

Golangでサーバサイド プログラミング勉強会

2018年2月22日&3月2日

@立命館大学BKC



スライド

goo.gl/jv6z3F

[https://drive.google.com/open?
id=1ZKtoB2dwW7BWmzUdEE8gyeie9xGB9I08](https://drive.google.com/open?id=1ZKtoB2dwW7BWmzUdEE8gyeie9xGB9I08)



ScrapBox(スクリプトなど)

<https://scrapbox.io/konatsup/>

Golang Seminar 0302



sli.do(質問箱)

<https://app2.sli.do/event/>

[ajxypgmh/ask](#)



ハッシュタグ

#RCC_Golang



みなさん



おはようございます！！



Golang勉強会へようこそ！

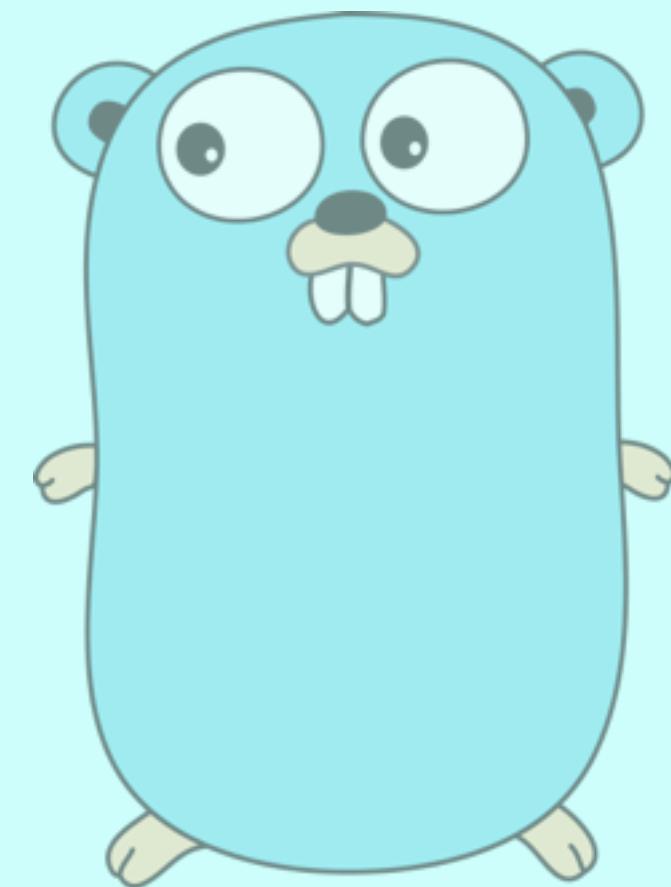


本日のゴール



本日のゴール

Golangと仲良くなる



勉強会の流れ



勉強会の流れ



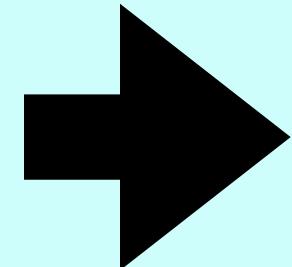
入門



勉強会の流れ



入門



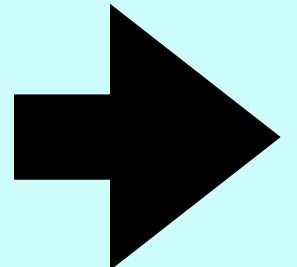
楽しい



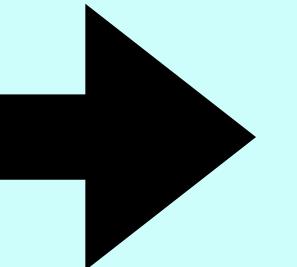
勉強会の流れ



入門



楽しい

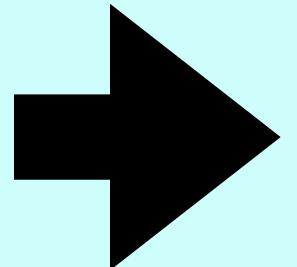


圧倒的成长

勉強会の流れ



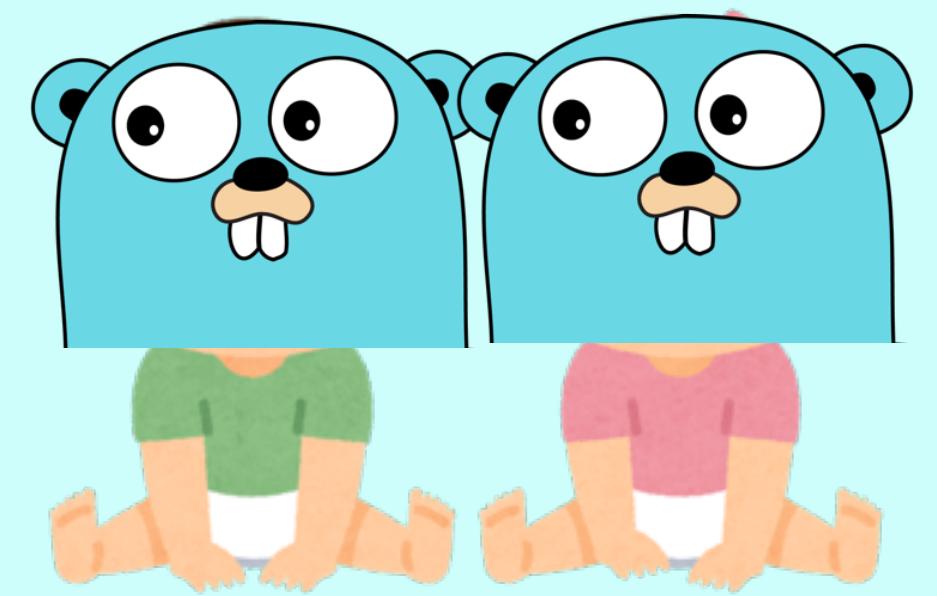
入門



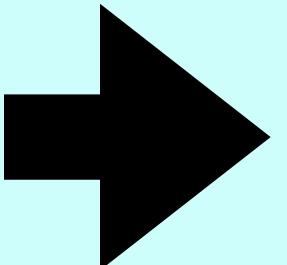
楽しい



圧倒的成长



入門



勉強



楽し

庄倒的成长



使用するもの



使用するもの

- go1.9(<https://golang.org/dl/>)



本日の流れ



本日の流れ

- 10:30~11:00 受付
- 11:00~11:15 イントロダクション
- 11:15~12:30 勉強会(午前の部)
- 12:30~13:30 お昼休憩
- 13:30~18:00 勉強会(午後の部)(途中休憩あり)
- 18:00~ ☆解散☆



內容



勉強会内容

1. goの環境構築
2. goの言語仕様・文法
3. サーバの基本
4. チャットアプリ作成
5. (並行処理を使う)
6. (API作成)



守って欲しい2つこと

守って欲しい2つのこと



1. すぐに質問する



2. 周りに教える



早速ですが始めて行きます！



今日も



- □





頑張るぞい！

よろしくお願いしまーす！



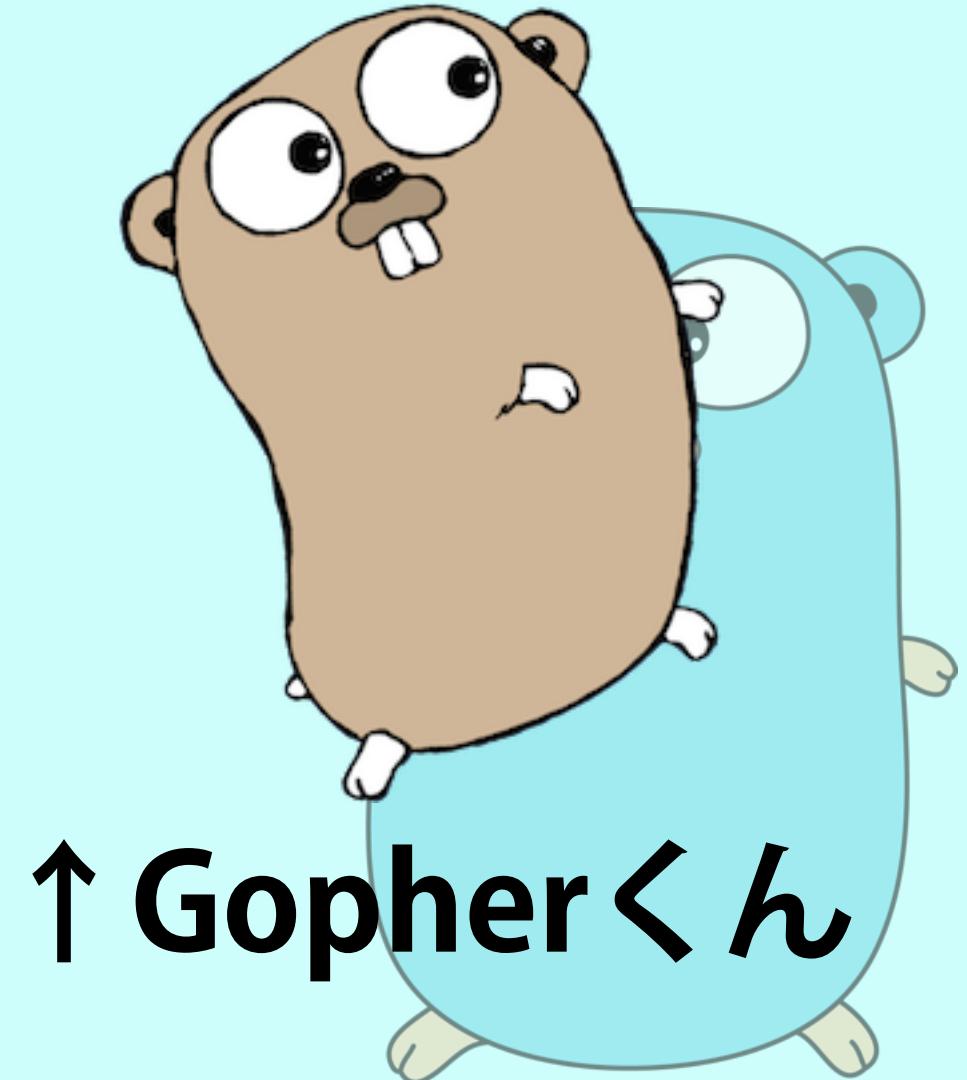
0. Golangについて

Golangとは？



Golang(Go言語)とは？

- Googleが開発したプログラミング言語
- 2012年3月ver1.0リリース
- コンパイラ型言語
- バックエンドで使われることが多い

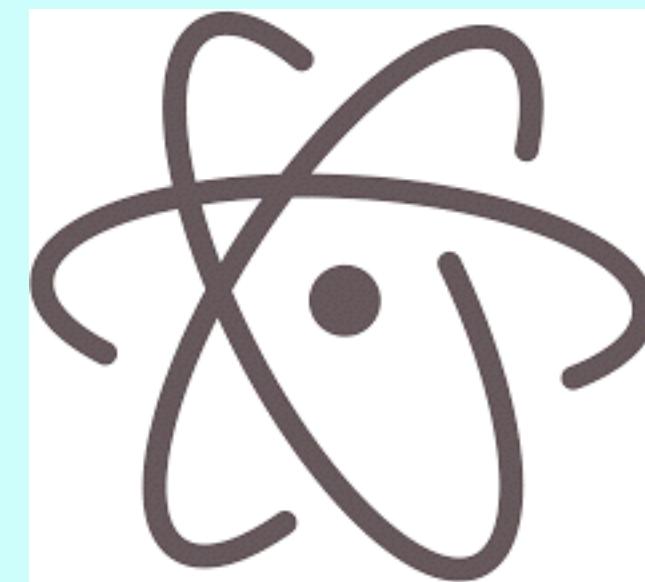


おすすめエディタ(とIDE)



おすすめエディタ(とIDE)

- Atom(プラグインが良い)
- VSCode(プラグインが良い)
- Vim(プラグインが良い)
- GoLand(Golang専用のIDE)



主な参考書籍



主な参考書籍



スターティングGo言語



Goプログラミング実践入門



Go言語によるWeb
アプリケーション開発

Go言語の特徴



Go言語の特徴

1. コンパイル型言語による圧倒的なパフォーマンス
2. マルチプラットフォームでの動作
3. OSへの非依存
4. シンプルな言語
5. 並行処理
6. ガベージコレクタ



1. コンパイル型言語

- ・ネイティブコードへのコンパイルされる
 - ・何らかの実行環境構築不要
- ・C, C++, C#, Java系列
- ・インタプリタ型(ruby, pythonなど)とは違う
- ・実行速度が早い(一般的なスクリプト言語より10~100倍早い)
 - ・VMやインタプリタが必要ないのでオーバーヘッド少



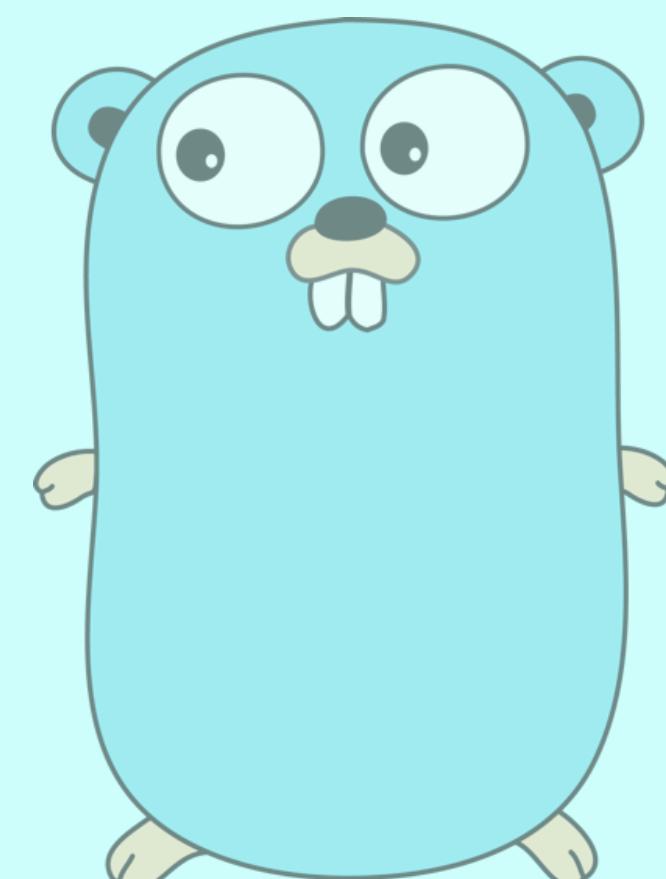
2. マルチプラットフォームでの動作

- OSやCPUによる実行環境の差異をほぼ完全に隠蔽
 - Goの標準ライブラリのみならプラットフォームの差を気にする必要なし
- クロスコンパイル機能搭載
 - 各実行環境で動作するプログラムを1つのコンパイル環境から生成する機能
- デプロイが容易



