WebSocket: How to automatically reconnect after it dies

Asked 11 years, 1 month ago Modified 1 year, 2 months ago Viewed 273k times

198

```
var ws = new WebSocket('ws://localhost:8080');
ws.onopen = function () {
  ws.send(JSON.stringify({
      .... some message the I must send when I connect ....
  }));

};

ws.onmessage = function (e) {
  console.log('Got a message')
  console.log(e.data);
};

ws.onclose = function(e) {
  console.log('socket closed try again');

}

ws.onerror = function(err) {
  console.error(err)
};
```

When I first connect to the socket, I must first send a message to the server to authenticate myself and subscribe to channels.

The problem I have is that sometimes the socket server is unreliable and that triggers the `onerror` and `onclose` events of the `'ws'` object.

Question: What is a good design pattern that would allow me, whenever the socket closes or encounters an error, wait for 10 seconds and then reconnect to the socket server (and resend the initial message to the server)

javascript websocket

Share Improve this question Follow

edited Apr 19, 2014 at 22:17

asked Mar 16, 2014 at 1:04
samol
20.8k 33 93 132

2 Possible duplicate of Reconnection of Client when server reboots in WebSocket – zoran404 Jun 1, 2018 at 7:08

clear-ws handles reconnects automatically (if you provide not null `reconnect.delayMs`). – grabantot Apr 25, 2022 at 10:43

12 Answers

Sorted by: Highest score (default)

339

Here is what I ended up with. It works for my purposes.

```
function connect() {
  var ws = new WebSocket('ws://localhost:8080');
  ws.onopen = function() {
    // subscribe to some channels
    ws.send(JSON.stringify({
        //.... some message the I must send when I connect ....
    }));
  };

  ws.onmessage = function(e) {
    console.log('Message:', e.data);
  };

  ws.onclose = function(e) {
    console.log('Socket is closed. Reconnect will be attempted in 1 second.',
e.reason);
    setTimeout(function() {
      connect();
    }, 1000);
  };

  ws.onerror = function(err) {
    console.error('Socket encountered error: ', err.message, 'Closing socket');
    ws.close();
  };
}

connect();
```

Share Improve this answer Follow

edited Jul 27, 2017 at 12:55
halfer
20.4k 19 109 203

answered Apr 19, 2014 at 22:18
samol
20.8k 33 93 132

7 Does that reconnect to the same websocket it was connected to before? Because I am using websocket id to send messages, but if it has new websocket id it would be hard to send messages to particular system. – Vishnu Y S May 11, 2017 at 6:28

33 @AlexanderDunaev, the time out is mainly added as an easy way to avoid too aggressive reconnect when the server is not availble, i.e. broken network, or shutdown of local debug server. But in general, I think an immediate reconnect followed by exponentially growing wait time for reconnect would be slightly better choice than fixed 1sec wait. – user658991 Jun 27, 2017 at 1:17

31 What happens to the websocket instance when the connection is closed. Is it garbage collected, or does the browser build up a pile of unused objects? – knobo Jun 21, 2018 at 7:05

21 setTimeout(connect,1000) is a more concise, resource efficient way of delaying the reconnect. also consider using setTimeout (connect ,Math.min(10000,timeout+=timeout)), resetting timeout to 250 before first connect and after every successful connect. this way error conditions during connect will add a backoff, but will quickly reconnect if it is a one time error situation - 250,500,1000,2000,4000,8000,10000,10000 msec delays is less agressive, but faster responding than 1000,1000,1000 msec – unsynchronized May 8, 2019 at 11:56

11 the issue i see with this code is that if the connection is closed and we try to open the connection again, and it fails, then we will never issue a retry. – Michael Connor Jul 31, 2019 at 19:36

12

This worked for me with `setInterval` , because client connection can be lost.

```
ngOnInit(): void {
    if (window.location.protocol.includes('https')) {
        this.protocol = 'wss';
    }

    this.listenChanges();
}


listenChanges(): void {
    this.socket = new
WebSocket(`${this.protocol}://${window.location.host}/v1.0/your/url`);

    this.socket.onmessage = (event): void => {
        // your subscription stuff
        this.store.dispatch(someAction);
    };

    this.socket.onerror = (): void => {
        this.socket.close();
    };


    this.socket.onopen = (): void => {
        clearInterval(this.timerId);

        this.socket.onclose = (): void => {
            this.timerId = setInterval(() => {
                this.listenChanges();
            }, 10000);
        };
    };
}
```

Don't forget to call `clearInterval` when the socket has been opened.

