情報化シンポジウム広島’08　10月30日(木)技術セミナー

『Rubyの基幹業務開発』

～COBOLエンジニアの復活～

2008. 10. 30

㈱テクノプロジェクト

吉 岡　宏

IPA公募事業　①医療・介護高額合算システム

住民記録システム

国民健康保険システム
（医療保険）

後期高齢者システム
（医療保険75歳以上）

介護保険システム

住民異動情報

自己負担額・
資格情報

支給額情報

自己負担額・
資格情報

支給額情報

自己負担額・
資格情報

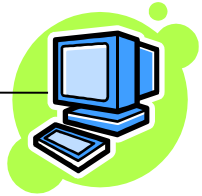
支給額情報

医療
介護
高額
合算
システム

既存業務システム

今回対象システム

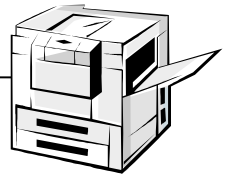
窓口業務



市民

相談支援
申請受付
証明書発行など

内部処理



自己負担額合算
支給決定
支払い登録
口座振替情報など

高額合算制度

医療・介護の自己負担額が、所得に応じて一定額を超したら還付をする。
今年4月からスタートする後期高齢者医療保険制度と同時に実施される。
1年間の自己負担合計額が対象となるため、実質平成21年度から窓口業務が開始される。

IPA公募事業　②Rubyの新しい取り組み

大量のデータ処理

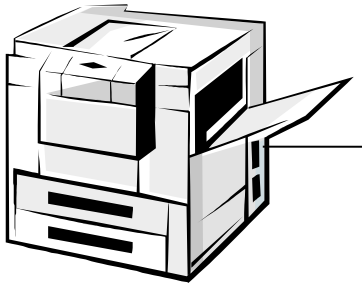
精度の高い帳票印刷

再印刷やページ指定印刷

夜間の無人運転支援

プログラムの保守性向上

COBOLエンジニアのシフト



今回、新たに取り組む分野

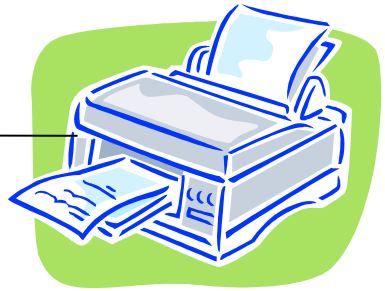
高額

合算

システム

画面からのデータ登録

少量のデータ処理



これまでに実績のある分野

基幹業務システム（大規模システム）への拡大

設計方式

Ruby　⇒ オブジェクト指向言語 ⇒ オブジェクト指向設計（UML）
COBOL ⇒ 手続き型言語 ⇒ 手続き型（構造化）設計

今回　手続き型（構造化）設計　・・・ 業務フロー図、画面・帳票、機能設計書（詳細）

開発方式

ウォータフォールを採用 ← アジャイルではない。
しっかり設計をやり、工程毎に顧客レビューを実施

プロジェクト体制

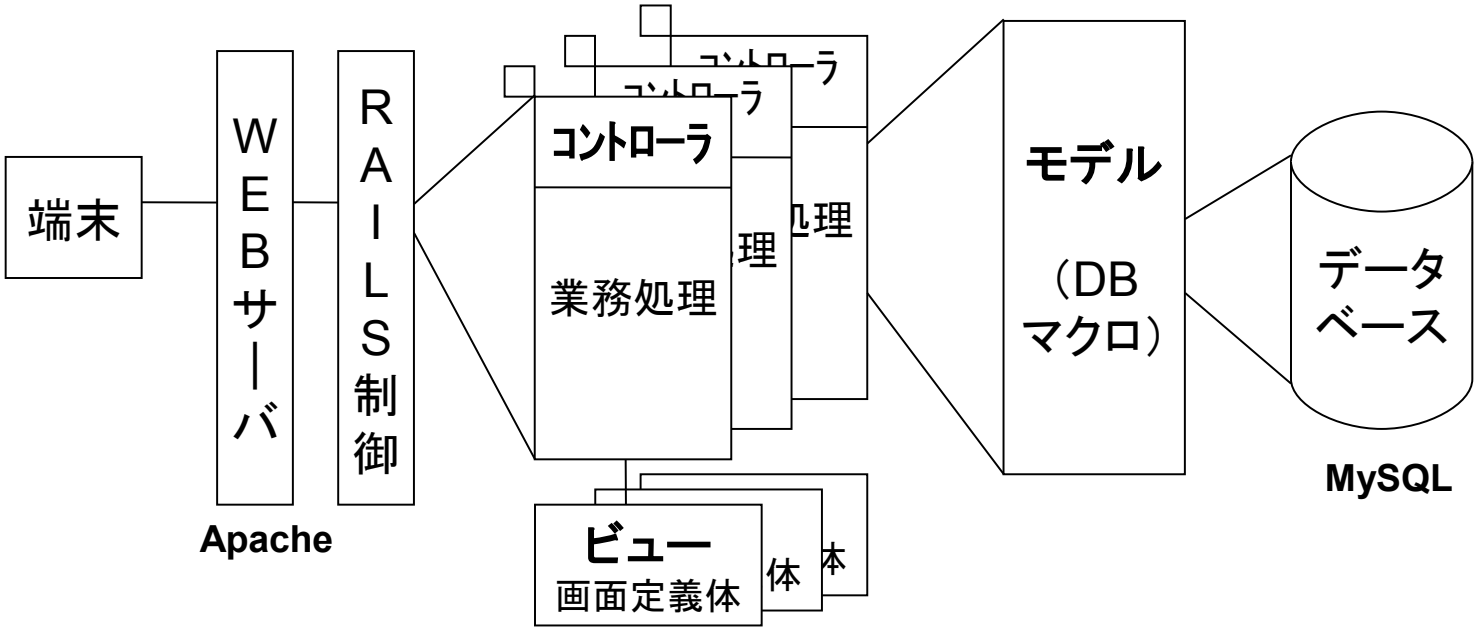
開発期間：2007．9～2008．2　オン：14本、バッチ：24本

業務開発チーム（TPJ：テクノプロジェクト・マツケイ中心）
50代SE　2名、30代SE／PG　3名、20代PG　3名
業務分析：顧客要件整理（含む厚生労働省の新制度説明資料）
設計（業務フロー、画面、帳票、詳細機能説明書）
開発（プログラミング、単体テスト、ソースレビュー）
テスト（結合テスト、システムテスト＝実証実験）

共通技術チーム（NaCl：ネットワーク応用通信研究所中心＋TPJ）
Ruby教育、プロトタイプ作成、コーディング規約作成＆ソース点検
ツール作成（大量帳票印刷・バッチ自動運用）
ライブラリィの組み込みと使用方法整理・・・祝日判定・DB複合キーなど
共通ルーチン作成（部品化）、開発環境整備

