raven-js Documentation

Release 1.0.8

Matt Robenolt

May 07, 2013

CONTENTS

Raven.js is an **under 5KB** standalone JavaScript client for Sentry.

This version of Raven.js requires Sentry 5.3 or newer.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

Raven is distributed in a few different methods, but they all should be included inside the `<head>` of your page.

You should try and include Raven as high up on the page as possible. Ideally, you'd like to include Raven first, in order to potentially catch errors from other JavaScript files.

1.1.1 Using our CDN

We serve our own builds off of Amazon CloudFront. They are accessible over both http and https, so we recommend leaving the protocol off.

```
<script src="//d3nslu0hdya83q.cloudfront.net/dist/1.0.8/raven.min.js"></script>
```

If you're feeling adventurous, we also host a **master** build, which should be considered *potentially* unstable, but bleeding edge.

```
<script src="//d3nslu0hdya83q.cloudfront.net/builds/master/raven.min.js"></script>
```

1.1.2 CDNJS.com

cdnjs.com also gives us SPDY support! Again, just leave the protocol off, and it'll do it's magic.

```
<script src="//cdnjs.cloudflare.com/ajax/libs/raven.js/1.0.8/raven.min.js"></script>
```

1.1.3 Bower

We also provide a way to deploy Raven via bower. Useful if you want serve your scripts instead relying on CDNs and mantain a `component.json` with a list of dependencies and versions.

```
bower install raven-js
```

Please note that it automatically deploys the `tracekit` requirement and you should link it **before** `raven-js`.

```
<script src="/components/tracekit/tracekit.js"></script>
<script src="/components/raven-js/src/raven.js"></script>
```

Also note that both files are uncompresed but are ready to pass to any decent JavaScript compressor like uglify or closure.

1.2 Configuration

We must first configure Sentry to allow certain hosts to report errors. This prevents abuse so somebody else couldn't start sending errors to your account from their site.

This can be found under the *Project Details* page in Sentry.

Now need to set up Raven.js to use your Sentry DSN.

```
Raven.config('https://public@getsentry.com/1').install()
```

At this point, Raven is ready to capture any uncaught exception.

Although, this technically works, this is not going to yield the greatest results. It's highly recommended to next check out *Usage*.

1.2.1 Putting it all together

```
<!DOCTYPE html>
<html>
<head>
    <title>Awesome stuff happening here</title>
    <script src="//d3nslu0hdya83q.cloudfront.net/build/master/raven.min.js"></script>
    <script>Raven.config('https://public@getsentry.com/1').install()</script>
</head>
<body>
    ...
    <script src="jquery.min.js"></script>
    <script src="myapp.js"></script>
</body>
</html>
```

1.3 Usage

By default, Raven makes a few efforts to try it's best to capture meaningful stack traces, but browsers make it pretty difficult.

The easiest solution is to prevent an error from bubbling all of the way up the stack to `window`.

1.3.1 How to actually capture an error correctly

try...catch

The simplest way, is to try and explicitly capture and report potentially problematic code with a `try...catch` block and `Raven.captureException`.

```
try {
    doSomething(a[0])
} catch(e) {
    Raven.captureException(e)
}
```

context/wrap

`Raven.context` allows you to wrap any function to be immediately executed. Behind the scenes, Raven is just wrapping your code in a `try...catch` block.

```
Raven.context(function() {
    doSomething(a[0])
})
```

`Raven.wrap` wraps a function in a similar way to `Raven.context`, but instead of executing the function, it returns another function. This is totally awesome for use when passing around a callback.

```
var doIt = function() {
    // doing cool stuff
}

setTimeout(Raven.wrap(doIt), 1000)
```

1.3.2 Tracking authenticated users

While a user is logged in, you can tell Sentry to associate errors with user data.

```
Raven.setUser({
    email: 'matt@example.com',
    id: '123'
})
```

If at any point, the user becomes unauthenticated, you can call `Raven.setUser()` with no arguments to remove their data. *This would only really be useful in a large web app where the user logs in/out without a page reload.*

1.3.3 Capturing a specific message

```
Raven.captureMessage('Broken!')
```

1.3.4 Passing additional data

`captureException`, `context`, `wrap`, and `captureMessage` functions all allow passing additional data to be tagged onto the error, such as `tags`.

```
Raven.captureException(e, {tags: { key: "value" }})

Raven.captureMessage('Broken!', {tags: { key: "value" }})

Raven.context({tags: { key: "value" }}, function(){ ... })

Raven.wrap({logger: "my.module"}, function(){ ... })
```

1.3.5 Dealing with minified source code

Raven and Sentry now support Source Maps. *Information coming soon*

CHAPTER

TWO

DEVELOPERS

2.1 Contributing

2.1.1 Setting up an Environment

To run the test suite and run our code linter, node.js and npm are required. If you don't have node installed, get it here first.

Installing all other dependencies is as simple as:

```
make develop
```

2.1.2 Running the Test Suite

The test suite is powered by Mocha and can both run from the command line, or in the browser.

From the command line:

```
make test
```

From your browser:

```
make runserver
```

Then visit: http://localhost:8888/test/test.html

2.1.3 Contributing Back Code

Please, send over suggestions and bug fixes in the form of pull requests on GitHub. Any fixes/features should include tests.

CHAPTER

THREE

RESOURCES

- Bug Tracker
- Code
- IRC (irc.freenode.net, #sentry)

CHAPTER

FOUR

INDICES AND TABLES

search