アプリケーションサーバにおける リクエスト処理とページレイアウトの分離

(株)東芝　情報・社会システム社
SI技術開発センター SI技術担当
松尾尚典

TOSHIBA

目次

- 1 Webアプリケーション開発の問題点
- 2 リクエスト処理とページレイアウトの分離
- 3 リクエスト処理オブジェクト
- 4 ページレイアウトオブジェクト
- 5 まとめ

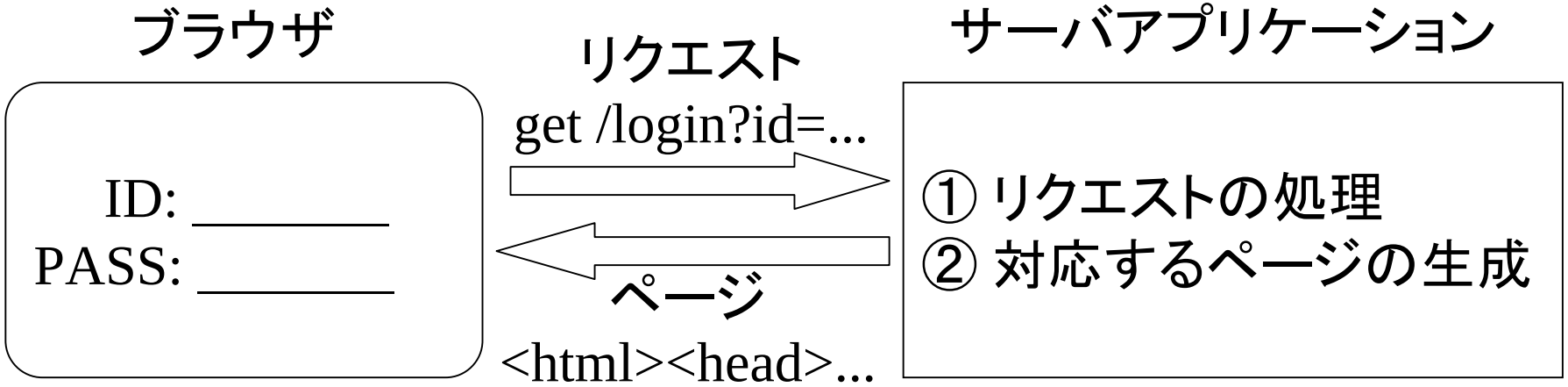
TOSHIBA

1 Webアプリケーション

- よく見かけるCGIなアプリケーション
  - 掲示板、ユーザ登録、情報サービス…
- 画面遷移が少ないものが殆ど
  - 一つの画面で入力、もう一つの画面で出力
- 一般のGUIアプリケーションなら数十、数百の画面を扱っているのに…
- もう少し画面数の多い、役に立つアプリケーションを作りたい

