

# Getting the reason why websockets closed with close code 1006

Asked 11 years, 2 months ago   Modified 9 months ago   Viewed 474k times



I want to get the reason websockets closed so that I can show the right message to the user.

161

I have



```
sok.onerror=function (evt)
  { //since there is an error, sockets will close so...
    sok.onclose=function(e){
      console.log("WebSocket Error: " , e);}
  }
```

The code is always `1006` and the reason is always `" "`. But I want to tell different closing reasons apart.

For example, the command line gives an error reason: "You cannot delete that because the database won't let you". But on Chrome's console, the reason is still `" "`.

Is there any other way to tell different closing reasons apart?

javascript

websocket

Share

Improve this question

Follow

edited Jan 4 at 9:30



Sorena

47 ● 1 ● 9

asked Oct 10, 2013 at 19:05



slevin

4,338 ● 20 ● 75 ● 140

I think this is because of how the server is handling the connected / disconnected events. I can't say for sure but the connection closing needs to be handled correctly on the server also with code. Try overriding the built in On Connected /Disconnected methods on the server and see. My assumption only is that you're closing it but the server isn't closing properly and therefore not relaying the proper closed response. – [Michael Puckett II](#) Dec 29, 2018 at 20:41

8 Answers

Sorted by: Highest score (default)



[Close Code](#) `1006` is a special code that means the connection was closed abnormally (locally) by the browser implementation.

197



If your browser client reports close code `1006`, then you should be looking at the `websocket.onerror(evt)` event for details.



However, Chrome will rarely report any close code `1006` reasons to the Javascript side. This is likely due to client security rules in the WebSocket spec to prevent abusing WebSocket. (such as using it to scan for open ports on a destination server, or for generating lots of connections for a denial-of-service attack).

Note that Chrome will often report a close code `1006` if there is an error during the HTTP Upgrade to WebSocket (this is the step before a WebSocket is technically "connected"). For reasons such as bad authentication or authorization, or bad protocol use (such as requesting a subprotocol, but the server itself doesn't support that same subprotocol), or even an attempt at talking to a server location that isn't a WebSocket (such as attempting to connect to `ws://images.google.com/`)

Fundamentally, if you see a close code `1006`, you have a very low level error with WebSocket itself (similar to "Unable to Open File" or "Socket Error"), not really meant for the user, as it points to a low level issue with your code and implementation. Fix your low level issues, and then when you are connected, you can then include more reasonable error codes. You can accomplish this in terms of scope or severity in your project. Example: info and warning level are part of your project's specific protocol, and don't cause the connection to terminate. With severe or fatal messages reporting also using your project's protocol to convey as much detail as you want, and then closing the connection using the limited abilities of the WebSocket close flow.

Be aware that WebSocket close codes are very strictly defined, and the close reason phrase/message cannot exceed 123 characters in length (this is an intentional WebSocket limitation).

But not all is lost, if you are just wanting this information for debugging reasons, the detail of the closure, and its underlying reason is often reported with a fair amount of detail in Chrome's Javascript console.

Share

Improve this answer

Follow

edited Oct 7, 2021 at 7:34



Community Bot

1 • 1

answered Oct 10, 2013 at 20:05



Joakim Erdfelt

49.4k • 7 • 88 • 141

9 Joakim, thanks, very detailed anser. If I use `sok.onerror=function (evt) {console.log(evt);}` the details are not so much. Not even a `reason` or something. So, no options at all? I just show to the user, `something is wrong`, or `not connencted`? Not so user-friendly, it would be nice if the user can see "You cannot delete, cause of database restrictions". Any options? Thanks – [slevin](#) Oct 10, 2013 at 20:42

You should use `sok.onclose` instead which triggers `close event`, it has `reason` and `code` in it – [Ihab Khattab](#) Oct 29, 2014 at 16:01

@lhabKhattab that would be close code specific, and also when the close occurs. having `sok.onclose` will work for many paths, but not all paths. Especially bad protocol, bad handshake errors (like some conditions that could cause close code `1006`). Will this change in the future? Probably. But when this answer was written it was true. – Joakim Erdfelt Oct 29, 2014 at 16:20

@JoakimErdfelt sorry, I was replying to @slevin question about he has no `reason` returned when he used `onerror` I was pointing that this properties `code` & `reason` specific to `close` event not `error` event. so it'd be better **for him** to use `onclose` instead, am I missing something? – lhab Khattab Oct 29, 2014 at 17:47