Share Improve this answer Follow

edited Jun 19, 2020 at 11:28
Jakye
6,665 3 23 40

answered Jun 19, 2020 at 8:18
Bulat
145 1 4

9

This isn't explicitly a react question but here is a react style answer:

TLDR: You can use `setInterval` to periodically check the websocket connection status and try to re-connect if the connection is closed. https://developer.mozilla.org/en-US/docs/Web/API/WebSocket/readyState

```
class TestComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};

    this.connect = this.connect.bind(this);
  }

  componentDidMount() {
    this.interval = setInterval(this.connect, 1000);
  }

  componentWillUnmount() {
    if (this.ws) this.ws.close();
    if (this.interval) clearInterval(this.interval);
  }

  connect() {
    // https://developer.mozilla.org/en-US/docs/Web/API/WebSocket/readyState
    if (this.ws === undefined || (this.ws && this.ws.readyState === 3)) {
      this.ws = new WebSocket(`ws://localhost:8080`);

      this.ws.onmessage = (e) => {
        console.log(JSON.parse(e.data));
      };
    }
  }

  render() {
    return <div>Hey!</div>;
  }
}
```

Share Improve this answer Follow edited May 12, 2021 at 14:24 answered May 12, 2021 at 14:18

Glen Thompson
10.1k 4 60 53

5

I have been struggling with this the last few weeks and decided to create a package called SuperSocket - in case it helps anyone! It should work as a drop in replacement for the native WebSocket. Existing packages I found seemed unmaintained.

SuperSocket sits on top of the existing WebSocket implementation and, amongst other features, make sure it reconnects until it successfully does. Of course you can setup a max retry to avoid infinite loop and unnecessary CPU load :)

```
//native implementation
var ws = new WebSocket('ws://localhost:8080');

//drop in replacement, embedding reconnect strategies
var ws = new SuperSocket('ws://localhost:8080');
```

Share Improve this answer Follow answered Aug 19, 2023 at 19:51

BaptisteC
116 1 2

Nice! Looking forward to use this package! Question, is the port 8080 mandatory? – LukyVj Aug 24, 2023 at 8:24

1 Thank you for the feedback! No, you can set any ports you want - and HTTP, TCP and WS can actually use the same port! – BaptisteC Aug 24, 2023 at 8:26

is this supposed to be used on node.js or on the browser? i see a ws library reference in the source code, hence asking – PirateApp Mar 5 at 7:48

4

I found that this package https://github.com/pladaria/reconnecting-websocket can solve the reconnection issues for Websocket connections. And it has the list of configurable options, one of them is `reconnectionDelayGrowFactor` which determines how fast the reconnection delay grows.

Share Improve this answer Follow

answered Mar 12, 2021 at 6:30
Dmytro Olefyrenko
379 1 6

1 does this reconnect with subscriptions or needs to manually subscribe the channel again?
– Mr. Kenneth Nov 22, 2022 at 0:55

3

using async-await if socket closed or any error occurred on the server the client will try to connect automatically every 5 sec forever have a look to my answer

Share Improve this answer Follow

edited Dec 21, 2021 at 12:34

answered Apr 14, 2021 at 23:50
Mohamed Farouk
1,229 1 16 31

2

Here's a simple version I use in my projects. It includes an incrementing wait timer for reconnects.

```
//wsURL - the string URL of the websocket
//waitTimer - the incrementing clock to use if no connection made
//waitSeed - used to reset the waitTimer back to default on a successful
connection
//multiplier - how quickly you want the timer to grow on each unsuccessful
connection attempt

const openSocket = (wsURL, waitTimer, waitSeed, multiplier) =>{
  let ws = new WebSocket(wsURL);
  console.log(`trying to connect to: ${ws.url}`);

  ws.onopen = () => {
      console.log(`connection open to: ${ws.url}`);
      waitTimer = waitSeed; //reset the waitTimer if the connection is made

      ws.onclose = () => {
        console.log(`connection closed to: ${ws.url}`);
        openSocket(ws.url, waitTimer, waitSeed, multiplier);
      };

      ws.onmessage = (message) => {
        //do something with messge...
      };
  };

  ws.onerror = () => {
    //increaese the wait timer if not connected, but stop at a max of 2n-1 the
check time
```

```
    if(waitTimer < 60000) waitTimer = waitTimer * multiplier;
    console.log(`error opening connection ${ws.url}, next attemp in :
${waitTimer/1000} seconds`);
    setTimeout(()=>{openSocket(ws.url, waitTimer, waitSeed, multiplier)},
waitTimer);
  }
}

openSocket(`ws://localhost:3000`, 1000, 1000, 2)
```

Share Improve this answer Follow

answered Jan 23, 2023 at 15:46
Trilaworks
21 2

2

ReconnectingWebSocket is a small library that addresses this by providing an API-compatible decorated WebSocket class that automatically reconnects.

Add the script to your page (e.g., via a `<script>` tag) and, as described by the README linked above:

> It is API compatible, so when you have:
>
> ```
> var ws = new WebSocket('ws://....');
> ```
>
> you can replace with:
>
> ```
> var ws = new ReconnectingWebSocket('ws://....');
> ```

Share Improve this answer Follow

edited Aug 25, 2023 at 23:05
Ryan M ♦
20.4k 34 73 82

answered Mar 1, 2022 at 10:31
anon

1 does this reconnect with subscription or needs to resubscribe to channels? – Mr. Kenneth Nov 22, 2022 at 0:54

1

UPDATED answer:

At last, (if you are not using java) I found you'd better implement your own "ping/pong" strategy. (if you are using java, please take a look at ping/pong "action type", I don't remember very clear...)

