go - aka golang

an uN4U7HOr123d PrOof OF ConC3p7

Craig Thomas (craig.thomas@bhnetwork.com)

Objectives of the brown bag

1. why another language
2. golang design characteristics
3. an experience report from a poc
   3.1. goals of the poc
   3.2. some code
   3.3. some demos
4. wrap

Language Design in the Service of Software Engineering

Rob Pike at the SPLASH 2012

(From keynote talk given by Rob Pike at the SPLASH 2012 conference in Tucson, Arizona, on October 25, 2012.)

Conceived in **late 2007** as an answer to some of the problems we were seeing developing software infrastructure at Google.

The computing landscape today is almost unrelated to the environment in which the languages being used, mostly C++, Java, and Python, had been created.

The problems introduced by multicore processors, networked systems, massive computation clusters, and the web programming model were being worked around rather than addressed head-on.

Moreover, the scale has changed: today's server programs comprise tens of millions of lines of code, are worked on by hundreds or even thousands of programmers, and are updated literally every day.

To make matters worse, build times, even on large compilation clusters, have stretched to many minutes, even hours.

Go was designed and developed to make working in this environment more productive. Besides its better-known aspects such as built-in concurrency and garbage collection, Go's design considerations include rigorous dependency management, the adaptability of software architecture as systems grow, and robustness across the boundaries between components.

golang design

- A syntax and environment adopting patterns more common in dynamic languages:
  - Concise variable declaration and initialization through type inference (x := 0 not int x = 0;).
  - Fast compilation times
  - Remote package management (go get) and online package documentation (godoc)

golang design

- Distinctive approaches to particular problems:
  - Built-in concurrency primitives: light-weight processes (goroutines), channels, and the select statement.
  - An interface system in place of virtual inheritance, and type embedding instead of non-virtual inheritance.
  - A toolchain that, by default, produces statically linked native binaries without external

golang design

- A desire to keep the language specification simple enough to hold in a programmer's head, in part by omitting features common to similar languages:
  - no type inheritance
  - no method or operator overloading
  - no circular dependencies among packages
  - no pointer arithmetic
  - no assertions
  - no generic programming

Experience report from poc

Goals of the poc

- Serve “standard” data sets with REST and json
  - Fed’s ACH data
  - ISO country, currency, language
- Not so standard sources:
  - Fed: fixed format text file served on the web
  - ISO country: gotta buy (or scrape from wikipedia)
  - ISO currency: XML served on the web
  - ISO language: pipe-delimited .csv file served on the web

some code

- https://github.com/musicbeat/stddata/blob/master/bank/bankprovider.go
- https://github.com/musicbeat/stddata/blob/master/country/countryprovider.go
- https://github.com/musicbeat/stddata/blob/master/curren

some code

- https://github.com/musicbeat/stddata/blob/master/stddata.go

demos

demos

- ➔ go get github.com/musicbeat/stddata
- ➔ go get github.com/musicbeat/stddata-cli
- ➔ cd $GOPATH/src/github.com/musicbeat/stddata/currency
- ➔ go test
- ➔ cd $GOPATH/src/github.com/musicbeat/stddata-cli
- ➔ go run stddata-cli.go
- ➔ browse http://localhost:6060
- ➔ godoc -http=:6062
- ➔ browse http://localhost:6062

“I like a lot of the design decisions they made in the [Go] language. Basically, I like all of them.”

–Martin Odersky, creator of Scala