httpsnoop

Package httpsnoop provides an easy way to capture http related metrics (i.e. response time, bytes written, and http status code) from your application's http.Handlers.

Doing this requires non-trivial wrapping of the http.ResponseWriter interface, which is also exposed for users interested in a more low-level API.

GO reference build passing

Usage Example

```
// myH is your app's http handler, perhaps a http.ServeMux or similar.
var myH http.Handler
// wrappedH wraps myH in order to log every request.
wrappedH := http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    m := httpsnoop.CaptureMetrics(myH, w, r)
    log.Printf(
        "%s %s (code=%d dt=%s written=%d)",
        r.Method,
        r.URL,
        m.Code,
        m.Duration,
        m.Written,
    )
})
http.ListenAndServe(":8080", wrappedH)
```

Why this package exists

Instrumenting an application's http.Handler is surprisingly difficult.

However if you google for e.g. "capture ResponseWriter status code" you'll find lots of advise and code examples that suggest it to be a fairly trivial undertaking. Unfortunately everything I've seen so far has a high chance of breaking your application.

The main problem is that a `http.ResponseWriter` often implements additional interfaces such as `http.Flusher`, `http.CloseNotifier`, `http.Hijacker`, `http.Pusher`, and `io.ReaderFrom`. So the naive approach of just wrapping `http.ResponseWriter` in your own struct that also implements the `http.ResponseWriter` interface will hide the additional interfaces mentioned above. This has a high change of introducing subtle bugs into any non-trivial application.

Another approach I've seen people take is to return a struct that implements all of the interfaces above. However, that's also problematic, because it's difficult to fake some of these interfaces behaviors when the underlying `http.ResponseWriter` doesn't have an implementation. It's also dangerous, because an application may choose to operate differently, merely because it detects the presence of these additional interfaces.

This package solves this problem by checking which additional interfaces a `http.ResponseWriter` implements, returning a wrapped version implementing the exact same set of interfaces.

Additionally this package properly handles edge cases such as `WriteHeader` not being called, or called more than once, as well as concurrent calls to `http.ResponseWriter` methods, and even calls happening after the wrapped `ServeHTTP` has already returned.

Unfortunately this package is not perfect either. It's possible that it is still missing some interfaces provided by the go core (let me know if you find one), and it won't work for applications adding their own interfaces into the mix.

However, hopefully the explanation above has sufficiently scared you of rolling your own solution to this problem. httpsnoop may still break your application, but at least it tries to avoid it as much as possible.

Anyway, the real problem here is that smuggling additional interfaces inside `http.ResponseWriter` is a problematic design choice, but it probably goes as deep as the Go language specification itself. But that's okay, I still prefer Go over the alternatives ;).

Performance

```
BenchmarkBaseline-8                 20000             94912 ns/op
BenchmarkCaptureMetrics-8           20000             95461 ns/op
```

As you can see, using `CaptureMetrics` on a vanilla http.Handler introduces an overhead of ~500 ns per http request on my machine. However, the margin of error appears to be larger than that, therefor it should be reasonable to assume that the overhead introduced by `CaptureMetrics` is absolutely negligible.

License

MIT