How to Handle Errors in Meteor Methods

posted Thu, Sep 3, 2015 · 13 Comments

If you define methods for altering your collections, you get the convenience of not having to define security-issue-prone `allow` / `deny` rules.

If you define these methods on **both** the client and the server, you get the benefit of Meteor's latency compensation feature.

You can run into trouble, however, when you throw an error in a client-side method. You get the following unhandled exception:

```
Exception while simulating the effect of invoking 'yourMethodName'
```

Throwing errors on the client and the server

In order to get around this, you can create a wrapper function that will `throw` an error on the server and simply `return` it on the client.

Edit 9/30/15: The return values of Method stubs are ignored, so returning a `Meteor.Error` on the client doesn't really do anything.

This will let you write the same code on the client and server, while also giving you the ability to display errors to the user when the server method returns.

The wrapper function

If you return the following `throwError` function in a method, it won't cause any unhandled exceptions on the client. And when the server method returns, it will hand off the error to the client.

The error will be returned as the first argument to your `Meteor.call` callback in the version of the `Meteor.call` callback that's triggered by the server-side code.

```
throwError = function(error, reason, details) {
  var meteorError = new Meteor.Error(error, reason, details);

  if (Meteor.isClient) {
    // this error is never used
    // on the client, the return value of a stub is ignored
```

```
      return meteorError;
    } else if (Meteor.isServer) {
      throw meteorError;
    }
  };
```

This function was taken from this helpful blog post: Smooth error handling for Meteor.methods.

You can pass it the same arguments you would pass to a `new Meteor.Error` call, that is: `error`, `reason`, and `details`.

See the `Meteor.Error` docs for more info.

Handling errors

If a method call causes an error to be thrown, you most likely want to notify the user in some way. You can handle this in your client-side code by checking the type of error that's returned from your `Meteor.call` call.

```
Meteor.call('saveProject', projectName, function (error, project) {
  if (error.error === "no-project-name") {
    sAlert.error("Please specify a project name.");
```

```
    }
  });
```

We check to see if the error returned is equal to an error name we know (i.e. “no-project-name”) and then return the appropriate error if it is.

The above `sAlert.error` function is from the `juliancwirko:s-alert` package (see here). It will cause a user-facing error message to pop up.

Consolidating error messages

When you call and return your new `throwError` function, if you provide a nice user-facing error message as the `reason` text (the second argument you pass to `throwError`) you can simplify your client-side error handling code:

```
Meteor.call('saveProject', projectName, function (error, result) {
  if (error) {
    sAlert.error(error.reason);
  }
});
```

The above code will take the `reason` that you provided to the `throwError` function and display it to the user. This will cut down on the amount of custom code you have to write on the client.

Edit 9/4/15: However! Since some of the errors returned to a `Meteor.call` callback function won't be one's you've defined (e.g. database errors and errors thrown by package code), I recommend the following approach instead:

```
// defined globally somewhere
isKnownError = function (error) {
  var errorName = error && error.error;
  var listOfKnownErrors = ['no-project-name', 'user-not-logged-in', 'projec

  return _.contains(listOfKnownErrors, errorName);
};

Meteor.call('saveProject', projectName, function (error, result) {
  if (isKnownError(error)) {
    sAlert.error(error.reason);
  } else if (error) {
    sAlert.error("An unknown error occurred while saving your project.");
  } else {
    sAlert.success("Successfully saved your project.");
```

```
      successCallback();
    }
  });
```

This will save you from displaying potentially confusing errors to the client by checking to see if they're listed in your approved `listOfKnownErrors` array before displaying their `reason` to the client.

If you want, you can view this code as part of a sample project: Method Demo - isKnownError Branch. Thank you @Siyfion for helping me correct this code.

Example application

If you want to see how this works in a demo Meteor application, go here: Method Error Demo.

After loading that page, try clicking the `Save Project` button without filling in the `input` box. It will display an error using the method described in this article.

You can find the code used for making this example application here: Method demo.

Questions?

If you have any questions, you can leave a comment here, contact me on twitter at artisfyhq, or email me at david@storylog.com.

Thank you for reading!

13 Comments Smashing 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

jonathan • 2 years ago

Thanks you for typing up this post! I do have a quick question:

1. Why do you return the error if the code is running on the client and throw error if the code is running on the sever? Specifically, what is the point of returning the error if the code is running on the client vs throwing an exception?

• Reply • Share ›

David Miranda Mod jonathan • 2 years ago

If you throw an error on the client, you will get an error message in the console. It's an unhandled exception and it's not good to leak these. It means something went wrong and you didn't do anything about it.