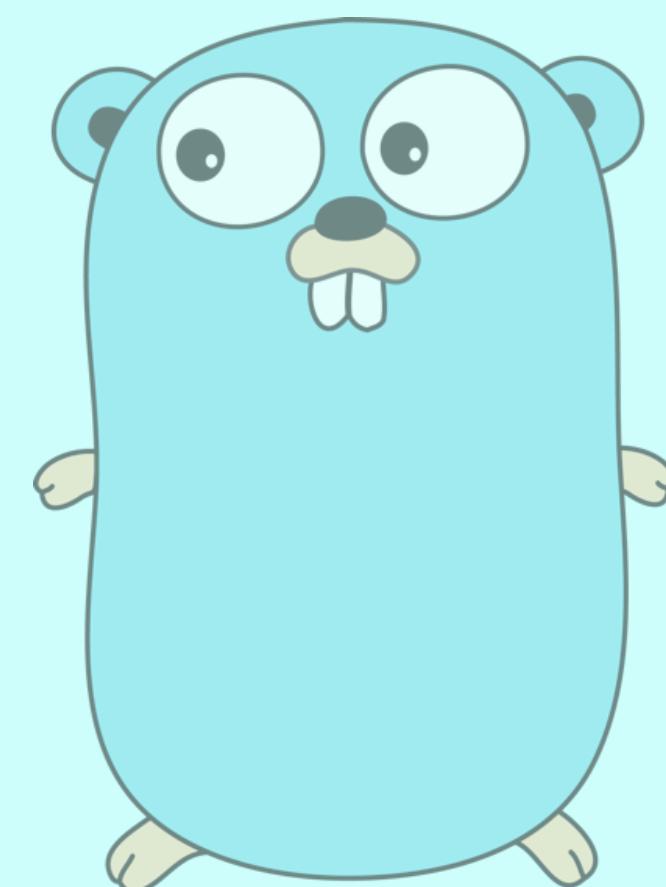
3. OSへの非依存

- ・ 各OSの標準的な共有ライブラリにすら依存しない実行ファイル
- ・ ただ、実行ファイルのサイズが多少大きくなる
 - ・ ストレージ的に問題になることはほぼない
- ・ デプロイが容易
 - ・ 実行ファイルのコピーのみ



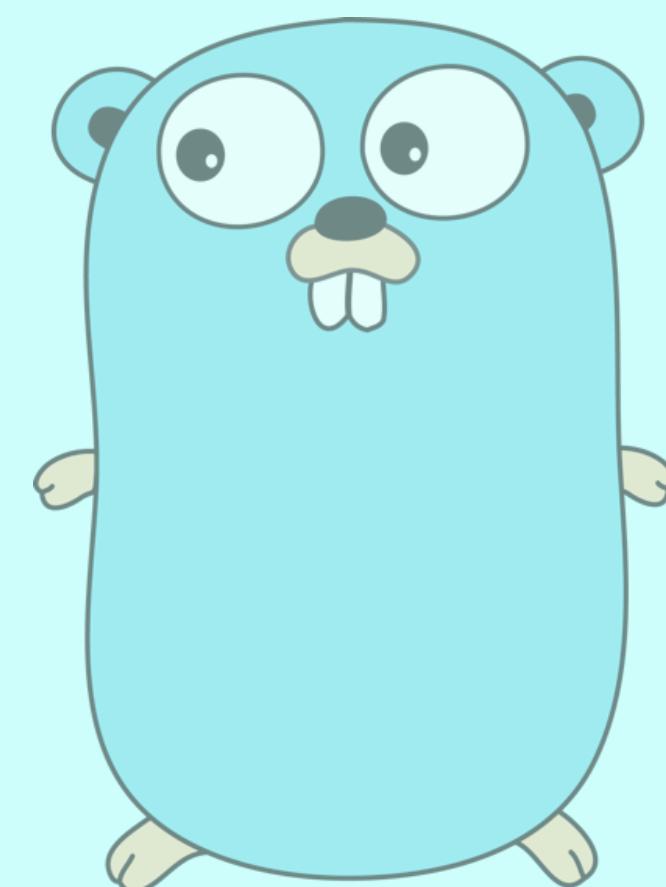
4. シンプルな言語

- オブジェクト指向型ではない(継承がない)
- 標準付属パッケージが豊富
 - net/httpなどサーバ関連のものもサポート
- 文法、実行、テスト、コードの整形などが簡単にできる
- 無駄なインポート文があるとコンパイルエラー(可読性向上)



5. 並行処理

- ・ ゴルーチン(goroutine)という並行処理の枠組み
 - ・ OSが提供するスレッドよりも小さな単位で動作する実行単位
- ・ チャネルというデータ構造を使う
 - ・ 安全なデータ共有
- ・ シンプルかつ効率的に並行処理が実行できる



6. ガベージコレクタ

- ・ 安全なメモリ操作
- ・ C言語におけるメモリリーク、バッファオーバーフローを防止
 - ・ メモリ解放漏れ、意図しないメモリへの書き込み
- ・ ポインタが使える
 - ・ しかし、ポインタ演算はできない(そもそもやる必要性…)
- ・ C言語をやってきてた人におすすめしたい言語



Go言語の特徴

1. コンパイル型言語による圧倒的なパフォーマンス
2. マルチプラットフォームでの動作
3. OSへの非依存
4. シンプルな言語
5. 並行処理
6. ガベージコレクタ



近年の動向

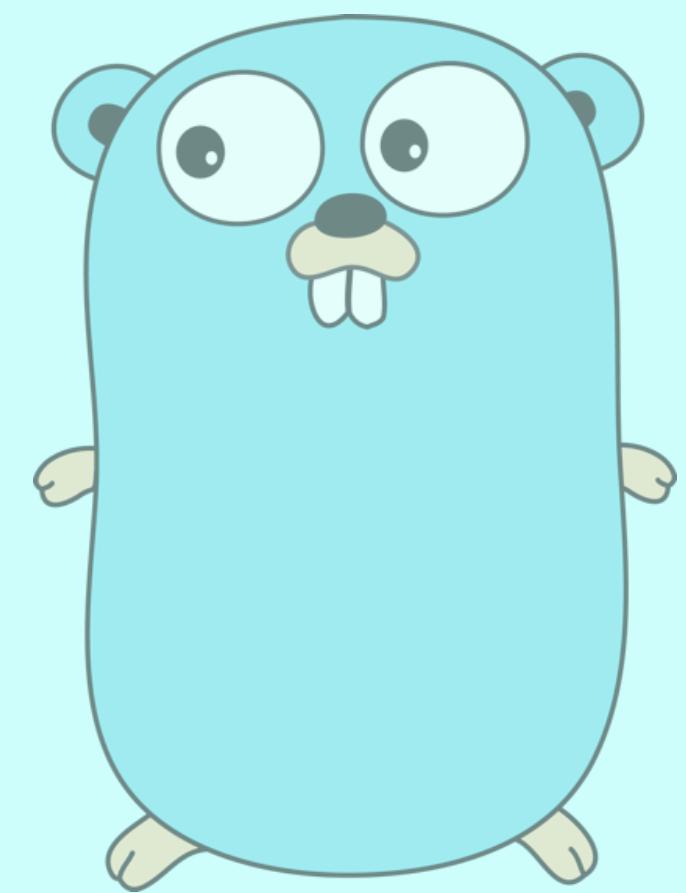


近年の動向

- 王道のフレームワークがまだない
 - しいていえばrevel, gin, Buffaloなど
 - 現在は標準ライブラリのみで頑張ることが多い(らしい)
- ミドルウェアなどのインフラ寄りで使われることが多い
- キーワード：アドテクノロジー、キャッシュサーバ、動画配信



終わり



勉強会内容

- 1. goの環境構築
- 2. goの言語仕様・文法
- 3. サーバの基本
- 4. チャットアプリ作成
- 5. (並行処理を使う)
- 6. (API作成)



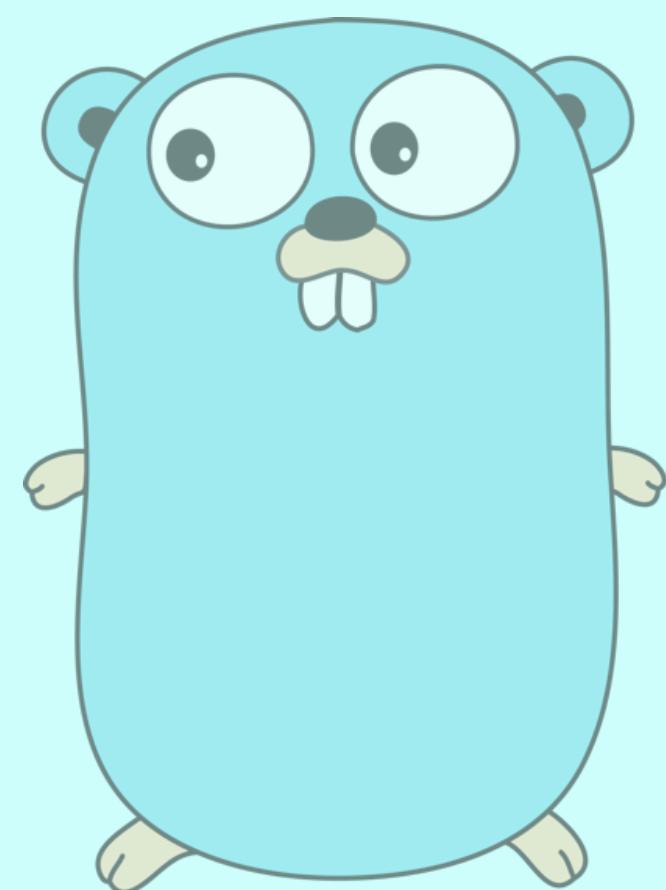
1. goの環境構築

スライドなし
(各自ググってほしい…)



要設定項目

- go ver1.9のインストール
- GOPATHの指定



GoのHPで落としてこよう

<https://golang.org/dl/>



MacOS

- \$ brew install go
- ~/.zshrcまたは~/.bashrcに以下を追加

```
export GOROOT=/usr/local/Cellar/go/1.10/libexec
```

```
export GOPATH=$HOME/go
```

```
export PATH=$PATH:$GOPATH/bin
```



勉強会内容

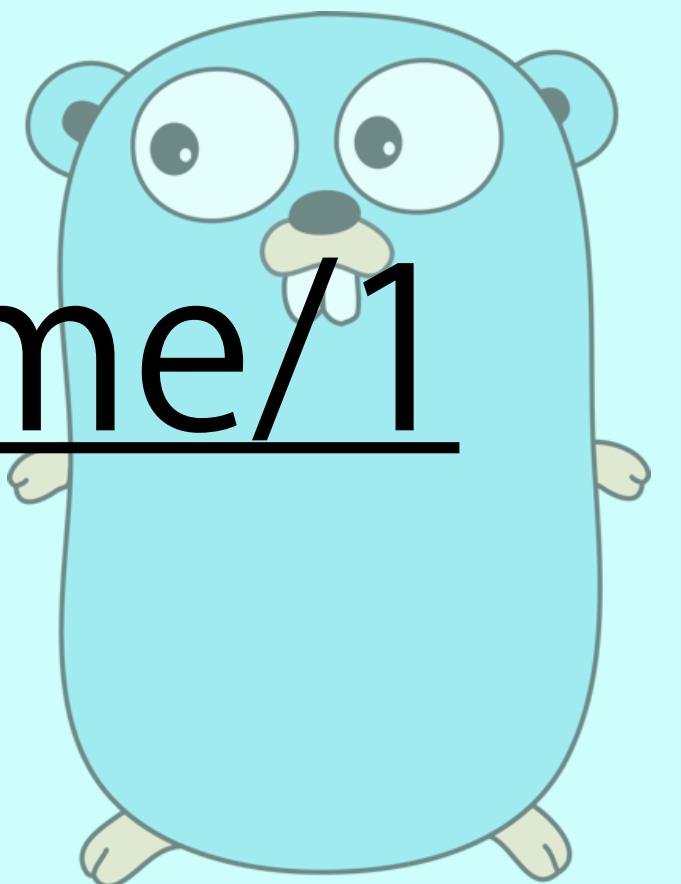
1. goの環境構築
2. goの言語仕様・文法
3. サーバの基本
4. チャットアプリ作成
5. (並行処理を使う)
6. (API作成)



