

Rmenu基礎知識

「Rmenu とは」に加え、基礎知識を学びます

Rmenu基礎知識

もくじ

- ▶ ビジュアルツールとは
- ▶ RACKの詳細
- ▶ 名前空間とは
- ▶ JSONlintの使い方
- ▶ オープンLDAPとは
- ▶ ER図とは
- ▶ DBのコピー作成
- ▶ プロジェクトのコピー作成
- ▶ プログラムのコピー作成
- ▶ URLの調べ方（ローカルテスト用）

Rmenu基礎知識

ビジュアルツールとは

Rmenu のデータの流を確認するために使用します

ビジュアルツールとは もくじ

- ▶ 「ビジュアルツールとは」
- ▶ 「ログビューア」
- ▶ 「サーバービューア」
- ▶ 「クライアントビューア」

ビジュアルツールとは

▶ プログラムの流れを確認するツール

プログラムの流れを調べるときに、複数のウィンドウを開きデータやプログラムを追いかけますが
ビジュアルツールは一画面で同時に関係するプログラムを開くことができるので

プログラムの修正箇所確認や学習に便利です

ビジュアルツールを使用時はブラウザを2つ用意します1つ目は『Rmenu』を開き、2つ目は『ビジュアルツール』を開きます

Rmenu の『クライアント側』『サーバー側』『ログの確認』の3種類に分けてデータの流れが確認できます

※それぞれどの部分が見れるのか図で確認

ブラウザ①Rmenuを開く

請求内容	請求形態	商品名	単価	N/Pの表示
Webサービス利用料	月課	Webサービス利用料 シンプルプラン	30,000	表示する
Webサービス利用料	年課	Webサービス利用料 プレミアムプラン	300,000	表示しない
Webサービス利用料	年課	Webサービス利用料 プレミアムプラン (特約割引料)	200,000	表示する
電子手帳送料	支払時	電子手帳送料	30,000	表示しない
パンフレット	月課	パンフレット	90,000	表示しない

ブラウザ②ビジュアルツールを開く

Rmenu Visual Tools メニュー 戻る: Esc
平成29年10月17日 (火) ログイン日時: 2017/10/17 11:30
ユーザ氏名: システム部担当

開発支援ツール テスト支援ツール システムドキュメント マスター管理

クライアント ビューア
サーバ ビューア
ログ ビューア

Rmenuファイル構成

Rmenuのファイルの構成です。右に列挙されているファイルはランタイムで読み込まれます。

各Rmenuが実行するプログラム名は各Rmenuごとに異なります。

共通のパターン名は各Rmenuごとに異なります。

クライアント側の作成はここで作成します

サーバー側の作成はここで作成します

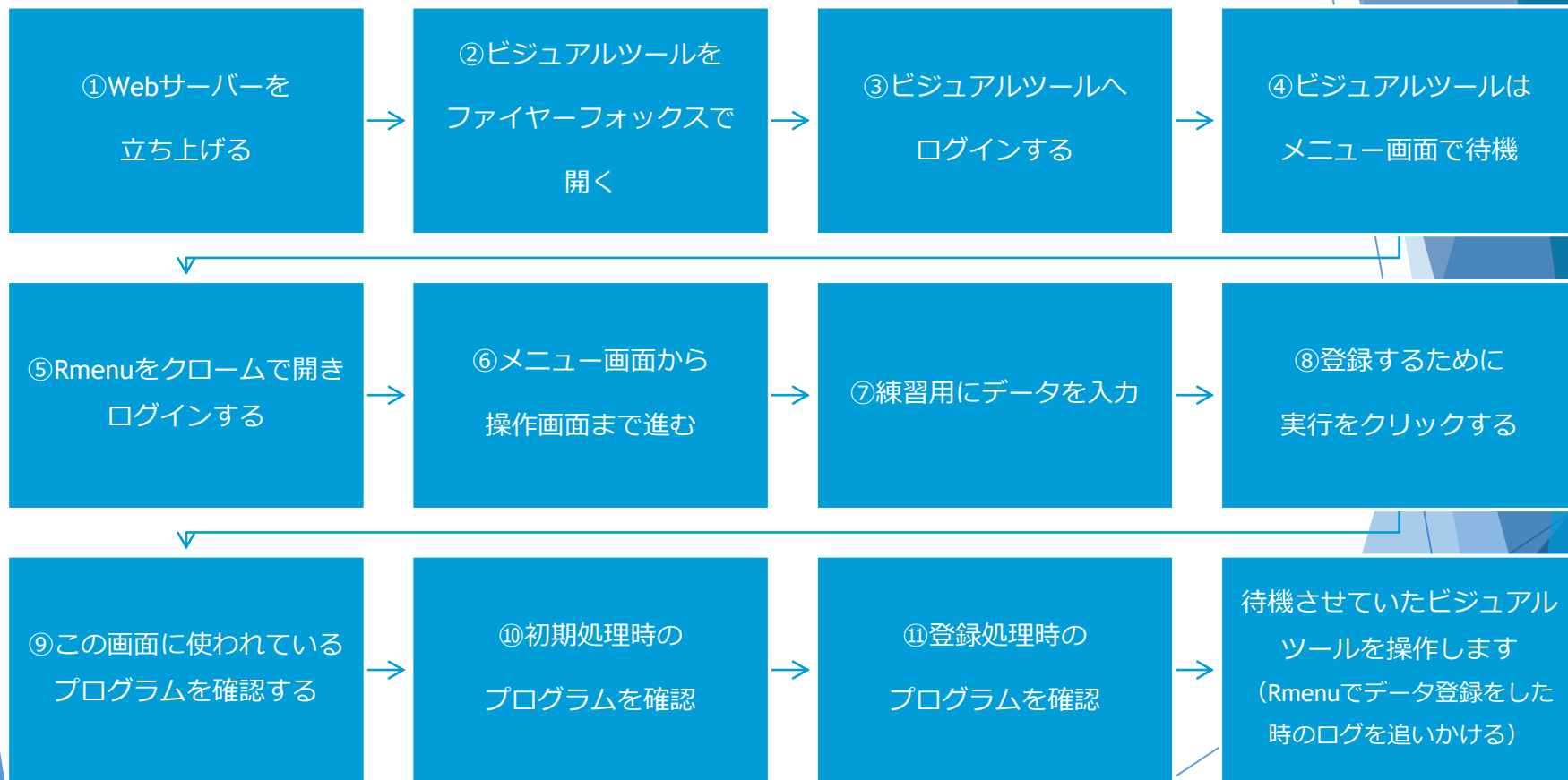
サーバー側のRmenuの作成はここで作成します

プログラムのコピーをする場合は、各クライアント側のRmenuの構成を参考にしてください。

プログラムの構成を確認できるため、制作時によく使われます。

ビジュアルツールとは

▶ ビジュアルツールを開く手順



ビジュアルツールとは

▶ ビジュアルツールを開く

- ① Webサーバーを立ち上げる
- ② URLを開く

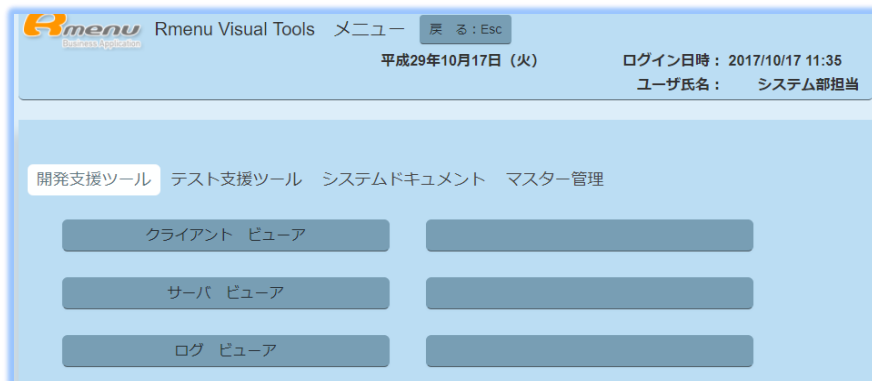
<http://127.0.0.1:9292/Application/RmenuVisualTools/Html/Apps/Login/index.html>

- ③ ログインID/PW [ログイン : guest PW : guest]



ビジュアルツールを開くブラウザは**ファイヤーフォックスを推奨**クロームで調べるプロジェクトを開きファイヤーフォックスでビジュアルツールを開く

- ④ メニュー画面から各ページへ入ります (ビジュアルツール用意完了)



※クロームで開くとエラーになる時があります
クロームの設定をポップアップ [許可] にすると改善



ビジュアルツールとは

▶ ビジュアルツールを開く

⑤ ビジュアルツールが用意できたら、データの流れ方を追いかける練習用に『Rmenu sales』をブラウザはクロームで開きます

※WEBサーバーはビジュアルツールで立ち上げ済

URL : <http://127.0.0.1:9292/Application/RmenuSales/Html/Apps/MyLogin2/index.html>

ID : guest

PW : guest

Rmenu Rmenu販売管理 ログイン クリア: Esc ログイン: F1

ユーザ quest

パスワード 必須入力

⑥ ログインし、メニュー画面から [マスター管理] → [商品入力] → [商品マスター一覧] を開く

Rmenu Rmenu販売管理管理 メニュー ログイン日時: 2017/10/19 11:14:52 ユーザ氏名: Rmenu管理者

売上/入金 請求売掛管理 仕入/支払 支払買掛管理 マスター管理

売上入力 売上チェックリスト

契約売上 訂正 入金チェックリスト

入金入力

仕入入力

日次処理 月次処理

Rmenu Rmenu販売管理管理 メニュー ログイン日時: 2017/10/19 11:14:52 ユーザ氏名: Rmenu管理者

売上/入金 請求売掛管理 仕入/支払 支払買掛管理 マスター管理

商品入力 支払方法

契約者入力 請求締切り メンテ

売上区分 メンテ 消費税 メンテ

入金区分 メンテ アカウント管理 メンテ

請求形態 テーマ メンテ

請求内容

Rmenu Rmenu販売管理管理 商品マスター一覧表 ログイン日時: 2017/10/19 11:14:52 ユーザ氏名: Rmenu管理者

登録: F1 転用: F2 訂正: F3 削除: F4 検索: F5

請求内容	請求形態	商品	単価	NPO表示
Webサービス利用料	月額	Webサービス利用料 シンプルプラン	30,000	表示する
Webサービス利用料	年額	Webサービス利用料 プレミアムプラン	300,000	表示しない
Webサービス利用料	年額	Webサービス利用料 プレミアムプラン (特別割引料金)	200,000	表示する
セミナー受講料	スポット	セミナー受講料	30,000	表示しない
バナー広告料	月額	バナー広告料	50,000	表示しない

総計: F9 戻へ: F10 10ページ(5件) 次へ: F11 検索: F12

ビジュアルツールとは

▶ ビジュアルツールを開く

⑦『商品マスター一覧画面』は
商品のデータが表示されています
検索を行い欲しいデータのみ検索して表示ができます
新規登録は登録ボタンへ進みます

練習用に、新規登録時のデータの流をビジュアルツール
で追いかけます [登録] ボタンへ進み新規登録を行います

商品マスター 一覧表 戻る: Esc
平成29年10月19日 ログイン日時: 2017/10/19 11:14:52
ユーザ氏名: Rmenu管理者

登録: F1 転用: F2 訂正: F3 削除: F4 照会: F5

検索する所

請求内容	請求形態	商品名	単価	NPO表示
Webサービス利用料	月額	Webサービス利用料 シンプルプラン	30,000	表示する
Webサービス利用料	年額	Webサービス利用料 プレミアムプラン	300,000	表示しない
Webサービス利用料	年額	Webサービス利用料 プレミアムプラン (特別割引料金)	200,000	表示する
セミナー受講料	スポット	セミナー受講料	30,000	表示しない
バナー広告料	月額	バナー広告料	50,000	表示しない

登録済データが表示されている (検索されると該当商品だけが表示される)

最初: F9 前へ: F10 次へ: F11 最後: F12

⑧ビジュアルツールでデータを追う為に操作を実行します
『商品マスター入力 (新規) 登録画面』で実行します

赤枠内に仮のデータを入れて [実行] し、準備完了

次に、この画面を動かす為に、
何が使用されているのかを詳細確認をします

商品マスター 入力 (新規) 戻る: Esc 実行: F1
平成29年10月19日 ログイン日時: 2017/10/19 15:35:50
ユーザ氏名: Rmenu管理者

商品ID

請求内容: その他

請求形態: スポット

商品名: aaaaaaaaaa

単位: 12345

単価: 12345

NPO表示: 表示しない

備考

簡単に流れを見るために
単価「12345」がDBへ入り
表示するまでの流れを追います

ビジュアルツールとは

▶ ビジュアルツールを開く

⑨この画面の情報

プロジェクト名 : Rmenusales
プログラム名 : ProductItemInput
使用プログラム :



- index.html
- ProductItemInput.appspec.js
- ProductItemInput.controller.js
- ProductItemInput.main.js
- ProductItemInput.model.js
- ProductItemInput.view.js

- ProductItemInput_dataset.json
- ProductItemInput_delete_sql.json
- ProductItemInput_delete_tran.json
- ProductItemInput_init_sql.json
- ProductItemInput_init_tran.json
- ProductItemInput_insert_sql.json
- ProductItemInput_insert_tran.json
- ProductItemInput_select_sql.json
- ProductItemInput_select_tran.json
- ProductItemInput_update_sql.json
- ProductItemInput_update_tran.json
- ProductItemInput_validation.json

- ProductItemInput_controller.rb
- ProductItemInput_model.rb
- ProductItemInput_view.rb