TOSHIBA

サーバアプリケーションの処理



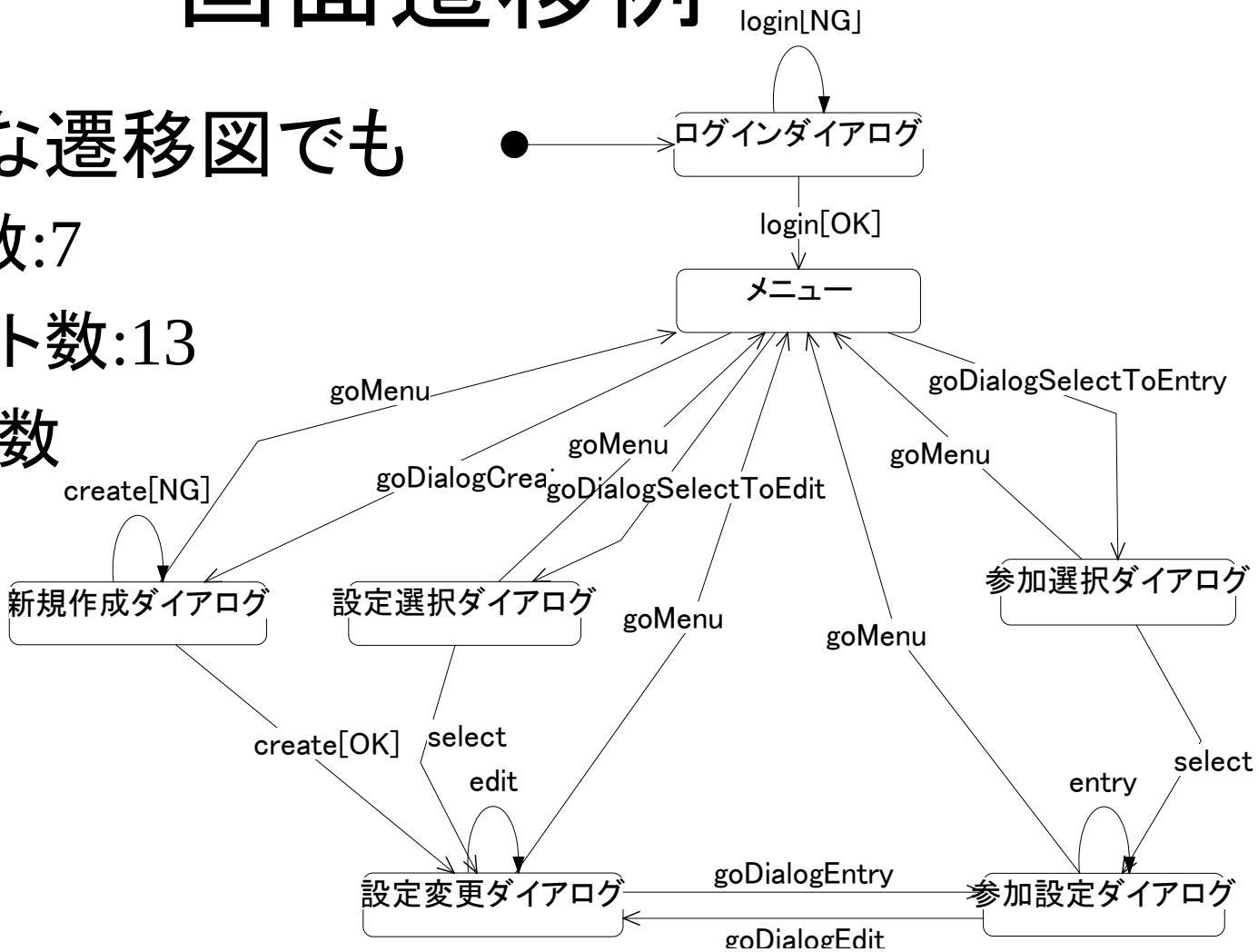
- 多くは①と②を一緒に記述
  - 教本のサンプルコード等殆ど全て
- 単純なアプリケーションではそれで十分
- しかし、リクエストやページの種類、関連数が多くなると、途端に開発が難しくなる

TOSHIBA

画面遷移例

こんな小さな遷移図でも

- ページ数:7
- リクエスト数:13
- 関連:多数



TOSHIBA

リクエストとページ

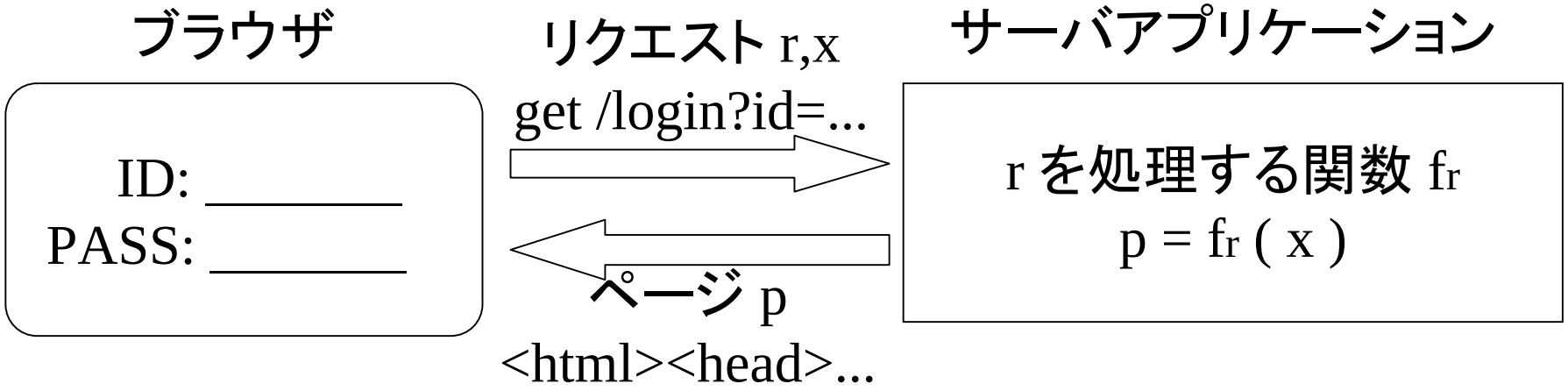
- リクエストとページは多対多
  - リクエストとページを、1対1として一緒に処理するアプローチでは対応できない
  - 一部のリクエストとページは密接に結びついているので、うまい工夫が出来そう
  - だが全てを奇麗に解決する手段は?
- 多数の画面を扱うのはやっぱり面倒
  - みんないろいろ工夫している

