Add TeeReader to service.DecodeRequest #803

Open with

Closed ZjMNZHgG5jMXw wants to merge 1 commit into `goadesign:master` from `ZjMNZHgG5jMXw:master`

Conversation 20 Commits 1 Checks 0 Files changed 1 +10 −2

ZjMNZHgG5jMXw commented on Oct 7, 2016 Contributor

Hi,

I changed the function `DecodeRequest` such that middlewares still can read the request body afterwards. In my case, this is required to implement an authorization middleware which computes a hash sum over the plain request body as part of the protocol. Currently, this does not work for non-trivial bodies, since `DecodeRequest` closes the body.

Would be nice to see the patch upstream.

Thanks!

Cheers,
Stefan

`Add TeeReader to service.DecodeRequest` ✓ 095a4bc

raphael commented on Oct 7, 2016 Member

Thank you for the PR! This seems like a useful addition. There is the obvious tradeoff of having to keep a copy of the body in memory - probably OK in practice. Do you think you could add a test?

panzerdev commented on Oct 10, 2016

Doesn't the std lib handler take care of closing the body as a server? At least that's what the docs say. So keeping this in memory shouldn't be a dealbreaker.

derekperkins commented on Oct 10, 2016 Member

Could that be implemented as a custom decoder? I'm mobile, so I haven't seen the code yet, but I wouldn't want to make that the default implementation. It definitely has value, but not worth the overhead for the majority of people who don't need to read the body multiple times.

raphael commented on Oct 10, 2016 Member

@**panzerdev** not sure what you mean. Here the code makes a copy of the whole body content in a in-memory buffer before closing the body. It probably doesn't change much for e.g. JSON payloads but could have a big impact on streaming encodings.

@**derekperkins** hmm yeah you could actually and it would be better.

@**ZjMNZHgG5jMXw** do you think you could rework this PR as a custom decoder? Depending on what you need it could be a JSON-only custom decoder that does the copy first. Or it could be a more generic "Tee" decoder that you'd have to configure with an actual underlying decoder. Custom decoders are described here: https://goa.design/implement/encoding/

panzerdev commented on Oct 10, 2016 • edited

@**raphael** sorry wrong thought there.

What would you think about a `[]byte` as `RawPayload` in the `RequestData` struct

https://github.com/goadesign/goa/blob/master/context.go#L26

if you have the need to use the raw bytes again later?

derekperkins commented on Oct 10, 2016 Member

@**panzerdev** if you really needed it, you could attach it to the context inside your custom decoder. I don't think it's that common of a use case to warrant special handling when it can be done easily with the existing setup.

panzerdev commented on Oct 10, 2016

@**derekperkins** I came across that when thinking that the body of the Request logger is actually logging the raw body but instead just the parsed Payload.

I'll do what you suggest for now but I really think that if you can access the raw Request then it should be the complete thing. It's a bit confusing when the body is missing and you have to go through the code to find out why.

raphael commented on Oct 10, 2016 Member

I have to admit I'm a little bit of two minds on this issue.

@**derekperkins**: you wouldn't "read the body multiple times" with this PR. The code reads it once, stores it in a buffer and wraps it with a `ReqCloser`. The next "read" is a no-op really. This is nice because it has the property that the request object `Body` field works as expected as noted by @**panzerdev**.

On the other hand if you have a case where your body is actually streaming then this approach requires loading everything in memory at once which could be problematic. That being said today to use streaming with goa you have to bypass the decoding entirely (that is not specify a media type in the design) as you can use e.g. a mime multipart file decoder or whichever streaming decoder you need directly on the body reader. That's because the goa decoder interface expects to be able to return an object.

One could argue that the current goa decoder interface already requires holding the decoded object in memory so also requiring that the raw body bytes be kept in memory until the request body is closed is probably not a huge problem.

So at this point I would be of the opinion that merging the initial PR (ideally after tests are added) is the best approach, @**derekperkins** do you agree?

derekperkins commented on Oct 10, 2016 Member

If you're ok with it, I won't make too much of a ruckus. Just feels like it's creating double garbage for a feature that 99% of people will never use. I'd love to see a benchmark.

panzerdev commented on Oct 10, 2016

@**derekperkins** what do you think about printing the raw body in the LogRequest middleware instead of the already processed body?

goa/middleware/log_request.go
Line 57 in 6a4b156

```
57      js, err := json.Marshal(r.Payload)
```

This would be a usecase for a lot of people I would imagine.

derekperkins commented on Oct 11, 2016 Member

I've definitely used `httputil.DumpRequest` for testing in the past, so I think it's a great option when turned on, but not as an always on thing.