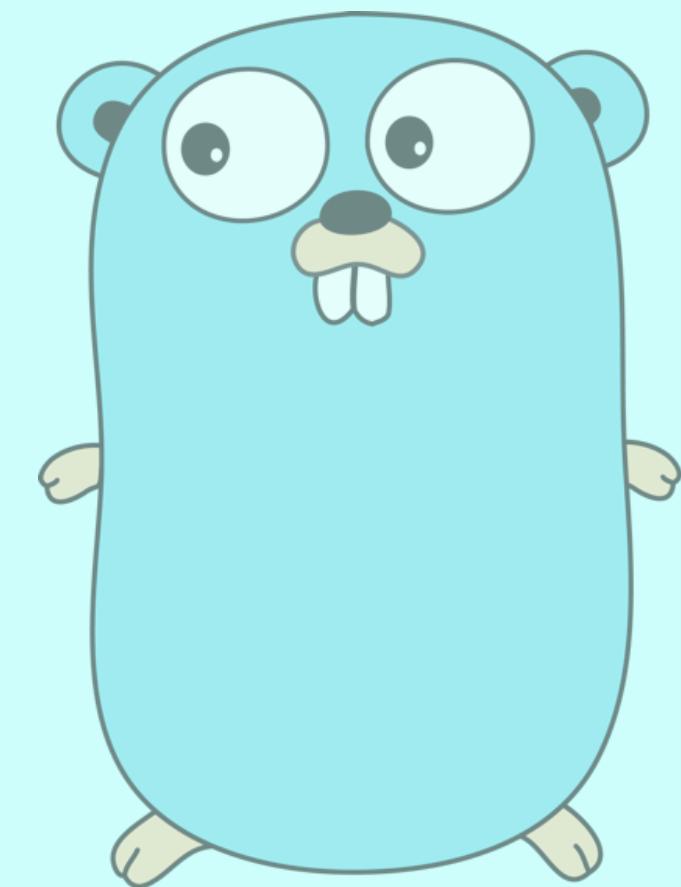
2. goの言語仕様・文法

「A Tour of Go」をやろう！

<https://go-tour-jp.appspot.com/welcome/1>



各種コマンド



各種コマンド

- go run [ファイルパス] : プログラムの実行
- go build : 実行ファイルの作成
- go get [リンク先(urlなど)] : packageのダウンロードとインストール
 - 例 : go get github.com/hoge/hogelibrary
- go fmt : フォーマット整形
- go test [package] : テスト



HelloWorld



Hello World

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello World!")
}
```



実行してみよう

- go run main.go
- またはgo buildしてから実行ファイルを起動



主な文法



主な文法

- 変数定義
- 関数
- if文
- for文
- switch文
- 構造体
- (他にもたくさんある)



变数定義



変数定義

- int型のnを定義
 - var n int
 - var [変数名] [型]
- int型のx, y, zを定義
 - var x, y, z int
- int型のnとstring型のsを定義
 - var(
 - n int
 - s string)



暗黙的な定義

- int型のnを定義し1を代入

`n := 1`

- string型のsを定義し文字列"hello"を代入

`s := "hello"`



配列

- 要素数が5であるint型の配列aを定義

```
a := [5]int{1, 2, 3, 4, 5}
```

- 要素数の省略

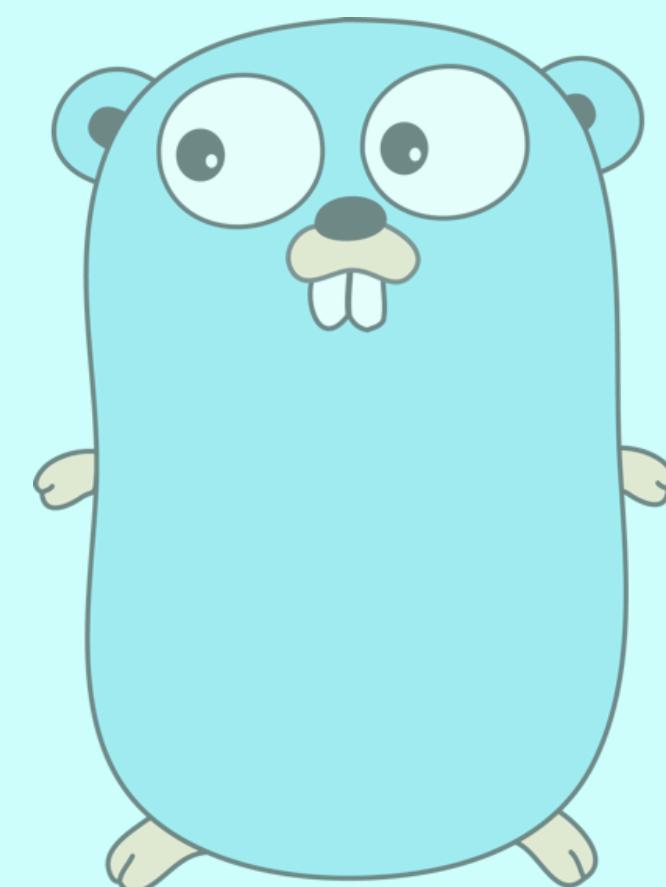
```
a := [...]int{1, 2, 3, 4, 5}
```



定数

- constを使う
- 定数Xの値は整数1

const X = 1



定数

- 複数定数定義
- 値の省略

```
const (
```

```
X = 1
```

```
Y = 1
```

```
Z = 1
```

```
)
```

```
const (
```

```
X = 1 //X ==1
```

```
Y //Y ==1
```

```
Z //Z ==1
```

```
)
```



算術演算子

算術演算子	意味	対象の型
+	和	整数、浮動小数点数、複素数、文字列
-	差	整数、浮動小数点数、複素数
*	積	整数、浮動小数点数、複素数
/	商	整数、浮動小数点数、複素数
%	剰余	整数
&	論理積	整数
	論理和	整数
^	排他的論理和	整数
&^	ピットクリア	整数
<<	左シフト	整数
>>	右シフト	整数



関数



関数

- int型のパラメータa, bを受け取り、足し合わせた数値をint型で返す

```
func plus(a, b int) int {
```

```
    return a + b
```

```
}
```

```
func [関数名]([引数の定義]) [戻り値の型] {
```

```
[関数の本体]
```

```
}
```



関数(戻り値なし)

```
func hello() {  
    fmt.Println("Hello")  
}
```



関数(複数の戻り値)

- int型のパラメータa, bを受け取り、それぞれint型で返す

```
func div(a, b int) (int, int) {
```

```
    q := a / b
```

```
    r := a % b
```

```
    return q, r
```

```
}
```



関数(戻り値の放棄)

- 戻り値の放棄

```
q, _ := div(19, 7)
```

```
_ , r := div(19, 7)
```



関数とエラー処理

- Goには例外機構がないため、エラーが発生したかどうかを戻り値の一部で記述

```
result, err := doSomething()
```

```
if(err != nil){  
    //エラー処理 "log.Fatal(err)", "panic(err)"など  
}
```

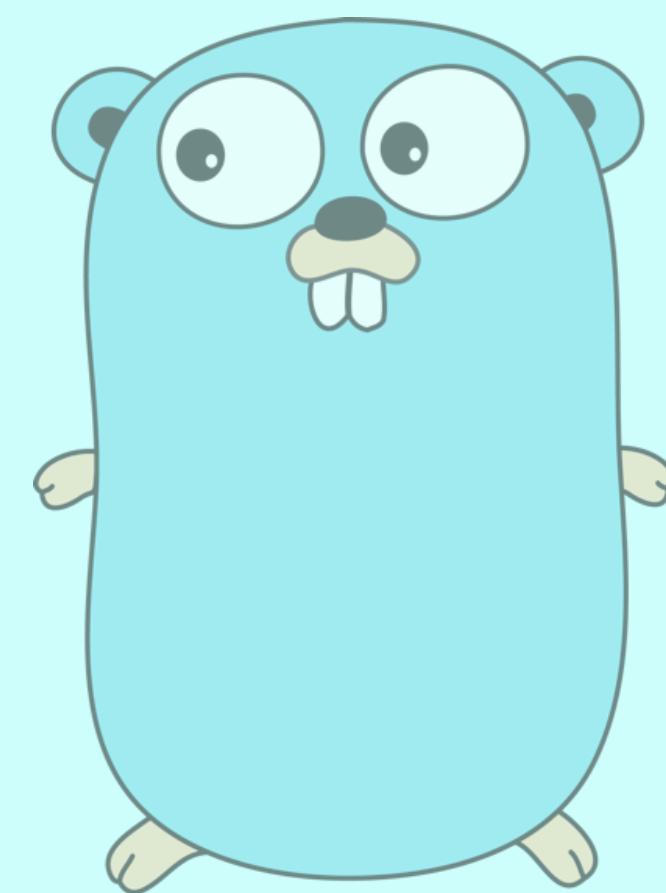


if文



ifの基本形

```
if x == 1 {  
    /*変数xの値が1のときに実行されるブロック*/  
}
```



if, else if

```
if [条件式A] {
```

[条件式Aが成立した場合の処理内容]

```
}else if [条件式B] {
```

[条件式Bが成立した場合の処理内容]

```
}else{
```

[すべての条件式が成立しなかった場合の処理内容]

```
}
```



if文の{}省略はできない！

- ・ 以下はコンパイルエラー

```
if [条件式A]
```

```
/*処理内容*/
```



簡易文付きif文

- 簡易文：式、代入式、暗黙的な変数定義など

```
if [簡易文] ; [条件式] {
```

```
}
```

```
if x, y := 1, 2 ; x < y {
```

```
    fmt.Printf("x(%d) is less than y(%d) \n", x, y)
```

```
}
```



for文

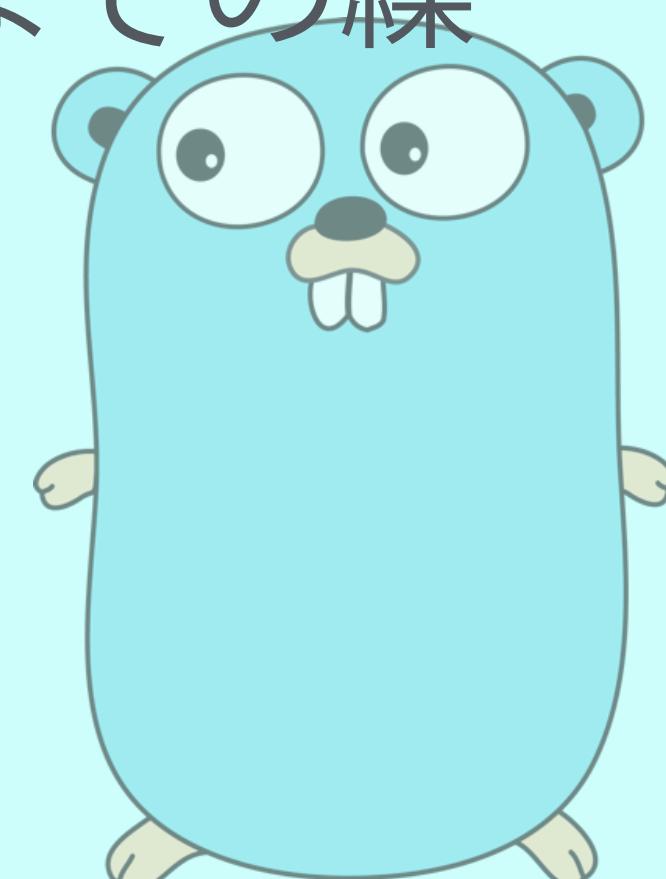


for文(基本形)

```
for i := 0; i < 10; i++ {  
    /*変数iの値が0から9までの繰り返  
    し*/  
}
```

C言語だと

```
for(int i = 0; i < 10; i++ ){  
    /*変数iの値が0から9までの繰  
    り返し*/  
}
```



for文(無限ループ)

```
for {  
    /*無限ループ*/  
}
```



break

- ・ ループを中斷する

```
i := 0
```

```
for{
```

```
    fmt.Println(i)
```

```
    i++
```

```
    if i == 100 {
```

```
        /* 変数iの値が100に到達したら forループを抜ける */
```

```
        break
```

```
}
```



forでwhileっぽい書き方

```
i := 0
```

```
for i < 100 {
```

```
    /* Do Something */
```

```
    i++
```

```
}
```



switch文



switch文

- 式によるswitch文
- 型によるswitch文



式によるswitch文

```
switch [式]{
```

```
    case [値]:
```

```
        default :
```

```
}
```

case 内に基本breakはいらない



式によるswitch文

```
n := 3

switch n {

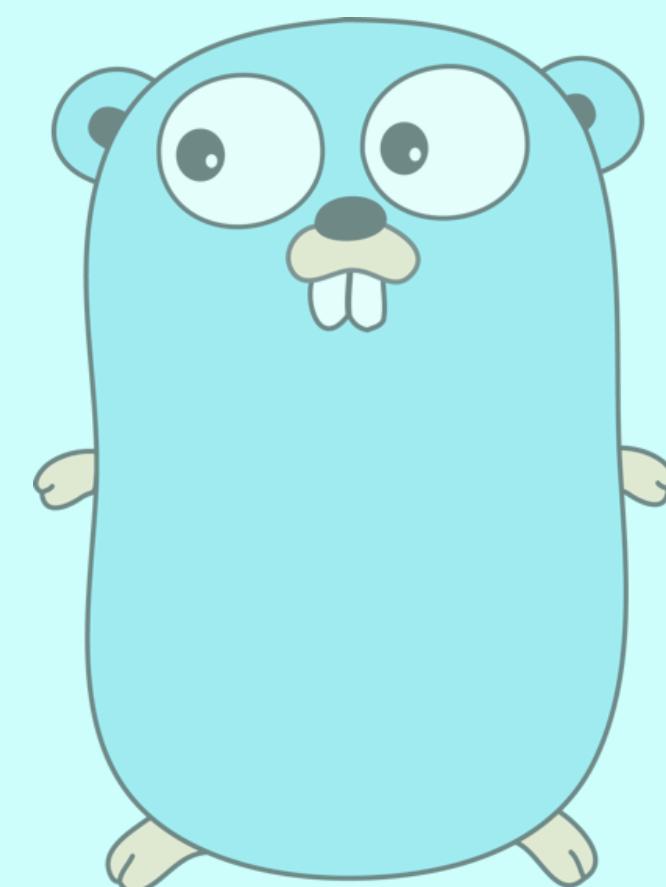
    case 1, 2:
        fmt.Println("1 or 2")

    case 3, 4:
        fmt.Println("3 or 4")

    default:
        fmt.Println("unknown")

}
```

出力 : 3 or 4



式によるswitch文

- 予約後fallthroughを使うと次のcase
内も実行する(C言語だとbreakを書
かなかったときの挙動)

出力：

1 or 2

3 or 4

unknown

```
n := 3  
  
switch n {  
    case 1, 2:  
        fmt.Println("1 or 2")  
        fallthrough  
    case 3, 4:  
        fmt.Println("3 or 4")  
        fallthrough  
    default:  
        fmt.Println("unknown")  
}
```



型によるswitch

- すべての型と互換性のある

interface{}型

