30th November 2019

Long polling

Long polling is the simplest way of having persistent connection with server, that doesn't use any specific protocol like WebSocket or Server Side Events.

Being very easy to implement, it's also good enough in a lot of cases.

Regular Polling

The simplest way to get new information from the server is periodic polling. That is, regular requests to the server: "Hello, I'm here, do you have any information for me?". For example, once in 10 seconds.

In response, the server first takes a notice to itself that the client is online, and second – sends a packet of messages it got till that moment.

That works, but there are downsides:

1. Messages are passed with a delay up to 10 seconds (between requests).
2. Even if there are no messages, the server is bombed with requests every 10 seconds, even if the user switched somewhere else or is asleep. That's quite a load to handle, speaking performance-wise.

So, if we're talking about a very small service, the approach may be viable, but generally, it needs an improvement.

Long polling

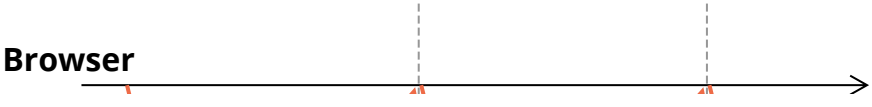
So-called "long polling" is a much better way to poll the server.

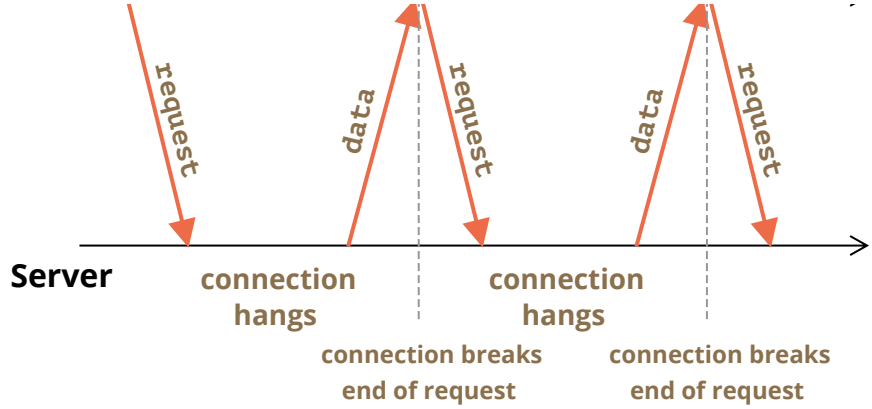
It's also very easy to implement, and delivers messages without delays.

The flow:

1. A request is sent to the server.
2. The server doesn't close the connection until it has a message to send.
3. When a message appears – the server responds to the request with it.
4. The browser makes a new request immediately.

The situation when the browser sent a request and has a pending connection with the server, is standard for this method. Only when a message is delivered, the connection is reestablished.





If the connection is lost, because of, say, a network error, the browser immediately sends a new request.

A sketch of client-side `subscribe` function that makes long requests:

```
async function subscribe() {
  let response = await fetch("/subscribe");

  if (response.status == 502) {
    // Status 502 is a connection timeout error,
    // may happen when the connection was pending for too long,
    // and the remote server or a proxy closed it
    // let's reconnect
    await subscribe();
  } else if (response.status != 200) {
    // An error - let's show it
    showMessage(response.statusText);
    // Reconnect in one second
    await new Promise(resolve => setTimeout(resolve, 1000));
    await subscribe();
  } else {
    // Get and show the message
    let message = await response.text();
    showMessage(message);
    // Call subscribe() again to get the next message
    await subscribe();
  }
}

subscribe();
```

As you can see, `subscribe` function makes a fetch, then waits for the response, handles it and calls itself again.

Server should be ok with many pending connections

The server architecture must be able to work with many pending connections.

Certain server architectures run a process per connect. For many connections there will be as many processes, and each process takes a lot of memory. So many connections just consume it all.

That's often the case for backends written in PHP, Ruby languages, but technically isn't a language, but rather implementation issue. Most modern language allow to implement a proper backend, but some of them make it easier than the other.

Backends written using Node.js usually don't have such problems.

Demo: a chat

Here's a demo chat, you can also download it and run locally (if you're familiar with Node.js and can install modules):

Result browser.js server.js index.html

All visitors of this page will see messages of each other.

Send

Browser code is in `browser.js` .

Area of usage

Long polling works great in situations when messages are rare.

If messages come very often, then the chart of requesting-receiving messages, painted above, becomes saw-like.

Every message is a separate request, supplied with headers, authentication overhead, and so on.

So, in this case, another method is preferred, such as Websocket or Server Sent Events.

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)