Go: how to get and read the headers of HTTP requests

When working with Go for web application development, you often need to access HTTP request headers. Headers contain crucial information, such as the type of content accepted by the client, cookies, and many other useful information to handle requests effectively. In this article, we will explore how to read HTTP request headers using the Go programming language.

To illustrate the process of reading headers, let's start by creating a simple HTTP server in Go. We can use the `net/http` package provided by Go to handle HTTP requests and responses. Here is a sample code to create a basic HTTP server:

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request)
{
    fmt.Fprintf(w, "Welcome to the basic HTTP
server!")
}

func main() {
    http.HandleFunc("/", handler)
```

```
    http.ListenAndServe(":8080", nil)
}
```

We now have an HTTP server that will respond with a welcome message when a request is made. However, we need to add logic to read the request headers.

To read the headers of an HTTP request in Go, we can do it inside the `handler` function that we defined in our example. The `Request` structure of the `net/http` package provides us with a `Header` field, which is an object of type `http.Header` >. This object contains all the headers of the current request.

Here's how we can modify our `handler` function to read and print the request headers:

```
func handler(w http.ResponseWriter, r *http.Request)
{
    // Read the request headers
    headers := r.Header

    // Print the headers
    fmt.Println("Request Header:")
    for key, values := range headers {
        fmt.Printf("%s: %s\n", key, values)
    }

    // Reply with a welcome message
    fmt.Fprintf(w, "Welcome to the basic HTTP
server!")
}
```

In the code above, we are iterating over all the request headers and printing both the key and associated values. A specific header can be accessed using the `headers["HeaderName"]` notation.

Now your HTTP server will not only respond with a welcome message, but will also print the headers of the received request. This is a useful starting point for more advanced handling of HTTP requests in your Go application.