By returning an error from a method on the client, we'll ensure that we'll get that error in our Meteor.call callback.

where we can handle it. If the method was only defined on the server, we wouldn't need to do this. The server automatically handles server errors and passes them back to us in the Meteor.call callback.

What we're doing here is ensuring that whether the method is defined on the server OR the client OR both, it'll be passed to our Meteor.call callback in the same way (that is, as the first argument to the Meteor.call callback).

Let me know if this makes sense.

1 • Reply • Share ›

jonathan ➚ David Miranda • 2 years ago

Hi David,

Thanks for taking the time to reply to my question.

I see what you mean by "If you throw an error on the client, you will get an error message in the console. It's an unhandled exception." That makes sense.

Also, I downloaded your sample code and tinkered with it. Specifically, I changed the code so that the client will return a different error message than the error message that the server throws.

With that change, I expected to see two different error messages when I tried to save a project with no project name, one error message from the client and one error message from the server. However, I only see the error from the server.

Here's the gist of the code (you can see that I altered line 76 and 84 to change the error msgs):
https://gist.github.com/che...

I have taken a screenshot of what I see.

Can you explain to me why the client error is not shown? I thought the callback function for Meteor.call('saveProject' ...) should capture both errors from the client and the server? Or am I mistaken.

thank you!

Reply • Share ›

David Miranda Mod → jonathan • 2 years ago

Hi Jonathan,

Thank you for this question. This is something I thought I knew, but it turns out I didn't.

According to the Meteor Docs (http://docs.meteor.com/#/fu..., "on the client, the return value of a stub is ignored." Stubs, according to the docs, are "run for their side-effects," not their return values. So, database updates, pretty much.

This is why you're not seeing the client's error message. I actually prefer it this way, so we don't get two different error messages (one from the client and then one from the server a little later). I thought Meteor somehow took care of this automatically, but it turns out they just ignore return values from stubs altogether.

I think I still prefer using the code in this blog post, as it lets you write the same code on the client and the server. But some documentation should be added to it to let people know that returning a Meteor.Error from a stub won't actually do anything.

1 • Reply • Share ›

jonathan → David Miranda • 2 years ago

Hi David,

Thanks for taking the time to investigate into this issue for us. What you said makes sense.

I guess the takeaway is don't expect for the client to see any errors that may have occurred during the client part of latency compensation.

I guess, if we need to do any error handling on the client, we should do it before calling the latency compensation function.

Thanks again for the help.

1 • Reply • Share ›

David Miranda Mod → jonathan • 2 years ago

Yup, very good point jonathan. That's good to keep in mind for form validation, for example.

1 ▲ ▼ • Reply • Share ›

DevIO • 3 years ago

Nice, thanks!

▲ ▼ • Reply • Share ›

Siyfion • 3 years ago

Your new "edited" code has a bug, it will *always* throw an error, even if there wasn't one!

▲ ▼ • Reply • Share ›

David Miranda Mod → Siyfion • 3 years ago

Thank you! I think it's fixed now.

▲ ▼ • Reply • Share ›

Siyfion → David Miranda • 3 years ago

This block of code (in your edited section of the blog post):

```
Meteor.call('saveProject', projectName, function (error, result) {
if (isKnownError(error.error)) {
sAlert.error(error.reason);
} else {
sAlert.error("An unknown error occurred while saving your project.");
}
});
```

is the bit I was referring too, but I still believe it has the same issue.

▲ ▼ • Reply • Share ›

David Miranda Mod → Siyfion • 3 years ago

Thank you again. It seems I had multiple errors in this blog post.

I fixed the two errors in that code. I wasn't handling the possibility of success at all

I fixed the two errors in that code. I wasn't handling the possibility of success at all.

Reply • Share ›

cstrat • 3 years ago

Do you know if there is an easy way to add your consolidated error message snippet to all method returns by default? I have similar code in my app and I have added the same:

if (error) {
sAlert.error(error.reason);
}

At the start of ever single callback...

Reply • Share ›

David Miranda Mod → cstrat • 3 years ago

Yes, there's a way to do this. However, keep in mind that some errors returned to the client won't be one's that you've thrown. There are database errors as well as errors returned by package code, to name a few.

A safer way to check for a client-side error would actually be the following, and it's what I recommend:

```
if (error.error === "project-no-name" || error.error === "user-not-logged-in") {
  sAlert.error(error.reason);
} else {
  sAlert.error("An unknown error occurred while saving your project.");
}
```

This gives you the benefit of only passing errors to the client that you want them to see. There are some errors that would be too confusing for the average user and you'd much rather display a generic error message in that case.

The above code can also be simplified if you make a function to test if an error is one that you want to display to the client.

```
// defined globally client-side somewhere
var isKnownError = function (errorName) {
  // returns true if it's an error you know about and want to display to the client
```

see more