Journey Before Destination

Jon Calhoun
Twitter: @joncalhoun
Email: jon@calhoun.io

About Me

- Go Time Panelist
- Go course creator
- I write a Go newsletter
- I write Go tutorials
- Working on a Go book

That's Me!

About Me

- Go Time Panelist
- Go course creator
- I write a Go newsletter
- I write Go tutorials
- Working on a Go book

About Me

- Go Time Panelist
- Go course creator
- I write a Go newsletter
- I write Go tutorials
- Working on a Go book

About Me

- Go Time Panelist
- Go course creator
- I write a Go newsletter
- I write Go tutorials
- Working on a Go book

Crash course on Go interfaces

Hi there,

In today's email we are going to take a crash course on interfaces in Go, and then next week we are going to dive a bit deeper into them.

If you are already using interfaces in your day-to-day Go code this email can probably be skipped, but I would suggest you skim next week's email as you might learn something new reading it 😀

First, let's start by talking a bit about how Go differs from dynamic languages like JavaScript, Ruby, and Python. When creating functions in dynamic languages, you don't define the type you expect to recieve. Instead, anything can be passed in. Below is an example of this using JavaScript.

```
function Greeting(name) {
  return "Hello, " + name
}

// The following is all valid in JS, even if it doesn't make much sense.
Greeting("Jon")
// Output: "Hello, Jon"

Greeting(123)
// Output: "Hello, 123"

Greeting(document)
// Output: "Hello, [object HTMLDocument]"
```

In Go you have to explicitly state types, which means when you are creating a function you have to state that you want the name argument to be a string.

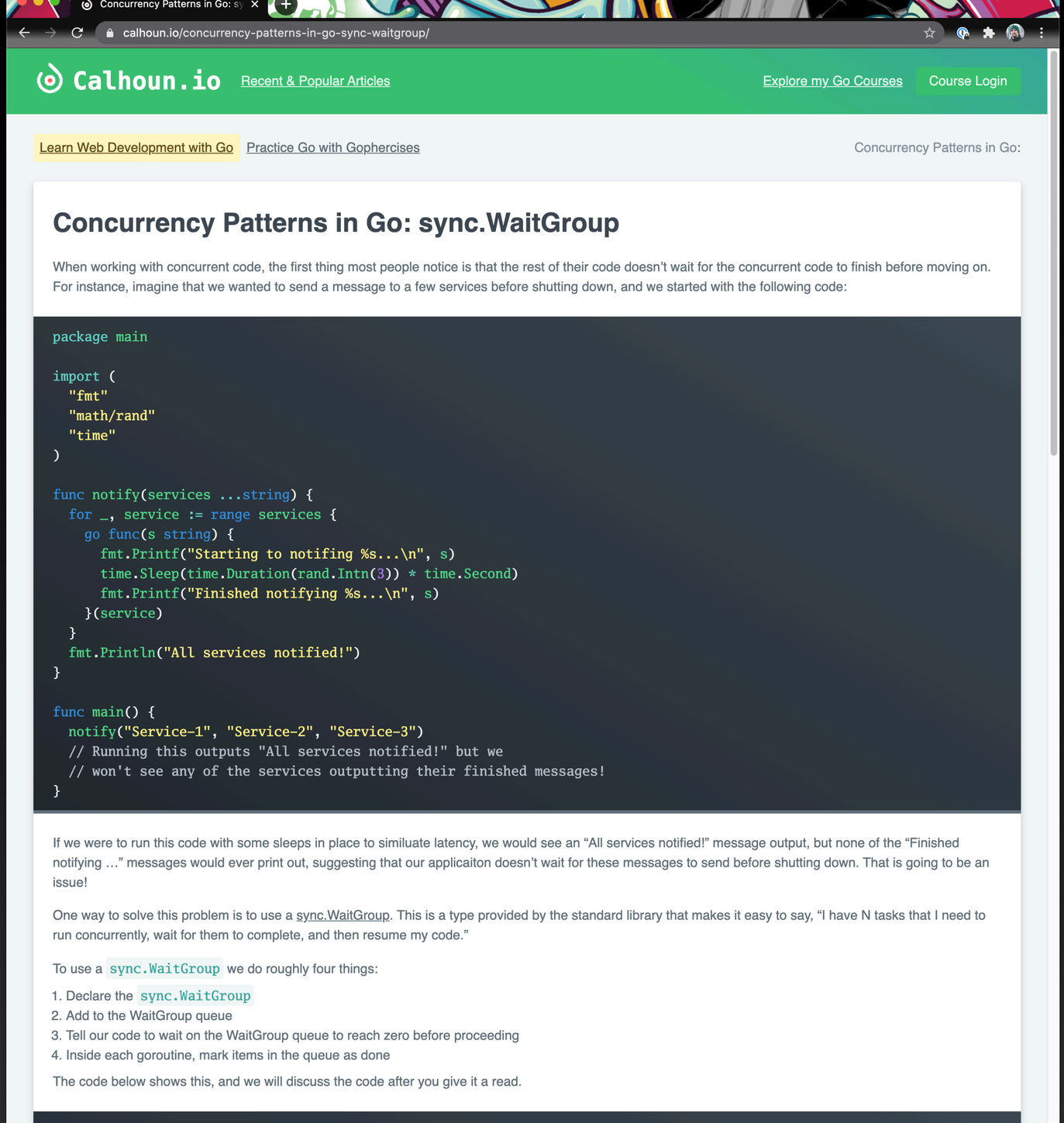
```
func Greeting(name string) string {
  return fmt.Sprintf("Hello, %v!", name)
}
```