オンライン開発（画面）

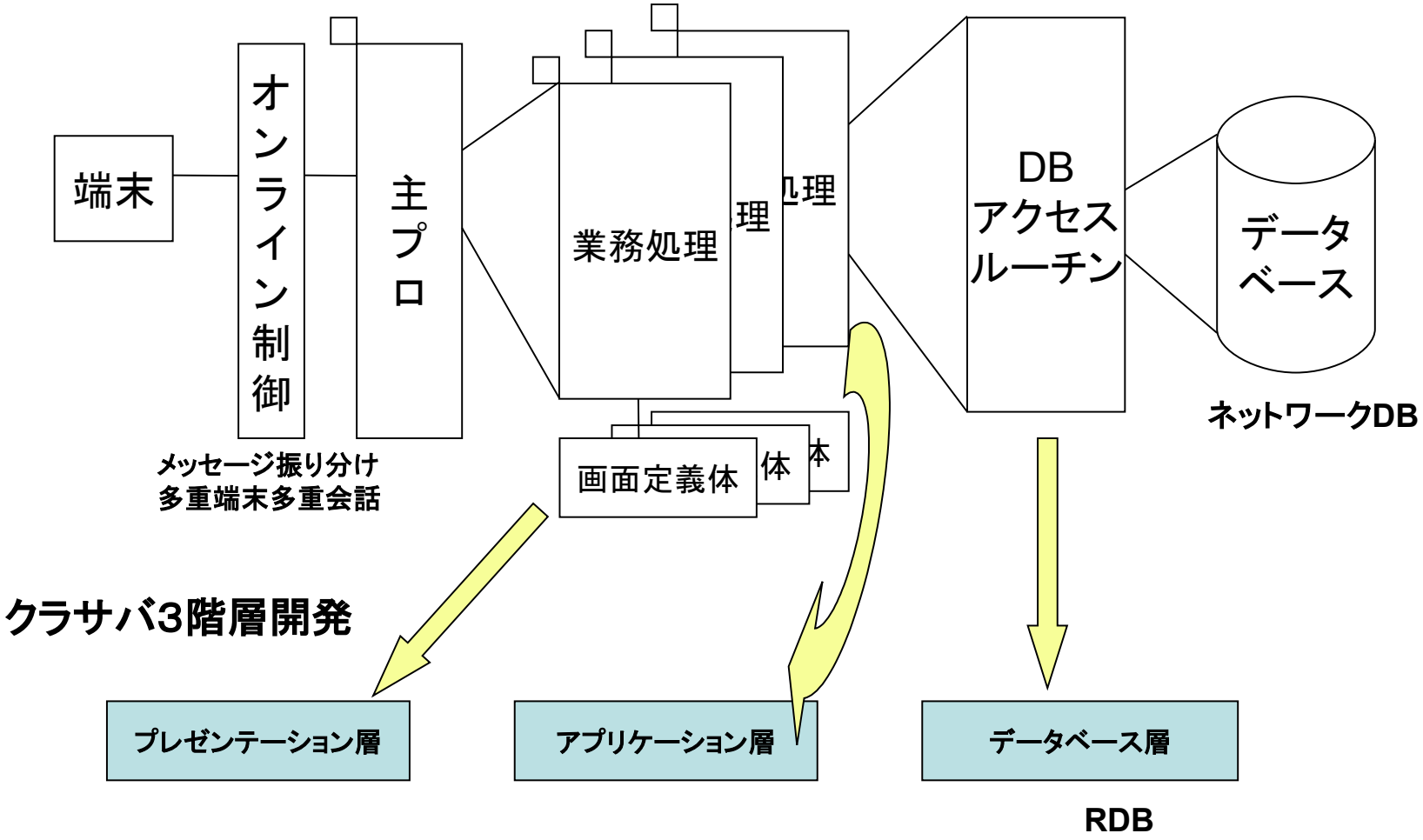
Ruby　on Rails（機能分割：ﾓﾃﾞﾙ・ﾋﾞｭｰ・ｺﾝﾄﾛｰﾗ）



※Ruby on Rails
・モデル（DBマクロ）　検索・挿入・更新・削除のSQL
・ビュー（画面定義）　DBの追加・更新・削除の画面HTML
・コントローラ（オンライン制御）
フレームを自動生成してくれ、Webアプリが動くようになる。

※ビュー
Webのホームページ作成ツールでHTML定義のひな形を作成する。
画面の統一性を出すため、１人のデザイナーが全画面作成する。
プログラマーはHTML定義にプログラムとの変数を挿入し、ビューを完成させる。

レガシー開発（画面）



業務処理に集中して開発できる環境があった。
オンラインやDBで、予期せぬ動きはない。安心して、開発できる環境であった。

バッチ業務

必要性

複数部門で利用するシステムの場合、オンライン中に大量のデータ処理を実行すると　①オンラインのレスポンス悪化　②排他制御によりオンライン停止あるいは　③デッドロックが起こる。
このため、大量のデータ処理は、オンライン終了後に実行する。

スケジューリング

他システムとのデータ受け渡しやバックアップ、当日の登録データー覧表作成などの定常処理とその日に担当者が要求する特定帳票の出力などの処理がある。
⇒　自動的に運用したい。実行する順番がある。

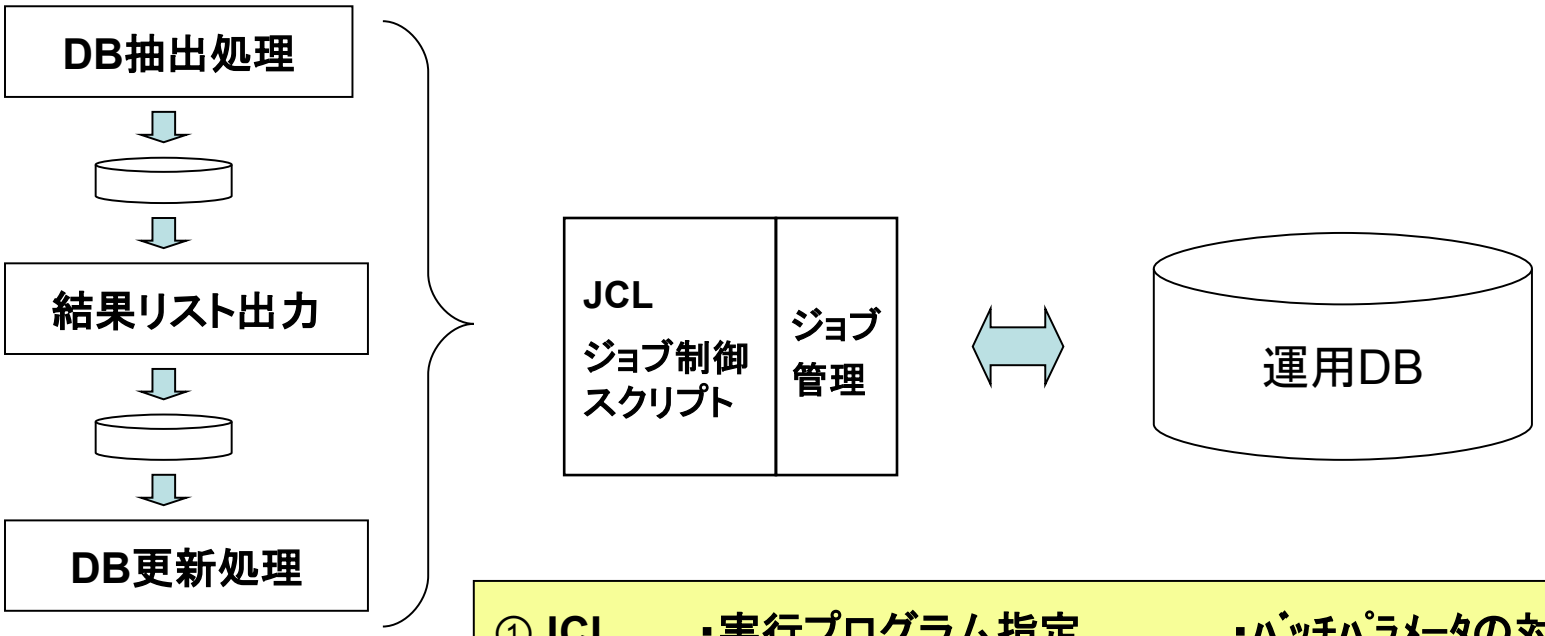
帳票出力

大量の帳票出力処理は、特定のページのみ出力したいケースやプリンターのトラブルのために再印刷する機能が必要である。
住民に渡す帳票なので、フォームを使用した判り易さが求められる。