⑩初期処理で動くプログラム

開いたばかりの最初の画面（未入力状態）



使用JSON :  ProductItemInput_init_sql.json
 ProductItemInput_init_tran.json

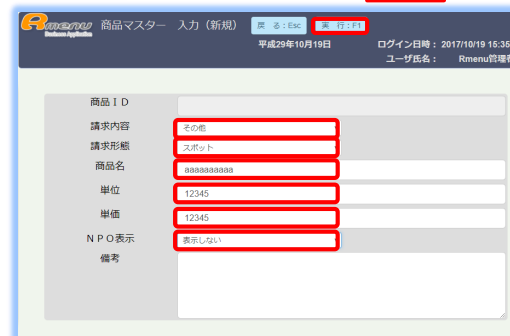


⑨と⑩は同じ画面ですが
操作によって
使用するtran.jsonと
sql.jsonが変わります

⑩登録処理で動くプログラム

新規登録する（入力済実行）

使用JSON :  ProductItemInput_insert_sql.json
 ProductItemInput_insert_tran.json



ビジュアルツールの
練習用
データの用意が完了

ログビューア

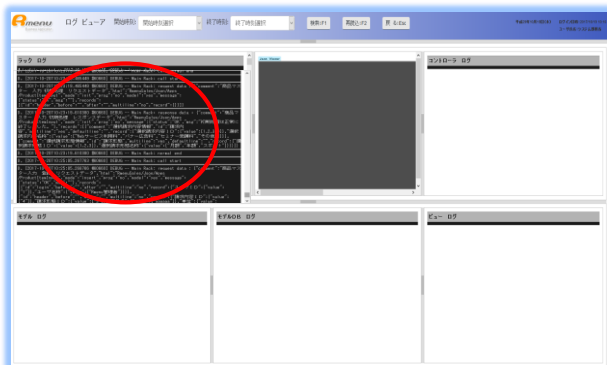
▶ ログビューア操作方法

① Rack.logは操作の履歴が見れます

ラックログでデータの流を確認したい操作の履歴を探し、**時間を確認**します

スクロールで1番下へ進むと最新の操作があります

※秒刻みで表示を選択できる為秒まで確認する ※リクエストの下にレスポンスがありこれで1操作分です



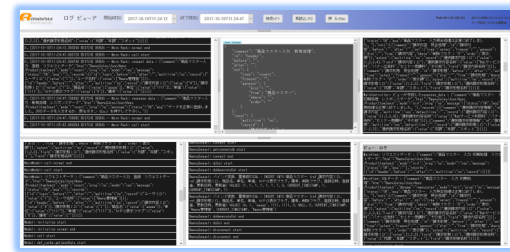
```
D [2017-10-20T10:23:09.465449 #80660] DEBUG -- Main Rack: request data : {"comment":"商品マ  
スター 入力 初期処理 リクエストデータ","html":"RmenuSales/Json/Apps  
/ProductItemInput","mode":"init","prog":"no","model":"yes","message":  
{"status":"OK","msg":"","records":  
[{"id":"header","before":"","after":"","multiline":"no","record":{}}]}
```

```
D [2017-10-20T10:23:09.610380 #80660] DEBUG -- Main Rack: response data : {"comment":"商品マ  
スター 入力 初期処理 レスポンスデータ","html":"RmenuSales/Json/Apps  
/ProductItemInput","mode":"init","prog":"no","message":{"status":"OK","msg":"初期処理は正常に  
終了しました。"},"records":[{"comment":"選択請求内容情報","id":"請求内  
容","multiline":"yes","defaultline":"","record":{"value":[1,2,3,4]},"選択  
請求内容名称":{"value":["Webサービス利用料","バナー広告料","セミナー受講料","その他"]}},  
{"comment":"選択請求形態情報","id":"請求形態","multiline":"yes","defaultline":"","record":{"選  
択請求形態ID":{"value":[1,2,3]},"選択請求形態名称":{"value":["月額","年額","スポット"]}}]}
```

② 時間選択・検索

ラックログで確認した **開始時刻** **終了時刻** を選択し、
検索 をクリックする 全画面に選択した時間帯のログが表示される

ログビューア 開始時刻: 開始時刻選択 終了時刻: 終了時刻選択 検索:F1 再読み込み:F2 戻る:Esc



ログビューア

▶ ログビューア操作方法

備考：①ラックログの詳細（ラックログを見て簡単にわかる事）

[時間・リクエスト・レスポンス・mode・データの表示・データの入力・画面に表示するメッセージ]が見れます

↓ Mode:initは画面の初期処理です

```
D, [2017-10-20T10:23:19.465449 #60660] DEBUG -- Main Rack: request data : {"comment": "商品マスター 入力 初期処理 リクエストデータ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "init", "prog": "no", "model": "yes", "message": {"status": "OK", "msg": "", "records": [{"id": "header", "before": "", "after": "", "multiline": "no", "record": {}}]}}
```

操作した時間

リクエスト「このデータをください」

レスポンス「このデータを表示してください」

mode「initのトランJSONとSQLJSONを使う」

データの表示「登録画面を開いた最初に表示する物を用意している（セレクトBOXの項目名等）」

```
D, [2017-10-20T10:23:19.610380 #60660] DEBUG -- Main Rack: response data : {"comment": "商品マスター 入力 初期処理 レスポンスデータ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "init", "prog": "no", "message": {"status": "OK", "msg": "初期処理は正常に終了しました。", "records": [{"comment": "選択請求内容情報", "id": "請求内容", "multiline": "yes", "defaultline": "", "record": {"選択請求内容ID": {"value": "[1,2,3,4]"}, "選択請求内容名称": {"value": "[\"Webサービス利用料\", \"バナー広告料\", \"セミナー受講料\", \"その他\"]"}}, {"comment": "選択請求形態情報", "id": "請求形態", "multiline": "yes", "defaultline": "", "record": {"選択請求形態ID": {"value": "[1,2,3]"}, "選択請求形態名称": {"value": "[\"月額\", \"年額\", \"スポット\"]"}}]}}
```

操作した時間

リクエスト「このデータをDBへ入れて」

レスポンス「このmessageを表示してください」

mode「insertのトランJSONとSQLJSONを使う」

データの入力「登録画面で入力した物が表示されています（例：単価12345）」

↓ Mode:insertはデータの登録処理です

```
D, [2017-10-20T10:25:05.268766 #60660] DEBUG -- Main Rack: request data : {"comment": "商品マスター入力 登録 リクエストデータ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "insert", "prog": "no", "model": "yes", "message": {"status": "OK", "msg": "", "records": [{"id": "login", "before": "", "after": "", "multiline": "no", "record": {"ユーザID": {"value": "[1]"}, "ユーザ名称": {"value": "[\"Rmenu管理者\"]"}}, {"id": "header", "before": "", "after": "", "multiline": "no", "record": {"請求内容ID": {"value": "[4]"}, "請求形態ID": {"value": "[3]"}, "商品名": {"value": "[\"aaaaa\"]"}, "単位": {"value": "[\"12345\"]"}, "単価": {"value": "[\"12345\"]"}, "NFC表示フラグ": {"value": "[\"1\"]"}, "備考": {"value": [\""]}}]}}
```

画面に表示するメッセージ「データ入力され実行され、サーバー側の処理が終わり、最後画面に表示するメッセージがレスポンスで戻った」

```
D, [2017-10-20T10:25:05.400521 #60660] DEBUG -- Main Rack: response data : {"comment": "商品マスター入力 新規処理 レスポンスデータ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "insert", "prog": "no", "message": {"status": "OK", "msg": "データを正常に登録しました。次のデータを入力するか、戻るボタン (Esc) を押下して下さい。"}}
```

ログビューア

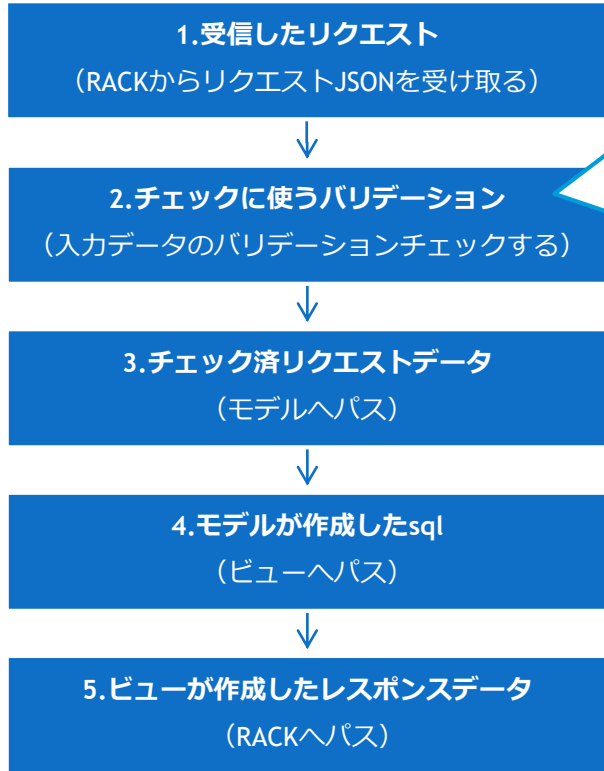
▶ ログビューア操作方法

③コントローラログ

コントローラは入ってきたデータをDBに入れてもいいかチェックを行います
 チェック済データをモデルへ送信し、返信のSQLを貰い、
 これをビューへ送り、返信のレスポンスを貰い、RACKへ送信し1つの処理が完了します
 1処理が5つの行に分かれて確認できます

```

コントローラ ログ
MainController: call start
MainController: 受信したリクエスト(OS)データ: {"comment": "商品マスター 入
初期処理 リクエストデータ {"html": "/RenuSales/Json/Apps
/ProductItenInput", "mode": "init", "prog": "no", "model": "yes", "message":
{"status": "OK", "msg": ""}, "records":
[{"id": "header", "before": "", "after": "", "multiline": "no", "record": [{}]}]
Controller: initialize start
Controller: initialize normal end
Controller: call start
Controller: def_cache.getJsonData start
Controller: def_cache.getJsonData normal end
Controller: app_cache.getControllerObject start
Controller: app_cache.getControllerObject normal end
Controller: checkRequestData start
Controller: チェックするリクエスト情報:
{"id": "header", "before": "", "after": "", "multiline": "no", "record": []}
Controller: チェックに使うバリデーション情報:
{"id": "header", "multiline": "no", "record": [{"テーブルID": "validation",
["nonrequired", "integerP"], "min": "1", "max": "11"}, {"テーブルID": "validati
MainController: モデルが作成したsql_data: [{"comment": "商品マスター入
理", "html": "/RenuSales/Json/Apps
/ProductItenInput", "dbname": "renusales", "mode": "insert", "prog": "no", "
["status": "OK", "msg": "登録処理は正常に終了しました。"}, {"sels": [{"come
マスター入力 情報処理 {"id": "header", "before": "", "after": "", "sels":
{"type": "insert", "frasesq": "", "genesq": [{"dist": "", "from": "商品マス
ター", "where": "", "order": ""}], "input": [{"multiline": "no", "record": [{"購
ID": [{"value": "[4]", "field": "ref_購求内容ID"}, {"購求形態ID": [{"valu
[3]}, {"field": "ref_購求形態ID"}, {"商品名": [{"value": "[aaaaa ]", "単位
["value": "[100g]"}, {"単位": [{"value": "[100g]"}, {"AIQ表示フラグ": [{"va
[""]}], {"備考": [{"value": "[ ]"}, {"削除フラグ": [{"value": "[1]"}, {"funct
["value": "[ ]"}, {"func": "CURRENT_TIMESTAMP"}, {"登録者": [{"value": "[Renu
者"], "frontype": "request", "fromid": "login", "fromio": "froname": "ユー
称"], "更新日時": [{"value": "[ ]"}, {"func": "CURRENT_TIMESTAMP"}, {"更新者": [{"
"/Renu管理時"], "frontype": "request", "fromid": "login", "fromio": "fron
ユーザー名体"}]]]}]
MainController: ビューが作成したresponse_data: [{"comment": "商品マス
初期処理 レスポンスデータ {"html": "/RenuSales/Json/Apps
/ProductItenInput", "mode": "insert", "prog": "no", "message": {"status": "OK
データを正常に登録しました。 次のデータを入力するか、 戻るボタン (Esc) を
下さい。"}]}]
MainController: call normal end
MainController: call start
    
```



コントローラログの動き：
 入ってきたデータを
**バリデーションJSONで
 チェックを行う**
 コントローラは**処理を役割
 分担**して各所へデータを
 パスしながらレスポンスを
 作成します

1~5の動きがinit処理
 とInsert処理の
 二回実行されます

ログビューア

▶ ログビューア操作方法

備考：③コントローラログの詳細 (insert登録処理を追いかけます)

『1.受信したリクエスト』画面で入力されたものが記載されています
[商品名 : aaaaa] [単位 : 12345] を追いかけます

```
MainController: 受信したリクエスト (CGI) データ: {"comment": "商品マスター登録 リクエストデータ", "url": "/RmenuSales/Json/Apps/ProductItemInput", "mode": "insert", "prog": "no", "model": "yes", "message": {"status": "OK", "msg": ""}, "records": [{"id": "login", "before": "", "after": "", "multiline": "no", "record": {"user": {"value": "1"}, "user_name": {"value": ["Rmenu管理者"]}}], [{"id": "header", "before": "", "after": "", "multiline": "no", "record": {"request_id": {"value": "4"}, "request_form_id": {"value": ["3"]}, "商品名": {"value": ["aaaaa"]}, "単位": {"value": ["12345"]}, "単価": {"value": ["12345"]}, "NPO表示フラグ": {"value": ["1"], "備考": {"value": [""]}}}]}
```

Controller: initialize start

```
{  
  "商品名": {  
    "value": [ "aaaaa" ]  
  },  
  "単位": {  
    "value": [ "12345" ]  
  }  
}
```

←リクエストJSON
「これをインサートしてください」

『2.チェックに使うバリデーション』

```
Controller: チェックするリクエスト情報: {"id": "header", "before": "", "after": "", "multiline": "no", "record": {"request_id": {"value": "4"}, "request_form_id": {"value": ["3"]}, "商品名": {"value": ["aaaaa"]}, "単位": {"value": ["12345"]}, "単価": {"value": ["12345"]}, "NPO表示フラグ": {"value": ["1"], "備考": {"value": [""]}}}
```

```
Controller: チェックに使うバリデーション情報: {"id": "header", "multiline": "no", "record": {"table_id": "validation", "validation": {"nonrequired": {"integerP": {"min": "1", "max": "11"}, "request_content_id": {"validation": {"required": {"integerP": {"min": "1", "max": "11"}, "商品名": {"validation": {"required": "free", "min": "1", "max": "100"}, "単位": {"validation": {"required": "free", "min": "1", "max": "20"}, "単価": {"validation": {"required": "integerP", "min": "1", "max": "7"}, "NPO表示フラグ": {"validation": {"required": "free", "min": "1", "max": "1"}, "備考": {"validation": {"nonrequired": {"free": {"min": "1", "max": "100"}}}}}}}}}}}}
```

```
Controller: チェックするメソッド情報: rmenu_required("aaaaa", "商品名")  
Controller: チェックするメソッド情報: rmenu_free("aaaaa", "商品名")  
Controller: チェックするメソッド情報: min(1, "aaaaa", "商品名")  
Controller: チェックするメソッド情報: max(100, "aaaaa", "商品名")  
Controller: チェックするメソッド情報: rmenu_required("12345", "単位")  
Controller: チェックするメソッド情報: rmenu_free("12345", "単位")  
Controller: チェックするメソッド情報: min(1, "12345", "単位")  
Controller: チェックするメソッド情報: max(20, "12345", "単位")  
Controller: チェックするメソッド情報: rmenu_required("12345", "単価")  
Controller: チェックするメソッド情報: rmenu_integerP("12345", "単価")  
Controller: チェックするメソッド情報: min(1, "12345", "単価")  
Controller: チェックするメソッド情報: max(7, "12345", "単価")
```

[バリデーションJSONの役割]

入力されたデータをDBに入れて良いかをチェックしている
※クライアント側とサーバー側で2回チェックする
(web環境でデータをいじられる恐れもある為)

エラー時→ビューヘエラーメッセージ表示し処理を止める
OK時→モデルへデータを送り通常処理を行う

```
{  
  "商品名": {  
    "validation": [ "required", "free" ],  
    "min": "1",  
    "max": "100" },  
  "単位": {  
    "validation": [ "required", "free" ],  
    "min": "1",  
    "max": "20" }  
}
```

