– Go –

A Distributed-First Language

Adam Sanghera // April 20, 2018

– Go –
A Distributed-First Language

Topics

- Origins
- Status Today
- Syntax & Features
  - Runtime & Stdlib **Example 0**
  - Accessible Concurrency & **Example 1**
  - Error-Handling & Call-Stack & **Example 2**
- Dev and Workflows
  - Tooling
  - Interface-driven design
  - Microservices with gRPC

– Go –
A Distributed-First Language

Origins (1 of 2)

- Google
  - Lots of big, distributed systems (Borg, Chubby, BigTable, Spanner, … Skynet?)
  - Mostly written in **C/C++**, or **Java**
  - Pain points with these languages:
    - :(Error handling kind of sucks
    - :(Concurrency primitives hard to reason about
    - :(Fast code tends to be unruly, dense, illegible
    - :(Steep/divergent learning curve(s)

– Go –
A Distributed-First Language

Origins (2 of 2)

- Took survey of options out there (Erlang, Python, Lua, Ruby) and found them variously lacking †
- Ideal language would be:
  - Terse & legible
  - Statically-typed
  - Fast to develop with
  - Scalable (?)
  - Good at networking, asynchronous, & concurrency
- Russ Cox, Robert Griesemer, Rob Pike, Ian Tayler, Ken Thompson got together build this language ~2009.

– Go –
A Distributed-First Language

Status Today

- $HIP_LANG
- Written-in-go:
  - Kubernetes (Container orchestrator / “datacenter OS”)
  - Docker (container daemon & run-time)
  - InfluxDB (“self-contained, distributed time-series DB”)
  - CockroachDB (open source SQL database)
- Companies using go (besides Google duh):
  - Dropbox, CloudFlare, Microsoft, Netflix, MongoDB, Heroku, Uber, Twitch.tv, Hyperleger, …

– Go –

Syntax & Features

Adam Sanghera // April 20, 2018


Example
Package 0
~
Hello World

– Go –
A Distributed-First Language

Syntax & Features – Accessible Concurrency (1 of 3)

– Key ideas
  – "Go Routines", M:N (M ≤ N) related to OS Threads
  – One process, many routines? Channels
  – Many processes? gRPC

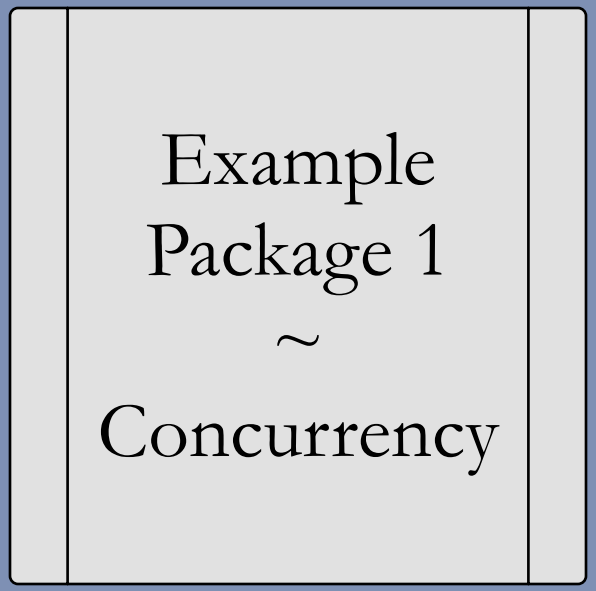
– Go has traditional sync tools (mutex, semaphore, etc.)
– Also has some higher-level tools (channels)
– Channels pass ownership
– Mutexes etc. control access

– Go –
A Distributed-First Language

Syntax & Features – Accessible Concurrency (2 of 3)

– High-performance “async” event loop in 10 LoC

```
func EventLoop(eventChannel chan Event) {
   for event := range eventChannel {
      switch event.type {
         case “get”:
            go handleGet(event)
         case “post”:
            go handlePost(event)
         // …
      }
   }
}
```


Example
Package 1
~
Concurrency

– Go –
A Distributed-First Language

Syntax & Features – Accessible Concurrency (3 of 3)

- Takeaways:
  - Spawning routines is easy
  - Effective parallelism, minimal effort
  - Channels are surprisingly useful
  - Garbage collector, efficient runtime scheduler empowers developers to write performant code sans necromancy

– Go –
A Distributed-First Language

Syntax & Features – Error-Handling & Call-Stack (1 of 3)

– Philosophy: (1) errors are **data**, treat them as such,
(2) error-handling should not impact on legibility
– Result: a typical function definition

```
func doSomethingRisky() (outcome, err) { /* risky biz */}
```

– If the function succeeds, err will be `nil` (similar to `null`)

```
result, err := doSomethingRisky()
if err != nil {
   panic(err)
}
```

– Go –
A Distributed-First Language

Syntax & Features – Error-Handling & Call-Stack (2 of 3)

– Panic blows down the call stack until *recovered*. If recover is never called, the run-time will catch it – and call Exit()

– However, sometimes you **want** to recover. Pattern:

```
func riskTaker() (outcome, err) {
   defer func(){
      if err := recover(); err != nil {
         if canSurvive(err) {
            handleErr(err)
         } else {
            panic(err)
         }
      }
   }
   result, err := doSomethingRisky()
   // …
```