バッチ業務（ジョブ管理）

機能　　・バッチジョブの実行パラメータを事前に設定できること（オンライン中）
　　・バッチ処理の実行結果の確認を翌日以降にできること（夜間自動運用）

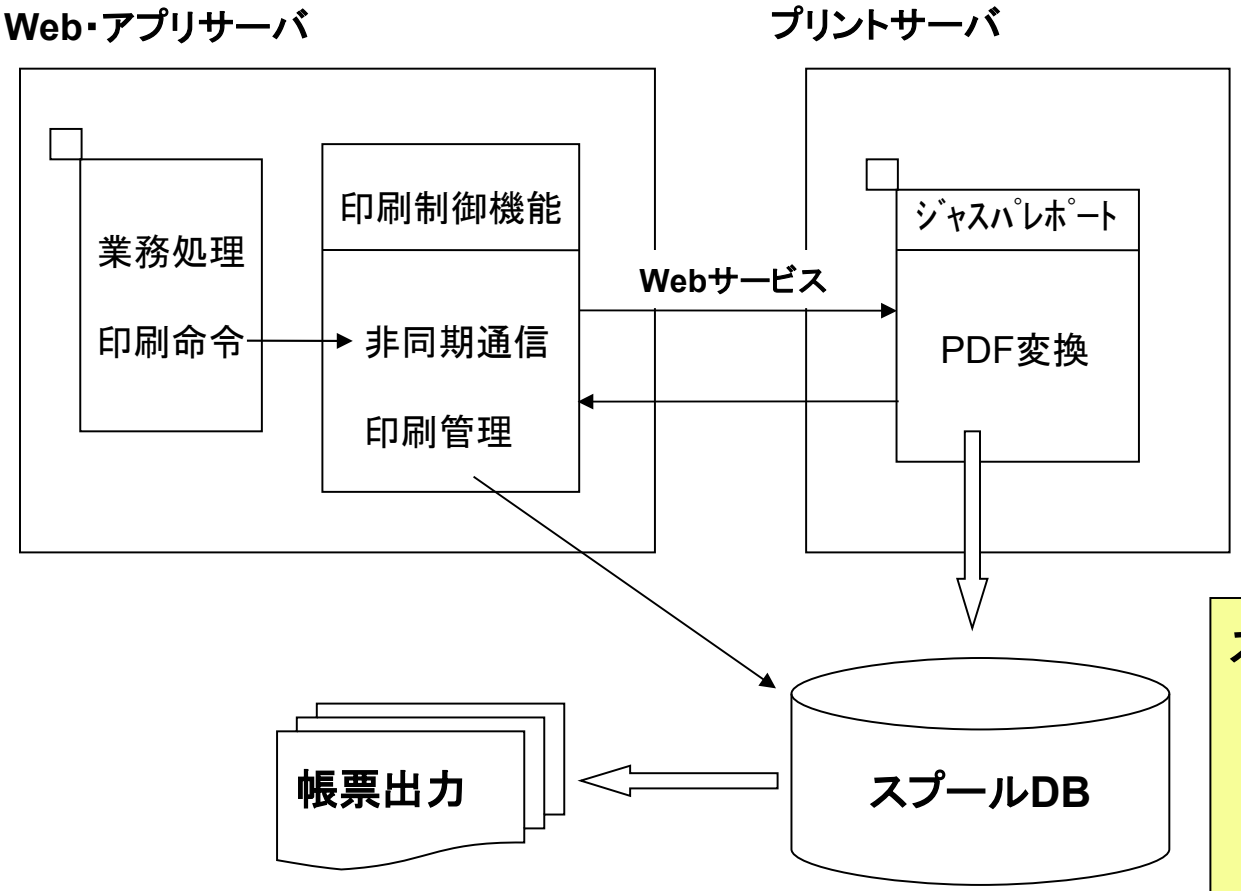
ツール新規作成
　　バッチプログラムの初期処理・終了処理時に、自動運用の関数（サブルーチン）



①JCL　・実行プログラム指定　・ﾊﾞｯﾁﾊﾟﾗﾒｰﾀの対応付け
②ﾌﾟﾛｸﾞﾗﾑ　・ログの書き出し　・ﾊﾞｯﾁﾊﾟﾗﾒｰﾀの受け取り
　　・ﾜｰｸﾌｧｲﾙの定義（SQL）
③ジョブ管理機能　・実行結果の判定　・ﾊﾞｯﾁﾊﾟﾗﾒｰﾀの事前設定
　　・実行結果のログ参照（ジョブ単位）

バッチ業務（スプール機能）

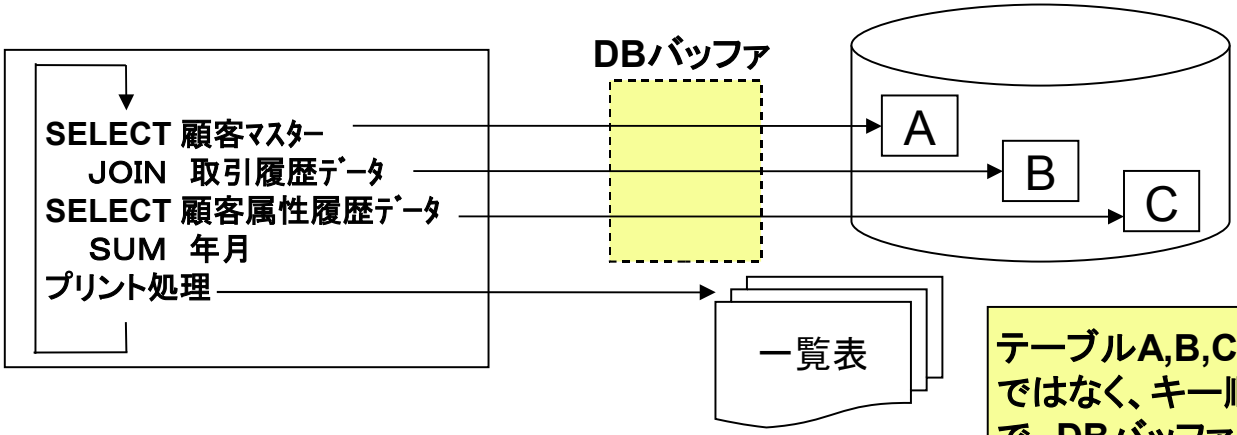
帳票印刷
　　iReport と ジャスパーレポートの活用（Javaオープンソース、フォーム作成・PDF変換）
ツール新規作成
　ジャスパーレポートの運用管理機能を新規作成する。JRubyで開発。