バリデーションJSONチェック指示	
nonrequired	空白OK
required	何か絶対入力
integerp	数字だけOK
min1	最低1桁は入力
max10	最大10文字までOK
free	文字入力OK

ログビューア

▶ ログビューア操作方法

備考：③コントローラログの詳細 (insert登録処理を追いかけます)

『3.チェック済リクエストデータ』バリデーションチェックが終わり、DBに入れても良いデータをモデルへ送ります

```
MainController: チェック済リクエストデータ: {"comment": "商品マスター入力  
リクエストデータ", "html": "RmenuSales/Json/Apps  
/ProductItemInput", "mode": "insert", "prog": "no", "model": "yes", "message":  
{"status": "OK", "msg": ""}, "records":  
[[{"id": "login", "before": "", "after": "", "multiline": "no", "record": {"ユーザ  
{"value": ["1"]}, "ユーザ名称": {"value": ["Rmenu管理者"]}}}],  
{"id": "header", "before": "", "after": "", "multiline": "no", "record": {"請求中  
ID": {"value": ["4"]}, "請求形態ID": {"value": ["3"]}, "商品名": {"  
["aaaaaa"]}, "単位": {"value": ["12345"]}, "単価": {"value": ["12345"]}, "NFC  
フラグ": {"value": ["1"]}, "備考": {"value": [""]}}}]}
```

この後④モデルログ⑤モデルDBログへ処理が移り、
モデルログから戻ってきたデータが次の「4.」です

『4.モデルが作成したsql』モデルから戻ってきたSQLデータを受け取り、つぎはビューへ送る

```
MainController: モデルが作成したsql_data: {"comment": "商品マスター入力  
理", "html": "RmenuSales/Json/Apps  
/ProductItemInput", "dbname": "rmenusales", "mode": "insert", "prog": "no", "me  
{"status": "OK", "msg": "登録処理は正常に終了しました。"}, "sqls": [{"comment  
マスター入力 新規処理", "id": "header", "before": "", "after": "", "sql":  
{"type": "insert", "freesql": "", "genesql": [{"dist": "", "from": "商品マス  
ター", "where": "", "order": ""}], "input": {"multiline": "no", "record": {"請求  
ID": {"value": ["4"]}, "field": "ref_請求内容ID", "請求形態ID": {"value  
["3"]}, "field": "ref_請求形態ID", "商品名": {"value": ["aaaaaa"]}, "単位":  
{"value": ["12345"]}, "単価": {"value": ["12345"]}, "NFC表示フラグ": {"valu  
["1"]}, "備考": {"value": [""]}, "削除フラグ": {"value": [""]}, "funct": "0"}, "登  
時": {"value": [""], "funct": "CURRENT_TIMESTAMP"}, "登録者": {"value": ["Rmenu  
者"], "fromtype": "request", "fromid": "login", "fromio": ""}, "fromname": "ユー  
"}], "fromtype": "request", "fromid": "login", "fromio": ""}, "fromname": "ユー
```

【JSON表示】SQL

```
"comment": "商品マスター入力 新規処理",  
"html": "RmenuSales/Json/Apps/ProductItemInput",  
"dbname": "rmenusales",  
"mode": "insert",  
"prog": "no",  
"message": {"  
  "status": "OK",  
  "msg": "登録処理は正常に終了しました。"  
},  
"sqls": [  
  {"comment": "商品マスター入力 新規処理",  
  "id": "header",  
  "before": "",  
  "after": "",  
  "sql": {"  
    "type": "insert",  
    "freesql": "",
```

【JSON表示】input

```
"field": "ref_請求形態ID"  
},  
"商品名": {"  
  "value": ["  
    "aaaaaa"  
  ]  
},  
"単位": {"  
  "value": ["  
    "12345"  
  ]  
}
```

DBに登録できた内容が返ってきた

ログビューア

▶ ログビューア操作方法

備考：③コントローラログの詳細 (insert登録処理を追いかけます)

『5.ビューが作成したレスポンスデータ』 ビューから戻ってきたレスポンスデータをRACKへ返す

```
MainController: ビューが作成したresponse_data : {"comment": "商品マスター  
新規処理 レスポンスデータ", "html": "RmenuSales/Json/Apps  
/ProductItemInput" "mode": "insert", "prog": "no", "message": {"status": "OK",  
データを正常に登録しました。次のデータを入力するか、戻るボタン (Esc) を押  
下さい。"}}  
MainController: call normal end  
MainController: call start
```

【JSON表示】

```
"comment": "商品マスター入力 新規処理 レスポンスデータ",  
"html": "RmenuSales/Json/Apps/ProductItemInput",  
"mode": "insert",  
"prog": "no",  
"message": {  
  "status": "OK",  
  "msg": "データを正常に登録しました。次のデータを入力す  
}"
```

Messageを表示する (validationがOK→sqlもOK→DBにデータが入ったので表示するメッセージが正常に登録OK)

ログビューア

▶ ログビューア操作方法

⑤ **モデルDBログ** (モデルから指示を貰いSQLを作成し、DBを更新し、結果をモデルへ返す役割)

SQLが確認できる (コピーしてポスグレに張り付けるとテーブルが見れます)

```
RmenuSequel: connect end
RmenuSequel: doSql start
RmenuSequel: doGenerateSql start
RmenuSequel: バインド変数、置換前のSQL : SELECT 請求内容ID AS 選択請求内容ID, 請求内容名称 AS 選択請求内容名称 FROM 請求内容 WHERE 削除フラグ = '0' ORDER BY 表示順
RmenuSequel: バインド変数、置換後のSQL : SELECT 請求内容ID AS 選択請求内容ID, 請求内容名称 AS 選択請求内容名称 FROM 請求内容 WHERE 削除フラグ = '0' ORDER BY 表示順
RmenuSequel: SQL実行前の出力レコード : [{"選択請求内容ID": {"value": "", "field": "請求内容ID"}, "選択請求内容名称": {"value": "", "field": "請求内容名称"}}]
RmenuSequel: SQL実行後の出力レコード : [{"選択請求内容ID": {"value": [1,2,3,4], "field": "請求内容ID"}, "選択請求内容名称": {"value": ["Webサービス広告料", "セミナー受講料", "その他"], "field": "請求内容名称"}}]
RmenuSequel: doGenerateSql end
RmenuSequel: doSql end
```

```
RmenuSequel: connect end
RmenuSequel: doSql start
RmenuSequel: doGenerateSql start
RmenuSequel: バインド変数、置換前のSQL : INSERT INTO 商品マスター (ref_請求内容ID, ref_請求形態ID, 商品名, 単位, 単価, NPO表示フラグ, 備者, 削除フラグ, 登録日時, 更新日時, 更新者) VALUES (?, ?, ?, ?, ?, ?, ?, 0, CURRENT_TIMESTAMP, ?, CURRENT_TIMESTAMP, ?)
RmenuSequel: バインド変数、置換後のSQL : INSERT INTO 商品マスター (ref_請求内容ID, ref_請求形態ID, 商品名, 単位, 単価, NPO表示フラグ, 備者, 削除フラグ, 登録日時, 更新日時, 更新者) VALUES (4, 3, 'aaaaaa', 12345, 12345, 1, NULL, 0, CURRENT_TIMESTAMP, 'Rmenu管理者', CURRENT_TIMESTAMP, 'Rmenu管理者')
RmenuSequel: doGenerateSql end
```

④モデルから指示↓

バインド変数 置換前SQL

「?を出して」

バインド変数 置換後SQL

「呼びました」

SQL実行前の出力レコード

(空のSQL)

SQL実行後の出力レコード

(データの入ったSQL)

この結果を④モデルへ返す↑

上のSQLを使って値を入れる

ログビューア

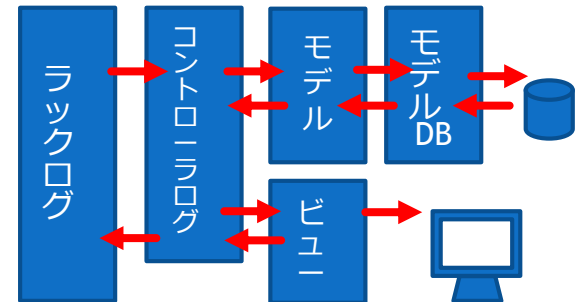
▶ ログビューア操作方法

⑥ビューログ (コントローラからリクエストデータとSQLデータを貰い、レスポンスデータを作成する役割)

```
MainView: call start
MainView: リクエストデータ : [{"comment": "商品マスター入力 登録 リクエストデー
タ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "insert", "prog": "no", "model": "yes", "message":
{"status": "OK", "msg": ""}, "records": [{"id": "login", "before": "", "after": "", "multiline": "no", "record": {"ユーザID":
{"value": ["1"]}, "ユーザ名称": {"value": ["Rmenu管理者"]}},
{"id": "header", "before": "", "after": "", "multiline": "no", "record": {"請求内容ID": {"value": ["4"]}, "請求形態ID":
{"value": ["3"]}, "商品名": {"value": ["aaaaaa"]}, "単位": {"value": ["12345"]}, "単価": {"value": ["12345"]}, "NPO表示フラ
グ": {"value": ["1"]}, "備考": {"value": [""]}}]}]}
MainView: SQLデータ : [{"comment": "商品マスター入力 新規処
理", "html": "RmenuSales/Json/Apps/ProductItemInput", "dbname": "rmenusales", "mode": "insert", "prog": "no", "message":
{"status": "OK", "msg": "登録処理は正常に終了しました。"}, "sqls": [{"comment": "商品マスター入力 新規処
理", "id": "header", "before": "", "after": "", "sql": {"type": "insert", "freesql": "", "genesql": {"dist": "", "from": "商品マスタ
ー", "where": "", "order": ""}}, "input": {"multiline": "no", "record": {"請求内容ID": {"value": ["4"]}, "field": "ref_請求内容I
D"}, {"請求形態ID": {"value": ["3"]}, "field": "ref_請求形態ID"}, {"商品名": {"value": ["aaaaaa"]}, "単位": {"value":
["12345"]}, "単価": {"value": ["12345"]}, "NPO表示フラグ": {"value": ["1"]}, "備考": {"value": [""]}, "削除フラグ": {"val
ue": [""]}, "funct": "0"}, {"登録日時": {"value": [""], "funct": "CURRENT_TIMESTAMP"}, "登録者": {"value": ["Rmenu管理
者"], "fromtype": "request", "fromid": "login", "fromio": "", "fromname": "ユーザ名称"}, "更新日時": {"value":
[""], "funct": "CURRENT_TIMESTAMP"}, "更新者": {"value": ["Rmenu管理
者"], "fromtype": "request", "fromid": "login", "fromio": "", "fromname": "ユーザ名称"}]}]}]}
View: initialize start
View: initialize normal end
View: call start
View: def_cache.getJsonData start
View: Json レスポンスデータ : [{"comment": "商品マスター入力 新規処理 レスポンスデー
タ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "insert", "prog": "no", "message": {"status": "OK", "msg": "デー
タを正常に登録しました。次のデータを入力するか、戻るボタン (Esc) を押下して下さい。"}]}
View: def_cache.getJsonData normal end
View: app_cache.getViewObject start
View: app_cache.getViewObject normal end
View: fromSqlToResponse start
View: fromSqlToResponse normal end
View: call normal end
MainView: クライアントへ送信する、レスポンスデータ : [{"comment": "商品マスター入力 新規処理 レスポンスデー
タ", "html": "RmenuSales/Json/Apps/ProductItemInput", "mode": "insert", "prog": "no", "message": {"status": "OK", "msg": "デー
タを正常に登録しました。次のデータを入力するか、戻るボタン (Esc) を押下して下さい。"}]}
MainView: call normal end
```

「リクエストデータ」
(↑入力されたデータ)
「SQLデータ」
(↑DBに入ったデータ入り)
「レスポンスデータ」
「編集前 レスポンス情報」
「編集後 レスポンス情報」
「クライアントへ送信するレスポンスデータ」
↓
①ラックログへレスポンスJSONが送られる

【ログの流れ】



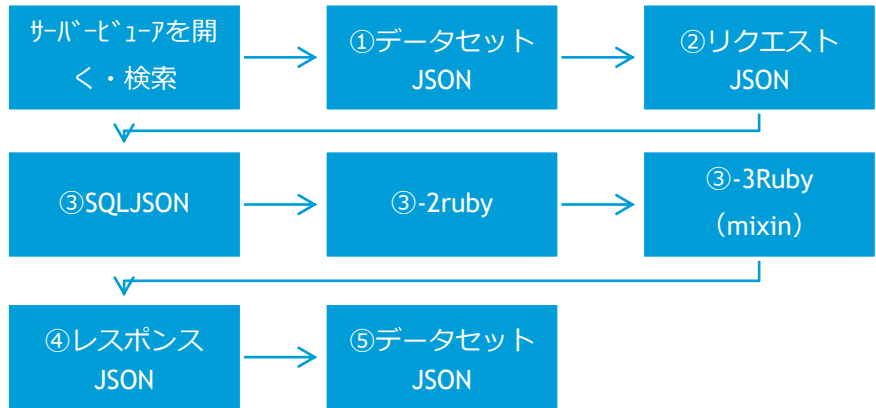
サーバービューア

▶ サーバービューアとは

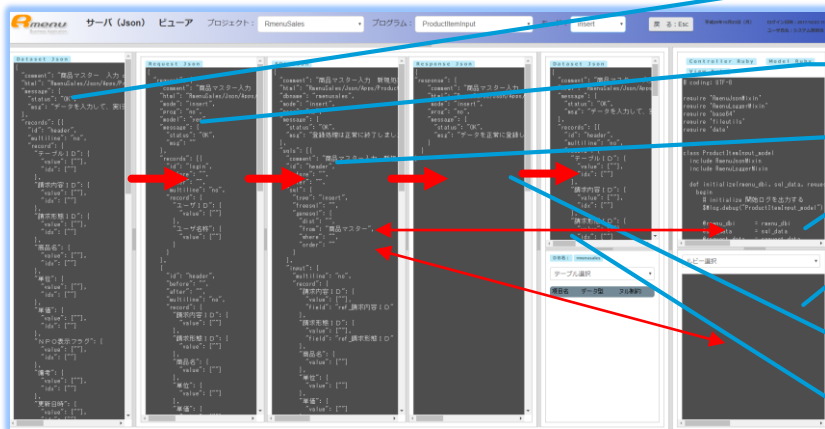
【サーバービューアで確認できるもの】
サーバー側 (JSON/ruby) が確認できます



【手順】



【データの流れ】
矢印がデータの流れる順をあらわします



①データセットJSON (リクエスト状態)

②トランJSON (リクエスト)

③-1 SQLJSON

③-2 ruby (コントローラ・モデル・ビュー)

③-3 ruby (Mixin共通のプログラム)

④トランJSON (レスポンス)

