文件下载实际情景

浏览器原生下载

最简单的方式就是用一个 <a> 标签：

```
<a href="http://example.com/file.pdf" download="myFile.pdf">下载文件</a>
```

- `href` 指向文件地址。
- `download` 属性指定下载时保存的文件名。
- 这种方式完全由浏览器接管，不需要写 JavaScript。

简单，不需要额外代码。

适合 **静态文件、小文件。**

缺点：**无法控制进度 / 无法自定义请求头 / 无法做权限控制。**

而且这个请求是且只能是GET请求

为什么必须是 GET 请求?

1. `<a>` **标签的机制：**
   - 当用户点击 `<a href="http://example.com/file.pdf" download="myFile.pdf">` 时，浏览器会向 `href` 指定的 URL 发送一个 **HTTP GET 请求。**
   - `download` 属性只是告诉浏览器，在收到服务器返回的数据后，**将其作为文件下载到本地，**而不是尝试在浏览器中显示（例如，PDF 可能默认在浏览器中预览）。
   - 浏览器的 `<a>` 标签只支持 GET 请求，因为它本质上是导航或资源请求的行为，无法通过 `<a>` 标签直接发起 POST 或其他类型的请求。

2. **服务器响应：**

- 服务器接收到 GET 请求后，返回文件内容（通常是二进制数据）以及适当的响应头，例如：
  - `Content-Type: application/octet-stream`（通用二进制文件）或具体类型如 `application/pdf`。
  - `Content-Disposition: attachment; filename="myFile.pdf"`（建议浏览器下载文件并指定文件名）。
- 如果服务器不支持 GET 请求或 URL 无效，下载会失败。

JS触发下载

有时候文件 URL 是动态的，或者你要在点击按钮时触发下载，就可以用 JS 来模拟点击 `<a>`：

js 复制代码

```
function download(url, filename) {
  const a = document.createElement('a')
  a.href = url
  a.download = filename
  a.click()
}
```

设置URL
以及文件名字

- 可以配合后端返回的临时链接、鉴权后的下载地址。
- 原理和 `<a>` 标签相同。

要让下载的文件名使用后端返回的 Content-Disposition 头中的 filename，而不是前端硬编码的 filename，你需要通过 JavaScript（如 Fetch API）获取服务器响应头中的 Content-Disposition，然后提取文件名。因为 标签的 download 属性无法直接读取服务器的响应头。

问题分析

- 当你直接用 `<a href="url" download="filename">` 时，`download` 属性指定的文件名只在同源请求或服务器未明确指定 `Content-Disposition` 时有效。
- 如果服务器返回了 `Content-Disposition: attachment; filename="myFile.pdf"`，浏览器会优先使用这个文件名，忽略 `<a>` 标签的 `download` 属性（在跨域情况下尤其如此）。
- 你的函数 `download(url, filename)` 目前直接设置 `a.download = filename`，无法动态获取服务器的 `Content-Disposition`。

为什么浏览器优先使用 `Content-Disposition`？

- **HTTP 协议规范**：`Content-Disposition` 头是服务器明确指定文件下载行为的机制，包含文件名信息。浏览器会优先遵循服务器的指令，因为它被认为是更权威的来源。
- **跨域限制**：在跨域请求中，浏览器的安全策略（如 CORS）会导致 `<a>` 标签的 `download` 属性被忽略，服务器的 `Content-Disposition` 中的 `filename` 会直接生效。
- **同源情况**：如果是同源请求，`download` 属性可能会覆盖 `Content-Disposition` 的文件名（取决于浏览器实现），但现代浏览器通常仍优先尊重服务器的设置。

服务器未设置 Content-Disposition：

- 如果服务器没有返回 `Content-Disposition` 头，浏览器会尝试从 URL 提取文件名（例如 `url.split('/').pop()`），或者使用 `<a>` 标签的 `download` 属性指定的文件名。
- 为了确保文件名可控，你可以在代码中设置默认文件名：

```javascript
function download(url, filename) {
  const a = document.createElement('a');
  a.href = url;
  a.download = filename || url.split('/').pop() || 'download'; //
  document.body.appendChild(a);
  a.click();
  document.body.removeChild(a);
}
```

Axios控制

```
import axios from "axios"

async function downloadFile(url, filename) {
  const res = await axios.get(url, { responseType: "blob" })
  const blob = new Blob([res.data])
  const link = document.createElement("a")
  link.href = URL.createObjectURL(blob)
  link.download = filename
  link.click()
  URL.revokeObjectURL(link.href)
}
```

特点：

- 支持自定义请求头（如 `Authorization`）。
- 可以拿到响应头（**比如 `Content-Disposition` 获取真实文件名**）。
- 适合 **有权限控制的文件下载。**

```
import axios from "axios"
                                    期望blob类型，不然会默认成JSON
async function downloadFile(url, filename) {
  const res = await axios.get(url, { responseType: "blob" })
  const blob = new Blob([res.data])
  const link = document.createElement("a")
  link.href = URL.createObjectURL(blob)
  link.download = filename
  link.click()
  URL.revokeObjectURL(link.href)
}
```

`res.data` **拿到**的是一个 **Blob 对象**（二进制数据的文件块）。
但是 **浏览器的** `<a>` **标签** 不能直接**下载一个 Blob 对象**，它需要一个 **文件地址。**

为什么要用 `URL.createObjectURL`**?**

`URL.createObjectURL(blob)` **会给内存里的二进制数据（Blob）**分配一个临时的 **本地 URL 地址**

```
blob:http://localhost:3000/1a2b3c-xxxx-xxxx
```

这个地址浏览器就能识别了，就像一个真正的文件路径
然后你把它赋值给 `<a>` **的** `href`**，浏览器就能下载。**

```
link.href = URL.createObjectURL(blob)
link.download = filename
```

为什么要 `URL.revokeObjectURL`?

因为 `createObjectURL` 会在内存中占用资源。
如果你不释放，下载多了可能导致内存泄漏。
所以下载完后用：

```
URL.revokeObjectURL(link.href)
```

告诉浏览器：这个临时地址不用了，可以释放掉。

👉 类比一下：

- `Blob` = 文件内容（存在内存里）。
- `URL.createObjectURL` = 给这块内容贴个“临时门牌号”。
- `<a href="临时门牌号">` = 浏览器就能找到这块内容并下载。