スプール機能
- ・プレビュー
- ・ページ指定印刷
- ・再印刷
- ・印刷中キャンセル
- ・検索・削除

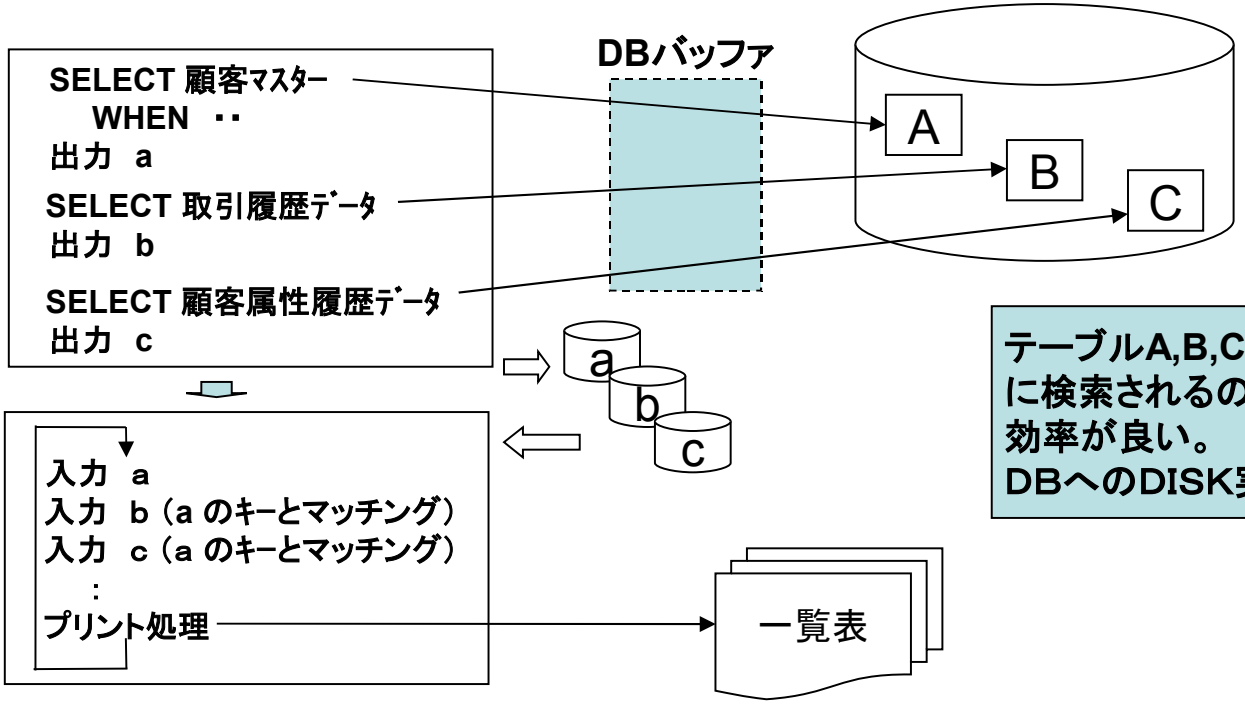
バッチ業務（大量データ処理）

遅くなる例



テーブルA,B,Cがそれぞれ格納順ではなく、キー順にアクセスするので、DBバッファの効率が悪い。
DBへのDISK実アクセスが多くなる。

速くなる例



テーブルA,B,Cがそれぞれ格納順に検索されるので、DBバッファの効率が良い。
DBへのDISK実アクセスが少ない。

バッチ業務（大量データ処理）

DISKアクセスとロジック（CPU）の処理時間比較　・・・　数値は想定値

DISKアクセス1回の処理時間　：　10m秒程度　＝　10×10-3秒　　←　①

サーチ時間：7200RPM(秒当り120回転=8m秒×1/2）＋シーク時間＋転送時間

Ruby 100Stepの実行時間

＝　100（Rubyｽﾃｯﾌﾟ数）×10（倍）×5（マシン語のステップ数/C言語1ｽﾃｯﾌﾟ）
×5（クロック数/マシン語1ｽﾃｯﾌﾟ）×0.5×10-9（秒）

＝　12.5×10-6秒　　←　②

※Rubyは実測レベルで、Cの数倍から10倍の実行速度である（インタプリタ）

※CPU：2GHz　（1クロックの時間＝0.5×10-9秒）

⇒　DISKアクセスとRuby100ステップの実行速度は1000倍程度の差がある。

20年程度前と現在の比較

	CPU速度	DISK アクセス	メモリ	大量 データ	ファイル	DB チューニング	言語
20年 程度前	MHz (FLOPS)	20～40 mS	MB	数十万 ～数百万	NDB 順編成	SEノウハウ ﾀﾞｲﾅﾐｯｸ ｽﾃｯﾌﾟ	COBOL PL/I
現　在	GHz	10mS	GB	数千万 ～数億	RDB	自動化 SEノウハウ	VB,Java Ruby

Ruby基幹業務開発（生産性）

①システム開発の工程別比率

設計工程(30%)＋製造工程(PG設計～単体テスト40%)＋テスト工程(30%)　⇒　全工程(100%)

②Rubyによる生産性（Rails研修後）

設計工程(30%)＋製造工程(40%)×1/2＋テスト工程(30%)×2/3　⇒　全工程(70%)

Ruby on Railsは、DBのテーブルを定義するとデータ操作用のWebアプリが簡単にできる。

テストは手順を登録すると、自動テストが可能である。（障害時の再テスト・レグレッションテスト）

③Rubyによる生産性（開発を終えて、開発者の感想）

設計工程(30%)＋製造工程(40%)×3/4＋テスト工程(30%)×1/1　⇒　全工程(90%)

・チェック処理が多い。1つのデータ登録に、100ステップ程度の整合性チェックをする。

・自動テスト機能は、プログラムの状態やデータベースの状態が同じでなければ、使えない。
短期開発では、わざわざテスト手順を登録し、データをバックアップする余裕はない。

・部品化 ： 設計中に共通部品の洗い出し　⇒　多くはできない。数個程度。
Railsの組み込み部品が有効。例）20件のﾃﾞｰﾀを検索表示 → 次の20件が部品。

・プログラム修正から単体テストがスムーズにできる。

・基本的に、1度限りの開発である。繰り返しによる部品の充実が図れない。

COBOL開発（ホスト・オフコン・クラサバ3階層）とRuby(Rails)は、ほぼ同じである。

Ruby基幹業務開発（メンテナンス性）

前提

・基幹業務は、企業活動になくてはならないシステムであり、変化に対応して
長い期間利用される。ロングライフであること。
・長いシステムの利用の中では、プログラムの担当SEが替わり、メンテナンスが
継続して行なわれる。誰でもメンテナンスできること。

管理された開発　⇒　可読性が良くなる
標準化＝コーディング規約を厳しく設ける。
ソースレビューの徹底（スパゲティ状態にならないこと）
豊富なRubyの言語仕様を制限して利用する。

Ruby：プログラミングのたのしさ
（プログラマーが想像力豊かに、自由に組める）

プロジェクトチームによるものづくりの楽しさ
基幹業務システム構築　by Ruby

Ruby基幹業務開発（その他のポイント）

①データのテーブル名・項目名 ： 日本語名称を使用しない　＝　ソースの可読性が低下する

受給申請者　→　jukyu-shinsei-sha

日本語サポート（RubyとMySQLは単独でOK、Railsもモデルに事例ナシ）、HTMLの中に、日本語名称が入るとエラーとなる。Input-id=“product_日本語名称”　⇒　XHTMLを利用すればOK。