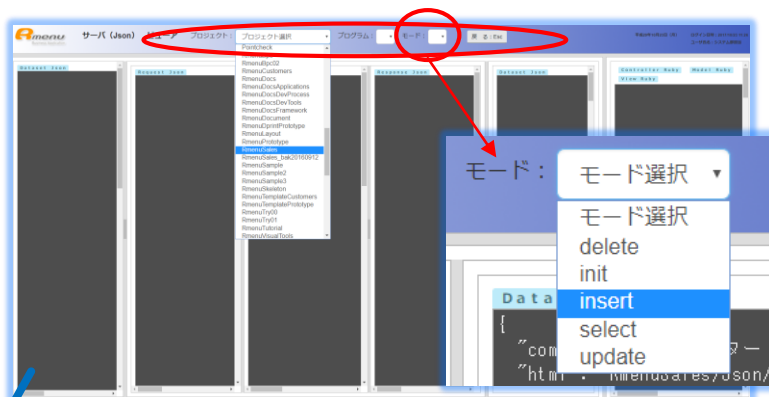
⑤データセットJSON (レスポンス状態)

サーバービューア

▶ サーバービューア操作方法

サーバービューアを開く

上部で【プロジェクト】【プログラム名】【モード】の検索を行うと各プログラムが表示されます



① データセットJSON (リクエスト状態)

WebサーバーからリクエストJSONを貰う

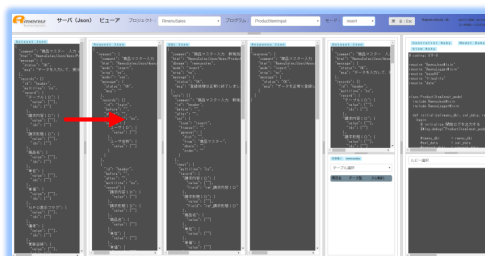
「id = 0000」

項目名が同じであれば自動でデータが流れる

動かないときはidが間違っていないか確認する事
(Html・JSON・SQL全てidをそろえる)

② トランJSON (リクエスト)

「この処理をして」
「このプログラムを使って」と指示が来る



サーバービューア

▶ サーバービューア操作方法

③-1 SQLJSON

SQLJSONを開いて確認できる

Before : に記載がある時はrubyのモデル内に

SetSql○○○があり、ここへ処理が飛びます

このrubyはsqlの処理不足を補ってデータを生成します

【SQLJSON】

```
SQL J son
{
  "comment": "商品マスター メンテ 初期画面処理",
  "html": "RmenuSales/Json/Apps/ProductItemList",
  "dbname": "rmenusales",
  "mode": "selectmaintable",
  "prog": "yes",
  "message": {
    "status": "OK",
    "msg": "照会処理は正常に終了しました。"
  },
  "sqls": [
    {
      "comment": "総件数を取得する",
      "id": "total_count",
      "before": [
        "setSql_商品マスターOfRmenuSales('header', 'total_count', 'genesql')",
      ],
      "after": "",
      "sql": {
        "type": "select",
        "freesql": "",
        "genesql": {
          "dist": "
          "from": "商品マスター AS A",
          "where": "A.削除フラグ = '0' &&",
          "order": ""
        }
      }
    }
  ]
}
```

ファイルの場所

処理内容 (どのJSONを使うか)

Yesはプログラムを使う意味

Before : Rubyのモデルを使う時の指示

このRubyを使う③-2へ行きます

Type : は処理の種類を記入select・insert等

freesql : SQL分を書き込む場所

genesql : JSONの書き方自動でsqlを生成パターン

③-2 ruby (コントローラ・モデル・ビュー)

上部でどのrubyを開くのが選択できる

```
Controller Ruby Model Ruby View Ruby
# coding: UTF-8
require 'RmenuSales/Server/Modules/RmenuSalesMasterMainteOfModelMixi

# SQL文の検索条件を変更する
def setSql_商品マスターOfRmenuSales(idname1, idname2, genesql, freesql)
  $Mlog.debug("ProductItemList(_model)") {"setSql_商品マスターOfRmenuSales st

  requestInfo = getJsonChunkById(@request_data, "records", idname1)

  searchString01 = requestInfo["record"]["検索請求内容ID"]["value"][0]
  searchString02 = requestInfo["record"]["検索請求形態ID"]["value"][0]
  searchString03 = requestInfo["record"]["検索商品名"]["value"][0]

  sqlInfo = getJsonChunkById(@sql_data, "sqls", idname2)
```

SQLJSONで不足部分はRubyを利用してプログラムを作成 (このプログラム用の追加ruby) ログビューアのモデルDBログのdosqlはこの事

③-3 ruby (Mixin共通のプログラム)

上記と同じSQLJSONで不足部分はRubyを利用して

プログラムを作成します

複数のプログラムが同じ指示を使う為1か所にまとめて書き、必要な時にそれぞれ呼び出して使います

共通のパターンとしてjsフォルダーにいます

サーバービューア

▶ サーバービューア操作方法

④ トランJSON (レスポンス)

```
Response Json
{
  "response": {
    "comment": "売上入力 照会処理"
    "html": "RmenuSales/Json/Apps/"
    "mode": "select",
    "prog": "no",
    "message": {
      "status": "OK",
      "msg": "照会処理は正常に終了"
    },
    "records": [
      {
        "id": "header",
        "multiline": "no",
        "before": "",
        "after": "",
        "multiline": "no",
        "record": {
          "ヘッダー日付": {
            "value": ""
          },
          "会計年度": {
            "value": ""
          },
          "伝票番号": {
            "value": ""
          },
          "契約区分": {
            "value": ""
          },
          "アカウント発行区分ID": {
            "value": ""
          },
          "契約者ID": {
            "value": ""
          },
          "契約者名": {
            "value": ""
          },
          "伝票"
        }
      }
    ]
  }
}
```

Value : の中に
データを入れて
データを移送する

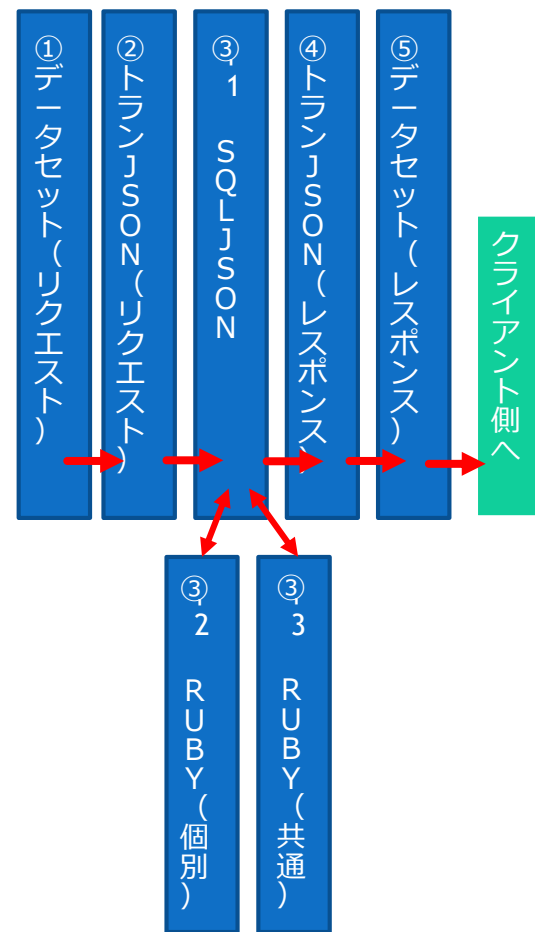
⑤ データセットJSON (レスポンス状態)

```
Dataset Json
{
  "comment": "売上入力 dataset"
  "html": "RmenuSales/Json/Apps/"
  "message": {
    "status": "OK",
    "msg": "データを入力して、"
  },
  "records": [
    {
      "id": "header",
      "multiline": "no",
      "record": {
        "ヘッダー日付": {
          "value": "",
          "idx": ""
        },
        "会計年度": {
          "value": "",
          "idx": ""
        },
        "伝票番号": {
          "value": "",
          "idx": ""
        },
        "契約区分": {
          "value": "",
          "idx": ""
        },
        "アカウント発行区分ID": {
          "value": "",
          "idx": ""
        },
        "契約者ID": {
          "value": "",
          "idx": ""
        },
        "契約者名": {
          "value": "",
          "idx": ""
        },
        "伝票"
      }
    }
  ]
}
```

Value : の中に
データを入れて
データを移送する

レスポンスは
Value : ヘデータが
入った状態になる

【項目名を追いかける順】



クライアントビューア

▶ クライアントビューアとは

クライアント側のプログラムを一覧で確認できます



index.html
.appspec.js
.controller.js
main.js
.model.js
.view.js

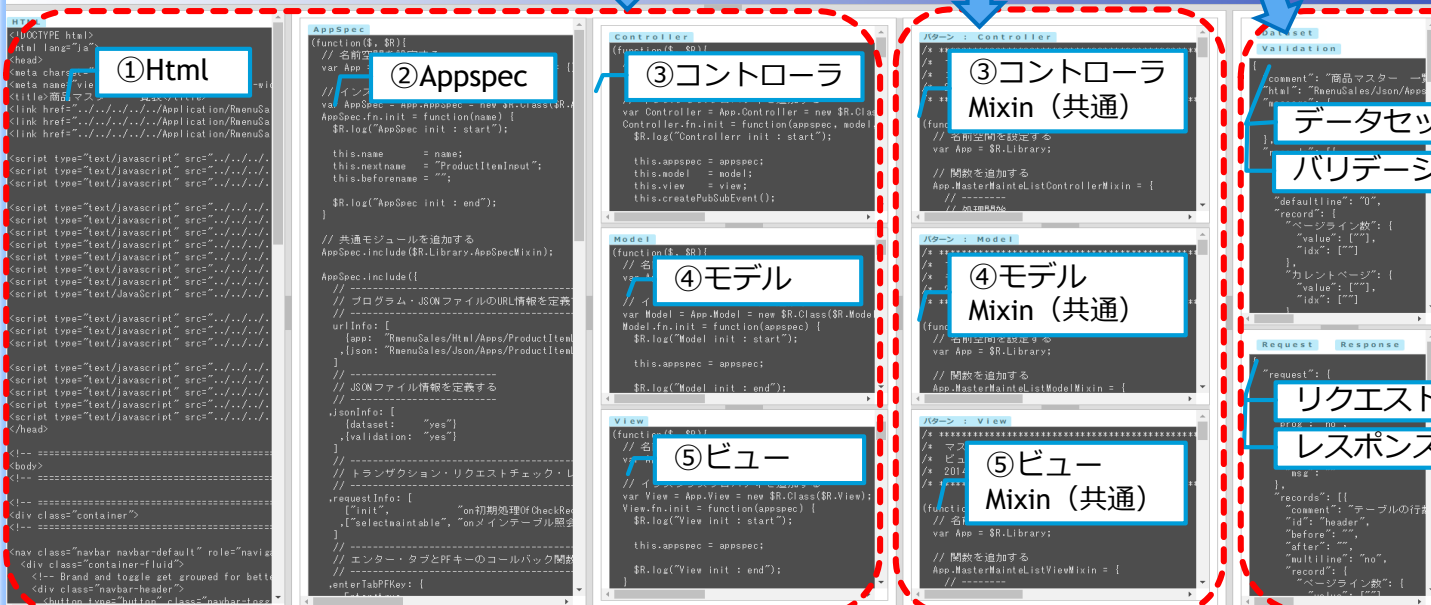
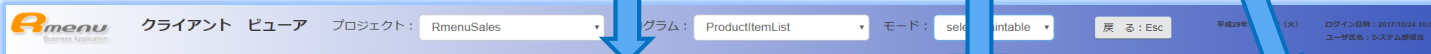
controller.mixin-2.1.1.js
.model.mixin-2.1.1.js
.view.mixin-2.1.1.js

JSON

クライアントビューアは大きく分けると3種類のフォルダーを確認しています

- ①クライアント側のプログラム
- ②共通のjs
- ③JSON

※main.jsは表示なし
(jqueryの宣言が入っているだけの為)



クライアントビューアのコードエディタのスクリーンショット。①Html、②Appspec、③コントローラ、④モデル、⑤ビュー、③コントローラ Mixin (共通)、④モデル Mixin (共通)、⑤ビュー Mixin (共通)、データセットJSON、バリデーションJSON、リクエストJSON、レスポンスJSON

