io/fsパッケージを用いたテスタブルなコード生成ツールの開発

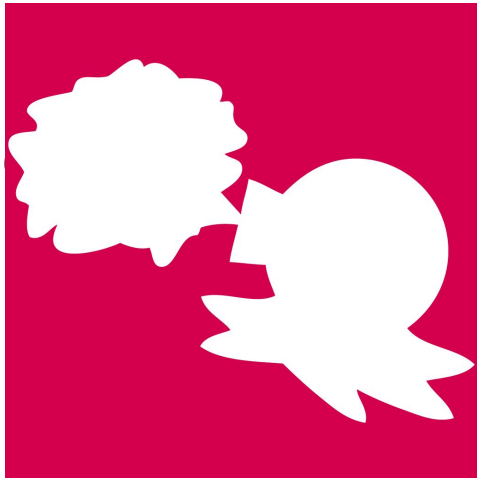
2021年11月13日（土）
Go Conference Online 2021 Autumn
資料：https://tenn.in/skeleton
ハッシュタグ：#gocon #goconA

自己紹介

上田拓也

@tenntenn

tenntenn.dev



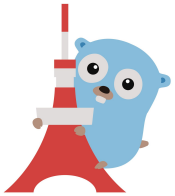
所属

merpay

コミュニティ活動

Google Developer Expert (Go)

Go ビギナーズ



golang.tokyo

Go Conference

GCPUG

Google Cloud Platform User Group

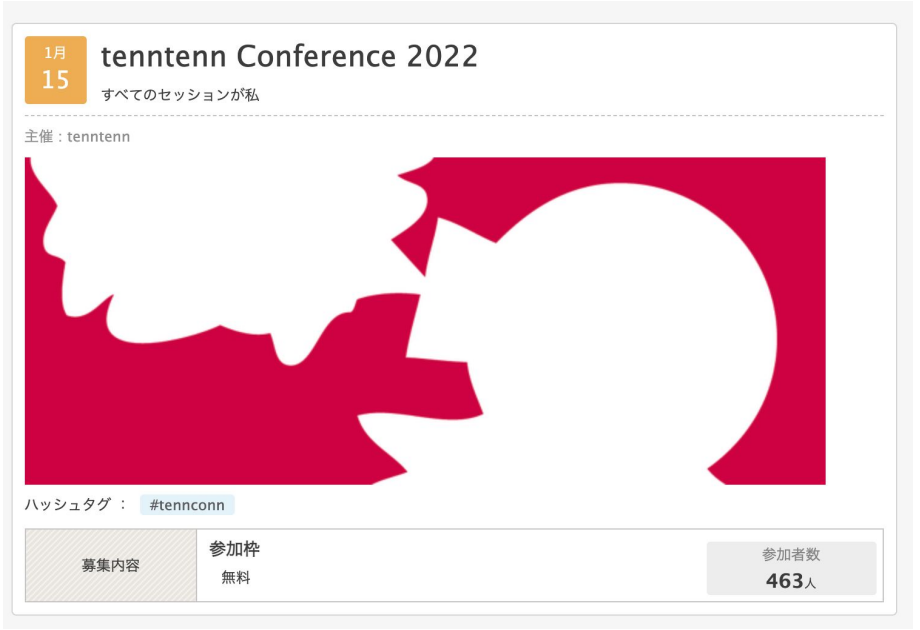
オンラインでセッションを見るときは

- 登壇者を孤独にさせないように！
  - Twitterなどのリアクションは大きめに！
  - 知らなかったことには「へぇ」
  - 知ってることには「そうそう、それ」
  - あなたの相槌で登壇者を孤独から救えます