Railsはテーブル名：複数形、クラスは単数形に自動変換する。

例）日本語のケース ： 顧客 → 顧客s　　英語のケース ： Customer → Customers

②計算精度 ： 固定小数点演算をサポートしている。

（RubyとMySQLでDecimal属性をサポート）

小数点2桁目を四捨五入し、小数点1桁目までを求める。

③データ交換 ： スペースを詰め込む関数で対応。

レガシーや金融機関とのデータ交換に固定長のデータ形式が必要となる。

④Ajax ：全面的に採用した。

画面の一部を書き換える処理が多い。画面のチラツキを押さえた使いやすい画面を実現。

⑤コーディング規約 ： インデント（字下げ）、1行の桁数80、メソッドのカッコを必須、改行コードはLF

文字列のクォートは‘（シングル）、unlessは使用禁止、グローバル変数の使用、SQL記述　など

⑥データベースの主キー ： 複合キー　（複数の項目を組み合わせたものを主キーとする）

例）　年度＋申請者番号＋履歴番号

⑦オープンソースを複数種類組み合わせた開発は、初期コストが確実に安くなる。但し、それぞれでバージョンアップをするので、要注意。　⇒　実績のある組合わせ

Ruby基幹業務開発（COBOL'erの復活）

業種・業務ノウハウ

・自治体の特性

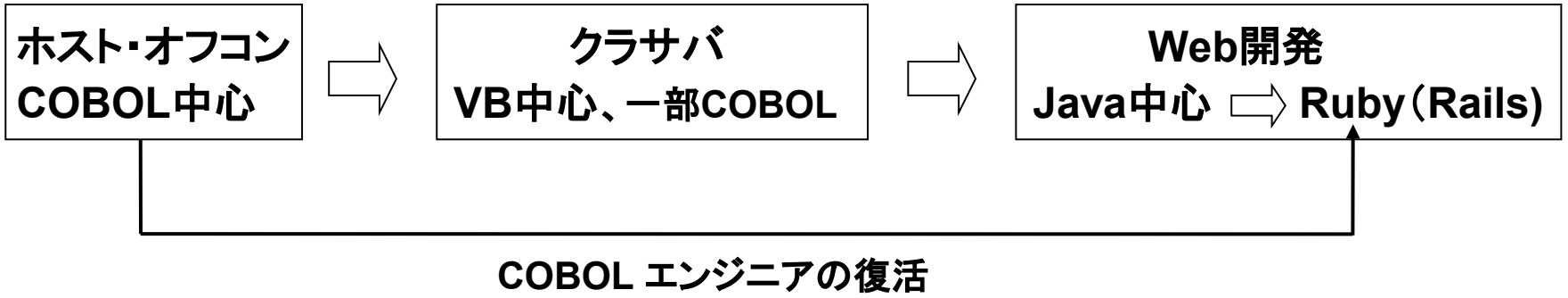
住民サービス　⇒　起算日処理が必要（申請や届け出の遅れを取り戻す）
データ修正の記録（履歴管理）
フォームを使った判りやすい帳票

職員の定期的な異動　⇒　他システムと同等な使い勝手
判りやすさ　＝　画面の階層を浅くする（細かな処理に分けない）
＝　1つの画面が複雑になる（登録・訂正・削除が1画面）

・業務の特性

医療保険や介護保険の自己負担の還付制度と類似している。
法律のあいまいさ　＝　詳細な箇所が記載されていない　＝　厚労省とのQAがロジック化

基幹業務開発：ベテランSE（COBOL'er）のノウハウが重要



※レガシー：過去の遺産・時代遅れ

COBOLエンジニア ＝ 業務ノウハウ

① 設計工程　構造化設計 ： 業務フロー、画面・帳票、機能設計書（概要・詳細）

~~UML設計（ユースケース図、シーケンス図、コンポーネット図・・・）~~

② 製造工程 （プログラム設計・プログラムング・単体テスト） 部品化（できる範囲で）

Rubyの理解できる人 ： 製造作業

Rubyの理解できない人 ： テスト仕様書作成、ソースレビュー（聞き手）

設計の残項目整理

~~アジャイル開発（小グループによるスピード開発、ドキュメントは後回し）~~

③ テストエ程 （結合テスト・システムテスト・運用テスト）

テスト仕様書＆成績書、テスト（テストデータ作成支援）

ドキュメント整備（操作マニュアル・運用マニュアル・設計書）

操作研修支援、本稼動支援

オブジェクト指向言語Rubyによる基幹業務開発

やることは同じである。言語が変っただけ。

Webアプリ開発

COBOLｴﾝｼﾞﾆｱの

業務ノウハウが必須

詳細設計書　と　Ruby ソースコード

〇高額合算システム　詳細設計書（支給審査処理__通知結果入力__kg030500）抜粋

【削除】:destroy
① 自己負担申請情報のロックバージョンをチェックする。
　7. その他共通処理【自己負担申請情報のロックバージョンチェック】を行う。
② 対象となる自己負担額申請情報を更新する。
　対象条件
　　・受付番号= {受付番号}
　更新項目
　　・他通知日= Null
　　・他通知審査区分= 0
　　・他通知審査不支給理由= Null
　　・他通知支給額= 0
　　・状態= 2:発行済み
　　・消込日時= Null
③ 対象となる自己負担証明書情報を削除
　対象条件
　　・受付番号= {自己負担申請情報. 受付番号}

○高額合算システム　ソースコード(支給審査処理__通知結果入力__kg030500)抜粋

```
 # 削除処理
 def destroy
  # 削除対象の検索
  del_shinsei_j = JikofutanShinseiJ.find(:first,
       :conditions => [' uketsuke_no = ? and updated_at = ?' ,
          my_session[:uketsuke_no],my_session[:shinsei_upd_at]])
    if del_shinsei_j = nil
       raise ActiveRecord::StaleObjectError,new()
    end
    del_gassan_j = KogakuGassanJ.find(:first,
       :conditions => [ 'shinsei_kbn = ? and uketsuke_no = ? and
          rireki_kbn = ? and updated_at = ?' , 2, my_session[:uketsuke_no],
          1, my_session[:gassan_upd_at]])
    if del_gassan_j == nil
       raise ActiveRecord::StaleObjectError.new()
    end
    JikofutanShinseiJ.transaction do
       # 自己負担証明書申請情報　更新
       if !del_shinsei_j.update_attributes(:tatsuchi_date => nil,
                                           :tatsuchi_shinsa_kbn =>  '0' ,
                                           :tatsuchi_shinsa_fushikyu_riyu => nil,
                                           :tatsuchi_shikyu_gaku => 0,
                                           :jotai => 2,
                                           :keshikomi_date_time => nil,
         msg = MessageM.create_message( 'EE0005' )
          raise KgStandardError.new(msg)
        end
        # 高額合算支給情報　削除
        del_gassan_j.destroy
     End
```

日本語の項目名称が使用できない


松江市高額合算システム - Mozilla Firefox
被保険者状況照会
窓口業務担当ユーザ　2008/01/22 16:24:18
制度：
年度：2008
被保険者番号：
個人番号：1000003490
被保険者番号検索
年度：2008
氏名
被保険者番号
8
9
10
11
12
1
2
3
4
5
6
7
計
主
高額太一
2003170000
７５:
34,000
49,000
0
36,000
0
119,000
3000005300
７５:
0
30,000
0
52,000
0
39,000
29,000
52,000
202,000
4000009700
5,000
3,000
3,000
4,000
7,000
5,000
3,000
4,000
8,000
3,000
4,000
5,000
54,000
高額幸子
2003170000
30,000
80,000
72,000
37,000
45,000
23,000
40,000
52,000
379,000
4000009800
30,000
5,000
2,000
3,000
2,000
4,000
5,000
3,000
4,000
2,000
5,000
3,000
68,000
高額太一郎
2003170000
0
前年度
国保
後期
介護
次年度
検索
確認
戻る
実行
印刷
終了