TOSHIBA

アプリケーションサーバでは

- 一つの解として、リクエストとページの分離
  - リクエストの処理とページレイアウトを分離出来る構造、メカニズムを提供
  - もちろんこれで全て解決できるわけではない
  - アプリケーションサーバのメリットがこれだけという訳でもない

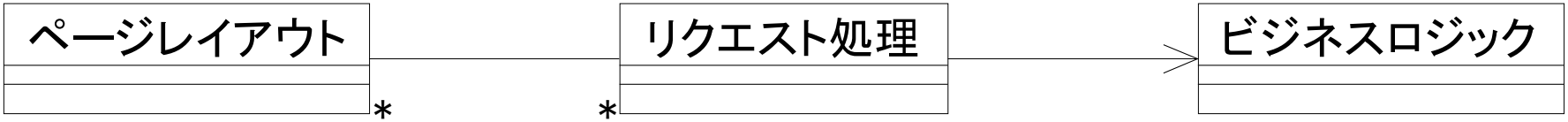
2 リクエストとページの分離



- データ x を持つリクエスト r に対応したページ p を選択する関数 f_r
- 関数 f_r とページ p は明確に分離出来る
  - f_r :リクエスト処理オブジェクト
  - p :ページレイアウトオブジェクト

TOSHIBA

分離することのメリット



- 責任と処理内容の明確化
  - 誰が何をするのか
- 開発効率、分業化、保守性
  - ページレイアウトはデザインの専門家に
  - リクエスト処理はプログラマに
  - ビジネスロジックは業務の専門家に

TOSHIBA

アプリケーションサーバにおけるリクエストとページの分離

- リクエスト処理とページレイアウト
  - 例えば Servlet / JSP、VBScript / ASP、Requestオブジェクト / Pageオブジェクト
- どちらもプログラムだから、やろうと思えばなんでも出来る
- だが、責任範囲以外のことはやるな!
  - わざわざ分離したんだから…

TOSHIBA

フレームワーク利用の心構え

- そもそもフレームワークは制約の塊
  - あれをしてはいけない、これをしてもいけない
  - このフォーマットに従え、範囲を逸脱するな…
- しかしその制約に従って初めて便利
  - 制約があるから迷わずに済む、ありがたい
  - 分業化、パターン化、コンポーネント化
- 開発では、制約に誠実に従うが吉
  - 現実には、構造だけ提供される場合が多い
  - 制約(=ノウハウ)部分は別売

TOSHIBA

3 リクエスト処理オブジェクト

- ① データを受け取り、
- ② 対応するビジネスロジックを実行し、
- ③ ページレイアウトオブジェクトを選択する

TOSHIBA

責任

- 渡されたデータをビジネスロジック用に正規化する
- ビジネスロジックを実行する
- 正常系、準正常系、異常系で分岐する
- ページレイアウトオブジェクト用にデータを正規化する
- ページレイアウトオブジェクトを呼び出す
- その他のことはやっては *いけない*

TOSHIBA

コードのパターン化(Ruby版)

```
begin  #  Java なら try - catch
    < データの正規化(ビジネスロジック用) >
    < ビジネスロジックの実行 >
    < データの正規化(ページレイアウトオブジェクト用) >
    < 正常系のページレイアウトオブジェクトの呼び出し >
rescue BusinessLogicException
    case $!.to_s
    when < 例外X >
        < データの正規化(ページレイアウトオブジェクト用) >
        < 例外Xのページレイアウトオブジェクトの呼び出し >
    when
        ...
    else
        raise  # 処理できなかった例外は再発生させる
    end
end
# rescue されなかった例外はスルーされる
```

正常系

準正常系

異常系

TOSHIBA

４ ページレイアウトオブジェクト

- 与えられたデータを用いて、HTML文字列を生成する
  - これだけ
  - デザイナがデザインに集中できる
  - データで表現可能な範囲なら、変更も容易

TOSHIBA

責任

- 渡されたデータをページとして表現する
- リクエストオブジェクト名をリンク先とする
- その他のことはやっては *いけない*

- 前提条件
  - 必要十分なデータが渡される
  - データは表示用に正規化されている
  - 必要なら、データは表示用の構造にパックされている

TOSHIBA

ページのパターン化

- 表示すべきデータは全て与えられているので、プログラミングレス
  → デザイナの手持ちのパターンが使える
- パターンの集積が可能
  → どんどん再利用

TOSHIBA

5 まとめ

- Webアプリケーション開発では、リクエスト処理とページレイアウトを明確に分離するとよい
- アプリケーションサーバは、分離するための構造を提供している
- 開発においては、分離する意義を十分理解して、与えられた構造を利用すべきである