次のスライドで練習してみましょう

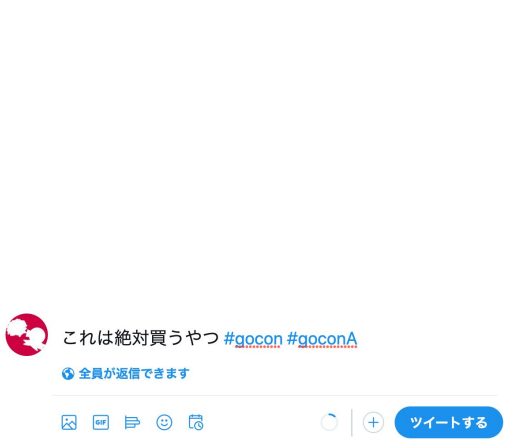
宣伝: tenntenn Conference 2022

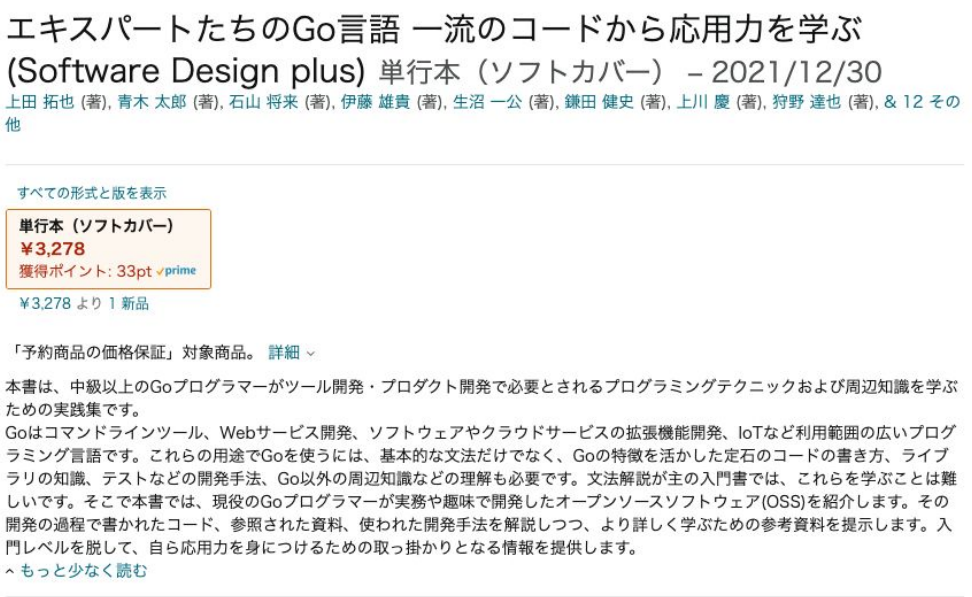
- tenntenn Conference 2022開催決定！
  - https://connpass.com/event/226562/
  - 私による私だけのカンファレンス
  - 地方移住の話もします



宣伝:エキスパートたちの Go言語

- 12月30日発売！
  - https://amazon.jp/dp/4297125196

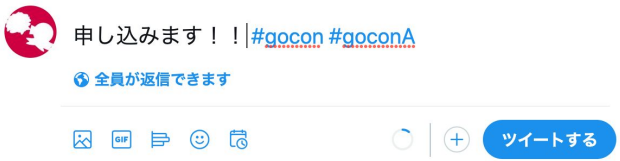




関連:技術専門誌編集者とメルペイエンジニアが2年間続けた連載「作品で魅せるGoプログラミング」のすべて

宣伝:パーソナライズ Gopher道場

- Goを使って働きたい人向け！
  - https://gopherdojo.org/personalized/
  - 性別・年齢・職歴(学年)不問
  - 参加費無料、Gopher道場 昇段審査 白帯または茶帯合格者
  - 1年間(途中6ヶ月で継続判定あり)の継続的な学習機会
  - 毎週1hの最大4人の少人数講義 + 隔週1hの1対1講義
  - 週1回まで何度でも設定可能な1on1
  - プログラミング言語Go完全入門をベースにした講義
  - 参加者の希望を最大限に反映