Railsでも、CSSと同様なキメ細かな画面

窓口に市民が来た時の対応（上段：申請者情報，、中段：受給履歴、下段：申請内容

Ruby の 可能性

1. フレームワーク

Ruby　：　Ruby on Rails （1つだけ）
Java　：　たくさんある（Slerの数？） 教育できない。

2. オープンソース

Sler に動作保証をだれがするか？
Ruby , Rails , MySQL , CentOS , Apache , FireFox
iReports , JasperReports
ディストリビュータが必要
品質重視のバージョン提供 & 長期サポート （開発が好きなコミュニティ）

3. 今後の展望

RubyのISO化（IPA=経済産業省がバックアップ）
オープンの時代、世界に認められた国産技術（ｵｰﾌﾟﾝｿｰｽ？）
⇒　数年内に政府調達（普及が早まる：Linuxの事例あり）

人材育成　文系の学生に言語教育を行なうとき
最適な言語Ruby = 判り易い
オブジェクト指向は一般の人に判り易い言語

共通基盤システムの開発

- Rubyで業務システムを開発するにあたり
- 他の言語（COBOL等）で業務システム開発に用いられる（用意されている）共通基盤システムを参考にして
- 以下のシステムを開発しました

１．バッチ処理環境

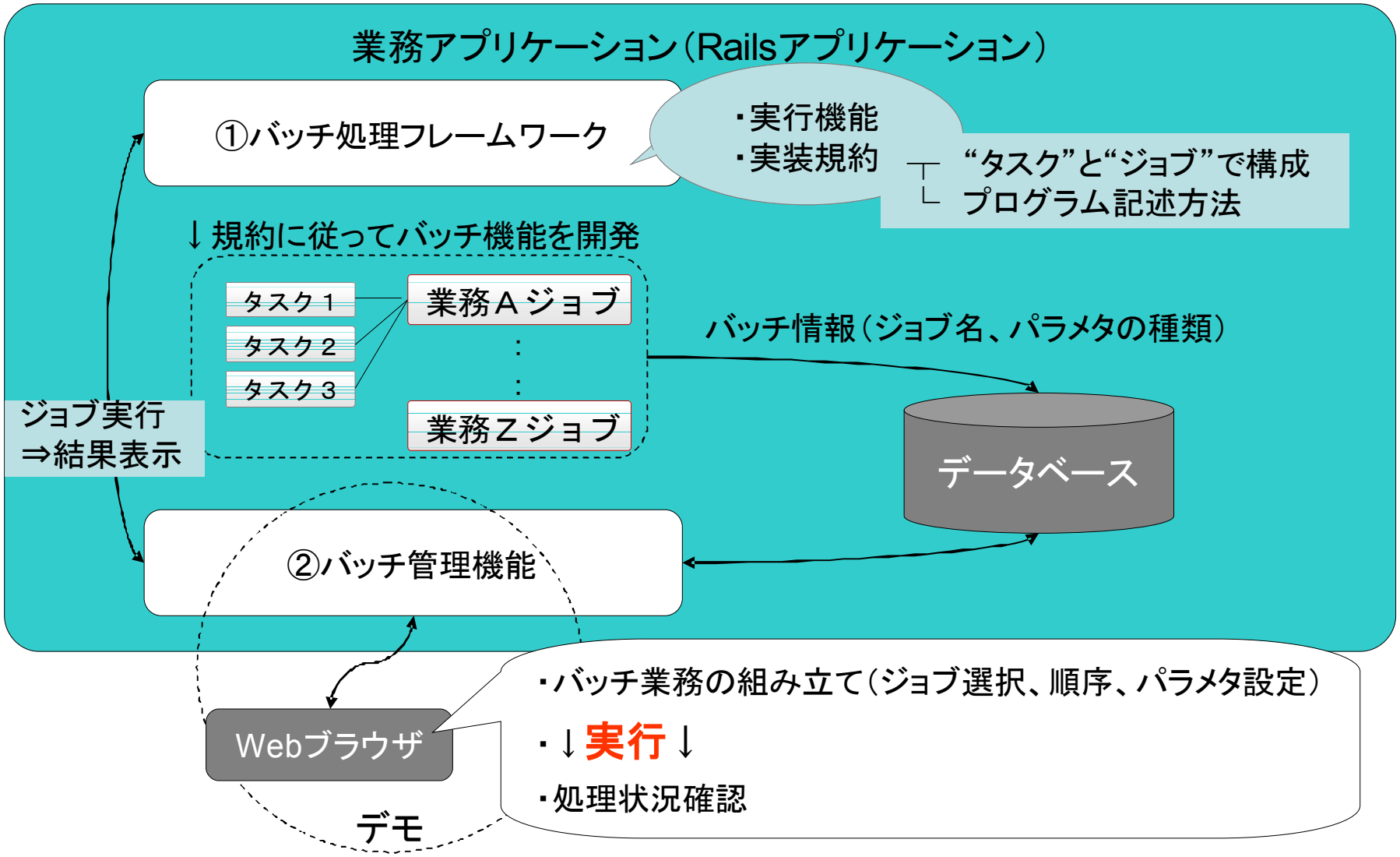
２．帳票出力環境

バッチ処理環境

- Ruby on Railsで開発するアプリケーションにバッチ処理機能を追加するためのもの

- 機能の構成
  1. バッチ処理フレームワーク
  2. バッチ管理機能
     （Webブラウザで動作）

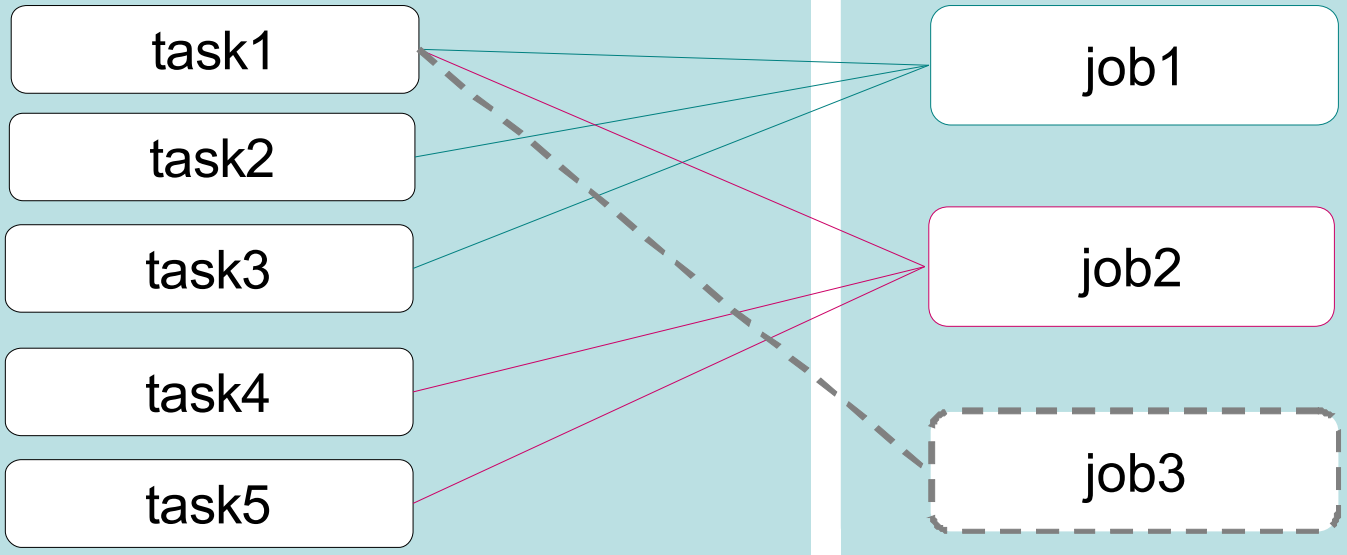
バッチ処理環境の構成イメージ



タスクとジョブ

【タスク】
- クラスとして開発
- 分割可能な業務ロジックを実装
- runメソッド

【ジョブ】
- タスクの組み合わせを記述
- タスク実行時パラメタを設定
- タスクエラー時の制御を指定



タスクとジョブ

【タスク】
- クラスとして開発
- 分割可能な業務ロジックを実装

【ジョブ】
- タスクの組み合わせを記述
- タスク実行時パラメタを設定
- 御を指定

```
# task1_task.rb
class Task1Task < ApplicationTask
  # ジョブから呼び出し時に実行されるメソッド
  def run
    # ここに業務ロジックを記述
    #
    return SUCCESS        # タスク処理成功(= 0)
  rescue Exception e
    add_app_log(:pg_name => 'task1', :error => e)
    return ERROR          # タスク処理失敗(= 0以外)
  end
end
```