```
func anything( a interface{}) {  
    fmt.Println(a)  
}
```

/* interface{} にはすべての型を指定できる */

anything(1)

anything(3.14)

anything(4 + 5i)

anything('海')

anything("日本語")

anything([... int{ 1, 2, 3, 4, 5}])



型によるswitch

こんな書き方ができる

```
/* 変数 x は interface{} 型 */

switch x.( type) {

    case bool:

        fmt.Println("bool")

    case int, uint:

        fmt.Println(" integer or unsigned integer")

    case string:

        fmt.Println(" string")

    default:

        fmt.Println("don't know")

}
```



構造体



構造体

type [型名] struct{

要素

}



構造体

```
type Point struct{
```

```
    X int
```

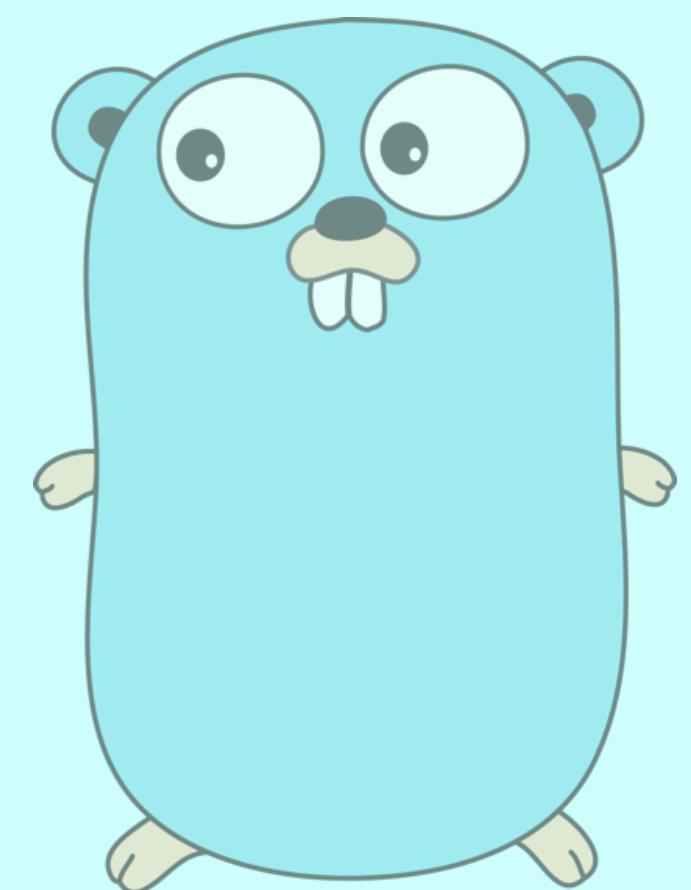
```
    Y float
```

```
}
```

```
var pt Point
```

```
pt.X = 1
```

```
pt.Y = 3.14
```



構造体

```
type [型名] struct{
```

要素

```
}
```

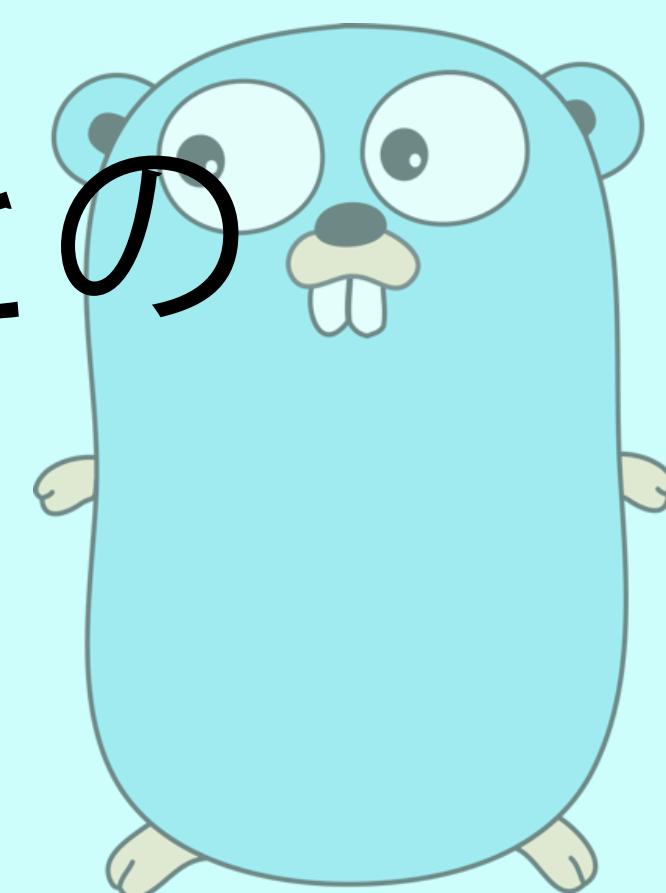


メソッド



メソッド

- ・オブジェクト指向のメソッドとは少し違う
- ・任意の型に特化した関数
- ・レシーバを定義する(funcとメソッド名との間)



メソッド

```
type Vertex struct {  
    X, Y float64  
}  
  
func (v Vertex) Abs() float64 {  
    return math.Sqrt(v.X*v.X + v.Y*v.Y)  
}  
  
func (v *Vertex) Scale(f float64) {  
    v.X = v.X * f  
    v.Y = v.Y * f  
}
```

```
func main() {  
    v := Vertex{3, 4}  
    v.Scale(10)  
    fmt.Println(v.Abs())  
}
```

出力

50

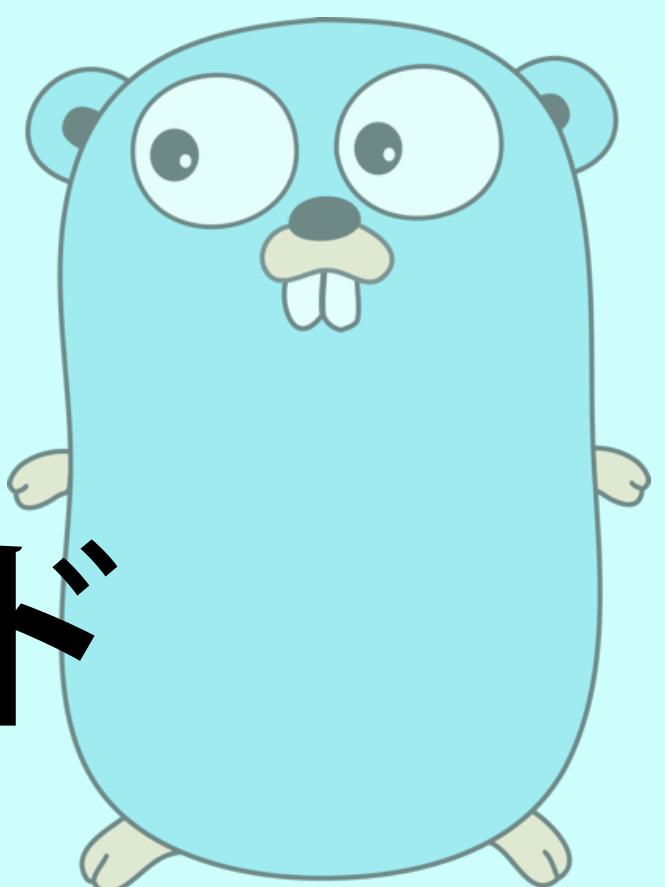


公開メソッドと非公開メソッド



公開メソッドと非公開メソッド

- ・ パッケージに定義された関数が、外部の関数
から参照可能かどうか
- ・ 関数の先頭文字が大文字 → 公開メソッド
- ・ 関数の先頭文字が小文字 → 非公開メソッド



例

```
package foo

const (
    MAX      = 100
    internal_const = 1
)

func FooFunc(n int) int {
    return internalFunc(n)
}
```

```
func internalFunc(n int) int {
    return n + 1
}
```

//internalFuncとinternal_constは
他パッケージから呼ぶとエラー



その他

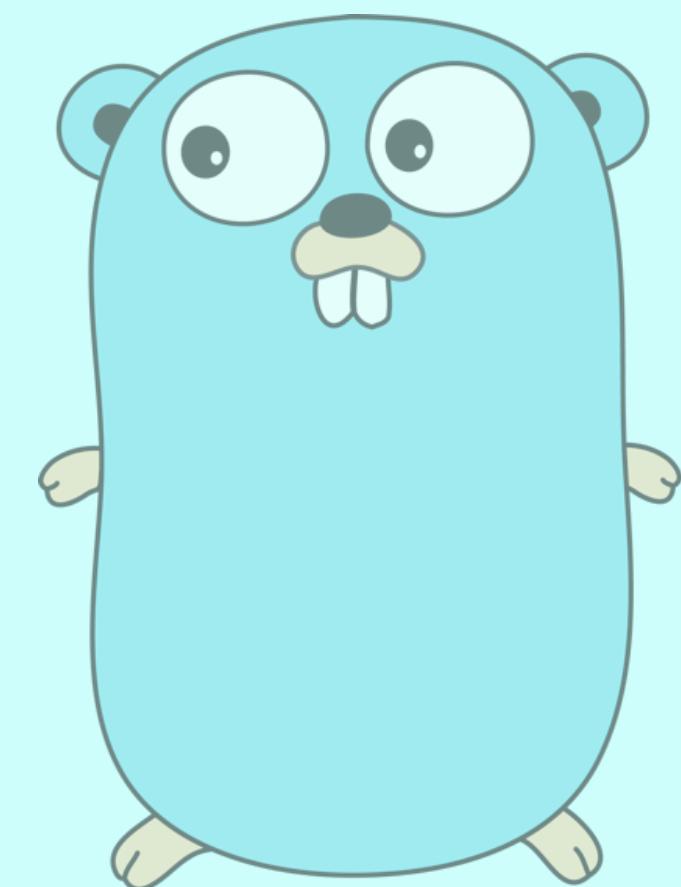


その他

- ポインタ
- go (goroutine)、チャネル
- スライス(可変長配列)、マップなど
- 各パッケージ(os, time, net/httpなど)



テストコード



テストコード

- foo.go