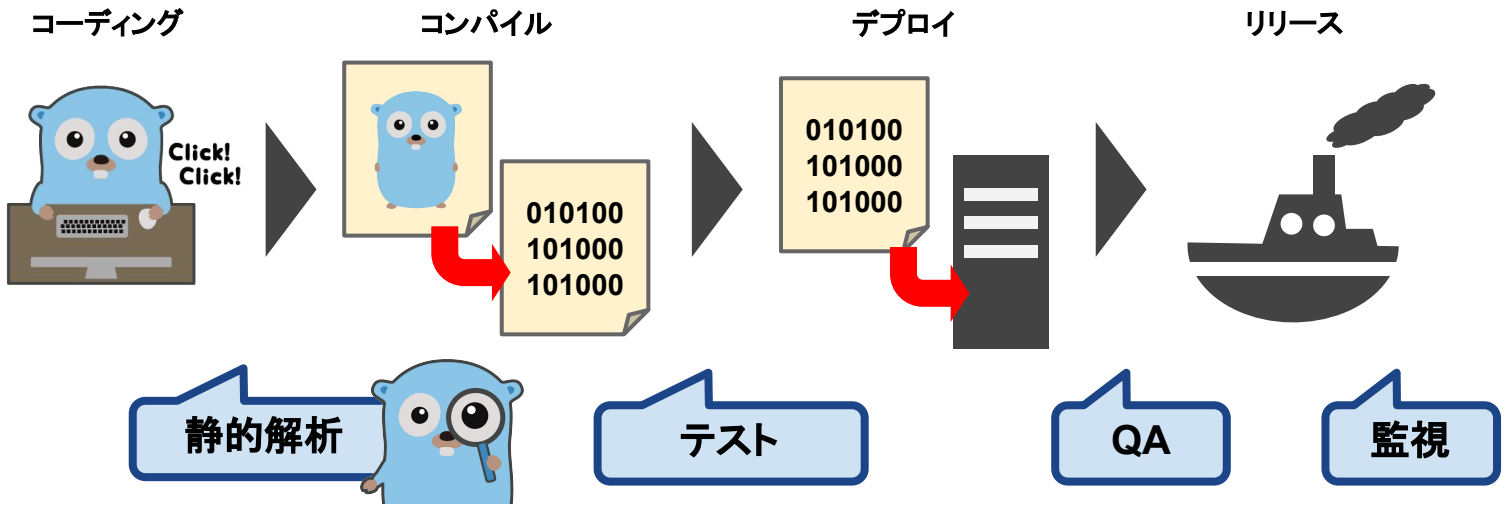
今日話すこと

- 静的解析ツール用スケルトンコードジェネレータskeleton
  - skeletonの説明とv1における課題
- io/fsパッケージを使ったテンプレート管理
  - io/fsパッケージとembedパッケージの説明
  - txtar形式とtxtarfsパッケージの紹介
  - skeletonkitの紹介
- io/fsパッケージを用いたゴールデンファイルテスト
  - ゴールデンファイルテストの説明
  - tenntenn/goldenの紹介

