Golangで支える
ハイパフォーマンスAPI

Kazuhisa Togo

LIG INC. × 

LIGにおけるGolangへの取り組み

“Gopher” illustration by © Renee French

LIG INC. × 

自社サービスのバックエンドに Golangを採用しています

“Gopher” illustration by © Renee French

LIG ID × 

マイクロサービスの橋渡しを担う統合API

“Gopher” illustration by © Renee French

開発中のAPI

×

膨大なリクエストを捌く大規模API

“Gopher” illustration by © Renee French

LIG ID

×

開発中のAPI

100% Golangで開発されています

“Gopher” illustration by © Renee French

なぜGolangなのか

LIG ID

LIGの統一アカウントサービス

LIG ID

OAuth 2 サーバー + API

LIG ID

あらゆるサービスのログイン処理を担当

LIG IDがダウン ＝

全サービスでログイン不可に

認証処理 ＝

セキュリティーホールは許されない

LIG ID

ミッションクリティカル

開発中のAPI

膨大な同時リクエスト

開発中のAPI

高負荷に耐えるパフォーマンスが必要

開発中のAPI

運用コストも減らしたい

開発中のAPI

単位性能の向上 =

サーバーの台数削減

開発中のAPI

サービスの停止は
大規模な機会損失に繋がる

開発中のAPI

信頼性が必要不可欠

開発中のAPI

長期的に運用 =

開発のしやすさも重要

高い信頼性

ハイパフォーマンス

開発のしやすさ

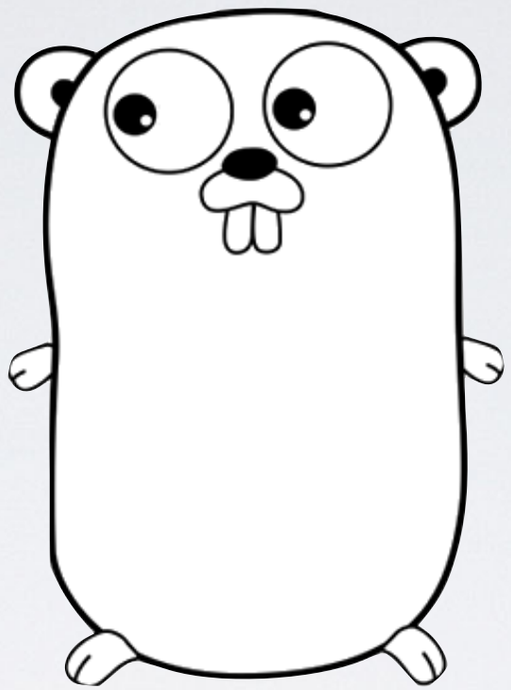
高い信頼性

ハイパフォーマンス

開発のしやすさ

＋

新しい言語を採用するなら
学習コストも抑えたい



Golangを採用しました

“Gopher” illustration by © Renee French

Golangの特徴

Golangの特徴

高い信頼性

Golangの特徴

コンパイラー言語

プログラムを予め機械語に翻訳
（コンパイル）

Golangの特徴

コンパイラー言語

コンパイルついでに
自動で処理の最適化まで行う

Golangの特徴

コンパイラー言語

機械語を直接実行するので
とても高速

Golangの特徴

コンパイラー言語

コンパイル時にコンパイルエラーを検出

バグがかなり減る

スクリプト言語あるある

Fatal error: Call to undefined function doSomething()

未定義の関数を呼び出し

スクリプト言語あるある

Fatal error: Call to undefined function doSomething()

実行されるまで分からない

スクリプト言語あるある

Fatal error: Call to undefined function doSomething()

