Polling Websites and Reading Web Page

This lesson gives insight into how to check the status of a website with Go and how to read a page on the web.

WE'LL COVER THE FOLLOWING

- Introduction
- Explanation

Introduction

Sending a website a very simple request and seeing how the website responds is known as **polling** a website.

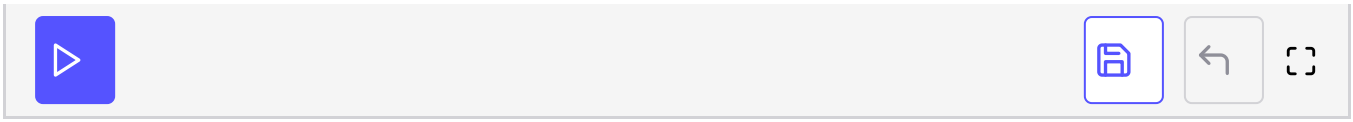
Explanation

Consider the following example where a request includes only an HTTP header.

```
package main
import (
"fmt"
"net/http"
)

var urls = []string{
  "http://www.google.com/",
  "http://golang.org/",
  "http://blog.golang.org/",
}

func main() {
  // Executes an HTTP HEAD request for all URL's
  // and returns the HTTP status string or an error string.
  for _, url := range urls {
    resp, err := http.Head(url)
    if err != nil {
    fmt.Println("Error:", url, err)
  }
  fmt.Println(url, ": ", resp.Status)
  }
}
```



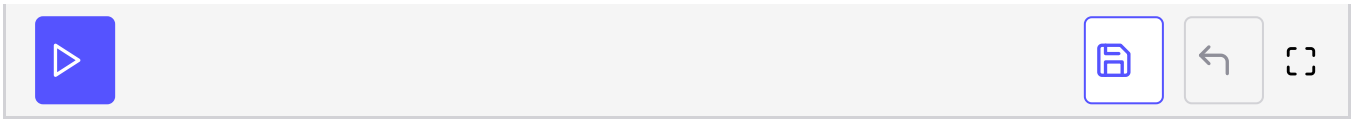
Requesting Polled URLs

In the program above, we import the package `net/http` (see **line 4**). All URLs in an array of strings `urls` (defined at **line 7**) are polled. At **line 16**, we start an iteration over `urls` with a for-range loop. At **line 17**, a simple `http.Head()` request is sent to each url to see how they react. The function's signature is: `func Head(url string) (r *Response, err error)`. When there is an error, we print it at **line 19**. If no error, the `Status` of `resp` is printed at **line 21**.

```
package main
import (
"fmt"
"net/http"
"io/ioutil"
"log"
)

func main() {
  res, err := http.Get("http://www.google.com")
  CheckError(err)
  data, err := ioutil.ReadAll(res.Body)
  CheckError(err)
  fmt.Printf("Got: %q", string(data))
}

func CheckError(err error) {
  if err != nil {
    log.Fatalf("Get: %v", err)
  }
}
```



Fetch

In the program above, we show the Html content of a web page by calling `http.Get()` at the front page of google at **line 10**. `Get` returns a result and a possible error. The error-handling here is performed by calling a function `CheckError(err)` at **line 11**, passing it the error as a parameter.

Now, look at the header of the function `CheckError(err error)` at **line 17**. When there is an error (**line 18**), we log it as *fatal* which stops the program (see **line 19**). When the program arrives at **line 12**, there is no error. The

response `res` returned from `Get` has the content in a field `Body`. We read this content in one slice of bytes called data with `ioutil.ReadAll` at **line 12**. A

possible error is again handled by `CheckError` at **line 13**. Then, we convert the slice to a string and print it out at **line 14**.

Here is a sample error output from CheckError when trying to read a non-existing web's site homepage: `2011/09/30 11:24:15 Get: Get http://www.google.bex: dial tcp www.google.bex:80: GetHostByName: No such host is known`.

Other useful functions in the `http` package which we will be using are:

- `http.Redirect(w ResponseWriter, r *Request, url string, code int)`: this redirects the browser to url (can be a path relative to the request path) and a `statuscode` code.
- `http.NotFound(w ResponseWriter, r *Request)`: this replies to the request with an HTTP 404 not found error.
- `http.Error(w ResponseWriter, error string, code int)`: this replies to the request with the specified error message and HTTP code.
- A useful field of an `http.Request object req is: req.Method`, this is a string which contains **GET** or **POST** according to how the web page was requested.

All HTTP status codes are defined as Go-constants, for example:

```
http.StatusContinue = 100
http.StatusOK = 200
http.StatusFound = 302
http.StatusBadRequest = 400
http.StatusUnauthorized = 401
http.StatusForbidden = 403
http.StatusNotFound = 404
http.StatusInternalServerError = 500
```

You can set the content header with `w.Header().Set("Content-Type", "../..")`, e.g., when sending Html-strings in a web application, execute `w.Header().Set("Content-Type", "text/html")` before writing the output, but this is normally not necessary.

Now that you're familiar with the basics of a web server, you're ready to learn

how to make a simple web application of your own.