静的解析ツール用
スケルトンコードジェネレータ
skeleton

静的解析

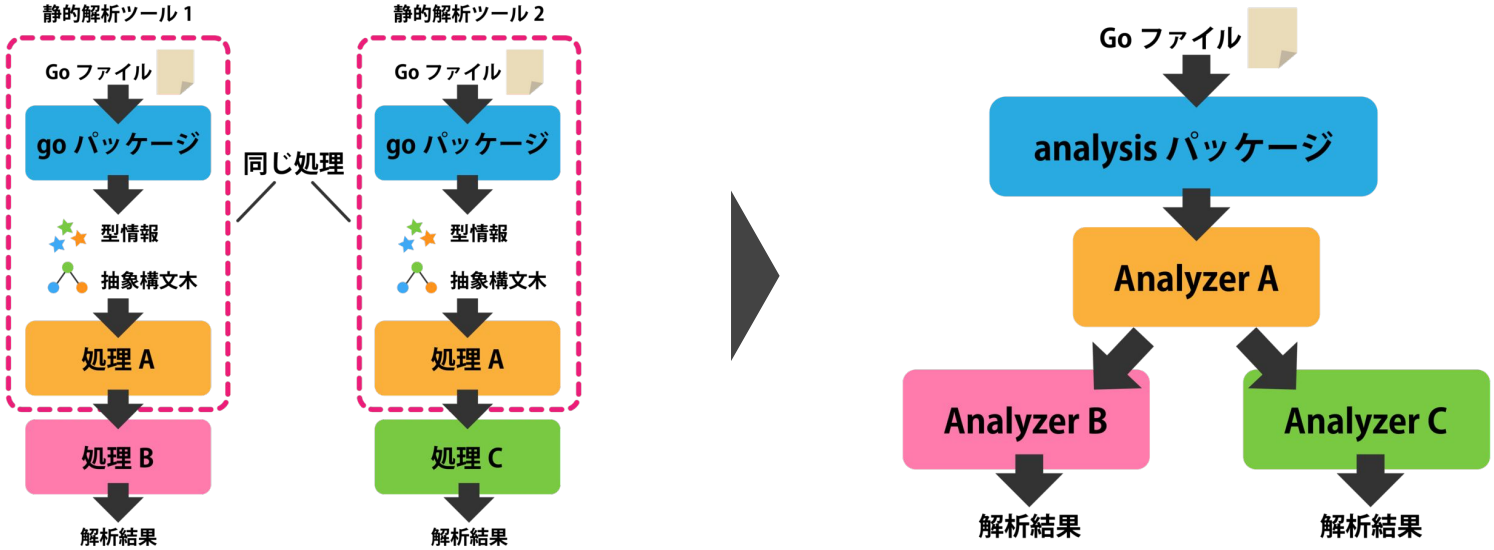
- 静的解析
  - プログラムを**実行せずに**解析すること
  - ソースコードの構造や意味を解析する
  - 例：lint、コード補完、コードフォーマッタ



参考：[プログラミング言語Go完全入門 - 14. 静的解析とコード生成]

静的解析ツールのモジュール化

- golang.org/x/tools/go/analysisパッケージ
  - 静的解析ツールのモジュール化を提供するパッケージ
  - Go1.12からgo vetでも使われるようになった



skeleton

- go/analysis用のスケルトンコードジェネレータ
  - https://github.com/gostaticanalysis/skeleton
  - 簡単に静的解析ツールを始めることができる
  - Analyzer、テストコード、main.goの雛形作ってくれる

```
$ skeleton myanalyzer
myanalyzer
├── cmd
│   └── myanalyzer
│       └── main.go
├── go.mod
├── myanalyzer.go
├── myanalyzer_test.go
└── testdata
    └── src
        └── a
            ├── a.go
            └── go.mod
```

Demo

skeletonが生成する雛形の種類

- -kindオプションで指定できる
  - inspect（デフォルト）
    - 抽象構文木の探索を行うコードの雛形
  - ssa
    - 単一代入（Static Single Assign）形式を扱うコードの雛形
  - codegen
    - コード生成を行うコードの雛形

```
$ skeleton -kind codegen mygenerator
mygenerator
├── cmd
│   └── mygenerator
│       └── main.go
├── go.mod
├── mygenerator.go
├── mygenerator_test.go
└── testdata
    └── src
        └── a
            ├── a.go
            ├── go.mod
            └── mygenerator.golden
```

その他のskeletonのオプション

- -kind以外にもオプションがある
  - 生成される雛形が異なってくる

```
$ skeleton -h
skeleton [-checker,-kind,-cmd,-plugin] example.com/path
  -checker value
     [unit,single,multi]
  -cmd
     create main file (default true)
  -kind value
     [inspect,ssa,codegen]
  -pkg string
     package name
  -plugin
     create golangci-lint plugin
```

skeletonを開発する上で考慮したこと

- テンプレートの管理が面倒
  - オプションによって生成する雛形が多岐にわたる
  - ファイルが複数ある
  - テンプレートをバイナリに含めたい
- テストがしづらい
  - 生成物が多い
  - ディレクトリやファイルを生成するためモックにしづらい
  - ディレクトリ構造も合わせてテストしたい

io/fsパッケージで解決！！

io/fsパッケージを使った
テンプレート管理

ファイルシステムの抽象化

- Go1.16からio/fsパッケージとして導入された
  - fs.FSインタフェースによって表現される
  - ファイルを開くという最低限の機能を定義している

```
type FS interface {
    Open(name string) (File, error)
}
```