- 1 In My case same 1006 Error is coming, but this is happening in case of Chrome, Firefox etc. not in case of Opera or Safari of iOS Devices. If anyone have idea then please help me out, [stackoverflow.com/questions/30799814/...](https://stackoverflow.com/questions/30799814/...) – Mrug Jun 18, 2015 at 7:40

In my and possibly @BIOHAZARD case it was `nginx proxy timeout`. In default it's `60` sec without activity in socket

78

I changed it to 24h in `nginx` and it resolved problem

```
proxy_read_timeout 86400s;
proxy_send_timeout 86400s;
```

Share Improve this answer Follow

answered Aug 29, 2019 at 8:30



[mixalbl4](#)

3,927 ● 1 ● 34 ● 47

- 1 where to set this? is this a terminal command after install the nginx? – Qadir Hussain May 23, 2022 at 7:25

@QadirHussain You need to add these lines in the nginx configuration file (placed in `/usr/local/nginx/conf`, `/etc/nginx`, or `/usr/local/etc/nginx`). I'm guessing you're using nginx as reverse proxy? – nulldevops Aug 4, 2022 at 17:13

This is a proxy issue for sure. I've tested passing my connection thru HAPROXY and then directly to the backend component. I can see even the sessionId is different. Passing thru the HAPROXY the sessionId is a full UUID and directly connecting it is a small hash like `ghvvedi3`. – Magno C Feb 10, 2023 at 17:58

- 1 You're the best! Thank you 😊 – Phatdunny Oct 1 at 1:53

Thought this might be handy for others. Knowing regex is useful, kids. Stay in school.

Edit: Turned it into a handy dandy function!

```
let specificStatusCodeMappings = {
  '1000': 'Normal Closure',
```



```
'1001': 'Going Away',
'1002': 'Protocol Error',
'1003': 'Unsupported Data',
'1004': '(For future)',
'1005': 'No Status Received',
'1006': 'Abnormal Closure',
'1007': 'Invalid frame payload data',
'1008': 'Policy Violation',
'1009': 'Message too big',
'1010': 'Missing Extension',
'1011': 'Internal Error',
'1012': 'Service Restart',
'1013': 'Try Again Later',
'1014': 'Bad Gateway',
'1015': 'TLS Handshake'
};

function getStatusCodeString(code) {
  if (code >= 0 && code <= 999) {
    return '(Unused)';
  } else if (code >= 1016) {
    if (code <= 1999) {
      return '(For WebSocket standard)';
    } else if (code <= 2999) {
      return '(For WebSocket extensions)';
    } else if (code <= 3999) {
      return '(For libraries and frameworks)';
    } else if (code <= 4999) {
      return '(For applications)';
    }
  }
  if (typeof(specificStatusCodeMappings[code]) !== 'undefined') {
    return specificStatusCodeMappings[code];
  }
  return '(Unknown)';
}
```

Usage:

```
getStatusCodeString(1006); //'Abnormal Closure'
```

```
{
  '0-999': '(Unused)',
  '1016-1999': '(For WebSocket standard)',
  '2000-2999': '(For WebSocket extensions)',
  '3000-3999': '(For libraries and frameworks)',
  '4000-4999': '(For applications)'
}

{
  '1000': 'Normal Closure',
  '1001': 'Going Away',
  '1002': 'Protocol Error',
  '1003': 'Unsupported Data',
  '1004': '(For future)',
  '1005': 'No Status Received',
  '1006': 'Abnormal Closure',
```

```
'1007': 'Invalid frame payload data',  
'1008': 'Policy Violation',  
'1009': 'Message too big',  
'1010': 'Missing Extension',  
'1011': 'Internal Error',  
'1012': 'Service Restart',  
'1013': 'Try Again Later',  
'1014': 'Bad Gateway',  
'1015': 'TLS Handshake'  
}
```

Source (with minor edits for terseness): [https://developer.mozilla.org/en-US/docs/Web/API/CloseEvent#Status\\_codes](https://developer.mozilla.org/en-US/docs/Web/API/CloseEvent#Status_codes)