```
package foo
```

```
func IsOne(n int) bool {
```

```
    if n == 1{
```

```
        return true
```

```
    } else {
```

```
        return false
```

```
}
```

```
}
```

- foo_test.go

```
package foo
```

```
import("testing")
```

```
func TestIsOne(t *testing.T){
```

```
    n := 1
```

```
    b := IsOne(n)
```

```
    if b != true {
```

```
        t.Errorf("%d is not one", n)
```

```
}
```

```
}
```

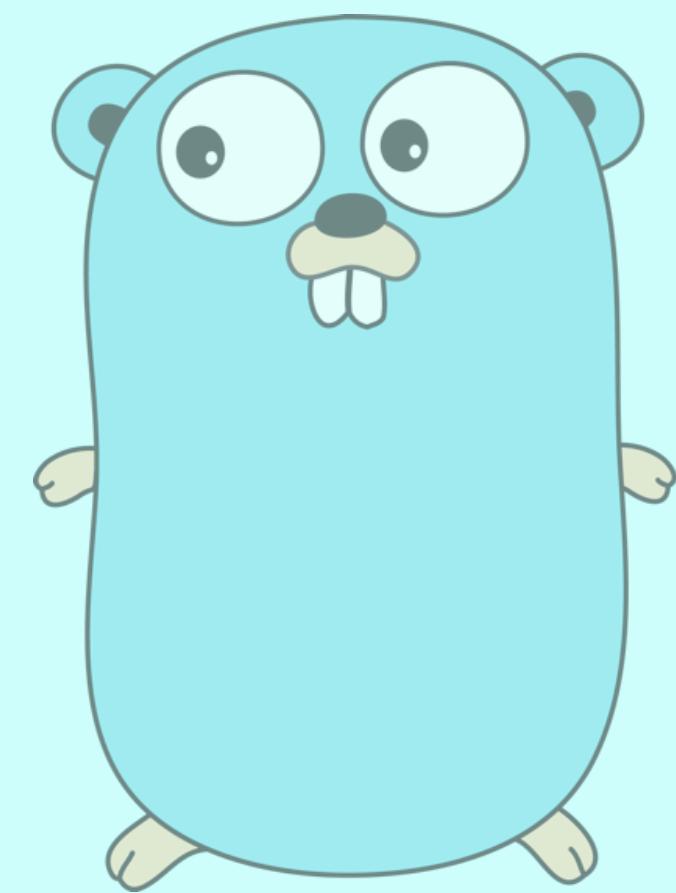


テストコード

- テストの実行
 - go test [package名]
- テストの実行
 - go test foo
- カバレッジ率の表示
 - go test -cover foo



演習



演習

1. 変数の型を出力するプログラム
2. テストを用いた素数判定をしよう
3. 平均、分散、標準偏差を出力するプログラム



1. 変数の型を出力するプログラム

- ・宣言した変数の型をprintしよう
- ・わからなければとりあえずぐぐってみよう(ヒント："%T")

例(出力)：

変数nの型はintです。

変数sの型はstringです。



2. テストを用いた素数判定をしよう

- ・ テストコードを利用する
- ・ 素数の条件は”1と自分以外の数字で割り算をしあまりが0にならないとき”
- ・ 出力：13のときok、12のときFAILが出るように
- ・ \$go test