1. client sent "ping" to server every 5 seconds.
2. server should echo a "pong" to the client once it receive "ping".
3. client should reconnect server if doesn't receive "pong" in 5 seconds.

Don't rely on any third party libs.

WARNING: DO NOT use these tools: (reason: they are not reliable and not stable and works in a very limited way.)

1. check if the network is available: https://github.com/hubspot/offline
2. to re-connect: https://github.com/joewalnes/reconnecting-websocket

Share Improve this answer Follow edited Aug 11, 2021 at 9:55 answered Aug 30, 2019 at 0:59 Siwei 21.7k 7 83 102

4 The github library github.com/joewalnes/reconnecting-websocket actually works as a simple drop in for `new WebSocket()` in a simple connection. I know this answer is a bit off the mark in general, but for simplicity, using the mentioned javascript library here does work. – TechnicalChaos Feb 4, 2020 at 10:08

1 Yes you are right ! Don't use those 2 github repos. – Siwei Feb 9, 2020 at 5:45

4 Why should we not use them? The second one looks pretty useful. – Shamoon May 28, 2020 at 21:04

3 you should implement your ping/pong strategy. don't trust the open/close event . – Siwei May 29, 2020 at 11:03

1 Note that as of my writing, ReconnectingWebSocket does not support the 'binaryType' option: it seems to fall back to 'blob' 50% of the time, and the minified JS does not contain the functionality at all. So I just rolled my own. – disconnectionist Dec 22, 2020 at 15:22

0

For the react users out there...

if there is no web socket on load the web socket is created checking to insure that another web socket does not exist already. if the connection is lost a timeout function is called to try and connect to the server again every time the websocket cannot connect.

```
    const host = process.env.REACT_APP_API_ROOT.replace(/^http/, 'ws')
    let ws
    if (ws === undefined || ws.readyState === 3) {
      ws = new WebSocket(host)
    }
    const connect_socket = () => {
      ws.onmessage = function (event, isBinary) {
        console.log(event.data)
      }
      ws.onopen = (event) => {
        ws.send('hey everyone')
        saveClaims((prev) => ({ ...prev, ws }))
      }

      ws.onerror = () => {
        console.log('error')
```

```
          if (ws) {
            ws.close()
          }
        }
        ws.onclose = (event) => {
          console.log('disconnected client')
          if (ws) {
            ws.close()
            setTimeout(() => {
              ws = new WebSocket(host)
              connect_socket()
            }, 300)
          }
        }
      }
      connect_socket()
      return () => {
        ws.close()
        ws = undefined
      }
    }, [])```
```

Share Improve this answer Follow edited Feb 9, 2024 at 19:48 answered Dec 21, 2023 at 20:16

Andrew Judd
13 1 4

1 As it’s currently written, your answer is unclear. Please edit to add additional details that will help others understand how this addresses the question asked. You can find more information on how to write good answers in the help center. – Community Bot Dec 25, 2023 at 1:35

Hello, please don't post code only and add an explantation as to why you think that this is the optimal solution. People are supposed to learn from your answer, which might not occur if they just copy paste code without knowing why it should be used. – Destroy666 Dec 28, 2023 at 5:45

-1

Try this:

```
const observable = Observable.create(
  (obs: Observer<MessageEvent>) => {
    this.ws.onmessage = obs.next.bind(obs);
    this.ws.onerror = obs.error.bind(obs);
    // this.ws.onclose = obs.complete.bind(obs);
    this.ws.onclose = function () {
      window.location.reload()
    }
    return this.ws.close.bind(this.ws);
  });

const observer = {
  next: (data: Object) => {
    if (this.ws.readyState === WebSocket.OPEN) {
      this.ws.send(JSON.stringify(data));
    }
  }
};
```

and component

```
getDatas() {
let url = environment.apiwebsocket
this.webSocketService.connect(url)
  .subscribe(evt => {
    let jsonObj = JSON.parse(evt.data)
  });}
```

Share Improve this answer Follow

answered Aug 4, 2022 at 3:33

GoldStreet VN

1 2

-1

I used to have this somewhere in project:

```
let rc = new WebSocket(
    'ws://'
    + window.location.host
    + `/ws/chat/${window.seen.pk}/`
)
```

now I switched to:

```
// ws create the websocket and returns it
function autoReconnect(ws_create){
    let ws = ws_create();
    function startReconnecting(){
        let interval = setInterval(()=>{
            console.log('trying')
            ws = ws_create();
            ws.onopen = () => {
                console.log('stop');
                ws.onclose = startReconnecting;
                clearInterval(interval);
            }
        }, 3000);
    }
    ws.onclose = startReconnecting;
}

let rc;
autoReconnect(()=>{
    rc = new WebSocket(
        'ws://'
        + window.location.host
        + `/ws/chat/${window.seen.pk}/`
    )
    return rc;
});
```

test it by running and stop local host, it works fine. (btw I found it weird this question has been posted for a long time, but there is not a short and elegant solution)

the benefit of this method, is that it allows you to pass in an arrow function, so that you can assign variable to any outer scope.

Share Improve this answer Follow

answered Aug 31, 2022 at 14:06

Weilory
3,117 23 39

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

javascript websocket

See similar questions with these tags.