@**raphael** What about adding some debug options to the context? Then in here, you could at least put it in an if statement, `if ctx.Debug == true` (pardon the shorthand syntax), then dump the request body.

If you still wanted to enable it all the time, you could be nicer on the GC using a pool like in https://github.com/spf13/hugo/tree/master/bufferpool. I stole this concept from Hugo recently because we use buffers a lot and it's a pretty easy drop in replacement.

derekperkins commented on Oct 11, 2016 Member

@**panzerdev** I forgot to answer your direct question. Yes, I think there's definitely value in seeing the raw body. On a quick glance, that code looks like it would obscure a number of issues, since it's just marshaling out the payload after it was already parsed.

raphael commented on Oct 11, 2016 Member

@**panzerdev** if all you are after is dumping the response body then that's already supported by the built-in `LogRequest` middleware. Simply initialize it with `true` and it will dump the request body to the logs (using code that is similar to `httputil.DumpRequest`).

The PR has a broader scope though, for example the initial use case presented by @**ZjMNZHgG5jMXw**. It also seems valuable that the request object accessible from within the context would have a body that can be read - that would probably be the expectation for most. But yes it does mean a copy of the request body bytes...

panzerdev commented on Oct 11, 2016

@**raphael** sadly the middleware you're mentioning in the version I linked in a previous comment doesn't show the raw request body but the state after parsing and potentially dropping data as @**derekperkins** also mentioned.

Or can you point me to the line of code you're referencing?

raphael commented on Oct 11, 2016 Member

Whoops you're right, I got mixed up with the client code.

Would be interesting to see what a custom decoder that does the "Teeing" would look like.

ZjMNZHgG5jMXw commented on Oct 11, 2016 • edited Contributor Author

Thanks for the comments!

I'd also prefer to switch the code off by default and so only on-demand, but I don't see how to achieve that with custom decoders. I need the parsed media type from the current decoder and the raw request body for the authentication check sum. That looks to me like I'd have to stack custom decoders. Would that be possible in the current framework?

raphael commented on Oct 11, 2016 Member

What you could do is "hijack" the service HTTP decoder. That is you'd set the service Decoder with a HTTP decoder built by passing in the current service Decoder.

The HTTP decoder constructor would register the same media types as the current service HTTP decoder but associate decoders which do the Teeing and call the original corresponding decoder.

This would work better than a proper custom decoder because with a proper custom decoder you'd have no way to know the media type which you said you needed.

ZjMNZHgG5jMXw commented on Oct 11, 2016 Contributor Author

I considered hijacking the decoder and got a rather bad gut feeling, since this solution would impose an order to the declaration of decoders in which the hijacker 'decoder' would have to be the last. That would at least be hard to put into words in the documentation. (Or where would you declare the hijacker in the design spec otherwise?)

Perhaps, it's possible to add a flag to the DSL which can be used to switch on the feature on demand. I have yet no idea how much effort that would be.

raphael commented on Oct 11, 2016 Member

What I was suggesting would be done purely in the implementation - not a design time. By the time the service is created the decoder is set. That seems OK to me since I don't see a lot of value with surfacing the Teeing decoder to the design (there's nothing docs or clients would do with it).

raphael mentioned this pull request on Oct 24, 2016

Use Decoder and Encoder based on Content-Encoding header #834 Closed

raphael commented on Dec 8, 2016 Member

I'm going to close this PR for now. This requires more thinking...

raphael closed this on Dec 8, 2016

Write Preview

Leave a comment

New **Video support! Upload MP4 and MOV file types.** Attach files by dragging & dropping, selecting or pasting them.

Comment

Remember, contributions to this repository should follow its contributing guidelines and code of conduct.

ProTip! Add `.patch` or `.diff` to the end of URLs for Git's plaintext views.

Reviewers
No reviews

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Linked issues
Successfully merging this pull request may close these issues.
None yet

Notifications Customize
Subscribe
You're not receiving notifications from this thread.

5 participants