3. 平均・分散・標準偏差の出力

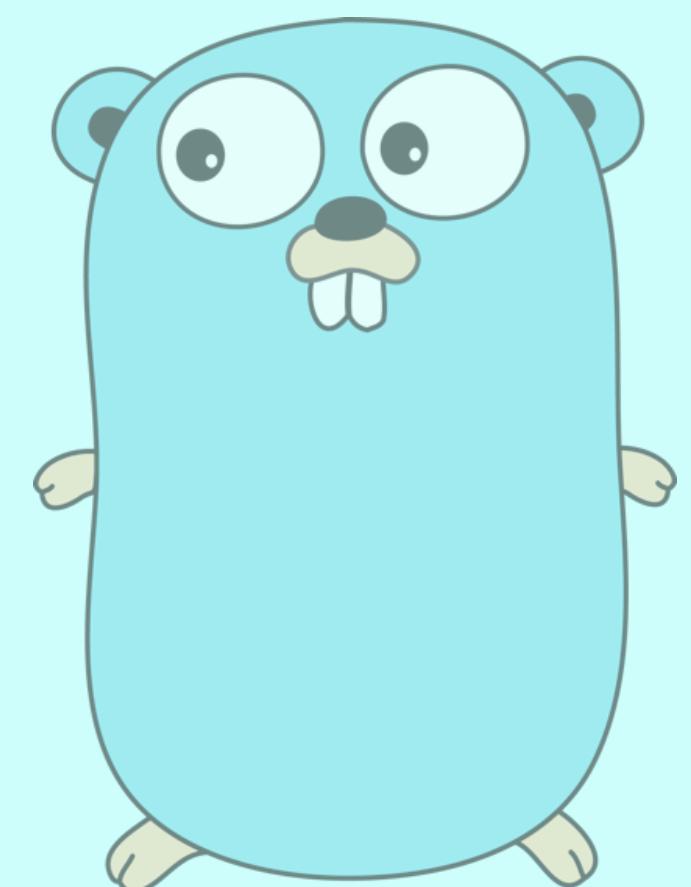
- ・配列を使ってみる
- ・今回のデータ : array := [...]float64{160, 161, 170, 180, 150}

出力例 :

平均 : ~~~~

分散 : ~~~~

標準偏差 : ~~~~



終わり

(もっと詳しくやりたい方はA Tour of Goで！)



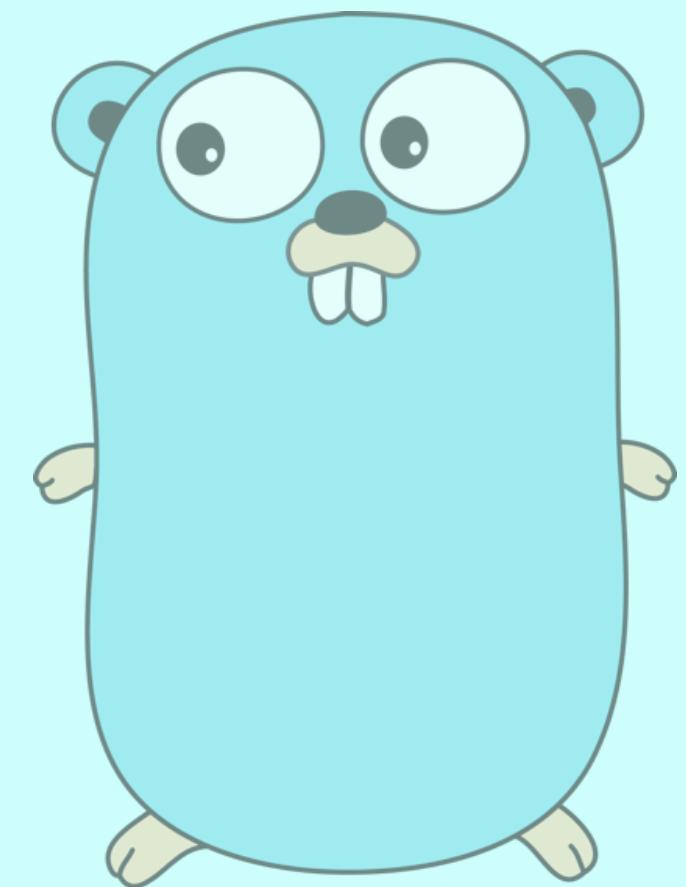
勉強会内容

1. goの環境構築
2. goの言語仕様・文法
3. サーバの基本
4. チャットアプリ作成
5. (並行処理を使う)
6. (API作成)



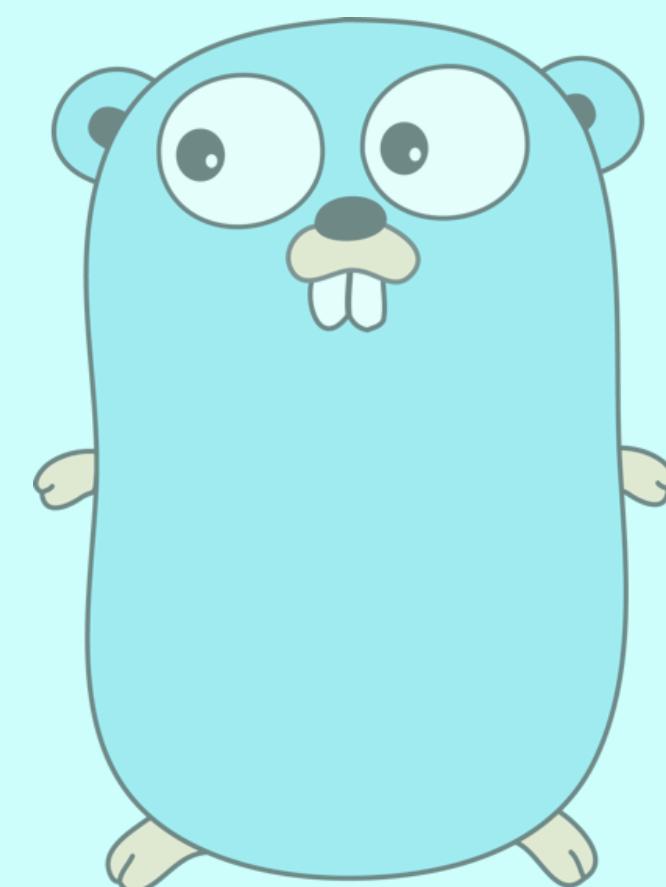
3. サーバの基本

Go言語はWebアプリ開発に最適



理由

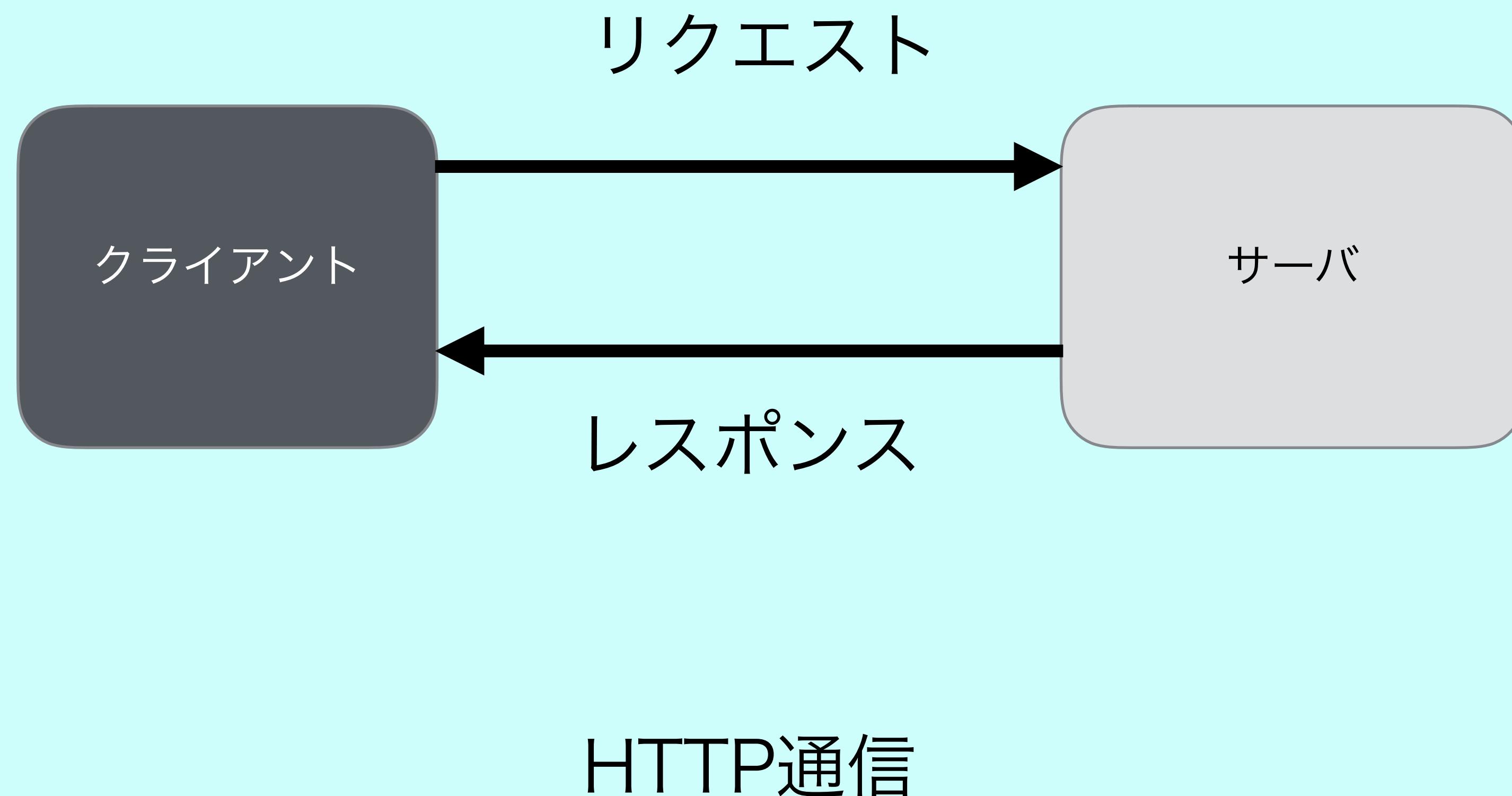
- ・スケーラビリティ(垂直、水平)
- ・モジュール性(インターフェース、関数っぽい書き方)
- ・保守性(テスト、ソースコードフォーマッタ)
- ・高い実行効率(並行実行、コンパイル型言語)



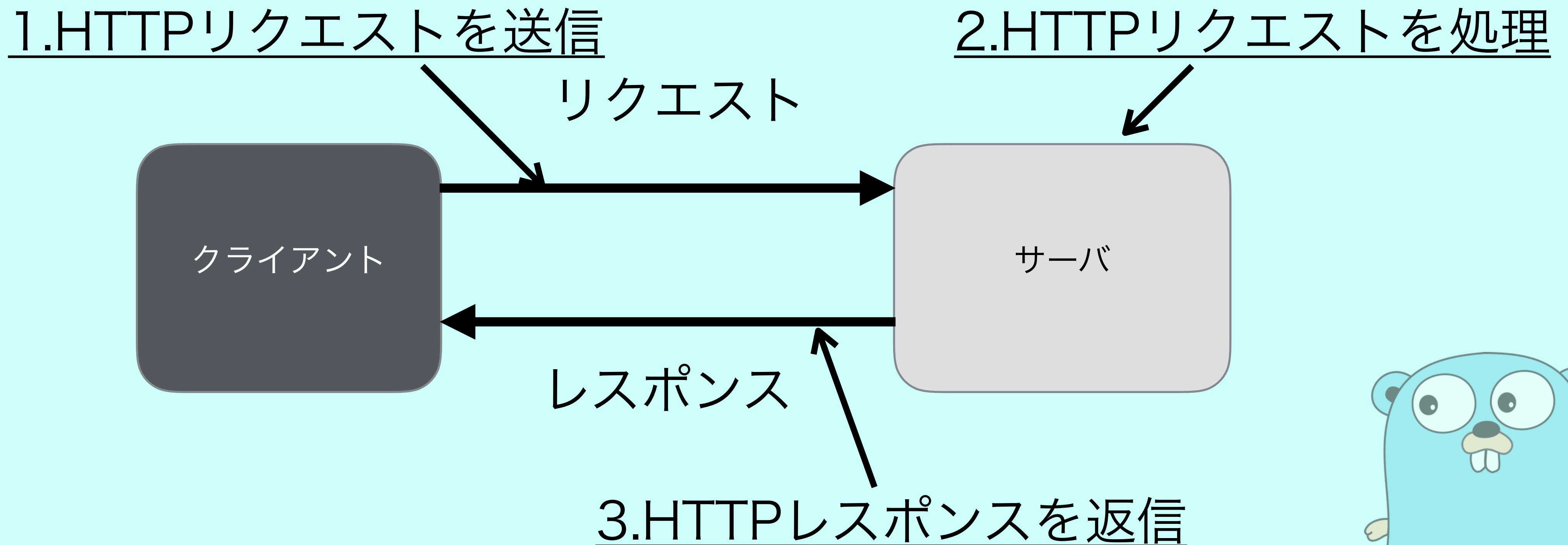
Webアプリケーションの仕組み



Webアプリケーションのリクエスト/レスポンス構造



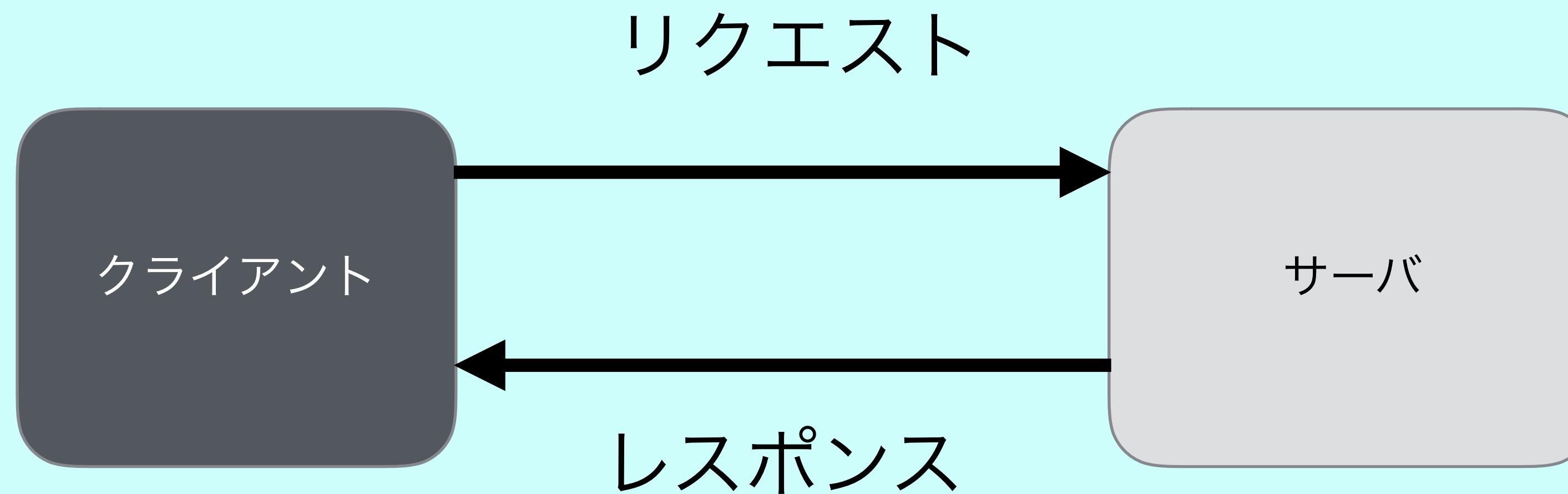
Webアプリケーションの一般的な仕組み



HTTPリクエストのURLフォーマット例

http://<サーバ名>/<ハンドラ名>?<パラメータ>

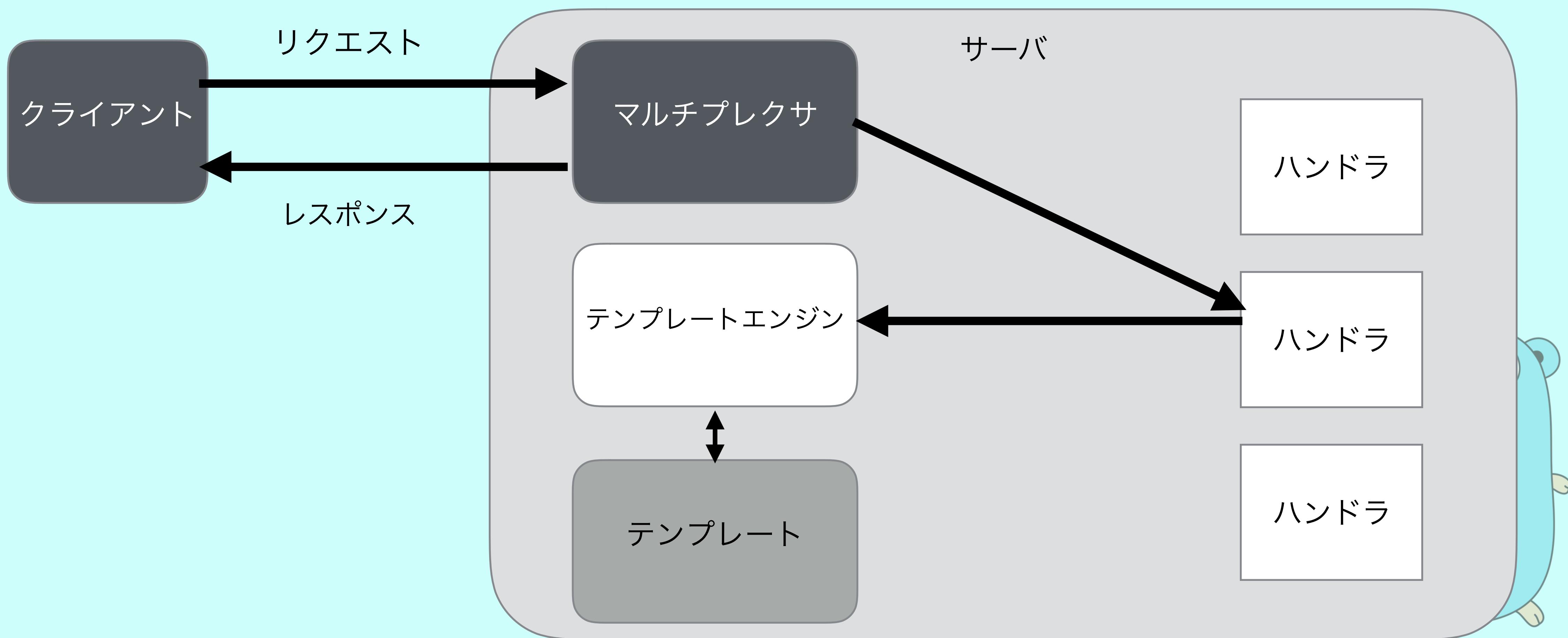
例：http://chitchat/thread?read?id=123



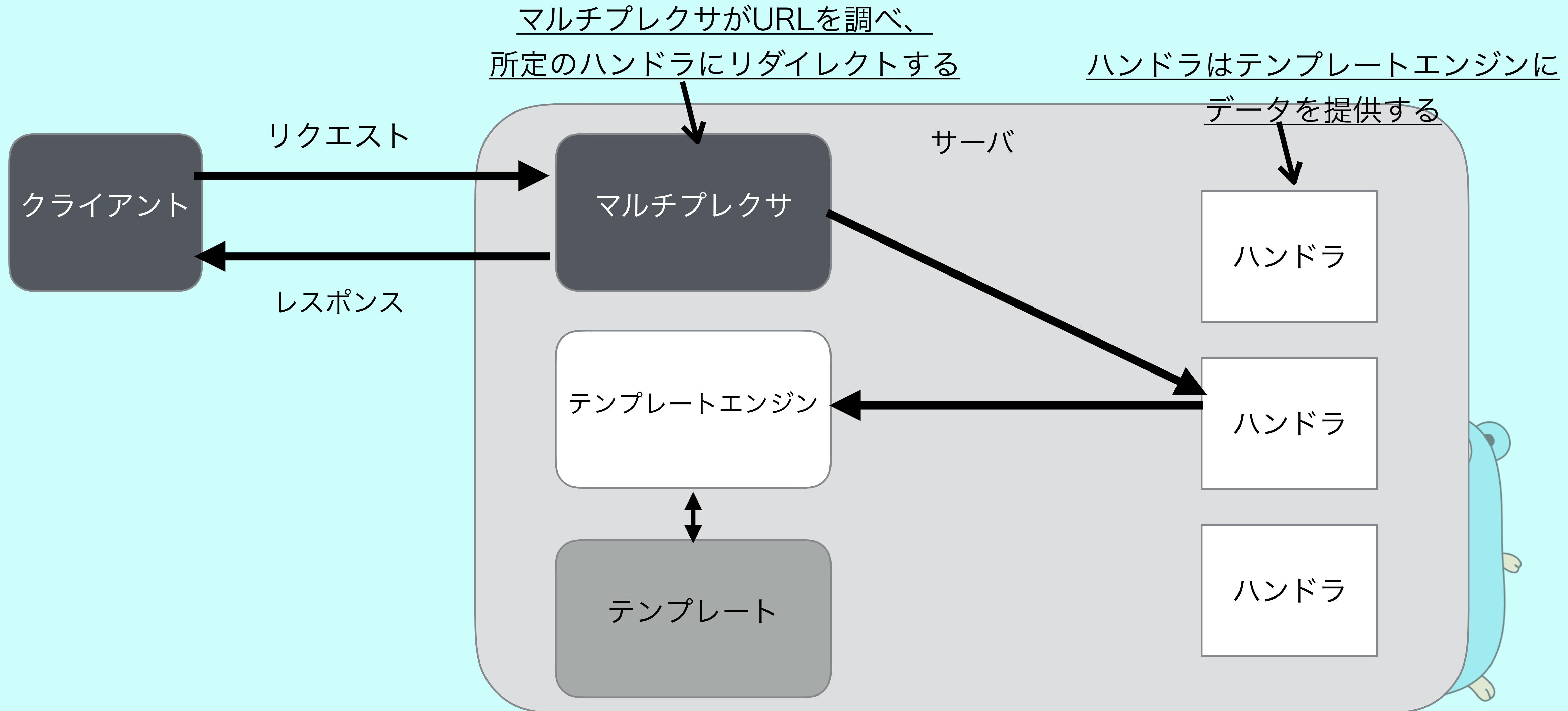
WEbアプリのサーバの働き



