

[Discover Packages](#) > [Standard library](#) > [net](#) > [http](#) > [httptrace](#) 





# httptrace

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [7](#) |Imported by: [1,522](#)

## Details

- ✓ Valid [go.mod](#) file 
- ✓ Redistributable license 
- ✓ Tagged version 
- ✓ Stable version 

[Learn more](#)

## Repository

[cs.opensource.google/go/go](https://cs.opensource.google/go/go)

## Links

 [Report a Vulnerability](#) Documentation 

## <> Documentation

### Overview

Package httptrace provides mechanisms to trace the events within HTTP client requests.

[► Example](#)

### Index

```
func WithClientTrace(ctx context.Context, trace *ClientTrace) context.Context
```

```
type ClientTrace
```

```
func ContextClientTrace(ctx context.Context) *ClientTrace
```

```
type DNSDoneInfo
```

```
type DNSStartInfo
```

```
type GotConnInfo
```

```
type WroteRequestInfo
```

### Examples

[Package](#)

### Constants

This section is empty.

### Variables

This section is empty.

## Functions

### func [WithClientTrace](#)

```
func WithClientTrace(ctx context.Context, trace *ClientTrace) context.Context
```

WithClientTrace returns a new context based on the provided parent ctx. HTTP client requests made with the returned context will use the provided trace hooks, in addition to any previous hooks registered with ctx. Any hooks defined in the provided trace will be called first.

## Types

### type [ClientTrace](#)

```
type ClientTrace struct {  
    // GetConn is called before a connection is created or  
    // retrieved from an idle pool. The hostPort is the  
    // "host:port" of the target or proxy. GetConn is called even  
    // if there's already an idle cached connection available.  
    GetConn func(hostPort string)  
  
    // GotConn is called after a successful connection is  
    // obtained. There is no hook for failure to obtain a  
    // connection; instead, use the error from  
    // Transport.RoundTrip.  
    GotConn func(GotConnInfo)  
  
    // PutIdleConn is called when the connection is returned to  
    // the idle pool. If err is nil, the connection was  
    // successfully returned to the idle pool. If err is non-nil,  
    // it describes why not. PutIdleConn is not called if  
    // connection reuse is disabled via Transport.DisableKeepAlives.  
    // PutIdleConn is called before the caller's Response.Body.Close  
    // call returns.  
    // For HTTP/2, this hook is not currently used.  
    PutIdleConn func(err error)  
  
    // GotFirstResponseByte is called when the first byte of the response  
    // headers is available.  
    GotFirstResponseByte func()  
  
    // Got100Continue is called if the server replies with a "100  
    // Continue" response.  
    Got100Continue func()  
  
    // Got1xxResponse is called for each 1xx informational response header  
    // returned before the final non-1xx response. Got1xxResponse is called  
    // for "100 Continue" responses, even if Got100Continue is also defined.
```

```
// If it returns an error, the client request is aborted with that error value.  
Got1xxResponse func(code int, header textproto.MIMEHeader) error
```

```
// DNSStart is called when a DNS lookup begins.  
DNSStart func(DNSStartInfo)
```

```
// DNSDone is called when a DNS lookup ends.  
DNSDone func(DNSDoneInfo)
```

```
// ConnectStart is called when a new connection's Dial begins.  
// If net.DialerDualStack (IPv6 "Happy Eyeballs") support is  
// enabled, this may be called multiple times.  
ConnectStart func(network, addr string)
```

```
// ConnectDone is called when a new connection's Dial  
// completes. The provided err indicates whether the  
// connection completed successfully.  
// If net.DialerDualStack ("Happy Eyeballs") support is  
// enabled, this may be called multiple times.  
ConnectDone func(network, addr string, err error)
```

```
// TLSHandshakeStart is called when the TLS handshake is started. When  
// connecting to an HTTPS site via an HTTP proxy, the handshake happens  
// after the CONNECT request is processed by the proxy.  
TLSHandshakeStart func()
```

```
// TLSHandshakeDone is called after the TLS handshake with either the  
// successful handshake's connection state, or a non-nil error on handshake  
// failure.  
TLSHandshakeDone func(tls.ConnectionState, error)
```

```
// WroteHeaderField is called after the Transport has written  
// each request header. At the time of this call the values  
// might be buffered and not yet written to the network.  
WroteHeaderField func(key string, value []string)
```

```
// WroteHeaders is called after the Transport has written  
// all request headers.  
WroteHeaders func()
```

```
// Wait100Continue is called if the Request specified  
// "Expect: 100-continue" and the Transport has written the  
// request headers but is waiting for "100 Continue" from the  
// server before writing the request body.  
Wait100Continue func()
```

```
// WroteRequest is called with the result of writing the  
// request and any body. It may be called multiple times  
// in the case of retried requests.  
WroteRequest func(WroteRequestInfo)
```

```
}
```

ClientTrace is a set of hooks to run at various stages of an outgoing HTTP request. Any particular hook may be nil. Functions may be called concurrently from different goroutines and some may be called after the request has completed or failed.

ClientTrace currently traces a single HTTP request & response during a single round trip and has no hooks that span a series of redirected requests.

See <https://blog.golang.org/http-tracing> for more.

## func ContextClientTrace

```
func ContextClientTrace(ctx context.Context) *ClientTrace
```

ContextClientTrace returns the ClientTrace associated with the provided context. If none, it returns nil.

## type DNSDoneInfo

```
type DNSDoneInfo struct {  
    // Addrs are the IPv4 and/or IPv6 addresses found in the DNS  
    // lookup. The contents of the slice should not be mutated.  
    Addrs []net.IPAddr  
  
    // Err is any error that occurred during the DNS lookup.  
    Err error  
  
    // Coalesced is whether the Addrs were shared with another  
    // caller who was doing the same DNS lookup concurrently.  
    Coalesced bool  
}
```

DNSDoneInfo contains information about the results of a DNS lookup.

## type DNSStartInfo

```
type DNSStartInfo struct {  
    Host string  
}
```

DNSStartInfo contains information about a DNS request.

## type GotConnInfo

```
type GotConnInfo struct {  
    // Conn is the connection that was obtained. It is owned by  
    // the http.Transport and should not be read, written or  
    // closed by users of ClientTrace.  
    Conn net.Conn  
  
    // Reused is whether this connection has been previously  
    // used for another HTTP request.  
    Reused bool
```

```
// WasIdle is whether this connection was obtained from an
// idle pool.
WasIdle bool

// IdleTime reports how long the connection was previously
// idle, if WasIdle is true.
IdleTime time.Duration

}
```

GotConnInfo is the argument to the ClientTrace.GotConn function and contains information about the obtained connection.

## type **WroteRequestInfo**

```
type WroteRequestInfo struct {
    // Err is any error encountered while writing the Request.
    Err error
}
```

WroteRequestInfo contains information provided to the WroteRequest hook.



## Source Files

[View all](#) 

[trace.go](#)

### Why Go

[Use Cases](#)

[Case Studies](#)

### Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

### Packages

[Standard Library](#)

[About Go Packages](#)

### About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

### Connect

[Twitter](#)

[GitHub](#)

[Slack](#)

[r/golang](#)

[Meetup](#)

[Copyright](#)

[Terms of Service](#)

[Privacy Policy](#)

[Report an Issue](#)