タスクとジョブ

【タスク】
- クラスとして開発
- 分割可能な業務ロジックを実装
- runメソッド

【ジョブ】
- タスクの組み合わせを記述
- タスク実行時パラメタを設定
- タスクエラー時の制御を指定

task1
task2
task3
task4
task5

```
#  job_name.rb
#  job  :ジョブ名，  :オプション
job (:job_name, :exit_on_error => true) do
   exec (:task1, :params => {:date => params[:date]})
   exec (:task2, :params => {:keyword => params[:keyword]})
   exec (:task3, :params => {:keyword => params[:keyword]})
end
```

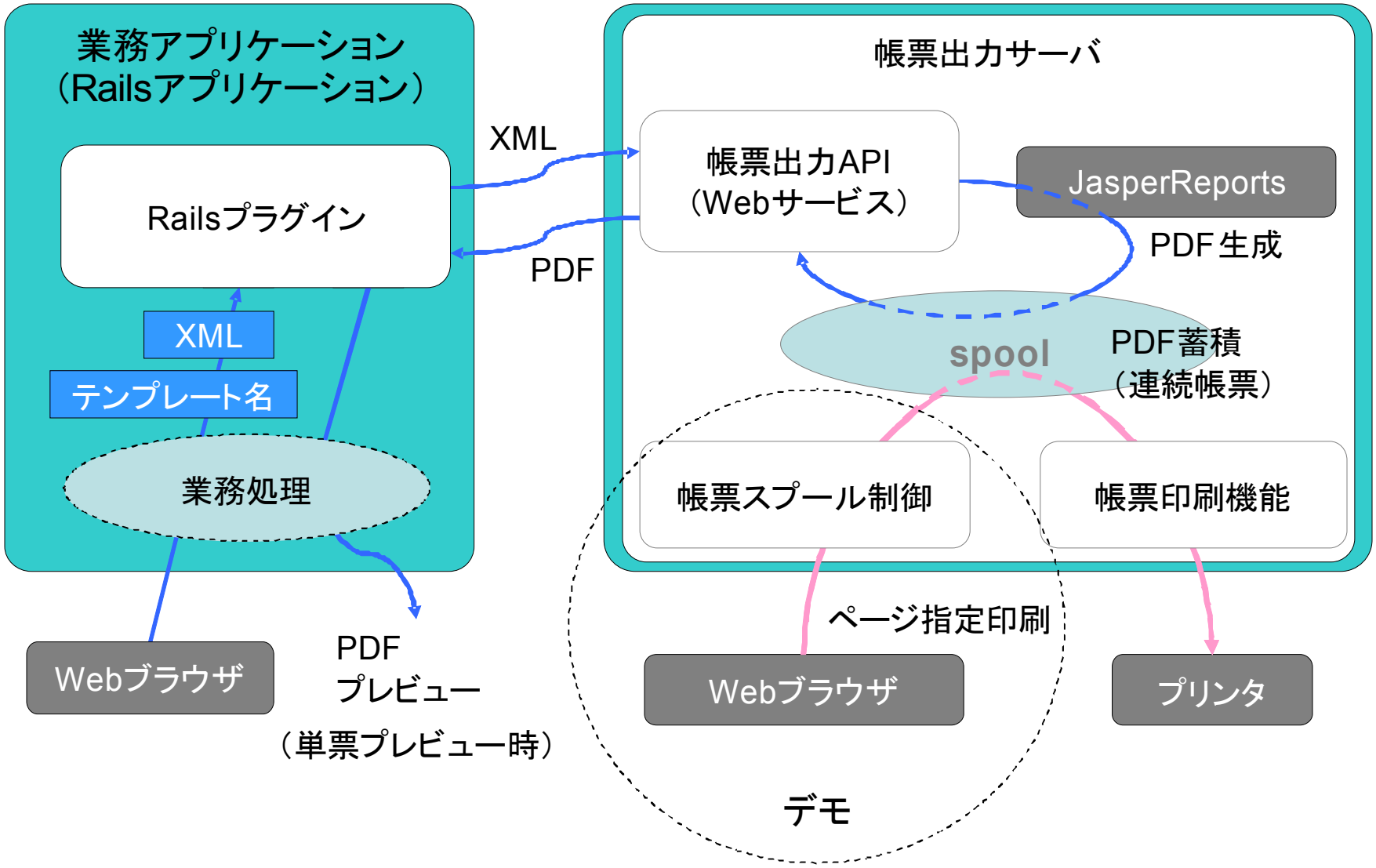
job3

帳票出力環境

- 複数の業務アプリケーションから共通で利用できる帳票出力システムです

- 機能の構成
  1. 帳票出力サーバ
  2. 帳票出力用Railsプラグイン

帳票出力環境の構成イメージ



帳票出力用Railsプラグイン

- 業務アプリケーションから帳票出力サービスへ印刷要求を発行するための機能
- 業務アプリケーションからは以下のように印刷要求を発行します
- 例（複数ページ帳票をスプール出力する場合）

