DOMException: The play() request was interrupted

By François Beaufort
Dives into Chromium source code

Did you just stumble upon this unexpected media error in the Chrome DevTools JavaScript Console?

> *Uncaught (in promise) DOMException: The play() request was interrupted by a call to pause().*

or

> *Uncaught (in promise) DOMException: The play() request was interrupted by a new load request.*

You're in the right place then. Have no fear. I'll explain what is causing this and how to fix it.

What is causing this

Here's some JavaScript code below that reproduces the "Uncaught (in promise)" error you're seeing:

DON'T

```
<video id="video" preload="none" src="https://example.com/file.mp4"></video

<script>
  video.play(); // <-- This is asynchronous!
  video.pause();
</script>
```

The code above results in this error message in Chrome DevTools:

> *Uncaught (in promise) DOMException: The play() request was interrupted by a call to pause().*

As the video is not loaded due to **`preload="none"`**, video playback doesn't necessarily start immediately after **`video.play()`** is executed.

Moreover since Chrome 50, a **`play()`** call on an a **`<video>`** or **`<audio>`** element returns a Promise, a function that returns a single result asynchronously. If playback succeeds, the Promise is fulfilled and the **`playing`** event is fired at the same time. If playback fails, the Promise is rejected along with an error message explaining the failure.

Now here's what happening:

1. **`video.play()`** starts loading video content asynchronously.
2. **`video.pause()`** interrupts video loading because it is not ready yet.
3. **`video.play()`** rejects asynchronously loudly.

Since we're not handling the video play Promise in our code, an error message appears in Chrome DevTools.

Note: Calling `video.pause()` isn't the only way to interrupt a video. You can entirely reset the video playback state, including the buffer, with `video.load()` and `video.src = ''`.

How to fix it

Now that we understand the root cause, let's see what we can do to fix this.

First, don't ever assume a media element (video or audio) will play. Look at the Promise returned by the **`play`** function to see if it was rejected. It is worth noting that the Promise won't fulfill until playback has actually started, meaning the code inside the **`then()`** will not execute until the media is playing.

Example: Autoplay

```
<video id="video" preload="none" src="https://example.com/file.mp4"></video

<script>
  // Show loading animation.
  var playPromise = video.play();

  if (playPromise !== undefined) {
    playPromise.then(_ => {
      // Automatic playback started!
      // Show playing UI.
    })
    .catch(error => {
      // Auto-play was prevented
      // Show paused UI.
    });
  }
</script>
```

👍 Example: Play & Pause

```
<video id="video" preload="none" src="https://example.com/file.mp4"></video>

<script>
  // Show loading animation.
  var playPromise = video.play();

  if (playPromise !== undefined) {
    playPromise.then(_ => {
      // Automatic playback started!
      // Show playing UI.
      // We can now safely pause video...
      video.pause();
    })
    .catch(error => {
      // Auto-play was prevented
      // Show paused UI.
    });
  }
</script>
```

That's great for this simple example but what if you use `video.play()` to be able to play a video later?

I'll tell you a secret. You don't have to use `video.play()`, you can use `video.load()` and here's how:

👍 Example: Fetch & Play

```
<video id="video"></video>
<button id="button"></button>

<script>
  button.addEventListener('click', onButtonClick);

  function onButtonClick() {
    // This will allow us to play video later...
    video.load();
    fetchVideoAndPlay();
  }

  function fetchVideoAndPlay() {
    fetch('https://example.com/file.mp4')
    .then(response => response.blob())
    .then(blob => {
      video.srcObject = blob;
      return video.play();
    })
    .then(_ => {
      // Video playback started ;)
    })
    .catch(e => {
      // Video playback failed ;(
    })
  }
</script>
```

Warning: Don't make your `onButtonClick` function asynchronous with the `async` keyword. You'll lose the "user gesture token" required to allow your video to play later.

Play promise support

At the time of writing, `HTMLMediaElement.play()` returns a promise in Chrome, Firefox, Opera, and Safari. Edge is still working on it.

Danger zone

`<source>` within `<video>` makes `play()` promise never rejects

For `<video src="not-existing-video.mp4"\>`, the `play()` promise rejects as expected as the video doesn't exist. For `<video><source src="not-existing-video.mp4" type='video/mp4'></video>`, the `play()` promise never rejects. It only happens if there are no valid sources.

Chromium Bug

Last updated July 2, 2018.