HTTP长轮询

长轮询（Long Polling）是一种让客户端和服务器保持通信的技术，特别适合需要**实时更新的场景**，比如聊天应用、股票价格变动等。

传统轮询（短轮询）

在长轮询出现之前，前端为了获取实时数据，最常见的做法是**短轮询。**

工作流程：

1. **客户端发起请求：** 浏览器通过 **`XMLHttpRequest` 或 `fetch` API** 向服务器的一个特定接口（例如 `/api/notifications`）发送 GET 请求。
2. **服务器处理请求：** 服务器立即处理这个请求，检查是否有新数据。
3. **服务器返回响应：**

   如果有新数据，服务器将数据封装在响应体中并返回

   如果没有新数据，服务器会返回一个空的数据结构（例如 `{ "data": null }`）或者一个状态码。
4. **客户端定时重复：** 客户端接收到响应后，使用 **`setTimeout` 或 `setInterval` 函数，等待一段固定的时间（比如 3 秒）**，然后再次发起一个新的请求。

在技术上，**这对应的是你的前端代码每隔几秒钟（比如3秒）就向服务器发送一个HTTP请求，询问是否有新数据。**

服务器从来就「不会主动」给客户端发一次消息，必须等待客户端去请求

这样就会造成一些**缺陷**：

效率低： 大部分请求都是无效的，因为没有新数据。这会浪费服务器资源和网络带宽。

延迟： 如果新数据刚好在你两次请求的间隔中间产生，你需要等到下一个请求发出后才能拿到，这就会有延迟。

示例代码

```
function pollForData() {
  fetch('/api/notifications')
    .then(response => response.json())
    .then(data => {
      if (data && data.newNotifications) {
        // 处理新数据
        console.log("新通知来了:", data.newNotifications);
      }
      // 无论有没有新数据，都定时发起下一个请求
      setTimeout(pollForData, 3000);
    })
    .catch(error => {
      console.error("请求失败:", error);
      setTimeout(pollForData, 3000);
    });
}

pollForData(); // 启动轮询
```

长轮询（Long Polling）

长轮询就是为了解决短轮询的低效和高延迟问题而诞生的。

工作流程：

1. **客户端发起请求：** 浏览器向服务器**发送一个 HTTP 请求，与短轮询不同的是，这个请求的预期处理时间会更长。**
2. **服务器挂起连接：** 服务器接收到请求后，**不立即返回响应。它会把这个 HTTP 连接保持在打开状态**，并让其进入一个“挂起”状态。在后端实现中，**服务器通常会把这个请求放入一个等待队列。**
3. **数据到达：** 当服务器**有新数据**（例如，新的聊天消息写入数据库或另一个系统事件发生）时，**它会从等待队列中找到相应的挂起连接。**
4. **服务器返回响应：** 服务器将新数据封装成 HTTP 响应，通过这个挂起的连接发送给客户端，然后**立即关闭这个连接。**
5. **客户端重新发起请求：** 客户端接收到响应并处理完新数据后，会**立即发起一个新的长轮询请求**，重复上述步骤。

示例代码

优点：

- **实时性好：** 数据一旦产生，**服务器会立即响应**，客户端几乎可以实时收到。
- **节省资源：** 大幅**减少了无效的 HTTP 请求**，客户端和服务器的压力都小很多。

缺点：

- **服务器连接数开销： 每个挂起的长轮询连接都会占用服务器的资源。**当连接数非常庞大时（例如百万级并发），服务器会面临很大的压力。
- **请求头开销：** 尽管请求次数少，每次新请求依然会带上完整的 HTTP 请求头，这在网络层面上会有一定的冗余开销。