Share

edited Apr 22, 2020 at 23:41

answered Apr 18, 2020 at 3:10

Improve this answer



Andrew

1

Follow

---

What about a npm package? :-)- [Piranna](#) Dec 18, 2020 at 12:51

---

[websocket-close-codes](#) npm package has the codes. - [whistling\\_marmot](#) Aug 4, 2021 at 15:58

---

@whistling\_marmot Only in one direction (for any reasonable efficiency). - [Andrew](#) Aug 6, 2021 at 16:27

---

@Andrew You can get the number from the word - wsCloseCodes['PolicyViolation'] - or the word from the number wsCloseCodes[1008] - [whistling\\_marmot](#) Aug 11, 2021 at 10:41

---

@whistling\_marmot Oh, this is Typescript... I looked at it as an object, because I thought we were dealing with raw JS here... - [Andrew](#) Aug 11, 2021 at 17:31

---



22



It looks like this is the case when Chrome is not compliant with WebSocket standard. When the **server initiates close** and sends close frame to a client, Chrome considers this to be an error and reports it to JS side with code 1006 and no reason message. In my tests, Chrome never responds to server-initiated close frames (close code 1000) suggesting that code 1006 probably means that Chrome is reporting its own internal error.

P.S. Firefox v57.00 handles this case properly and successfully delivers server's reason message to JS side.

Share Improve this answer Follow

answered Nov 16, 2018 at 14:45



[user10663464](#)

229 ● 2 ● 2

Arrrgh, I was getting a 404 due to a bad rewrite and Chrome didn't even try to show it... Thanks for the hint! – [RobM](#) Nov 1, 2022 at 7:37



11



I've got the error while using Chrome as client and golang gorilla websocket as server under nginx proxy

And sending just some "ping" message from server to client every x second resolved problem

Update: Oh boy I implemented dozens of websocket based apps after this answer and **PINGING FROM CLIENT** every 5 seconds is the correct way to keep connection with server alive (I do not know what was in my mind when I was recommending to ping from server)

Share

edited Oct 17, 2021 at 22:09

answered Aug 16, 2019 at 21:30

Improve this answer

Follow



[BIOHAZARD](#)

2,047 ● 22 ● 24

Correct answer I think, but about the Update, why do you think sending pings from the server is a bad idea? (based on Mozilla docs, you can send pings from any side, but when you send them from the server, the browser handles the pong automatically.)

– [Mohammad Reza Karimi](#) Jul 7 at 14:11

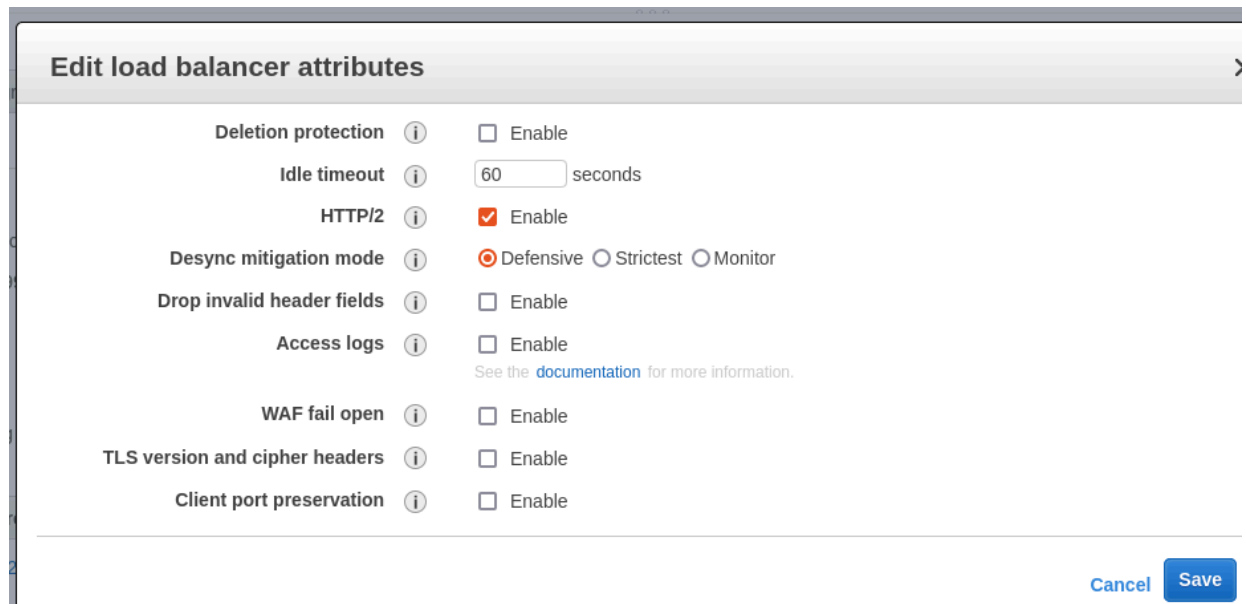
For me, the connection was getting closed every 5 minutes (for those looking for an optimized interval), and as an additional note, the issue is not specific to Chrome or Nginx (we had the same issue without using them e.g. with the Postman client). – [Mohammad Reza Karimi](#) Jul 7 at 15:55

