare("SELECT password FROM users WHERE username = :username");
    $stmt->execute(['username' => $user]);
    $result = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($result && password_verify($pass, $result['password'])) {
        $success['infoCode'] = 1; // 登录成功
    } else {
        $success['infoCode'] = 0; // 登录失败
    }
} catch (Exception $e) {
    $success = array('msg' => 'error', 'error' => $e->getMessage());
}
```

```
echo json_encode($success);
```

## 8. 总结

- 本代码展示了一个基本的登录逻辑，但实际场景中应引入更强的安全机制，包括：
  - 输入过滤与验证。
  - 数据库操作安全（使用参数化查询）。
  - 密码加密与验证。
  - 返回结果规范化与详细化。
- 按照以上建议进行改进后，登录系统会更加安全和可靠。