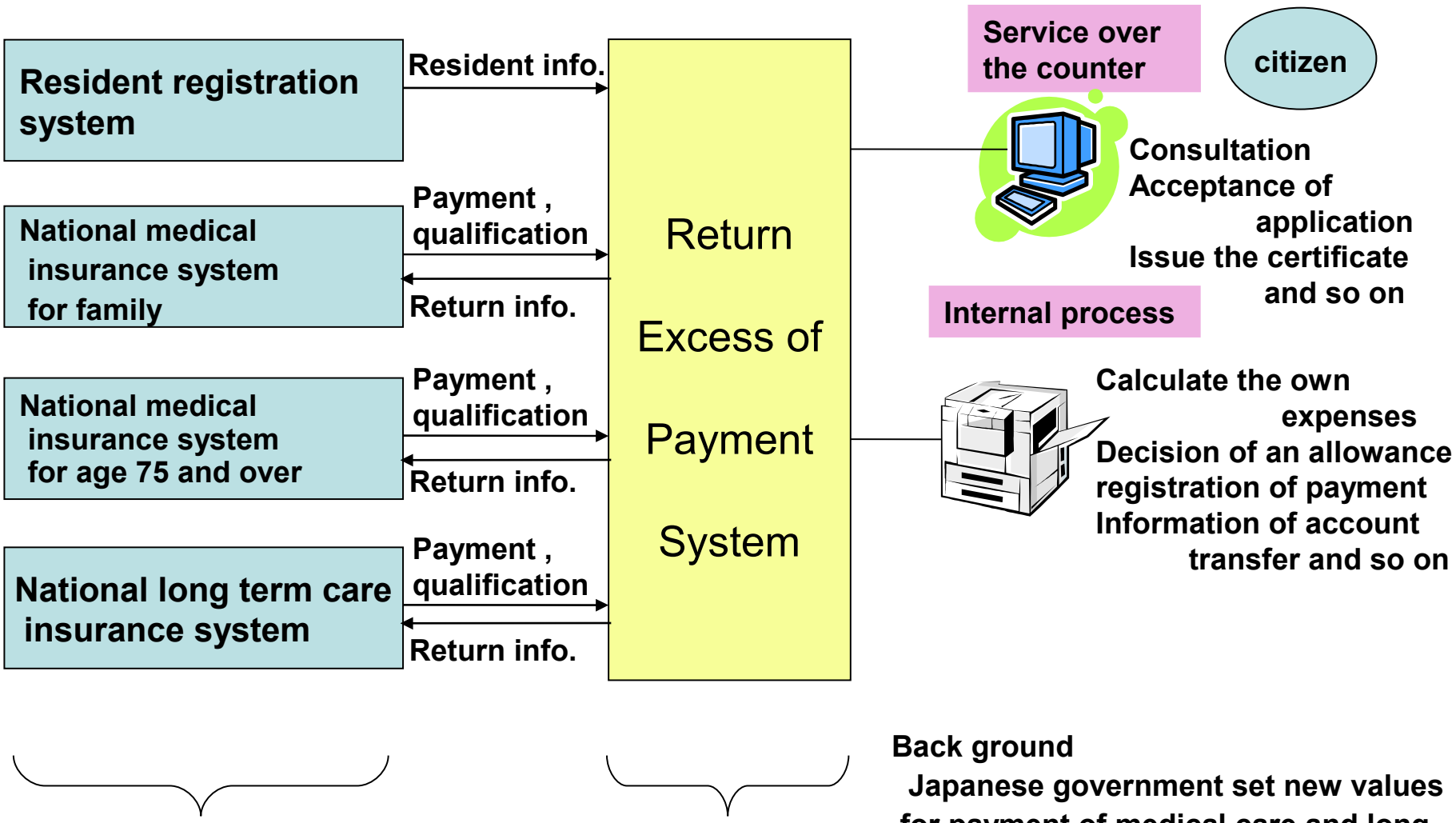
```
# 帳票データ（XML）の生成
xml_datas = create_report_xml(monthry_records)
# 帳票出力データ（XML）を1ページづつ出力
report = Rps::Report.create(:title => ‘一覧表’, )
xml_datas.each_with_index | xml_data, index | do
    report.add_entry(:title => “月次集計一覧表（#{index}項）”,
                     :data_path => ‘report/pages’,
                     :template_name => ‘monthry_report.jasper’,
                     :data_source => xml_ddata )
end
```

2008. 6. 23

JRuby 開発者
チャールズ・ナッター氏
松江訪問

Ruby基幹業務開発を
プレゼン

IPA's event (1) Return Excess of Payment System



Already existing system
(Almost legacy system on main frame)

Target system

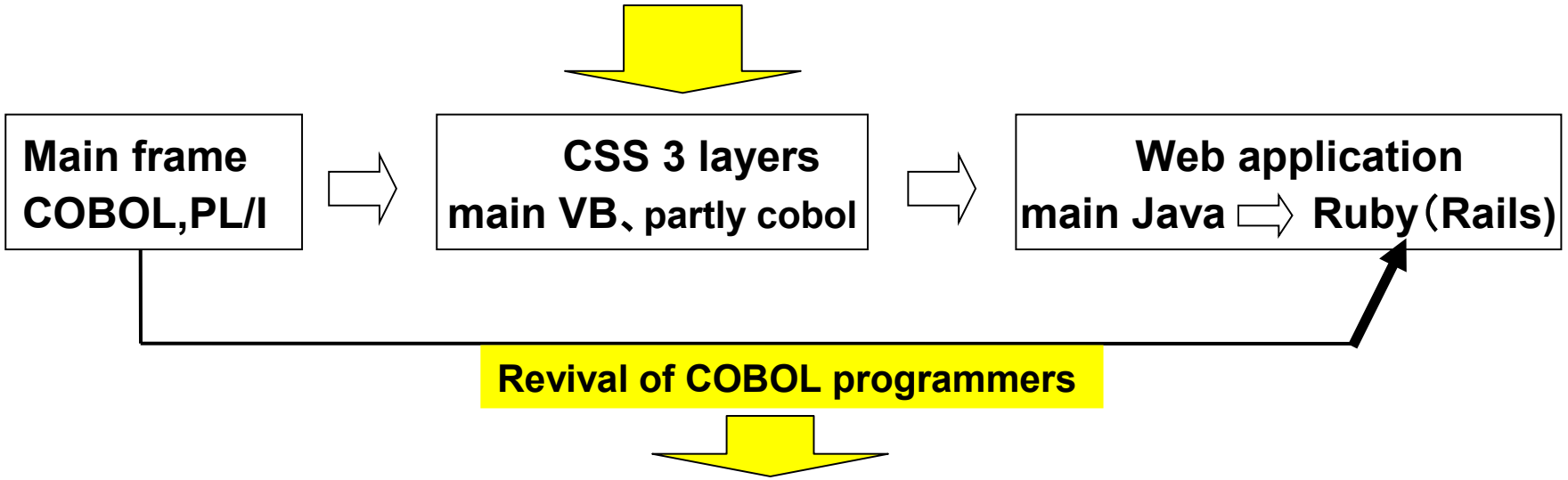
Back ground
Japanese government set new values for payment of medical care and long term care expenses. When total cost of the individual payment is over the standard, return excess of payment.

Ruby development in Enterprise System (COBOLer's revival)

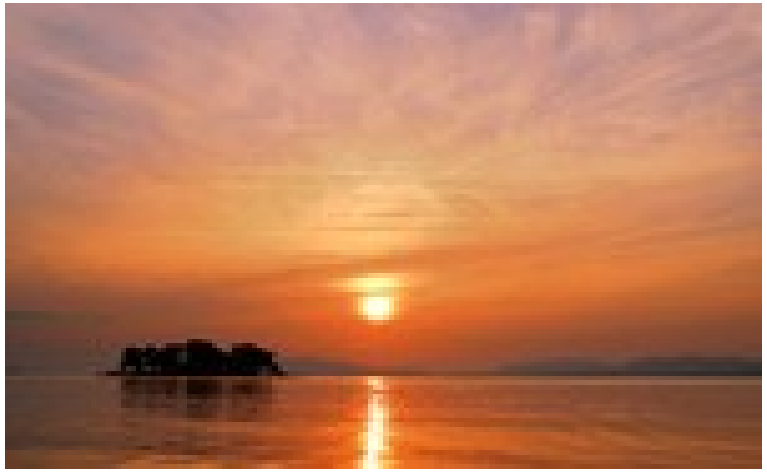
COBOL programmers

they have “Many Business Skills”.
rich experience of software development, human network of end users.
But, they are older, their technology is legacy.

Enterprise system development needs Many Business Skills



Ruby’s development is nearly equal to main flame or CSS 3 layers.
There is no silver bullet for Werewolf same as software development.
by Frederick Brooks Jr. IBM’s OS360



Sunset at Lake SINJIKO in Matsue City
where Matz lives.

公開ページ： www.tpj.co.jp

IPA公募事業成果「高額合算ｼｽﾃﾑ」

ITプロのWebﾏｶﾞｼﾞﾝ：エンタープライズジン
今日の日記記事 ランキング 先頭に表示
“エンタープライズ開発の現場で感じたRubyのメリット”
http://enterprisezine.jp/news/

The END