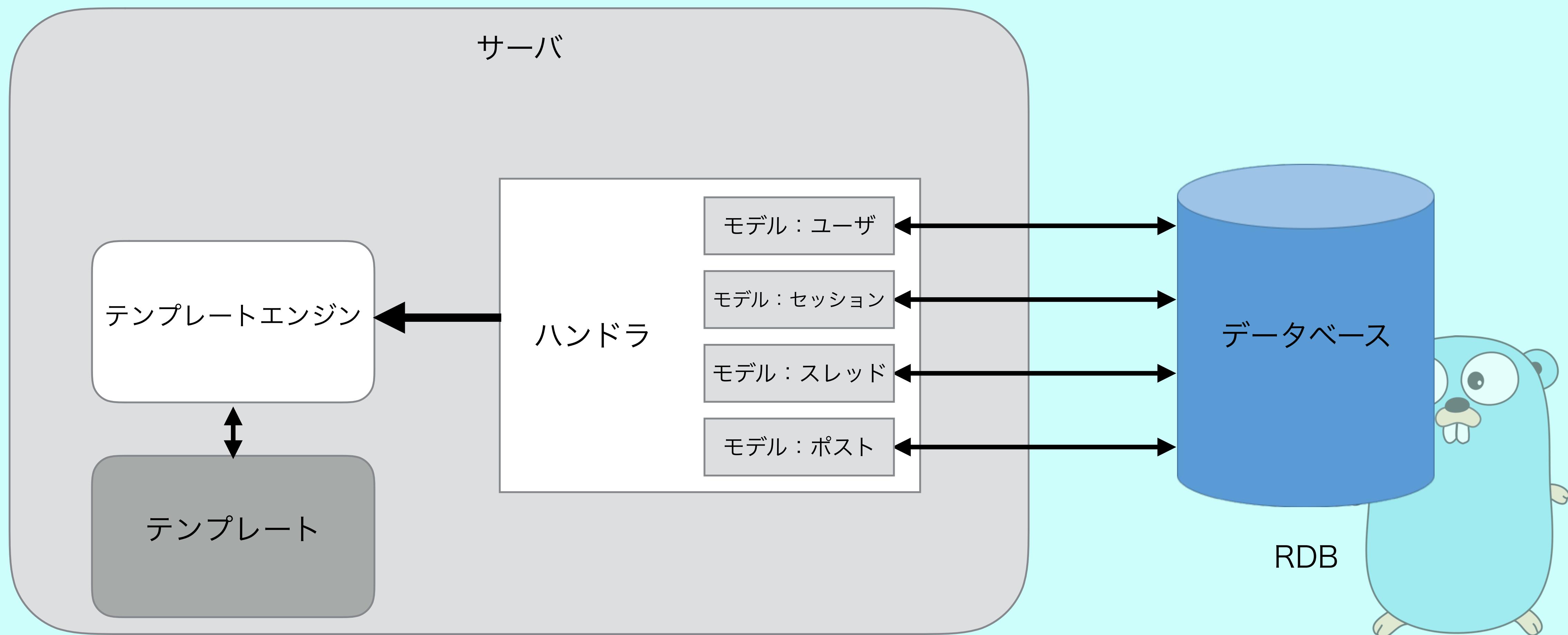
典型的なWebアプリでのサーバの働き



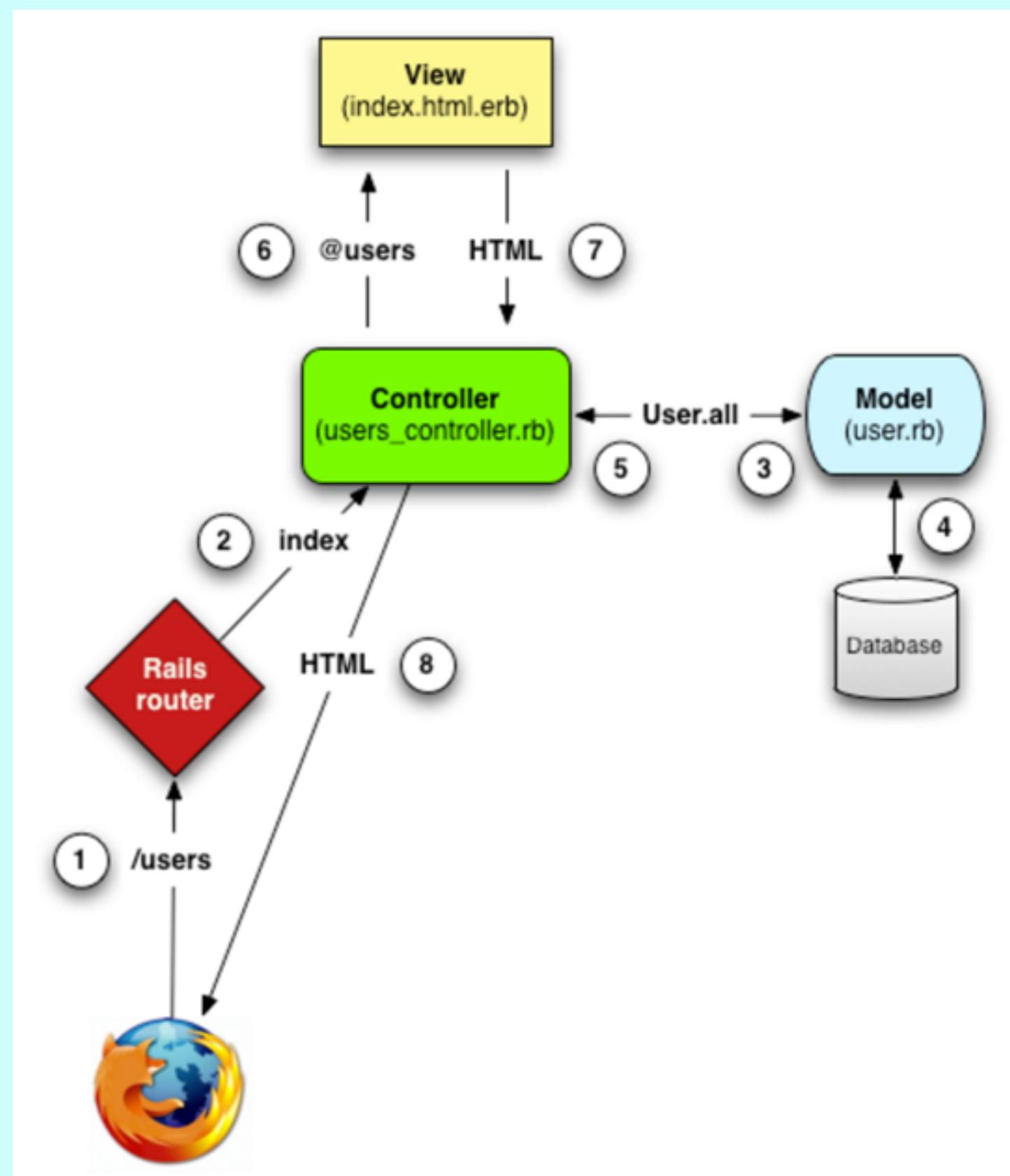
典型的なWebアプリでのサーバの働き



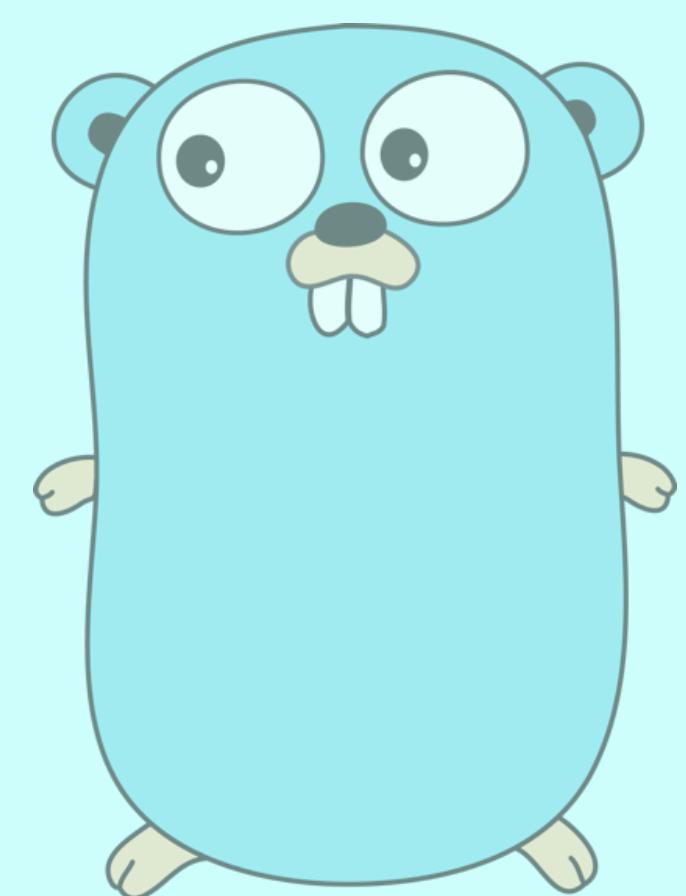
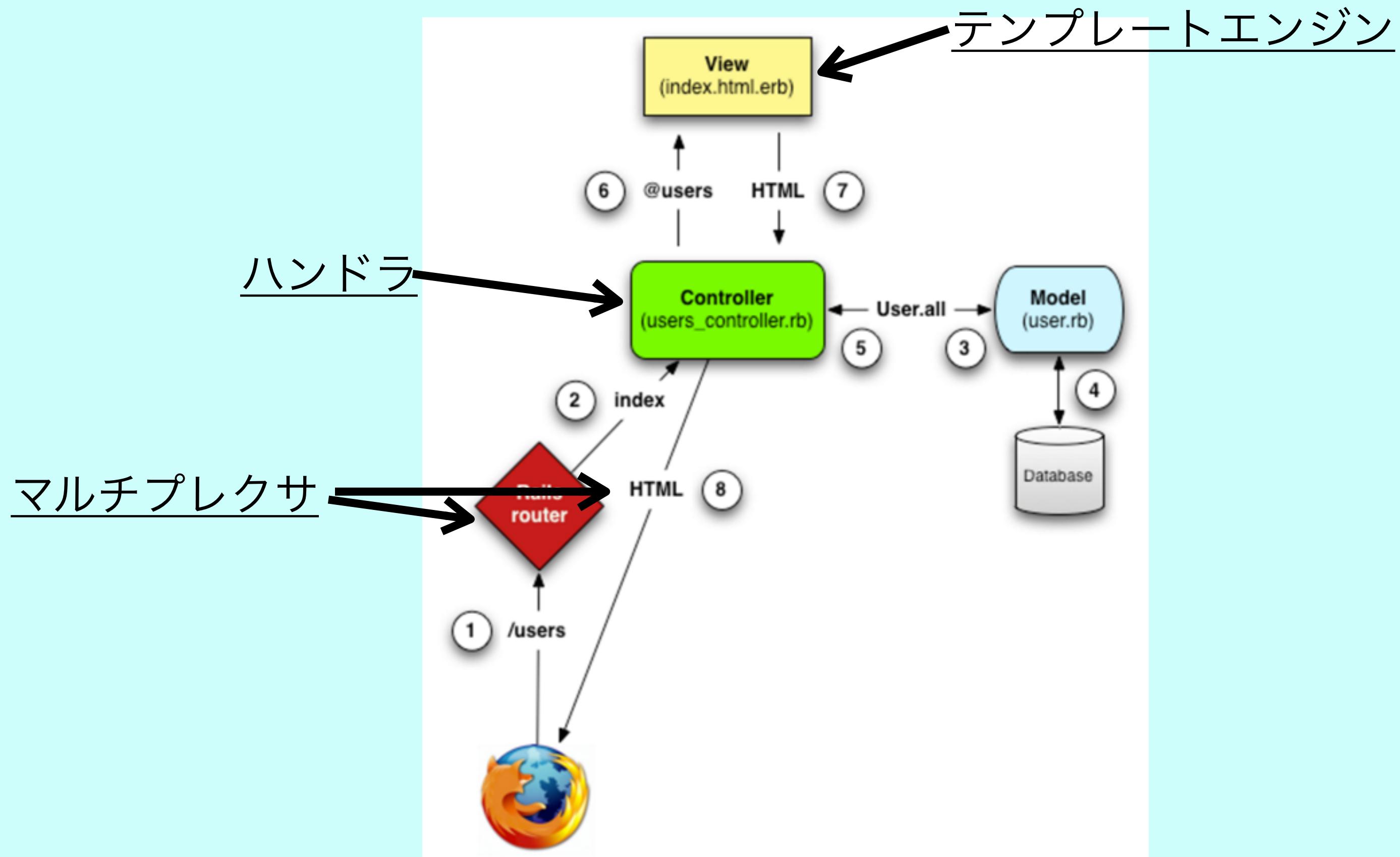
Webアプリケーションのデータへのアクセス



参考：Ruby on Rails の MVC モデル



参考：Ruby on Rails の MVC モデル



実装



HelloWorldをサーバ上で行う



テンプレートエンジンの利用



終わり



勉強会内容

1. goの環境構築
2. goの言語仕様・文法
3. サーバの基本
4. チャットアプリ作成
5. (並行処理を使う)
6. (API作成)



4. チャットアプリ

サンプルアプリ



**\$GOPATH/src内にclone(または
ダウンロード)してください**

github.com/mushahiroyuki/gowebprog



Postgresqlのインストール



Linux

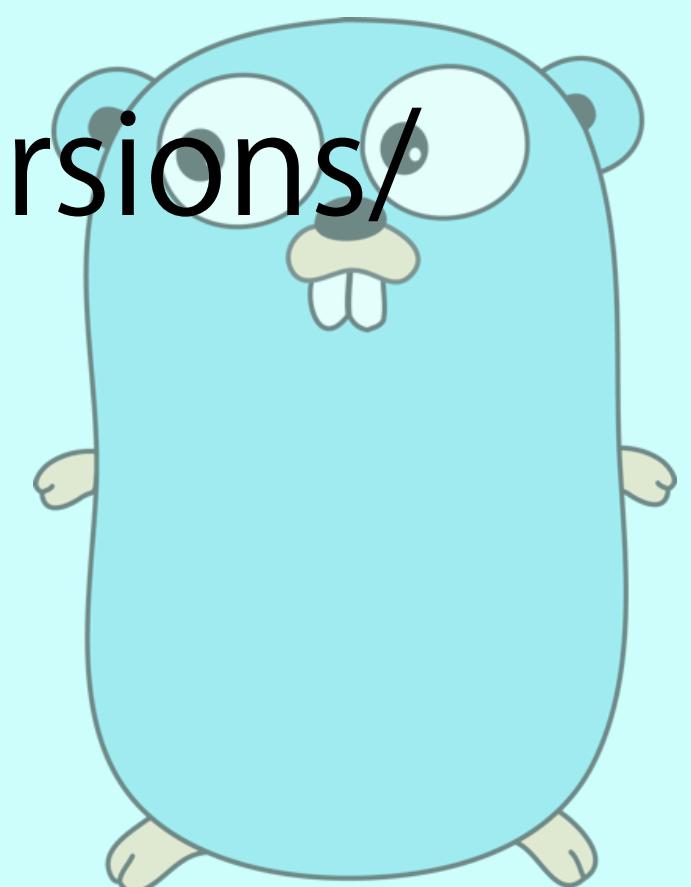
- \$ sudo apt-get install postgresql postgresql-contrib
- \$ sudo su postgres
- \$ create user -interactive
- \$ createdb <自分のアカウント名>
- <http://www.postgresql.org/download>



MacOS

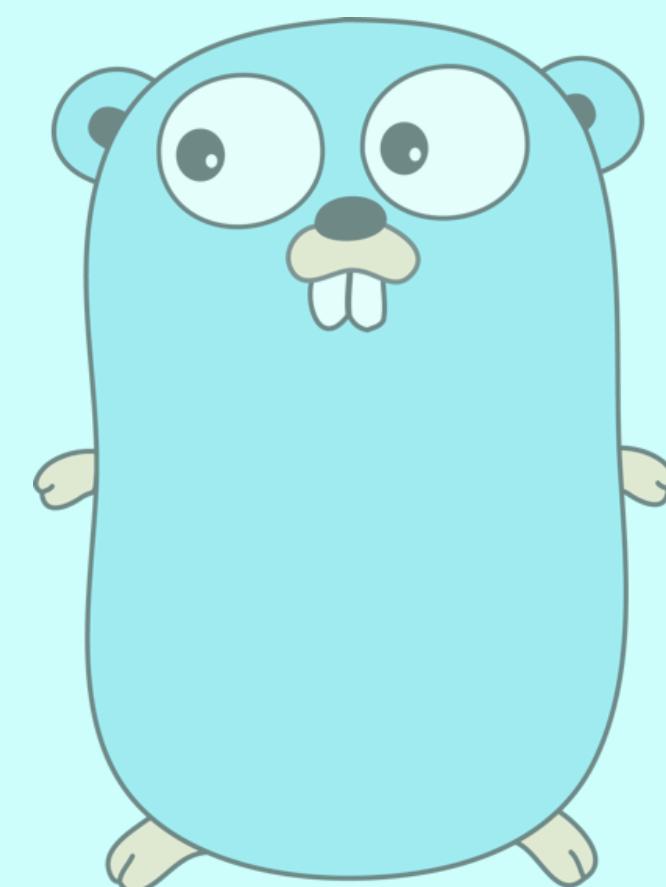
- <http://postgresapp.com/>
- `~/.profile`または`~/.bashrc`に以下を追加