宣伝：エキスパートたちの Go言語でも書いてます！

抽象化されて得るメリット

- 例：ZIPファイル
  - ZIPファイルをファイルシステムとして扱う
  - *zip.Reader型がOpenメソッドを実装
  - template.ParseFS関数など引数に渡せる

```
func main() {
    r, err := zip.OpenReader("templates.zip")
    if err != nil { /* エラー処理 */ }
    defer r.Close()

    tmpl, err := template.ParseFS(r, "*.html")
    if err != nil { /* エラー処理 */ }
    /* tmplを使う処理 */
}
```

embedパッケージ

- io/fsパッケージと同様にGo1.16で入った
  - ファイルやディレクトリを埋め込むことができる
    - go-bindataやstatikなどと同じようなことができる
    - go toolに統合されているためgo generate不要
  - embed.FS型によってファイルシステムとして埋め込める
    - fs.FSインタフェースを実装

```
// templateディレクトリ以下をバイナリに埋め込む
//go:embed template/*
var tmplFS embed.FS

// 埋め込まれたtemplateディレクトリをファイルシステムとして扱う
var tmpl = template.Must(template.ParseFS(tmplFS, "*.html"))
```

txtar形式

- 複数のテキストファイルをまとめて扱う
  - Goコンパイラのテストに使われる
    - Go Modules (vgo) とともに導入
  - The Go Playgroundでも利用可能
  - golang.org/x/tools/txtarパッケージでパースできる
  - `-- path/to/file --` でファイルを区切る

```
package main
import "play.ground/foo"
func main() { foo.Bar() }

-- go.mod --
module play.ground

-- foo/foo.go --
package foo
import "fmt"
func Bar() { fmt.Println("This function lives in an another file!") }
```

txtarfsパッケージ

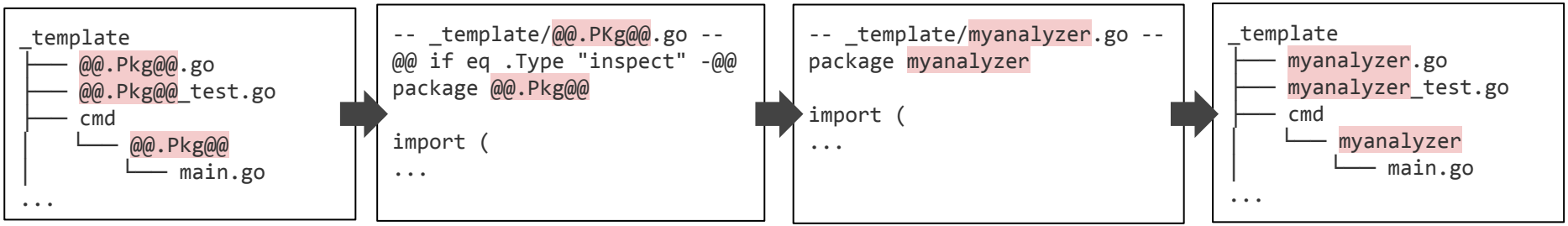
- txtar形式をファイルシステムとして扱うパッケージ
  - github.com/josharian/txtarfs
  - x/tools/txtarパッケージにプロポーザルが出されている
  - txtarfs.As関数でfs.FS型にする
  - txtarfs.From関数で*txtar.Archive型にする

Playgroundで動かす

```
ar := &txtar.Archive{Files: []txtar.File{
    txtar.File{Name: "a/a.txt", Data: []byte("A")},
    txtar.File{Name: "b/b.txt", Data: []byte("B")},
}}
fsys := txtarfs.As(ar)
fs.WalkDir(fsys, ".", func(path string, _ fs.DirEntry, _ error) error {
    fmt.Println(path)
    return nil
})
ar2, err := txtarfs.From(fsys)
if err != nil { /* エラー処理*/ }
fmt.Println(ar2)
```

txtar形式でテンプレートを管理

- skeleton v1の方法
  - 自前ツールで_templateディレクトリ以下をtxtar形式変換
  - txtar形式のテンプレートをtemplate.goに文字列型変数として定義
  - @@をデリミタとしてパース
  - テンプレートにデータを埋め込む
  - txtar形式をパースし、ディレクトリに変換



txtar形式でテンプレートを管理

- skeleton v2の方法
  - embed.FS型としてテンプレートをディレクトリごと埋め込む
  - txtarfsパッケージを用いてtxtar形式に変換
  - @@をデリミタとしてパース
  - テンプレートにデータを埋め込む
  - txtar形式をtxtarfsパッケージでfs.FS型にしファイルを作成

