Return greetings for multiple people - The Go Programming Language

Read time: 3 minutes

---

1. Documentation
2. Tutorials
3. Return greetings for multiple people

In the last changes you'll make to your module's code, you'll add support for getting greetings for multiple people in one request. In other words, you'll handle a multiple-value input, then pair values in that input with a multiple-value output. To do this, you'll need to pass a set of names to a function that can return a greeting for each of them.

But there's a hitch. Changing the `Hello` function's parameter from a single name to a set of names would change the function's signature. If you had already published the `example.com/greetings` module and users had already written code calling `Hello`, that change would break their programs.

In this situation, a better choice is to write a new function with a different name. The new function will take multiple parameters. That preserves the old function for backward compatibility.

1. In greetings/greetings.go, change your code so it looks like the following.

```
package greetings

import (
    "errors"
    "fmt"
    "math/rand"
)

// Hello returns a greeting for the named person.
func Hello(name string) (string, error) {
    // If no name was given, return an error with a message.
    if name == "" {
        return name, errors.New("empty name")
    }
    // Create a message using a random format.
    message := fmt.Sprintf(randomFormat(), name)
    return message, nil
}

// Hellos returns a map that associates each of the named people
// with a greeting message.
func Hellos(names []string) (map[string]string, error) {
    // A map to associate names with messages.
    messages := make(map[string]string)
    // Loop through the received slice of names, calling
    // the Hello function to get a message for each name.
    for _, name := range names {
        message, err := Hello(name)
        if err != nil {
            return nil, err
        }
        // In the map, associate the retrieved message with
        // the name.
        messages[name] = message
    }
    return messages, nil
}

// randomFormat returns one of a set of greeting messages. The
returned
// message is selected at random.
func randomFormat() string {
    // A slice of message formats.
    formats := []string{
        "Hi, %v. Welcome!",
        "Great to see you, %v!",
        "Hail, %v! Well met!",
    }
    // Return one of the message formats selected at random.
    return formats[rand.Intn(len(formats))]
}
```

In this code, you:

- Add a `Hellos` function whose parameter is a slice of names rather than a single name. Also, you change one of its return types from a `string` to a `map` so you can return names mapped to greeting messages.
- Have the new `Hellos` function call the existing `Hello` function. This helps reduce duplication while also leaving both functions in place.
- Create a `messages` map to associate each of the received names (as a key) with a generated message (as a value). In Go, you initialize a map with the following syntax: `make(map[`*key-type*`]`*value-type*`)`. You have the `Hellos` function return this map to the caller. For more about maps, see Go maps in action on the Go blog.

- Loop through the names your function received, checking that each has a non-empty value, then associate a message with each. In this `for` loop, `range` returns two values: the index of the current item in the loop and a copy of the item's value. You don't need the index, so you use the Go blank identifier (an underscore) to ignore it. For more, see The blank identifier in Effective Go.

2. In your hello/hello.go calling code, pass a slice of names, then print the contents of the names/messages map you get back.

   In hello.go, change your code so it looks like the following.

```
package main

import (
    "fmt"
    "log"

    "example.com/greetings"
)
```

```
func main() {
    // Set properties of the predefined Logger, including
    // the log entry prefix and a flag to disable printing
    // the time, source file, and line number.
    log.SetPrefix("greetings: ")
    log.SetFlags(0)

    // A slice of names.
    names := []string{"Gladys", "Samantha", "Darrin"}

    // Request greeting messages for the names.
    messages, err := greetings.Hellos(names)
    if err != nil {
        log.Fatal(err)
    }
    // If no error was returned, print the returned map of
    // messages to the console.
    fmt.Println(messages)
}
```

With these changes, you:

- Create a `names` variable as a slice type holding three names.
- Pass the `names` variable as the argument to the `Hellos` function.

3. At the command line, change to the directory that contains hello/hello.go, then use `go run` to confirm

that the code works.

The output should be a string representation of the map associating names with messages, something like the following:

```
$ go run .
map[Darrin:Hail, Darrin! Well met! Gladys:Hi, Gladys. Welcome!
Samantha:Hail, Samantha! Well met!]
```

This topic introduced maps for representing name/value pairs. It also introduced the idea of preserving backward compatibility by implementing a new function for new or changed functionality in a module. For more about backward compatibility, see Keeping your modules compatible.

Next, you'll use built-in Go features to create a unit test for your code.

< Return a random greeting Add a test >