We had the same problem and actually AWS was our problem.

Setup Websocket connection -> AWS EC2 Loadbalancer -> Nginx Proxy -> Node.js Backend

4

We increase the timeout based on [this](#) answer above in the Nginx conf but didn't see any improvements. We now found out that the AWS Loadbalancer also has a timeout that defaults to 60s. You can edit it under EC2 -> Loadbalancers



**Edit load balancer attributes**

Deletion protection	<input type="checkbox"/> Enable
Idle timeout	60 seconds
HTTP/2	<input checked="" type="checkbox"/> Enable
Desync mitigation mode	<input checked="" type="radio"/> Defensive <input type="radio"/> Strictest <input type="radio"/> Monitor
Drop invalid header fields	<input type="checkbox"/> Enable
Access logs	<input type="checkbox"/> Enable <small>See the <a href="#">documentation</a> for more information.</small>
WAF fail open	<input type="checkbox"/> Enable
TLS version and cipher headers	<input type="checkbox"/> Enable
Client port preservation	<input type="checkbox"/> Enable

[Cancel](#) [Save](#)

Note that we didn't implement a ping-pong scheme. We think that implementing one would also fix the problem until then we just use this workaround and increase the idle timeout.

Share Improve this answer Follow

answered May 16, 2022 at 10:36

 **John Doe**  
303 ● 3 ● 12

▲

3

Posting my solution here, maybe it helps someone down the line.

I had the issue that websockets worked in Firefox but not Chrome (or other Chromium based browsers). Failing with close code 1006 and I couldn't get any meaningful error description. Finally Postman gave me a reason - "Error: Server sent no subprotocol."

My setup is Django API with django-channels and React FE with react-use-websockets. To use token based authentication, I'd added the authorization token to the header "sec-websocket-protocol". It's a hack-ish way to send the token which defines a subprotocol. React snippet:

```
useWebSocket(`${WS_URL}`, {  
  protocols: ['authorization', `${accessToken}`],  
})
```

```
retryOnError: true,  
...}
```

This was enough for Firefox and the websocket connection was established. But Chrome refused the connection because the server is actually supposed to **send the subprotocol back**, which I'd missed.

In the end, it was just a matter of adding the same protocol to the response when accepting the websocket connection: `self.accept("authorization")` instead of `self.accept()`

```
class MyConsumer(WebSocketConsumer):  
    def connect(self):  
        user = self.scope.get("user")  
        self.group_name = "name"  
  
        if user.is_authenticated:  
            self.group_name = group_name  
  
            async_to_sync(self.channel_layer.group_add)(self.group_name,  
self.channel_name)  
  
            self.accept("authorization")  
  
        else:  
            self.close()
```

Relevant Django Channels documentation:

<https://channels.readthedocs.io/en/latest/topics/consumers.html#websocketconsumer>

Share

edited Feb 29 at 11:10

answered Feb 29 at 11:09

Improve this answer

 tinag  
31 ● 3

Follow

Holy hell, this is the same problem I've been having. I'm working with Spring Boot + React with `useWebSockets()` and I had the exact same issue. You saved me a huge headache.

– fudge Mar 18 at 6:22 



Adding this as one of the possible reasons rather than answer to the question.

1

The issue we had affected only chromium based browsers.



We had a load balancer & the browser was sending more bytes than negotiated during handshake resulting in the load balancer terminating the connection.



TCP windows scaling resolved the issue for us.





Share Improve this answer Follow

answered Oct 29, 2021 at 15:35



Aswin Saketh

11 ● 2

---