Golangなら

コンパイル時に検出可能

Golangの特徴

静的型付け

全ての変数のデータ形式（型）が固定

“Gopher” illustration by © Renee French

Golangの特徴

静的型付け

型が一致しないとコンパイルエラー

実行時エラーを阻止

スクリプト言語あるある

Fatal error: Call to a member function someMethodCall() on array

配列に対してメソッド呼び出し

スクリプト言語あるある

Fatal error: Call to a member function someMethodCall() on array

これも実行するまで分からない

スクリプト言語あるある

Fatal error: Call to a member function someMethodCall() on array

Golangなら

コンパイル時に検出可能

ユニットテストの標準サポート

実行時しか分からないロジックエラーを
機械的に自動でテスト

Golangの特徴

ユニットテストの標準サポート

go test コマンド

ユニットテストの標準サポート

*_test.go ファイルを置いて

テストコードを記述

Golangの特徴

ユニットテストの標準サポート

標準で用意されている

testing パッケージを利用

ユニットテストの標準サポート

コンソールで *go test* を実行

＝ テストが行われる

Golangの特徴

ユニットテストの標準サポート

ベンチマークも標準対応

go test -bench

Golangの特徴

ユニットテストの標準サポート

積極的にテストを書くようになった
（当社比）

Golangの特徴

開発のしやすさ

“Gopher” illustration by © Renee French

Golangの特徴

学習コストの低さ

できるだけ複雑なコードを避けた設計

Golangの特徴

学習コストの低さ

非常に短期間で学習可能

Golangの特徴

学習コストの低さ

弊社例：経験者ゼロの状態から

3週間でサービスローンチ

Golangの特徴

開発スピード

静的型付けだが、短い構文で表現可能

Golangの特徴

開発スピード

コンパイルエラーが検出できるので
実行時エラーが少なくなる

Golangの特徴

開発スピード

同じ品質のものを目指せば

決して遅くない

“Gopher” illustration by © Renee French

ISUCON

ISUCON

Iikanjini Speed Up CONtest

（いい感じにスピードアップコンテスト）

ISUCON

バックエンド界のスピードレース

ISUCON

限られた時間内で、どこまで
パフォーマンスが上げられるかを競う

ISUCON

限られた時間内で、どこまで
パフォーマンスが上げられるかを競う

ISUCON

限られた時間内で、どこまで
パフォーマンスが上げられるかを競う

↓

開発スピードが命

ISUCON

ISUCON本戦

利用言語ランキング

ISUCON

	ISUCON 3 (2013)	ISUCON 4 (2014)	ISUCON5 (2015)
1位	Perl 44%	Ruby 33.3% 10組	Ruby 44% 11組
2位	Ruby 36%	Go 30.0% 9組	Go 32% 8組
3位	Python 12%	Perl 20.0% 6組	Perl 28% 7組
4位	Go Node.js 4% PHP	PHP 16.6% 5組	LUA Shell 4% 1組 Javascript

Ref: http://isucon.net

ISUCON

	ISUCON 3 (2013)		ISUCON 4 (2014)			ISUCON5 (2015)		
1位	Perl	44%	Ruby	33.3%	10組	Ruby	44%	11組
2位	Ruby	36%	**Go**	30.0%	9組	**Go**	32%	8組
3位	Python	12%	Perl	20.0%	6組	Perl	28%	7組
4位	**Go** Node.js PHP	4%	PHP	16.6%	5組	LUA Shell Javascript	4%	1組

Golangが2年連続で本戦2位獲得

Ref: http://isucon.net

ISUCON

	ISUCON 3 (2013)		ISUCON 4 (2014)			ISUCON5 (2015)		
1位	Perl	44%	Ruby	33.3%	10組	Ruby	44%	11組
2位	Ruby	36%	**Go**	30.0%	9組	**Go**	32%	8組
3位	Python	12%	Perl	20.0%	6組	Perl	28%	7組
4位	**Go** Node.js PHP	4%	PHP	16.6%	5組	LUA Shell Javascript	4%	1組