For the most part this is a good thing. It makes it clear to anyone reading your code exactly what you expect, and it prevents other developers from accidentally passing in the wrong thing.

If you have ever used a dynamic language, chances are you have experienced errors because someone passed the wrong thing into a function - like passing in the entire document in the JS example! This type of error is far less likely to occur in Go because of the static type system.

About Me

- Go Time Panelist
- Go course creator
- I write a Go newsletter
- I write Go tutorials
- Working on a Go book



About Me

- Go Time Panelist
- Go course creator
- I write a Go newsletter
- I write Go tutorials
- Working on a Go book

I help people learn Go in a lot of ways

Learning to code is a means to an end

It is a mistake to rush to the end

Learning to code shouldn't suck

Seriously, don't bang your head off the wall

Discover what you love

Learn about alternative approaches

```
class FriendStatus extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isOnline: null };
    this.handleStatusChange = this.handleStatusChange.bind(this);
  }

  componentDidMount() {
    ChatAPI.subscribeToFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
  componentWillUnmount() {
    ChatAPI.unsubscribeFromFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
  handleStatusChange(status) {
    this.setState({
      isOnline: status.isOnline
    });
  }

  render() {
    if (this.state.isOnline === null) {
      return 'Loading...';
    }
    return this.state.isOnline ? 'Online' : 'Offline';
  }
}
```

```
class FriendStatus extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isOnline: null };
    this.handleStatusChange = this.handl
  }

  componentDidMount() {
    ChatAPI.subscribeToFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
  componentWillUnmount() {
    ChatAPI.unsubscribeFromFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
  handleStatusChange(status) {
    this.setState({
      isOnline: status.isOnline
    });
  }

  render() {
    if (this.state.isOnline === null) {
      return 'Loading...';
    }
    return this.state.isOnline ? 'Online' : 'Offline';
  }
}
```

```
componentDidMount() {
  ChatAPI.subscribeToFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}

// ... this could be separated by a lot of code

componentWillUnmount() {
  ChatAPI.unsubscribeFromFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}
```

```
useEffect(() => {
  // Teardown and setup are grouped together inside a single useEffect!
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  // Specify how to clean up after this effect:
  return function cleanup() {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

```
useEffect(() => {
  // Teardown and setup are grouped together inside a single useEffect!
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  // Specify how to clean up after this effect:
  return function cleanup() {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

```
useEffect(() => {
  // Teardown and setup are grouped together inside a single useEffect!
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  // Specify how to clean up after this effect:
  return function cleanup() {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

```
useEffect(() => {
  // Teardown and setup are grouped together inside a single useEffect!
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  // Specify how to clean up after this effect:
  return function cleanup() {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

Why are we looking at JS code?

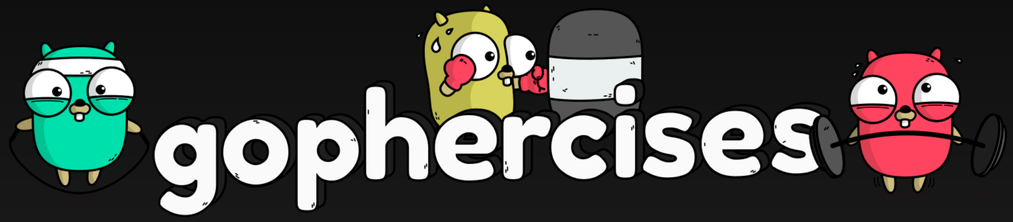
```
// Source code stolen shamelessly from Mat Ryer =D
func NewTimer(name string) func() {
  start := time.Now()
  return func() {
    metric.recordTime(name, start, time.Now())
  }
}

// Using NewTimer
func Something() {
  stop := metric.NewTimer("something")
  defer stop()
  // do work
}
```

Planning isn't doing

Practice, practice, practice

Practice, practice, practice



exercism

HackerRank

<YOUR PROJECT>

+ SO MANY MORE OPTIONS

Use an iterative approach

Learn with others

Journey Before Destination

Jon Calhoun
Twitter: @joncalhoun
Email: jon@calhoun.io