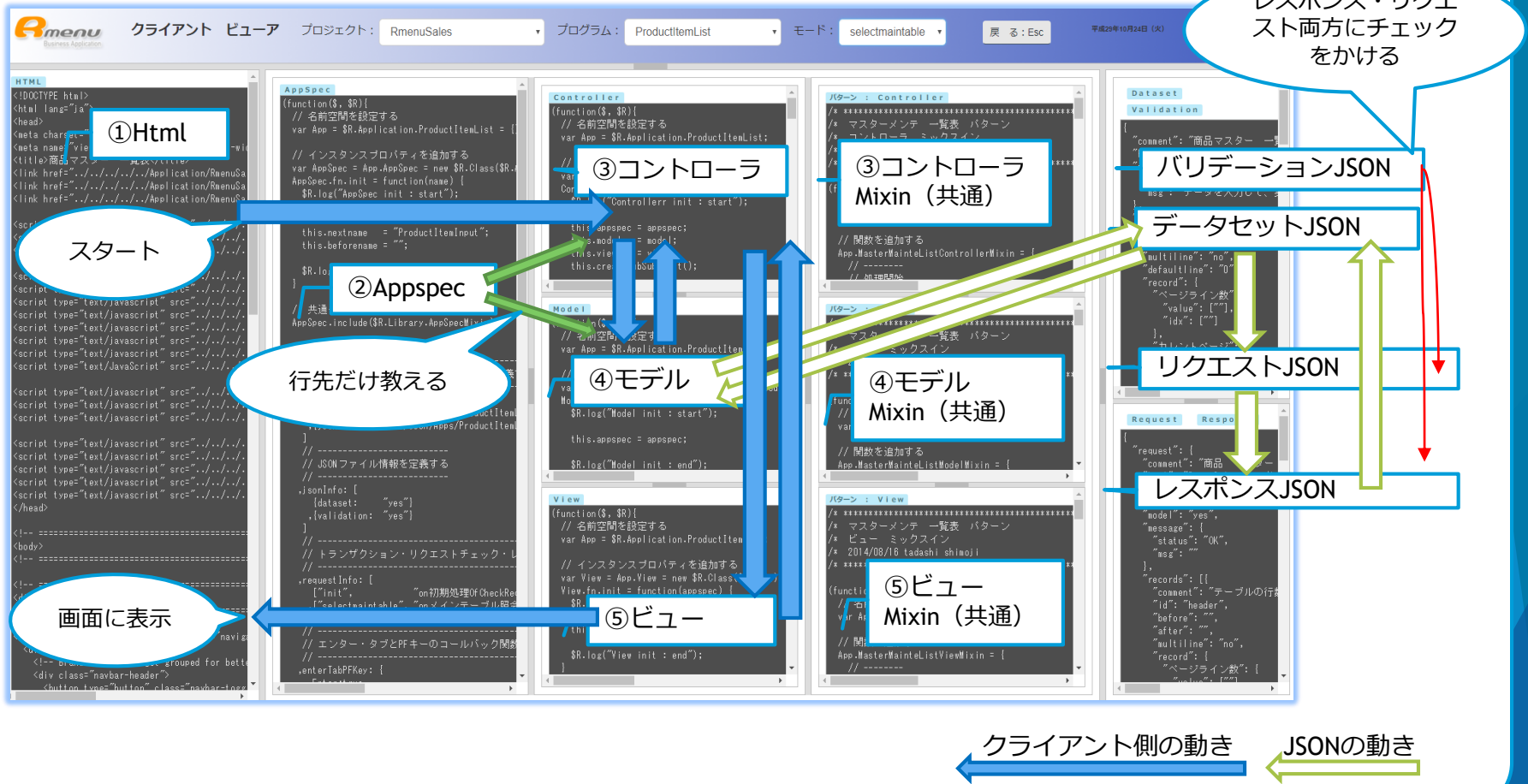
データセットJSON
バリデーションJSON

リクエストJSON
レスポンスJSON

クライアントビューア

データの流れ

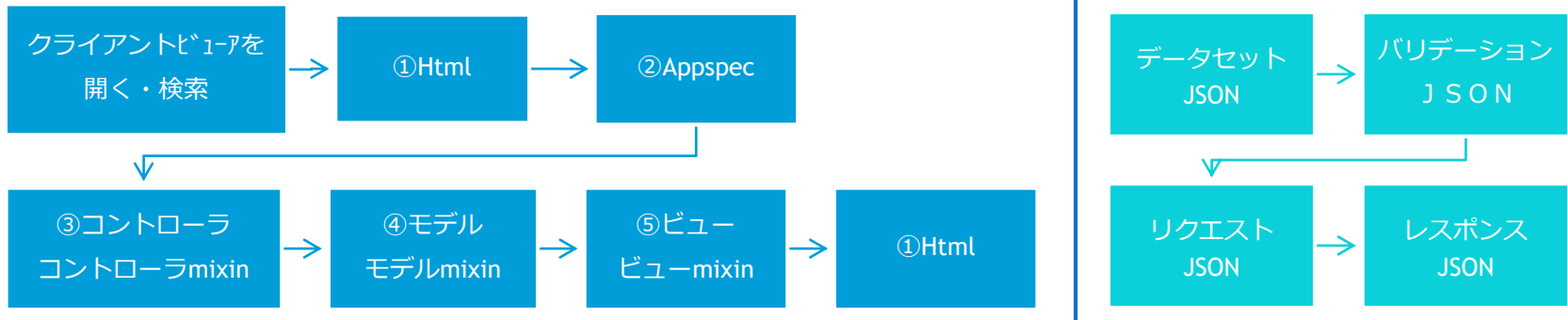
Htmlからスタートしモデルからサーバー側の処理へ移り、クライアント側で処理を実行している、クライアント側の流れがクライアントビューアで確認できます



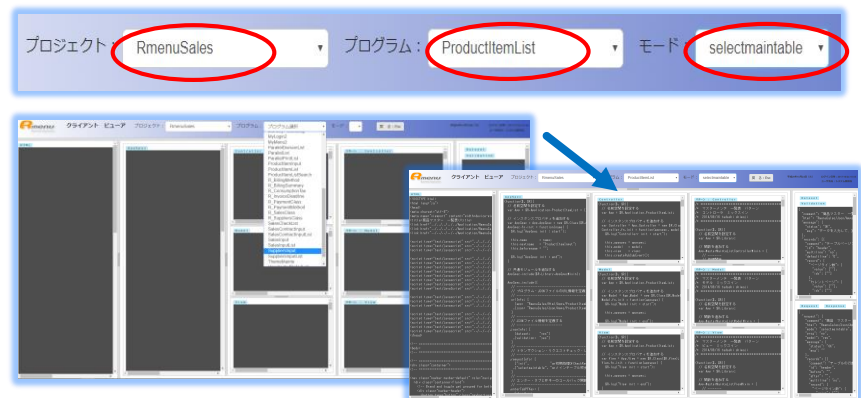
クライアントビューア

▶ クライアントビューア操作方法

【手順】



ログビューアを開く上部で【プロジェクト】【プログラム名】【モード】検索を行うと各プログラムが表示される



クライアントビューア

①Htmlの簡単な内容ここで画面が作成される（画面・ID・ボタン・・・）

```
HTML
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>商品マスター 入力</title>
<link href="..." rel="stylesheet" type="text/css">
<link href="..." rel="stylesheet" type="text/css">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
<script type="text/javascript" src="...">
</head>
<!-- ===== -->
<body>
<!-- ▼body部 スタート -->
<!-- ===== -->
<!-- ===== -->
<div class="container">
<!-- ▼container スタート -->
<!-- ===== -->
<nav class="navbar navbar-default" role="navigation">
<div class="container-fluid">
<!-- Brand and toggle get grouped for better mobile display -->
<div class="navbar-header">
```

1.画面タイトル

2.CSSの読み込み

3.各ライブラリーの読み込み

4.Rmenu共通の読み込み

5.Rmenu共通パターンの読み込み

6.このプログラム用のjs

- .appspec.js
- .controller.js
- .main.js
- .model.js
- .view.js

クライアントビューア

② Appspec イベントの設定・JSONファイルの場所の情報だけ入っている

```
// 共通モジュールを追加する
AppSpec.Include($R.Library.AppSpecMixin);

AppSpec.Include({
// -----
// プログラム・JSONファイルのURL情報を定義する (サブシステム名・種類・Apps・プログラムのURL)
// -----
urlInfo: {
  [app: "RmenuSales/Html/Apps/ProductItemInput/"],
  [json: "RmenuSales/Json/Apps/ProductItemInput/"]
}
// -----
// JSONファイル情報を定義する
// -----
jsonInfo: {
  [dataset: "yes"],
  [validation: "yes"]
}
// -----
// トランザクション・リクエストチェック・レスポンス編集・エラーのコールバック関数を定義する
// -----
requestInfo: {
  ["init", "on初期処理Of CheckRequestData", "on初期処理Of EditResponseData"],
  ["select", "on照会Of CheckRequestData", "on照会Of EditResponseData"],
  ["insert", "on登録Of CheckRequestData", "on登録Of EditResponseData"],
  ["update", "on訂正Of CheckRequestData", "on訂正Of EditResponseData"],
  ["delete", "on削除Of CheckRequestData", "on削除Of EditResponseData"],
}
// -----
// エンター・タブとPFキーのコールバック関数を定義する
// -----
enterTabPFKey: {
  Enter: true
  ,Tab: true
  ,F1: "on実行"
  ,F2: null
  ,F3: null
  ,F4: null
  ,F5: null
  ,F6: null
  ,F7: null
  ,F8: null
  ,F9: null
  ,F10: null
  ,F11: null
  ,F12: null
  ,ESC: "on戻る"
  ,Forms: 0
}
// -----
// イベント設定にセレクタ・イベント・コールバック関数の順に指定する
// -----
// NAVボタンのイベントを定義する
navButtonEvent: [
  ["戻る", "click", "on戻る"],
  ["実行", "click", "on実行"],
  // ["画像検索", "click", "on画像検索"],
  // ["画像検索後処理", "click", "on画像検索後処理"]
]
// バリデーションのイベントを定義する
validationEvent: [
```

URLインフォ：ファイルの置いている場所を定義 (Htmlとサーバー側のJSONをつなぐ)

JSONインフォ：上記の場所から持ってきてほしいJSONがyes

リクエストインフォ：上記の場所から持ってきてほしい (処理内容によって)

INSERT、リクエストデータ、レスポンスデータ、

ENTERキーは無効 (true) キーの指示

F1キーはon実行

他にもキーボードのキーが押された時の処理が書かれている

イベントの設定画面のボタンがクリックされたらどこへ移動するのかを書いている所

クライアントビューア

③コントローラ

Controller

```
function($, $R){  
  // 名前空間を設定する  
  var App = $R.Application.ProductItemInput;  
  
  // インスタンスプロパティを追加する  
  var Controller = App.Controller = new $R.Class($R.Controller);  
  Controller.fn.init = function(appspec, model, view) {  
    $R.log("Controllerr init : start");  
  
    this.appspec = appspec;  
    this.model = model;  
    this.view = view;  
    this.createPubSubEvent();  
  
    $R.log("Controllerr init : end");  
  };  
  
  // モジュールを追加する  
  Controller.include($R.Library.EnterTabPFKeyMixin);  
  Controller.include($R.Library.ControllerMixin);  
  
  // マスターメンテパターン ミックスインを追加する  
  Controller.include($R.Library.MasterMainteControllerMixin);  
  
  // -----  
  // パターンに含まれない処理を追加する  
  // また、パターン内の処理を変更するとき、オーバーライドする  
  // -----  
  Controller.include({  
    // -----  
    // 初期処理  
    // -----  
    on初期処理: function() {  
      $R.log("Controller on初期処理 : start");  
  
      this.model.on初期処理();  
      this.ajaxExecute("init");  
  
      $R.log("Controller on初期処理 : end");  
    }  
  });  
}
```

名前空間とは次の資料で説明

Thisはこのコントローラの事
コントローラは
Appspec/model/viewを呼出せ
ることがわかる

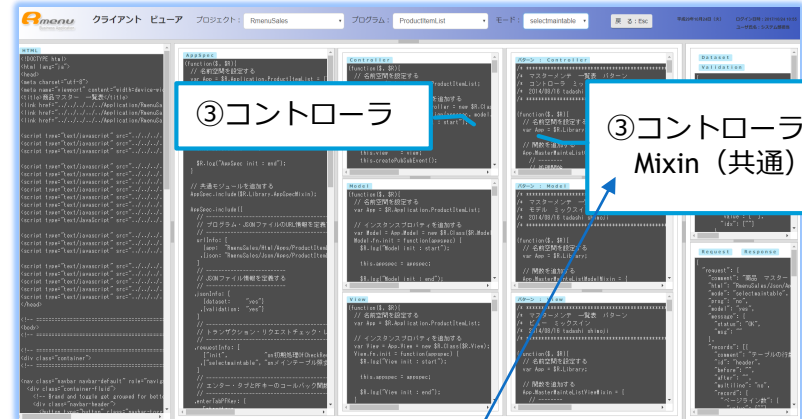
\$ R.log () はログビューアに表示される部分

共通のコントローラを読み込んでいる

共通のコントローラにプログラムを追加したいときだけ
ここから下に記載する（オーバーライドは上書の意味）

Modelへ処理をパスしている

Init処理をします



③コントローラ

③コントローラ
Mixin (共通)

クライアントビューア

④モデル

Model

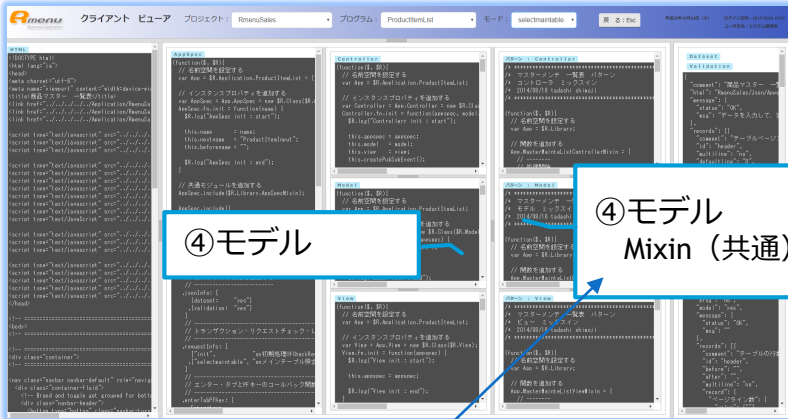
```
(function($, $R){  
  // 名前空間を設定する  
  var App = $.Application.ProductItemInput;  
  
  // インスタンスプロパティを追加する  
  var Model = App.Model = new $.Class($R.Model);  
  Model.fn.init = function(appspec) {  
    $R.log("Model init : start");  
  
    this.appspec = appspec;  
  
    $R.log("Model init : end");  
  }  
  
  // モジュールを追加する  
  Model.include($R.Library.ValidationMixin);  
  Model.include($R.Library.ModelMixin);  
  Model.include($R.Library.HtmlTransitionMixin);  
  
  // マスターメンテパターン ミックスインを追加する  
  Model.include($R.Library.MasterMainteModelMixin);  
  
  // -----  
  // パターンに含まれない処理を追加する  
  // また、パターン内の処理を変更するとき、オーバーライドする  
  // -----  
  Model.include({  
    // -----  
    // チェンジイベント処理 (追加処理)  
    // -----  
    onChangeNPO表示フラグ: function(event) {  
      $R.log("Model onChangeNPO表示フラグ : start");  
  
      var w_NPO表示フラグ = $("#NPO表示フラグ").val();  
      this.dataset.setElementData(w_NPO表示フラグ, "NPO表示フラグ");  
  
      $R.log("Model onChangeNPO表示フラグ : end");  
    }  
  });  
})
```

Thisはこのモデルの事
モデルはAppspecを呼出せること
がわかる (JSONの場所を知るため
に)

共通のコントローラを読み込んでいる

共通のコントローラにプログラムを追加したいときだけ
ここから下に記載する (オーバーライドは上書の意味)

データセットする→webサーバ→DB→レスポンス (データ
入りで戻ってくる) →コントローラへ返す



クライアントビューア

⑤ビュー

```
View
(function($, $R){
  // 名前空間を設定する
  var App = $.R.Application.ProductItemInput;

  // インスタンスプロパティを追加する
  var View = App.View = new $.R.Class($R.View);
  View.fn.init = function(appspec) {
    $.R.log("View init : start");

    this.appspec = appspec;
    $.R.log("View init : end");
  }

  // 共通モジュールを追加する
  View.include($R.Library.FormatMixin);
  View.include($R.Library.OutLineMixin);
  View.include($R.Library.LoadTemplateMixin);
  View.include($R.Library.ViewMixin);

  // マスターメンテパターン ミックスインを追加する
  View.include($R.Library.MasterMainteViewMixin);

  // -----
  // パターンに含まれない処理を追加する
  // また、パターン内の処理を変更するときは、オーバーライドす
  // -----
  View.include({
    // -----
    // レスポンスデータ 編集処理
    // -----
    on初期処理OfEditResponseData: function(arg) {
      $.R.log("View on初期処理OfEditResponseData : start");

      $("#請求内容 I D").val(arg["請求内容 I D"]);
      $("#請求形態 I D").val(arg["請求形態 I D"]);

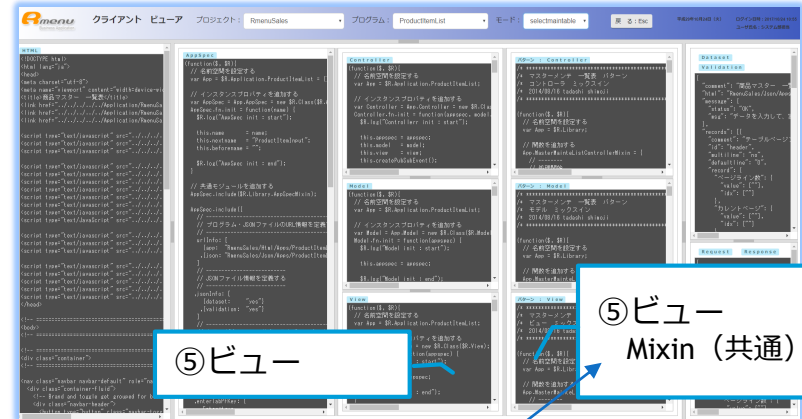
      $.R.log("View on初期処理OfEditResponseData : end");
    }
  });
});
```