つまり開発スピードは十分に速い

Ref: http://isucon.net

Golangの特徴

パフォーマンス

Golangの特徴

実行速度

APIで肝になるのがレスポンスの速度

“Gopher” illustration by © Renee French

Golangの特徴

実行速度

処理時間が短ければ

単位時間あたりにより多く捌ける

Golangの特徴

実行速度

Golangはコンパイラー型なので

実行速度は著しく速い

Golangの特徴

実行速度

簡単な処理ならマイクロ秒レベル

実行速度

弊社例：実際に開発中のAPIは

1ms未満で処理を終えることも

Golangの特徴

実行速度

仮に200msを20msに減らせれば

単位時間あたり10倍のリクエストを捌ける

Golangの特徴

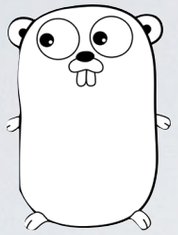
実行速度

APIに最適

“Gopher” illustration by © Renee French

どのように設計するか

“Gopher” illustration by © Renee French



どのように設計するか

パフォーマンスの向上

“Gopher” illustration by © Renee French

高速なパッケージの選択

実装によってパフォーマンスは
桁違いに異なる

高速なパッケージの選択

WAFの実例：*Martini* は非常に遅く

近年主流の *Echo* や *Gin* などは速い

高速なパッケージの選択

複数のパッケージを
ベンチマークでテストする

高速なパッケージの選択

わりとベンチマーク結果も
多く出回っています

Golangの特徴

fasthttp

実例

Golangの特徴

fasthttp

標準ライブラリよりも高速化した

HTTPサーバー

Golangの特徴

fasthttp

標準の *net/http* は

メモリアロケーションが非効率

Golangの特徴

fasthttp

メモリ効率を抜本的に改善

Golangの特徴

高速なパッケージの選択

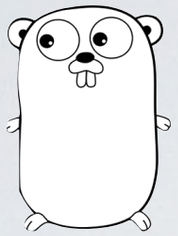
超高頻度なリクエストに対する
パフォーマンスは数倍改善

Golangの特徴

高速なパッケージの選択

本家 *net/http* よりはメンテ頻度低め

導入は慎重に



どのように設計するか

テストファースト

独立性の高い実装

TDDとは言わないまでも

テストはできる限り100%行いたい

独立性の高い実装

Golangはパッケージ単位で

細かくユニットテストを書く仕様

Golangの特徴

独立性の高い実装

DBアクセスを抽象化するなど、

依存のない実装を目指す

Golangの特徴

独立性の高い実装

DBアクセスのテストなどは

Dockerを利用

Golangの特徴

独立性の高い実装

テストコードから

Dockerが立ち上げられる

“Gopher” illustration by © Renee French

Golangの特徴

独立性の高い実装

テストごとにDockerコンテナを立ち上げ

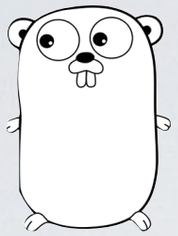
DBを初期化してテスト

Golangの特徴

独立性の高い実装

ここまでGoのコードだけで実現できます

“Gopher” illustration by © Renee French



どのように設計するか

デプロイメント

Golangの特徴

もっとも簡単なデプロイ

ビルドしたバイナリを
突っ込むだけ

“Gopher” illustration by © Renee French

Golangの特徴

デーモン化

デーモン化するパッケージを利用

Golangの特徴

デーモン化

またはエージェントを用いて

間接的にデーモン化

Golangの特徴

Docker

Dockerでコンテナ作成

コマンドひとつでどこでも実行

Golangの特徴

CIでビルド

ローカルでのビルドの代わりに

CIでテスト&ビルド

Golangの特徴

CIでビルド

CircleCIなどはGolangに対応

GitHubへのPushで自動ビルド

Golangの特徴

CIでビルド

ビルド後は自動デプロイ

Push以後は全て自動化

Golangの特徴

Amazon Web Service

AWSならCode Deployなどを利用

Golangの特徴

Google App Engine

GAEならコンソールから

数回のコマンドでデプロイ

時間があれば話すこと

時間があれば話すこと

- Goで良かったこと
  - 並列処理が容易 → 簡単にCPU性能を使い切れる
  - 標準ライブラリが充実 → SMTPやHTTPのサーバー、暗号化から画像処理まで！
  - 外部ライブラリの導入が非常に楽 → githubなどのリポジトリURLを指定するだけ
  - 依存関係の悩みがない → Go 1.5からVendoringに対応
  - 標準でテスト機能を内蔵 → ユニットテストやベンチマーク、コードの解析・検証まで
  - 標準でソースコードのコメントからドキュメントを自動生成
- 他のコンパイル型言語と比較した特徴
  - コンパイルが高速 → 効率的な依存関係解析や階層構造を持たない型など、高速にコンパイルできる設計
  - ガベージコレクション → メモリーの解放に頭を悩ませなくて済む。go1.5で大幅に高速化
  - 実行ファイル1つに全てをパッキング → ビルドさえすれば依存関係を気にしなくていい

時間があれば話すこと

- Goの設計思想 → 断捨離ベース。バグの生みにくさと理解のしやすさ、高速さを重視した設計
  - FAQがとても充実 → 設計思想が書き連ねてある
  - 例外がない → 「例外はコードを複雑にするため」
  - 関数や演算子のオーバーロードがない → 「混乱やバグの元。無いほうがずっとシンプル」
  - ジェネリクスがない → 「将来的には導入の可能性。現在はベストな設計を模索中」
  - 標準のテストスイートにアサーションがない → 「正しいエラー処理とレポートの記述を促すため」
- Goにおける問題点
  - interface{} → 型システムの利点が削がれる、やっぱりジェネリクス欲しい
  - 細かい仕様まで理解しないと思わずバグることはまだある → Sliceの仕様は罠
  - gccgoの開発が間に合ってない → Go1.5以降はビルド済みバイナリを導入するか1.4系のインストールが必要。そしてgcはgccgoより遅い（それでも十分速いが）

ご静聴ありがとうございました