Playgroundで動かす

```
//go:embed _template
var tmplFS embed.FS
```

```
fsys, _ := fs.Sub(tmplFS, "_template")
ar, _ := txtarfs.From(fsys)
tmplStr := string(txtar.Format(ar))
tmpl, _ := template.New("tmpl").Delims("@@", "@@").Parse(tmplStr)
var buf bytes.Buffer
_ = tmpl.Execute(&buf, struct{ Pkg string }{Pkg: "myanalyzer"})
outfs := txtarfs.As(txtar.Parse(buf.Bytes()))
fs.WalkDir(outfs, ".", func(path string, _ fs.DirEntry, _ error) error { /* 略 */ })
```

txtarfsパッケージを使う利点

- go generateコマンドを使う必要がない
  - ビルド時にembedパッケージの機能で埋め込まれる
  - _templateディレクトリ以下をいじるだけ済む
  - テストしやすくなる（後述）

embedパッケージを使う際の注意点

- go.modファイルを含めることができない
  - @@gomod@@のような名前にする
  - テンプレートのFuncMapの機能でgo.modを生成する関数を設定する
    - golang.org/x/mod/modfileパッケージを利用

```
// go.modの生成
var mf modfile.File
_ = mf.AddModuleStmt(path)
gov, _ := goVersion()
_ = mf.AddGoStmt(gov)
b, _ := mf.Format()
```

```
func goVersion() (string, error) {
    tags := build.Default.ReleaseTags
    for i := len(tags) - 1; i >= 0; i-- {
        version := tags[i]
        if strings.HasPrefix(version, "go") &&
              modfile.GoVersionRE.MatchString(version[2:]) {
            return version[2:], nil
        }
    }
    return "", errors.New("there are not valid go version")
}
```

skeletonkit

- skeletonライクなツールを作るためのライブラリ
  - https://github.com/gostaticanalysis/skeletonkit
  - skeleton v2.0.5から使用
  - gqlgo/gqlanalysis/cmd/gqlskeletonでも使用
  - ディレクトリ単位のテンプレートのパース
  - 任意のデータをテンプレートに埋め込む
  - fs.FS型としてテンプレートからファイルを生成する

```
tmpl, err := skeletonkit.ParseTemplate(tmplFS, "skeleton", "_template")
if err != nil { /* エラー処理 */ }
fsys, err := skeletonkit.ExecuteTemplate(g.template(), info)
if err != nil { /* エラー処理 */ }
if err := skeletonkit.CreateDir(skeletonkit.DefaultPrompt, dst, fsys); err != nil {
    /* エラー処理 */
}
```