Thisはこのビューの事
ビューはAppspecを呼出せること
がわかる (JSONの場所を知るため
に)

共通のコントローラを読み込んでいる

共通のコントローラにプログラムを追加したいときだけ
ここから下に記載する (オーバーライドは上書の意味)

Val ("") の中に表示するものが入る
(コントローラがモデルからもらったレスポンスデータ)
貰ったデータをテーブルセットしてHtmlへ送りデータを表示する



ビジュアルツールとは 完了

各プログラムの詳細な読み方はそれぞれの詳細説明で確認します
ここではデータ流れの方を確認して終了します

Rmenu基礎知識

RACKの詳細

「RACK内のデータの流れを確認します」

RACKの詳細

もくじ

- ▶ RACKとは
- ▶ RACKの種類
- ▶ RACKの流れ（Rmenu起動時）
- ▶ RACKの流れ（ログイン時）

RACKの詳細

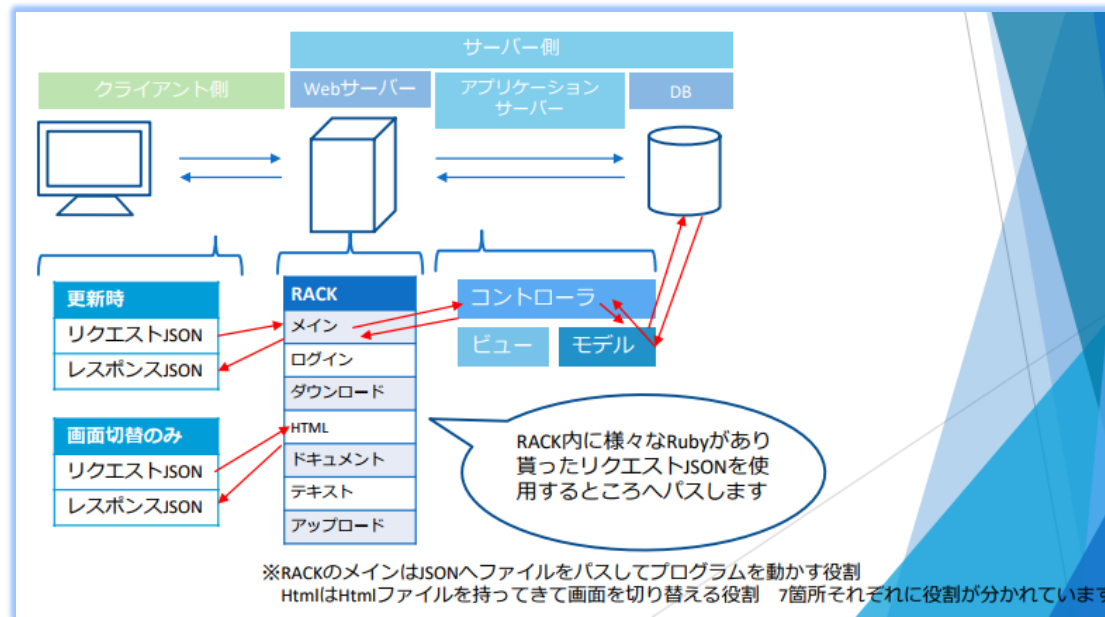
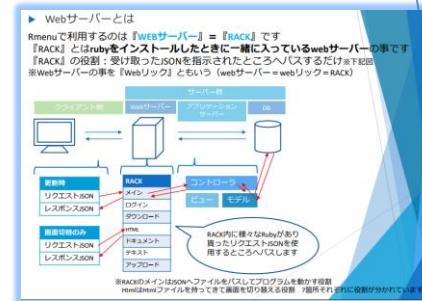
▶ 『RACK』とは

『RACK』とは：rubyをインストールしたときに一緒に入っているwebサーバーの事です

『RACK』の役割：受け取ったJSONを指示されたところへパスするだけ

※「Rmenuとは」の「webサーバーとは」の続きになります

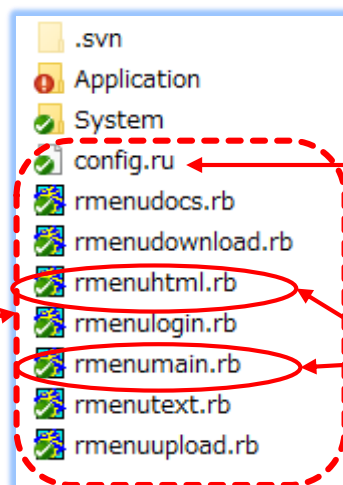
- RACKにはたくさん種類があります
- Rmenu 起動時に『NO.0』で立ち上げるのがRACKです
- RACKの詳細ではRACK内のデータの流を確認します



RACKの詳細

▶ RACKの種類

RACKはRmenuを開いて直ぐの場所であり、7種類に分かれています



No.0実行時に下の7つのラックを立ち上げる

よく使われる所

ファイル名	読みカナ	用途
config.ru	コンフィグ	NO.0で立ち上げる時（他のRACKを起動する指示が入っている）
rmenudocs.rb	ドキュメント	
rmenudownload.rb	ダウンロード	エクセルにDLする時
rmenuhtml.rb	HTML	画面の表示だけを変える時
rmenulogin.rb	ログイン	ログイン時・ログを残す時
rmenumain.rb	メイン	システムを使う時
rmenutext.rb	テキスト	
rmenuupload.rb	アップロード	ブラウザから写真を送る時

RACKの詳細

▶ RACKの流れ (Rmenu起動時)

Rmenu起動時に『no.0』と『no.1』を開きますが、
『No.0』は『webサーバー = RACK』を起動させます
ここでは『No.0』を実行する時のRACKの流れを確認します

RACKの流れ

① 「No.0」を立ち上げる
(config.rbを起動する指示がある)



② 「config.rb」が起動する
(各ラックを起動させる指示がある)



③各ラックが起動し
準備完了



Html表示を実行し
Rmenuを利用できる

① 「No.0」を立ち上げる (config.rbを起動する指示がある)

```
C:\Windows\system32\cmd.exe
C:\RmenuSVN\Rmenu\System> rem Rmenuフォルダに移動する
C:\RmenuSVN\Rmenu\System> cd ../
C:\RmenuSVN\Rmenu> rem Rackを起動する
C:\RmenuSVN\Rmenu> rackup
[2017-03-08 11:51:49] INFO WEBrick 1.3.1
[2017-03-08 11:51:49] INFO ruby 2.2.5 (2016-04-26) [i386-mingw32]
[2017-03-08 11:51:49] INFO WEBrick::HTTPServer#start: pid=7778 port=9292
```

Rackを起動する

WEBrick

RACKの詳細

▶ RACKの流れ (Rmenu起動時)

② 「config.rb」が起動する (各ラックを起動させる指示がある)

```
config.ru
1 #coding:UTF-8
2
3 $LOAD_PATH.push(File.dirname(__FILE__)+"../System/Server/Libraries/Main/")
4
5 require './rmenu/main'
6 require './rmenu/html'
7 require './rmenu/login'
8 require './rmenu/download'
9 require './rmenu/upload'
10 require './rmenu/docs'
11 require './rmenu/text'
12
13 use Rack::Static, :urls => ["/Application", "/System"]
14
15 # 業務画面用ラック
16 map '/RmenuRack/RmenuMain.rb' do
17   run RmenuMain.new
18 end
19
20 # Html画面用ラック
21 map '/RmenuRack/RmenuHtml.rb' do
22   run RmenuHtml.new
23 end
24
25 # ログイン用ラック
26 map '/RmenuRack/RmenuLogin.rb' do
27   run RmenuLogin.new
28 end
29
30 map '/RmenuRack/RmenuDownload.rb' do
31   run RmenuDownload.new
32 end
33
34 # アップロード用ラック
35 map '/RmenuRack/RmenuUpload.rb' do
36   run RmenuUpload.new
37 end
38
39 # ドキュメント画面用ラック
40 map '/RmenuRack/RmenuDocs.rb' do
41   run RmenuDocs.new
42 end
43
44 # テキストファイル用ラック
45 map '/RmenuRack/RmenuText.rb' do
46   run RmenuText.new
47 end
48 [EOF]
```

「config.rb」の中に図の様に指示があり、各ラックを起動させています

Require (リクワイア)
このRubyを順番に起動させる指示

メインラックを動かすときの指示

map : ブラウザからメインを使ってくださいと指示がきたら

run : メインのファイルを動かします (走らせます)

rmenu/main.rbへ移動してプログラムが実行されます
メインはJSONを受け取りコントローラへ

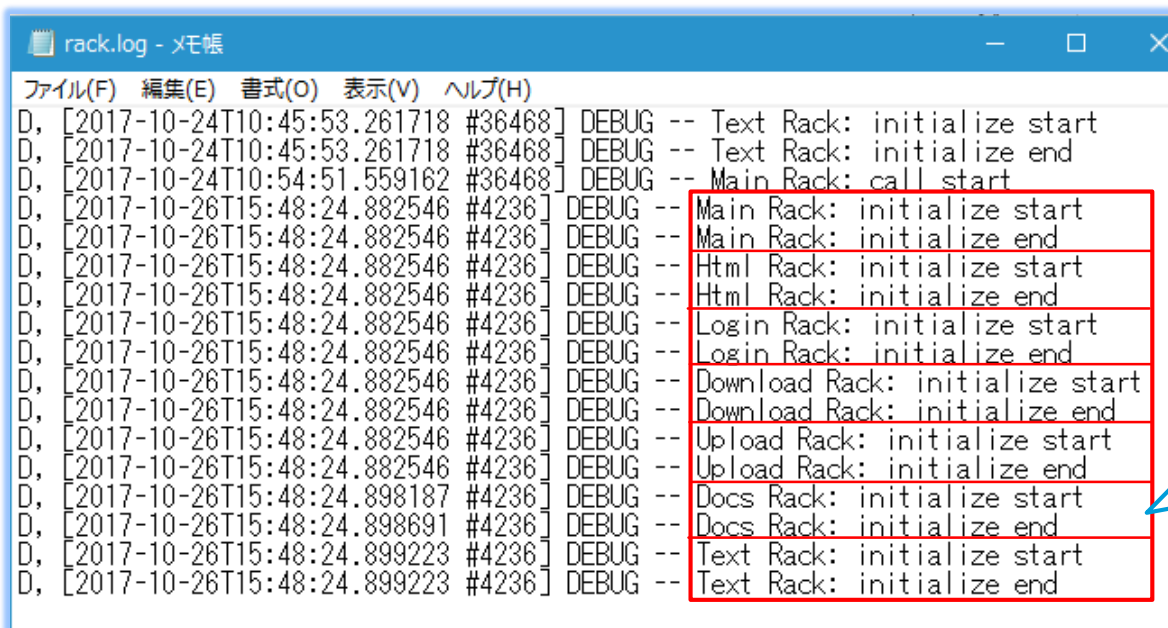
次のラックが順に起動します

```
rmenu/main.rb
1 #coding:UTF-8
2
3 require 'rubygems'
4 require 'rack'
5 require 'json'
6 require 'drb'
7 require 'Rmenu/Config'
8 require 'Rmenu/LosserMixin'
9
10 class RmenuMain
11   include RmenuLosserMixin
12
13   def initialize()
14     @controller = RmenuController.new
15     @logger = RmenuLogger.new("rack")
16
17     @controller.logger = @logger
18     @logger.debug("Main Rack") { "initialize start" }
19
20     @controller = DRBObject.new_with_uri($config['controller'])
21     @menu_controller = DRBObject.new_with_uri($config['menu_controller'])
22   end
23 end
```


RACKの詳細

▶ RACKの流れ (Rmenu起動時)

③各ラックが起動し準備完了
ラックログを開くと、各ラックが用意される様子がわかります



```
rack.log - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
D, [2017-10-24T10:45:53.261718 #36468] DEBUG -- Text Rack: initialize start
D, [2017-10-24T10:45:53.261718 #36468] DEBUG -- Text Rack: initialize end
D, [2017-10-24T10:54:51.559162 #36468] DEBUG -- Main Rack: call start
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Main Rack: initialize start
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Main Rack: initialize end
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Html Rack: initialize start
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Html Rack: initialize end
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Login Rack: initialize start
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Login Rack: initialize end
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Download Rack: initialize start
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Download Rack: initialize end
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Upload Rack: initialize start
D, [2017-10-26T15:48:24.882546 #4236] DEBUG -- Upload Rack: initialize end
D, [2017-10-26T15:48:24.898187 #4236] DEBUG -- Docs Rack: initialize start
D, [2017-10-26T15:48:24.898691 #4236] DEBUG -- Docs Rack: initialize end
D, [2017-10-26T15:48:24.899223 #4236] DEBUG -- Text Rack: initialize start
D, [2017-10-26T15:48:24.899223 #4236] DEBUG -- Text Rack: initialize end
```

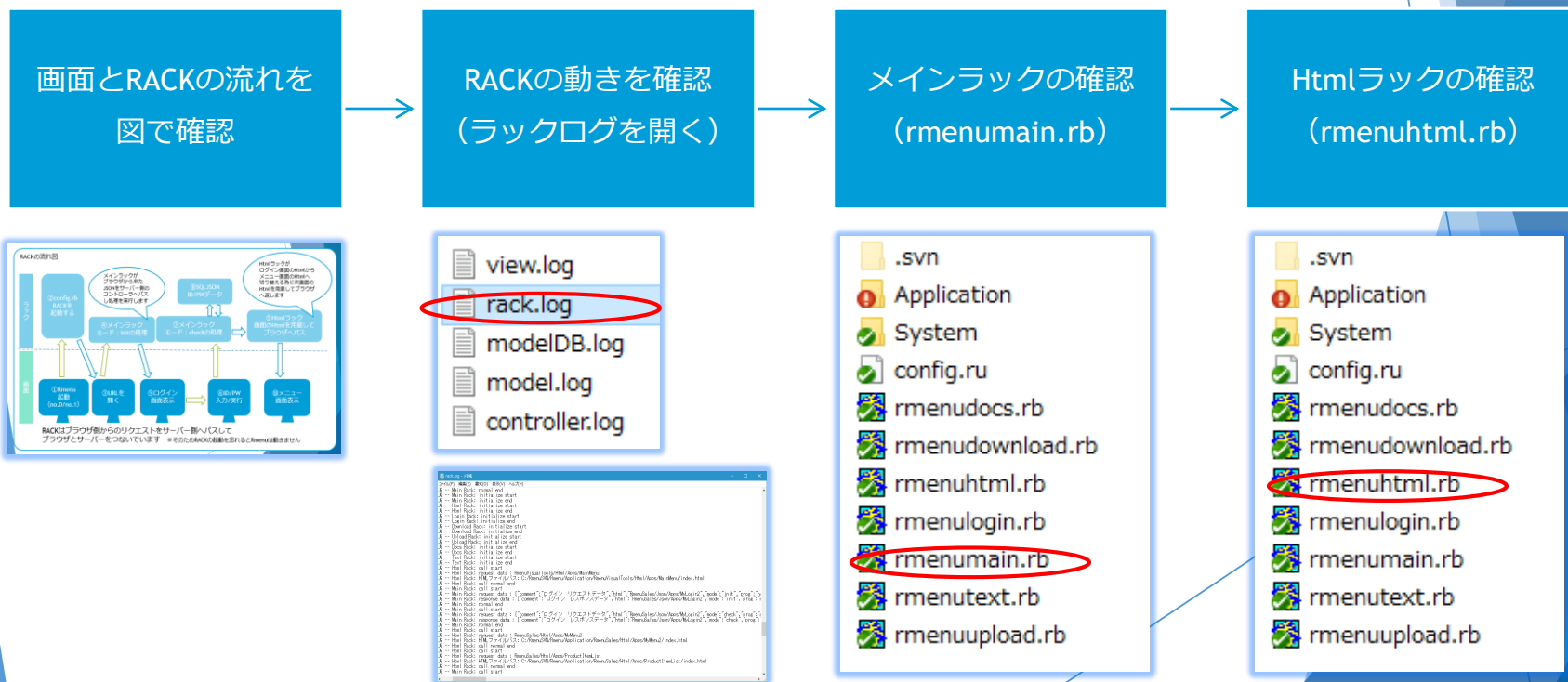
メインラックから順番
にスタート・エンドが
セットになってる状態
「正常にWEBサーバー
が起動しました」