– Go –
A Distributed-First Language

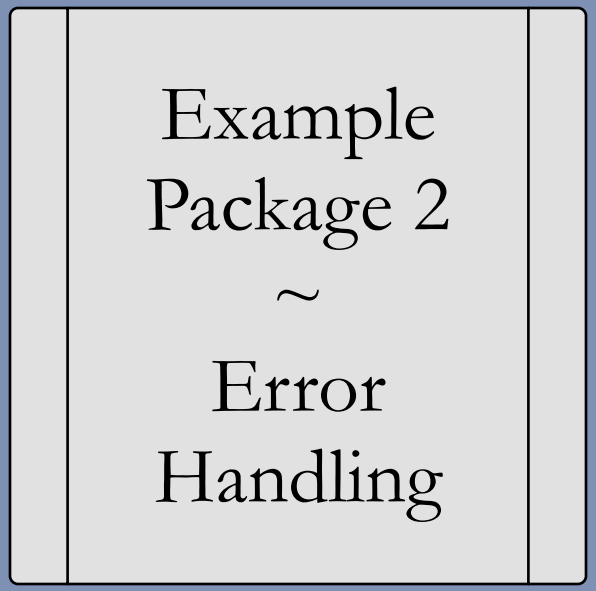
Syntax & Features – Error-Handling & Call-Stack (3 of 3)

– **defer** slips a function into the call stack
– Useful for handling errors, other things too:

```
mut.Lock()
defer mut.Unlock()

start := time.now()
defer log.Printf("Took %5.1f seconds", time.Since(start))
```

– Allows the developer to segregate specific variety of code, which tends to be as important as it is dry/ubiquitous.


Example
Package 2
~
Error
Handling

– Go –

Development & Workflows

Adam Sanghera // April 20, 2018

– Go –
A Distributed-First Language

Dev and Workflows – Tooling/Workflow (1 of 2)

– Package managers (pip, npm, apt-get, etc.) are handy!
  + go get
– Maintaining a consistent format is good for legibility
  + go fmt
– Wouldn't it be nice to auto-detect race conditions?
  + go run -race
– Run all of my unit tests, with cached results
  + go test
– Importing big packages creates bloated binaries
  + unused import is a compiler error

– Go –
A Distributed-First Language

Dev and Workflows – Tooling/Workflow (2 of 2)

– Managing dependencies can be a nightmare
  + dep init
– Linking packages can be fragile (relative/absolute paths)
  + coerces developer to root projects at $GOPATH/src
– Modularity and Compose-ability are powerful
  + given above, managing many-small packages is easy
– Compilation is good, but compile+run is annoying
  + go run
– Tell me how many ns are spent per routine/http-request
  + `go test -trace` | Zipkin

– Go –
A Distributed-First Language

Dev and Workflows – Interface-Driven Design (1 of 2)

– A **type** can be either a **struct** or an **interface**
– A **struct** can implement an **interface** implicitly

– Interfaces are typically the central element of a package
– Complex interfaces/structs are typically compositions of smaller, straightforward ones (Kubernetes for example)
– Trivial to *mock* complex packages, for end-to-end testing

– Idiomatic Go is **functional** and minimalist, with concise communication patterns. See standard library.

– Go –
A Distributed-First Language

Dev and Workflows – Interface-Driven Design (2 of 2)

– Takeaways of Interface-driven Design:
  – Encourage top-down engineering, simultaneously keeps code **easy to read**
  – Packages (stdlib included) are easily extensible
  – Every Go file layout is similar, helps snowball learning
  – Rich community of high-quality open source packages available to transfer-learn from

– Go –
A Distributed-First Language

Dev and Workflows – Microservices with gRPC (1 of 2)

– Creating a gRPC microservice in Go takes ~50 LoC
– Easy to mock and test client/server, leveraging interfaces
– Modular package design makes it easy to slap a high-performance gRPC server onto a local package
– Tracing tools make it trivial to centralize/analyze distributed logs, automatically generated by `-trace`
– Want to run in containers? Compiled, statically-linked binaries on minimal linux kernel yields containers ~**15MB**

– Go –
A Distributed-First Language

Go in Action – Microservices with gRPC (2 of 2)

– Takeaways:
  – Concise, performant networking package provides excellent primitives for distributed-systems building
  – Abstraction level is high. High level of abstraction means that there is usually only "one right way", accelerates development speed. GC is great. Very fast.
  – Minimal stdlib keeps import footprints very low
  – Great community, pumps out super high-quality tooling

– Go –
A Distributed-First Language

Done! Here is a bunch of links/sources/further reading

The Go Blog (and Rob Pike's various talks over the years) are tier-1
https://blog.golang.org/error-handling-and-go
https://blog.golang.org/http-tracing
https://blog.golang.org/concurrency-is-not-parallelism
https://golang.org/doc/faq <– absolutely tier 1 resource
Language sources of inspiration: https://talks.golang.org/2014/hellogophers.slide#21
https://web.stanford.edu/class/ee380/Abstracts/100428-pike-stanford.pdf
https://blog.golang.org/profiling-go-programs
https://blog.golang.org/race-detector
https://blog.golang.org/organizing-go-code
https://blog.golang.org/share-memory-by-communicating

less-official gold nuggets:
https://astaxie.gitbooks.io/build-web-application-with-golang/content/en/
https://blog.codeship.com/building-minimal-docker-containers-for-go-applications/
https://changelog.com/gotime
https://github.com/kubernetes/kubernetes
https://github.com/ethereum/go-ethereum
https://github.com/grpc/grpc-go
https://github.com/hashicorp/raft

Q & A