io/fsパッケージを用いた ゴールデンファイルテスト

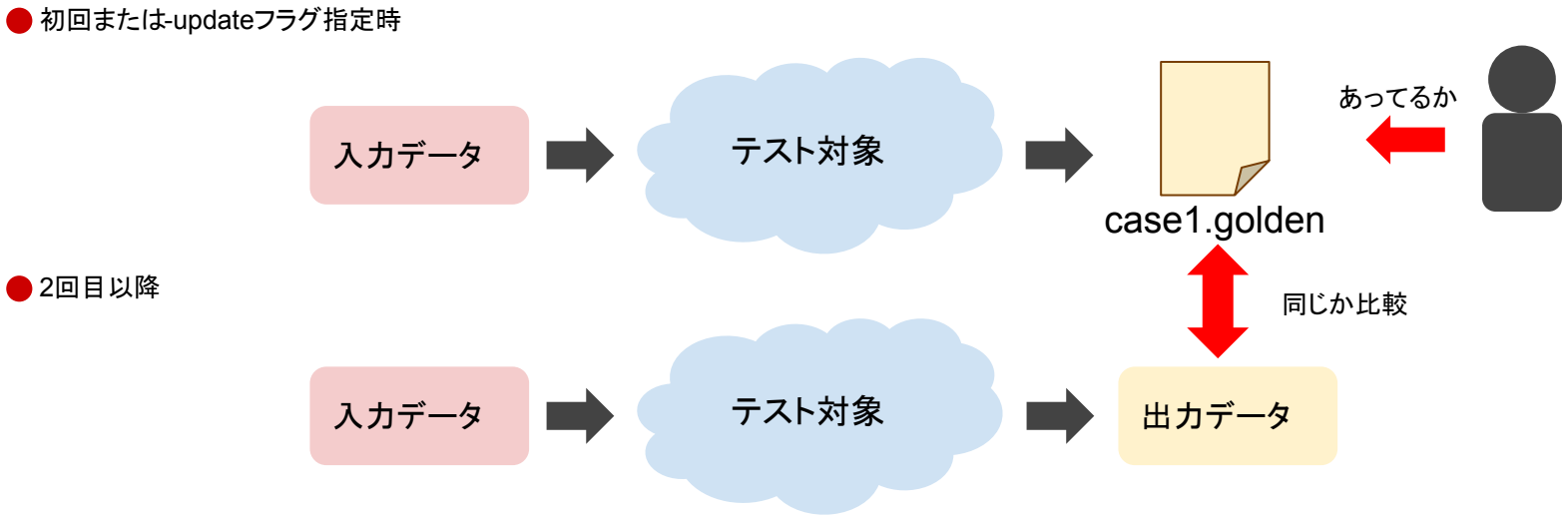
コード生成ツールのテスト

- オプションによって生成物が多岐にわたる
  - チェックが大変でテストケースが分かりづらい
  - ファイルを生成するため検証がめんどう
  - ディレクトリの構造もチェックが必要

txtar形式とio/fsパッケージを使った
ゴールデンファイルテストで解決！

ゴールデンファイルテスト

- 期待するデータをファイルで用意する
  - レスポンスなど実データを記録しておく
  - -updateフラグでアップデートできる（最初は人が確認）
  - .goldenという拡張子で記録する



tenntenn/golden

- ゴールデンファイルテストを行うライブラリ
  - https://github.com/tenntenn/golden
  - golden.Update関数でゴールデンファイルをアップデート
    - string, []byte, io.Reader, encoding.TextMarshalerに対応
    - それ以外はすべてJSONにする
  - golden.Diff関数で既存のゴールデンファイルとの差分を取得する
    - go-cmpを使って比較する

```
var flagUpdate bool
func init() { flag.BoolVar(&flagUpdate, "update", false, "update golden files")}
func Test(t *testing.T) {
    got := Target()
    if flagUpdate { golden.Update(t, "testdata", "mytest", got); return}
    if diff := golden.Diff(t, "testdata", "mytest", got); diff != "" { t.Error(diff) }
}
```

txtar形式とゴールデンファイルテスト

- 出力したディレクトリをゴールデンファイルにする
  - 出力先を(*testing.T).TempDirメソッドで作成
  - 出力ディレクトリをos.DirFS関数でfs.FS型として開く
  - txtarfs.From関数でtxtar形式に変換する
  - txtar形式の出力データをゴールデンファイルとして記録する
- tenntenn/goldenではtxtar形式に関する関数を提供
  - skeletonにおける使用例
  - golden.DirInit関数
    - txtar形式で指定した状態でディレクトリを初期化
  - golden.Txtar関数
    - 指定したディレクトリ以下をtxtar形式で取得
  - golden.TxtarWith関数
    - 引数で指定したファイルとデータでtxtar形式を作成

Demo

まとめ

- io/fsパッケージとembedパッケージは便利
  - ディレクトリをポータブルに扱える
  - ファイルシステムが透過的に使える
  - skeletonkit便利
- txtar形式とゴールデンファイルテストの相性は良い
  - txtar形式は抜群の汎用力
  - 複雑なテストはゴールデンファイルテストでスッキリ
  - tenntenn/golden便利