次は、ログイン画面からログインし、メニュー画面を開くまでの
RACKの流れを確認します

RACKの詳細

▶ RACKの流れ (ログイン起動時)

ログイン時のRACKの流れを確認します
URLを開くとログイン画面が表示され、ID/PWを入力し、
メニュー画面が開くまでを『画面』と『RACK』を一緒に追いかけます

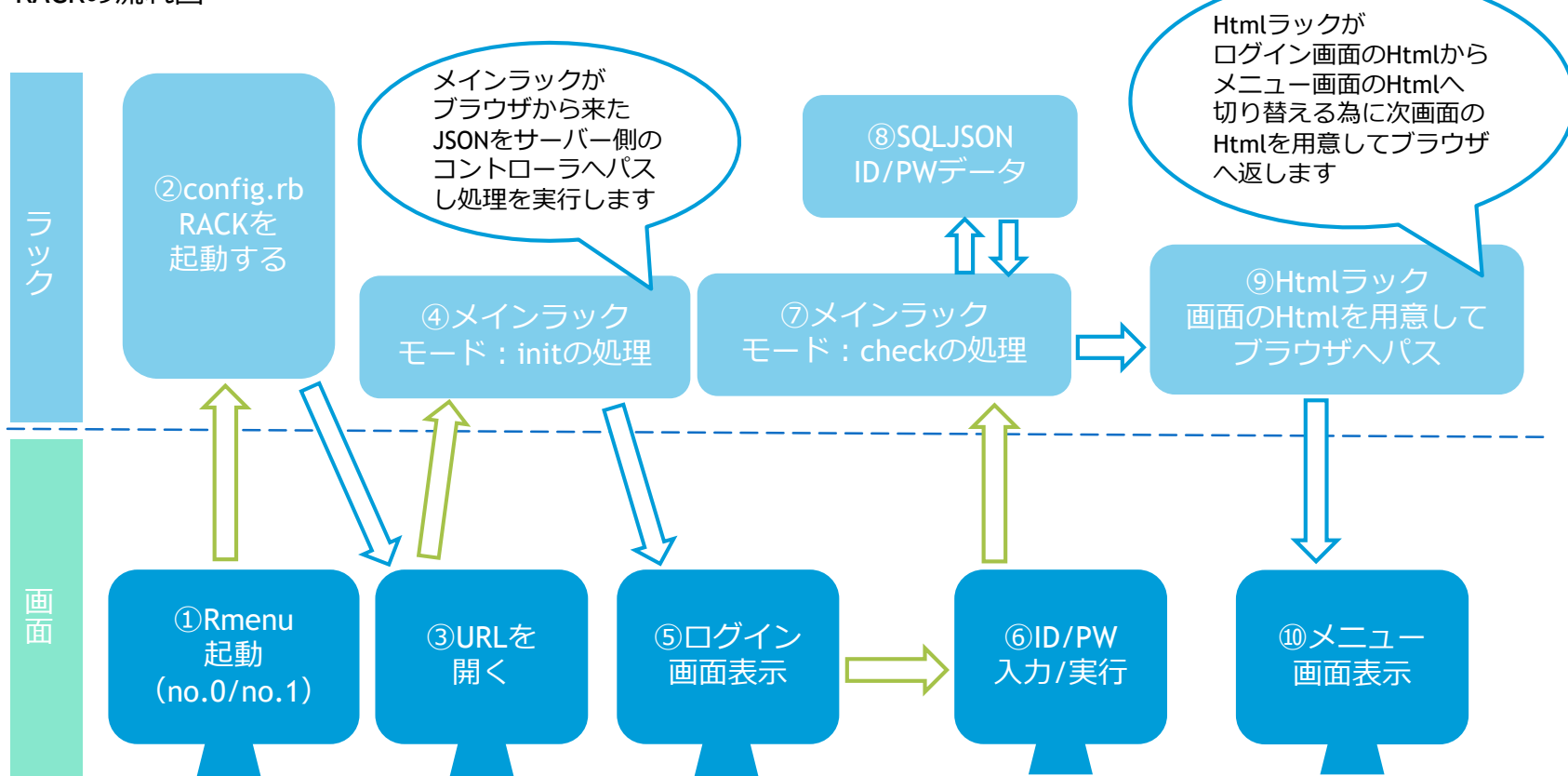


RACKの詳細

▶ RACKの流れ（ログイン起動時）

『画面側』と『RACKの動き』を図で確認します

RACKの流れ図



RACKはブラウザ側からのリクエストをサーバー側へパスしてブラウザとサーバーをつないでいます ※そのためRACKの起動を忘れるとRmenuは動きません

RACKの詳細

▶ RACKの流れ (Rmenuログイン時)

ラックログでRACKの動きを確認

The screenshot shows a log window titled "rack.log - メモ帳" with the following content:

```
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
JG -- Main Rack: normal end
JG -- Main Rack: initialize start
JG -- Main Rack: initialize end
JG -- Html Rack: initialize start
JG -- Html Rack: initialize end
JG -- Login Rack: initialize start
JG -- Login Rack: initialize end
JG -- Download Rack: initialize start
JG -- Download Rack: initialize end
JG -- Upload Rack: initialize start
JG -- Upload Rack: initialize end
JG -- Docs Rack: initialize start
JG -- Docs Rack: initialize end
JG -- Text Rack: initialize start
JG -- Text Rack: initialize end
JG -- Html Rack: call start
JG -- Html Rack: request data : RmenuSales/Json/Apps/MyLogin2/index.html
JG -- Html Rack: HTMLファイルパス : C:/RmenuSVN/Rmenu/Application/RmenuSales/Html/Apps/MyLogin2/index.html
JG -- Html Rack: call normal end
JG -- Main Rack: call start
JG -- Main Rack: request data : {"comment":"ログイン リクエストデータ","html":"RmenuSales/Json/Apps/MyLogin2","mode":"init","prog":"n
JG -- Main Rack: response data : {"comment":"ログイン レスポンスデータ","html":"RmenuSales/Json/Apps/MyLogin2","mode":"init","prog":"
JG -- Main Rack: normal end
JG -- Main Rack: call start
JG -- Main Rack: request data : {"comment":"ログイン リクエストデータ","html":"RmenuSales/Json/Apps/MyLogin2","mode":"check","prog":"
JG -- Main Rack: response data : {"comment":"ログイン レスポンスデータ","html":"RmenuSales/Json/Apps/MyLogin2","mode":"check","prog":"
JG -- Main Rack: normal end
JG -- Html Rack: call start
JG -- Html Rack: request data : RmenuSales/Html/Apps/MyMenu2
JG -- Html Rack: HTMLファイルパス : C:/RmenuSVN/Rmenu/Application/RmenuSales/Html/Apps/MyMenu2/index.html
JG -- Html Rack: call normal end
JG -- Html Rack: call start
JG -- Html Rack: request data : RmenuSales/Html/Apps/ProductItemList
JG -- Html Rack: HTMLファイルパス : C:/RmenuSVN/Rmenu/Application/RmenuSales/Html/Apps/ProductItemList/index.html
JG -- Html Rack: call normal end
JG -- Main Rack: call start
JG -- Main Rack: request data : {"comment":"ログイン リクエストデータ","html":"RmenuSales/Json/Apps/MyLogin2","mode":"init","prog":"n
JG -- Main Rack: response data : {"comment":"ログイン レスポンスデータ","html":"RmenuSales/Json/Apps/MyLogin2","mode":"init","prog":"
```

Annotations and callouts:

- Rmenu起動時で確認した部分**: Points to the initialization phase of the Main Rack.
- ここからログイン画面を表示する流れ**: Points to the start of the HTML Rack call.
- リクエストデータがJSONなのでメインラックがパスしている**: Points to the JSON request data in the log.
- ⑤ログイン画面表示**: Points to the first screenshot of the login page.
- ⑥ID/PW入力/実行**: Points to the input fields in the login page.
- ⑩メニュー画面表示**: Points to the second screenshot of the menu page.
- ここから次の画面を表示**: Points to the end of the HTML Rack call.
- リクエストデータがHtmlなのでHtmlラックがパスしている (メニュー画面には処理がないのでHtmlで終わり)**: Points to the HTML request data for the menu page.

RACKの詳細

▶ RACKの流れ (Rmenuログイン時)

メインラック確認 (役割: プログラムを使用するJSONをブラウザ側から受け取りサーバー側のコントローラへパスする)

```
#!/coding:UTF-8+
require 'rubygems'
require 'rack'
require 'json'
require 'drb'
require 'RmenuConfig'
require 'RmenuLoggerMixin'

class RmenuMain
  include RmenuLoggerMixin

  def initialize
    #_controller用ログを作成する+
    $Rlog = createLogger("rack")
  end

  #_controller用ログを作成する+
  $Rlog.debug("Main.Rack") {"initialize.start"}

  #_コントローラオブジェクトを参照する+
  @rmenu_controller = DRbObject.new_with_uri($Rconfig["controller_uri"])
  $Rlog.debug("Main.Rack") {"initialize.end"}
end

def call(env)
  begin
    $Rlog.debug("Main.Rack") {"call.start"}

    #_ブラウザからリクエストデータを受信する+
    req = Rack::Request.new(env)
    req_data = req.params["data"]

    #_json文字列をHashに変換する+
    request_data = JSON.load(req_data)

    #_開発支援プロジェクトはログを出力しない+
    $Rlog = changeLoggerMode($Rlog, request_data)

    $Rlog.debug("Main.Rack") {"request.data: #{req_data}"}

    #_コントローラオブジェクトのcallメソッドを実行する+
    response_data = @rmenu_controller.call(request_data)

    #_レスポンスデータ (hashオブジェクト) をJSON文字列に変換する+
    content = JSON.dump(response_data)
    $Rlog.debug("Main.Rack") {"response.data: #{content}"}

  rescue
    $Rlog.debug("Main.Rack") {"normal.end"}
  end

  #_エラーログを出力する+
  $Rlog.debug("Main.Rack") {"exception: #{$_}"}

  #_エラーメッセージを作成する+
  error_data = setErrorMessage()

  #_エラーデータ (hashオブジェクト) をJSON文字列に変換する+
  content = JSON.dump(error_data)
  $Rlog.debug("Main.Rack") {"abnormal.end"}
end

#_ブラウザへレスポンスデータを送信する+
res = Rack::Response.new([], r)
res.status = 200
res.r["Content-Type"] = "application/json;charset=utf-8"
res.write content

res.finish
end
[EOF]
```

Require (リクワイア)

@rmenu_controller = DRbObject.new_with_uri(\$Rconfig["controller_uri"])
①コントローラの手続きが書かれている
メインラックはここを見てコントローラへJSONをパスする

def call(env)
② (env) へ①が入るcallで呼び出す

req_data = req.params["data"]
③ブラウザからリクエストJSONを受け取る (リクエストデータが入っているトランJSONごと読んでいる)

request_data = JSON.load(req_data)
④JSONを変換する連想配列に変換する

response_data = @rmenu_controller.call(request_data)
④コントローラへリクエストJSONを渡す

content = JSON.dump(response_data)
⑤レスポンスデータが戻ってくる (JSONダンプがテキストに直す)

r["Content-Type"] = 'application/json;charset=utf-8'
r.write content
⑥utf8 (日本語を使う) にチャセットしてブラウザへ送信

※メインラックの役割の追加
上記のJSONの受け渡しともう一つ、
JSONを連想配列・テキストに変換しwebサーバーを通過させています 詳細は「連想配列とは」を参照してください

RACKの詳細

▶ RACKの流れ (Rmenuログイン時)

Htmlラック確認

```
#!/usr/bin/env ruby
# coding: UTF-8 ↓
require 'rubygems' ↓
require 'rack' ↓
require 'RmenuConfig' ↓
require 'RmenuLoggerMixin' ↓

class RmenuHtml ↓
  include RmenuLoggerMixin ↓
  def initialize() ↓
    # controller用ログを作成する ↓
    $Rlog.createLogger("rack") ↓
    # controller用ログを作成する ↓
    $Rlog.debug("Html_Rack") {"initialize.start"} ↓
    $Rlog.debug("Html_Rack") {"initialize.end"} ↓
  end ↓
  def call(env) ↓
    begin ↓
      $Rlog.debug("Html_Rack") {"call.start"} ↓
      # ブラウザからリクエストデータを受信する ↓
      req = Rack::Request.new(env) ↓
      req_data = req.params["game"] ↓
      $Rlog.debug("Html_Rack") {"request_data: #{req_data}"} ↓
      url = $Rconfig["base_application"] + "/" + req_data + "/index.html" ↓
      $Rlog.debug("Html_Rack") {"HTMLファイルパス: #{url}"} ↓
      content = readHtml(url) ↓
      # マルチユーザ環境の場合、ページ埋め込みのリンクを書き換える ↓
      # リンク先頭の「./」(たくさん)を「ユーザー名」に書き換え ↓
      if $Rconfig["multi_user_name"] != "" ↓
        conv_pattern = "%./%.*%" * (System::Application)("%.*%") ↓
        conv_replace = "/" + $Rconfig["multi_user_name"] + "%2F3" ↓
        content.gsub!(conv_pattern, conv_replace) ↓
      end ↓
      $Rlog.debug("Html_Rack") {"call.normal.end"} ↓
    rescue ↓
      # エラーログを出力する ↓
      $Rlog.debug("Html_Rack") {"exception: #{$!}"} ↓
      $Rlog.debug("Html_Rack") {"exception: #{$@}"} ↓
      # エラーメッセージを作成する ↓
      content = setErrorMessage() ↓
      $Rlog.debug("Html_Rack") {"call.abnormal.end"} ↓
    end ↓
    # ブラウザへレスポンスデータを送信する ↓
    res = Rack::Response.new([], |r| ↓
      r.status = 200 ↓
      r["Content-Type"] = "text/html; charset=utf-8" ↓
      r.write content ↓
    ) ↓
    res.finish ↓
  end ↓
end ↓
[EOF]
```