```
export PATH=$PATH:/Applications/Postgres.app/Contents/Versions/  
9.6/bin
```



サンプルチャットアプリの実行

- cloneしてきたディレクトリのch02/chitchatに移動
- \$ createdb chitchat
- \$ psql -f data/setup.sql -d chitchat
- \$ go build
- \$./chitchat



終わり



勉強会内容

1. goの環境構築
2. goの言語仕様・文法
3. サーバの基本
4. チャットアプリ作成
5. (並行処理を使う)
6. (API作成)



5. 並行処理

時間的に厳しそうなので

今回はなし



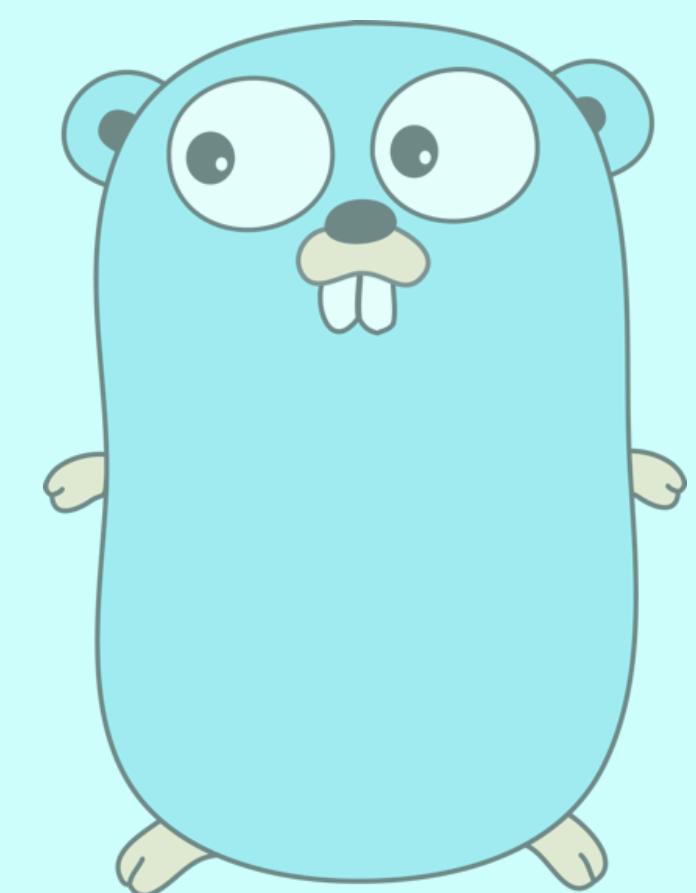
「A Tour of Go」の Concurrency部分をやろう！

<https://go-tour-jp.appspot.com/concurrency/1>



勉強会内容

1. goの環境構築
2. goの言語仕様・文法
3. サーバの基本
4. チャットアプリ作成
5. (並行処理を使う)
6. (API作成)



6. API作成

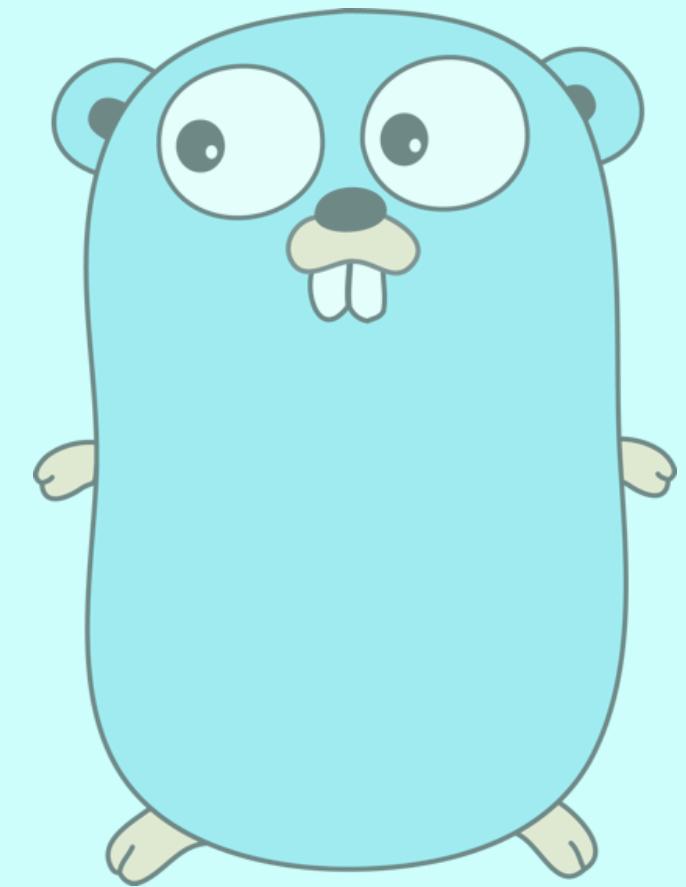
時間的に厳しそうなので

今回はなし



終わりに

本日はお疲れ様でした！



最後に紹介

- ・ おすすめエディタ(とIDE)
- ・ おすすめ書籍
- ・ Goを使いこなすためにこれからやっていくべきこと
- ・ 勉強会開催のすゝめ(サークル民向け)

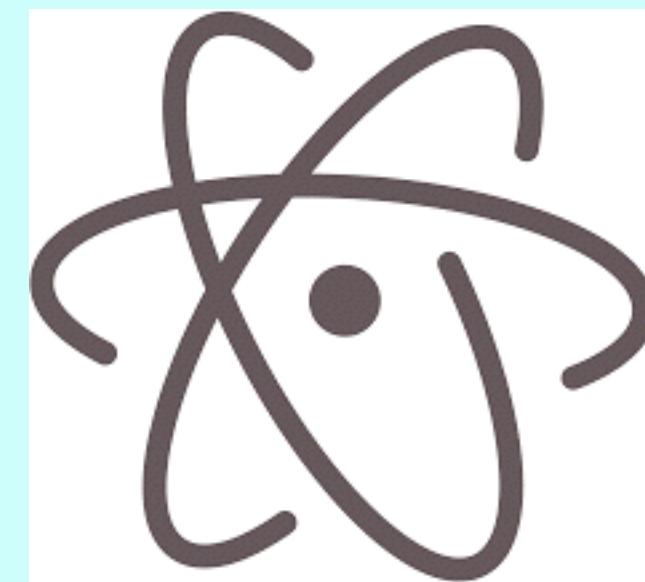


おすすめエディタ(とIDE)



おすすめエディタ(とIDE)

- Atom(プラグインが良い)
- VSCode(プラグインが良い)
- Vim(プラグインが良い)
- GoLand(Golang専用のIDE)



おすすめ書籍



おすすめ書籍①

- ・「スターティングGo言語」
- ・言語仕様・文法周り中心
- ・A Tour Of Go で物足りなくなった
ら買っても良いかも



https://www.amazon.co.jp/dp/B01FH3KRTI/ref=cm_sw_r_tw_dp_U_x_rprJAbQDQ0SVT

おすすめ書籍②

- ・「Goプログラミング実践入門」
- ・サーバサイドの基本からアプリケーション制作まで
- ・全般的に学びたい人、サーバサイドを学びたい人



https://www.amazon.co.jp/dp/B06XKPNVWV/ref=cm_sw_r_tw_dp_U_x_1trJAbQ6YH2DM

おすすめ書籍③

- ・「Go言語によるWebアプリケーション開発」
- ・どちらかというと応用メイン
- ・より実践的なことをしたい人



https://www.amazon.co.jp/dp/4873117526/ref=cm_sw_r_tw_dp_U_x_rtrJAb6436Q8B

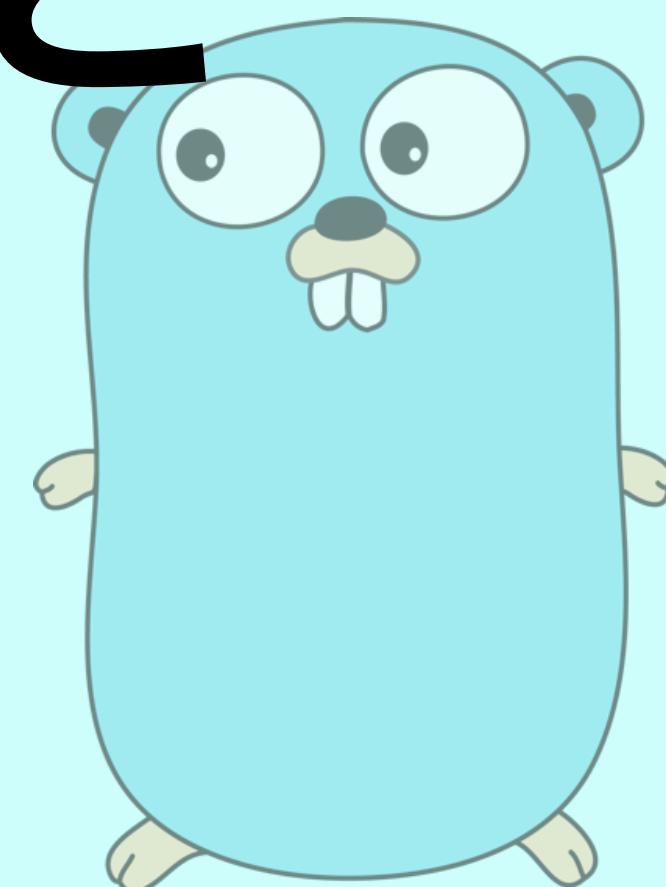
おすすめ書籍④

- ・「みんなのGo言語」
- ・現役で企業で使っている人らが書いた技術集
- ・さらに実践的なことがしたい人

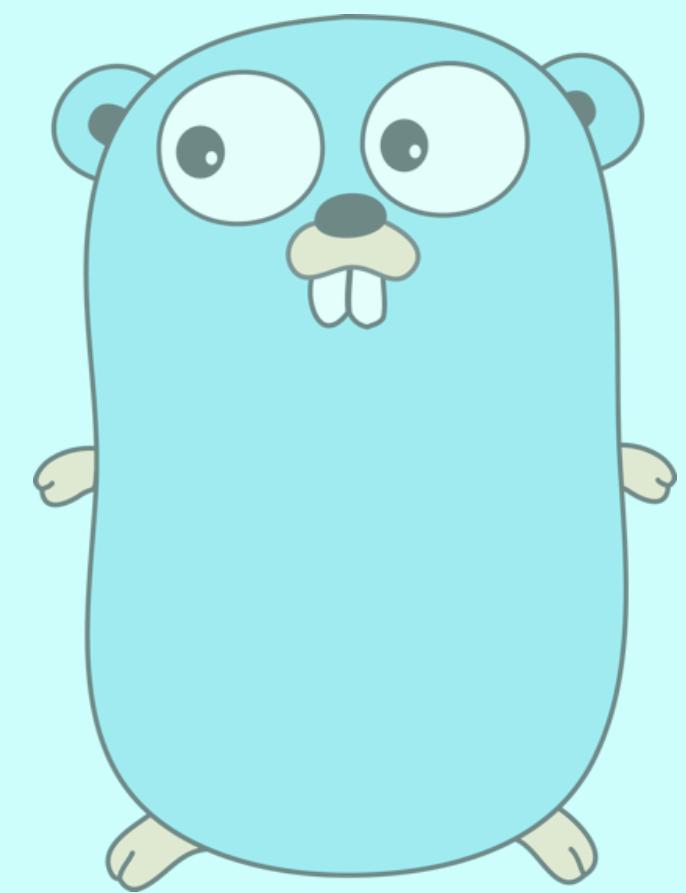


https://www.amazon.co.jp/dp/B01LMS7B1O/ref=cm_sw_r_tw_dp_U_x_avrJAb7EX63ZW

**Goを使いこなすために
これからやっていくべきこと**



(個人的な感想)



やっていくべきこと

- ・ GitHubの他人のソースコード(ライブラリ)を読もう
 - ・ 最初は本やWebの記事やチュートリアルからやると良い
 - ・ Goは学習用の書籍が他に比べ少ないため
- ・ Go以外でのサーバサイドの経験
 - ・ 言語はあくまで手段
 - ・ Webアプリ開発の基本を勉強するなら他の言語の方が情報量が多く最適
- ・ インフラ周りの勉強
 - ・ インフラやWebサービスで使われることが多い、実用的なコードを書くなら必須



勉強会開催のすゝめ



**勉強会開催には
メリットがたくさん！**



メリット

- ・ 良いアウトプットの機会
 - ・ 自分の中で言語化できるようになり、理解が高まる
 - ・ 間違った理解を直してくれる
- ・ その界隈の人と知り合える機会
- ・ 締め切りが発生することで、爆速でのインプットができる
- ・ 参加者との知見交換、布教(?)



デメリット

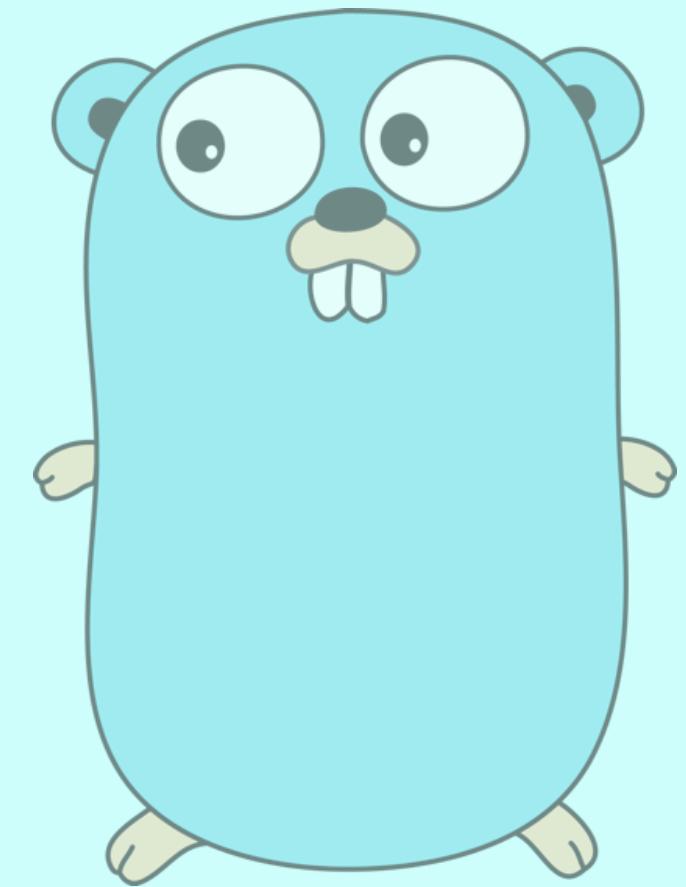
- ・準備、運営の手間
- ・参加者が少ないと寂しい…(´・ω・`)



アウトプットの重要性

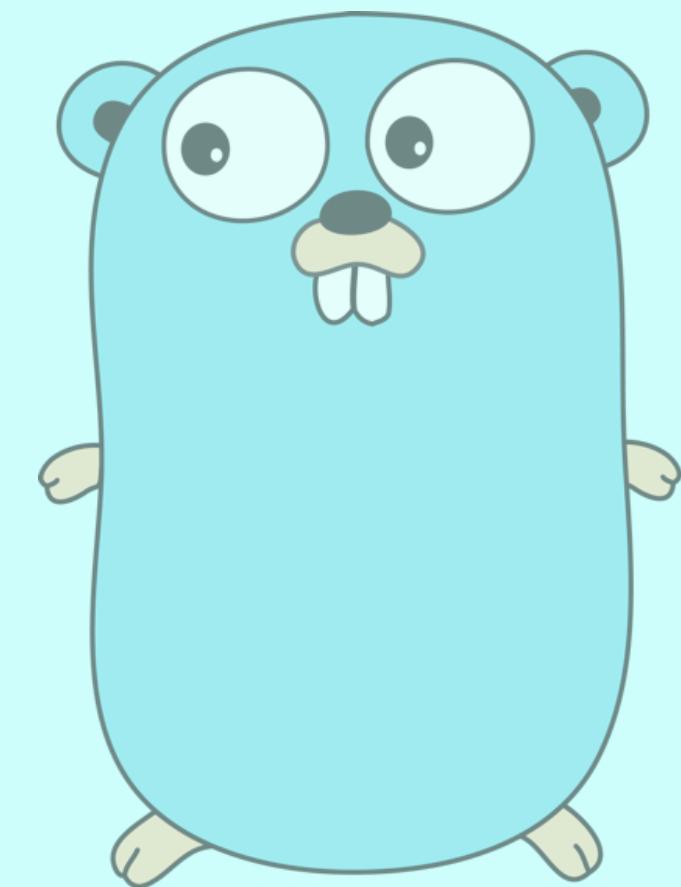


本日のゴール



本日のゴール

Golangと仲良くなる



1日お疲れ様でした！