Require (リクワイア)

url = \$Rconfig["base_application"] + "/" + req_data + "/index.html"
① どこを読み込むか書いている

content = readHtml(url)
② Htmlを読み込む

r.write content
③ ブラウザ側へ送る

RACKの詳細 完了

RACKの詳細はRmenuがどんな感じで動いているのか
なんとなくイメージができればOK
作成時に特に使うことはありません

ラックログは見方を覚えると作業が早いです
プログラム作成時のデバックに必要です

Rmenu基礎知識

名前空間とは

「名前空間」

「ブラウザメモリー」

「セッションストレージ」

名前空間とは

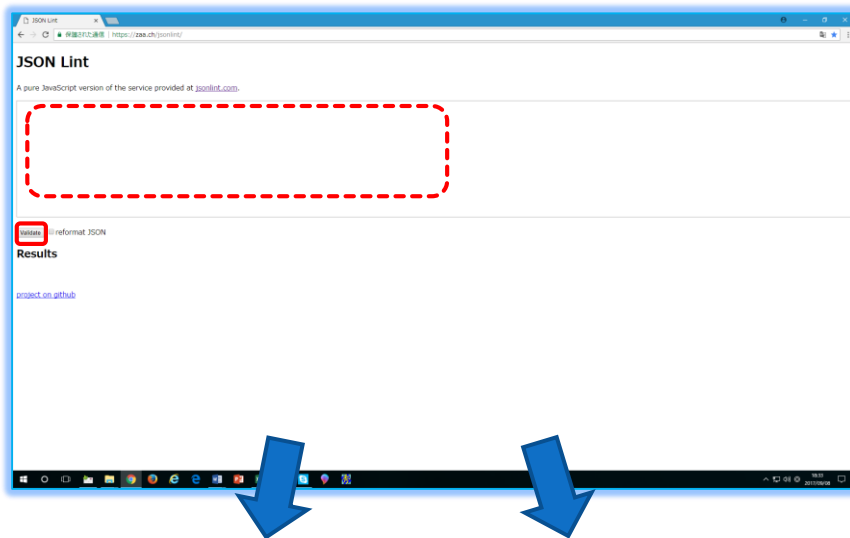
- ▶ Rmenuではセッションストレージに使う

JSONlintの使い方

▶ RmenuのプログラミングとはJSONを作成することです

JSONを作成できたら必ずJSONLintを行い、文法の誤りがないかチェックを行います

※JSONのプログラムを作成したときは必ず実行すること



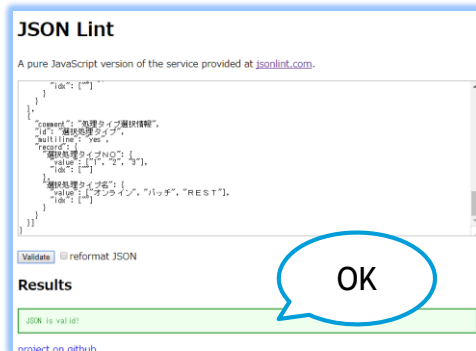
①JSONリントを開く

URL <https://zaa.ch/jsonlint/>

②点線の空欄へ作成したJSONをALLコピーし貼り付ける

③赤枠のボタンをクリックする

④結果が表示されるので、OKが出るまで修正をして正しい文法でJSONを作成すること



オープンLDAPとは

▶ LDAPとは

LDAP = ディレクトリーDBへアクセスするためのプロトコルのこと

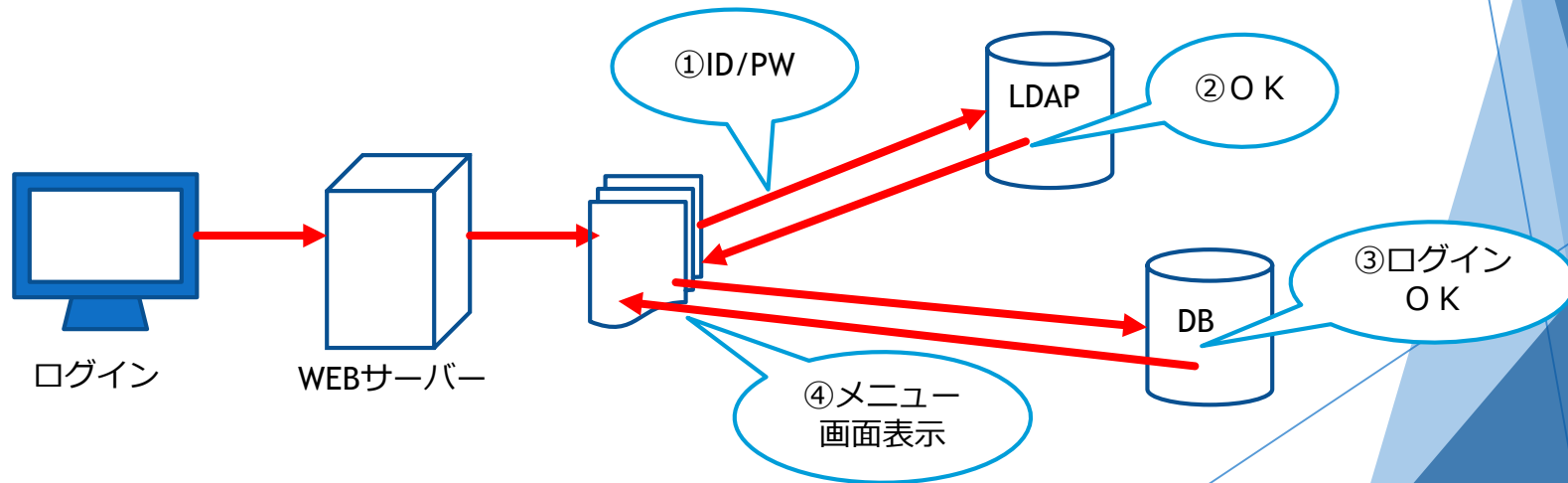
RmenuではプロジェクトのログインにLDAPを使用することがあります

LDAPはRmenuで利用しながら、同時に他のシステムでも利用でき

1つのID/PWで複数のシステムにログインすることが可能です

LDAPとはID/PWのみが入っているDBのことで大きなプロジェクトで使うことが多い。

(PWが暗号化されるので安全なため)



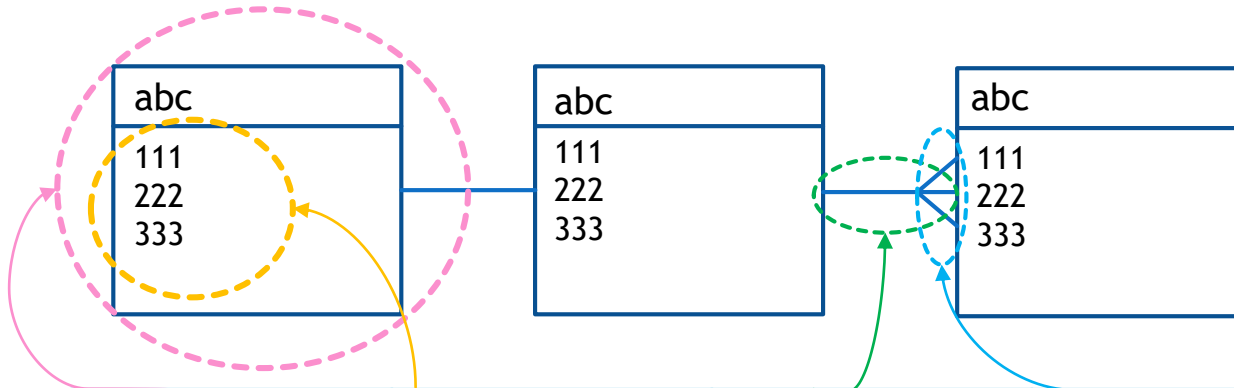
ER図とは

- ER図とはデータベース設計で使う設計方法（データモデリング）

エンティティ リレーション

ER図 『Entity Relgtinal』 = 『テーブル 関係』

Rmenuでは主に4パターンある ER図を見ると画面のパターンも決まる



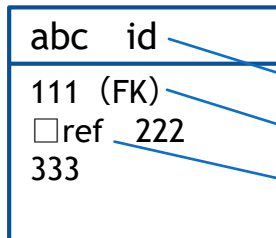
テーブル	レコード/属性	つなぐ線	つなぎ目
エンティティ	アトリビュート	リレーション/関連	カーディナリティ

枝分かれした書き方をIE記法という

【IE記法】

○	ゼロ
	1
∟	多/以上

【アトリビュートの見方】

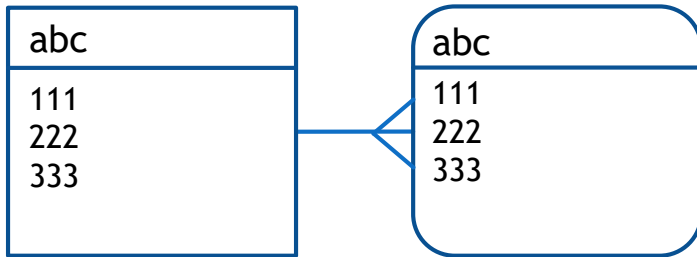


id	主キー
(FK)	外部キー
□ref	他のテーブルから持ってくる（マスターから）

ER図とは

- ▶ ER図とはデータベース設計で使う設計方法（データモデリング）

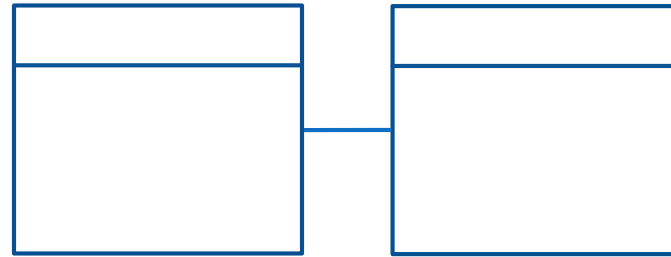
依存リレーションシップ



親エンティティ

子エンティティ

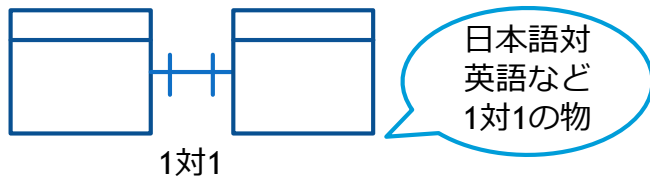
非依存リレーションシップ



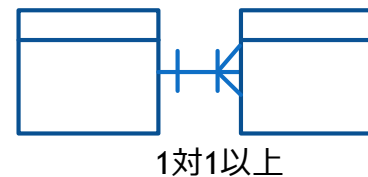
非依存

非依存

カーディナリティの見分け方



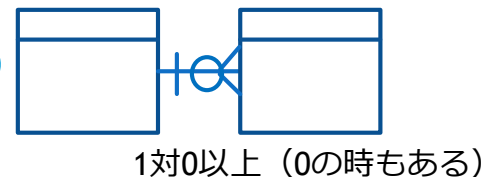
1対1



1対1以上



1対多（あいまい増加あり）



1対0以上（0の時もある）

ER図とは

▶ ER図と画面のパターン

パターン①一覧画面/選択画面

The screenshot shows a web application interface with a table of items. The table has columns for 'ヘッダID', 'ヘッダ項目1', 'ヘッダ項目2', 'ヘッダ項目3', 'ヘッダ項目4', 'ヘッダ項目5', 'サンプル項目1', and '詳細情報'. The first row contains the following data: '1', 'ヘッダ項目1.1', 'ヘッダ項目1.2', 'ヘッダ項目1.3', 'ヘッダ項目1.4', 'ヘッダ項目1.5', 'サンプル項目1.1', and '1'. The table is scrollable and has a search bar at the top.

パターン②登録画面/入力画面

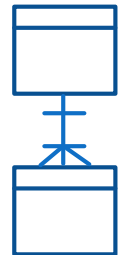
The screenshot shows a web application interface for a registration form. The form has a header with 'サンプル親 メンテナンス (新規)' and a date '平成29年10月16日 (月)'. The form contains several input fields labeled 'ヘッダID', 'ヘッダ項目1', 'ヘッダ項目2', 'ヘッダ項目3', 'ヘッダ項目4', and 'ヘッダ項目5'. There is a 'サンプル検索' button at the bottom.

パターン③スクロールタイプ

The screenshot shows a web application interface with a scrollable table. The table has columns for '項目', 'コード', '登録番号', 'ログインID', 'パスワード', 'メールアドレス', 'メニュー名', and '権限'. The first row contains the following data: '1', 'テスト1', 'user1', 'user1', 'test@example.com', 'メニュー1', and '10'. The table is scrollable and has a search bar at the top.

パターン④1件と一覧を同時表示

The screenshot shows a web application interface with a registration form and a table. The form is at the top and has the same fields as in pattern 2. Below the form is a table with columns for '項目', '項目詳細', 'サンプル項目1', 'サンプル項目2', 'サンプル項目3', 'サンプル項目4', 'サンプル項目5', and '詳細情報'. The first row contains the following data: '1', 'サンプル項目1.1', 'サンプル項目1.2', 'サンプル項目1.3', 'サンプル項目1.4', 'サンプル項目1.5', 'サンプル項目1.6', and '1'. The table is scrollable and has a search bar at the top.



DBのコピー作成

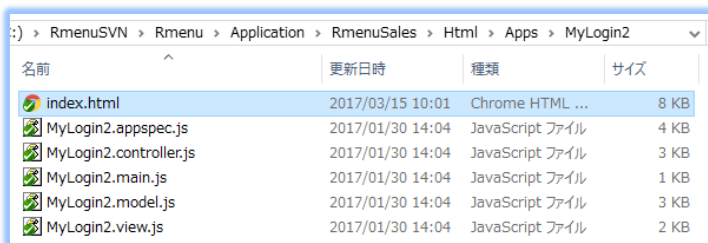
プロジェクトのコピー作成

プログラムのコピー作成

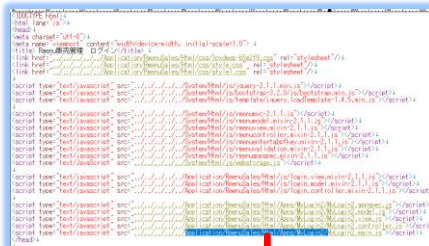
URLの調べ方

備考：URLが不明の時の調べ方

①開きたいプロジェクトのHtmlを開き、
[ログインプログラムのHtml] を開く



②Html上部からURL を作成するために、ファイルの場所をコピー
します（青部分の [Appli~プログラム名/] まで）



Application/RmenuSales/Html/Apps/MyLogin2/MyLogin2.controller.js
Application/RmenuSales/Html/Apps/MyLogin2/MyLogin2.main.js

③ブラウザを開きURLを作成します

ローカル環境で接続する為前に [ローカル] を足す [コピー] を貼り付け、後ろへ [index.html] を足し検索で完了

