# 簡易・軽量な Web アプリ方式

2021 年 1 月 8 日

## 内容

# 1. これまで

Struts が登場して以来、Web アプリは「Java で MVC フレームワーク」が主流になっている。MVC は Model、View、Controller に機能分割してアプリケーションを開発することで各種のメリット（開発速度、保守性...etc.）を得ようとするものだった。

# 2. MVC の難しさ

## 2.1. これまでの MVC は業務システムの構築には向いていない

View：以下の問題がある

① HTML、CSS や操作用の JavaScript 等、業務処理関係以外の情報が大量に必要

② 一つの資材（JSP 等）を出力と入力で共用するため解り難い。表示確認ができない

Model：以下の複数の機能を一つの実装とするため、保守性が落ちる（モジュール強度が弱）

・View の入力データチェック

・View の出力編集用データ準備

・DB 更新等の業務処理

## 2.2. 画面が決まらないと View／Model の開発に着手できない

デザイナーが HTML で画面を作成し、AP 開発者がデータ編集用のタグや画面効果用の JavaScript を仕込んで View に変換（JSP、xhtml、Thymeleaf 等の作成）する。悪いことに画面はユーザの興味が集中し易く、後続行程へのリリースが遅れることが多い。

# 3. 環境的な背景

・Oracle JDK の 2019 年有償化や JaveEE.8 からの Eclipse Foundation 移管に伴うパッケージ名の変更（javax⇒jakarta）等、Java を使い続けることへの不安がある。

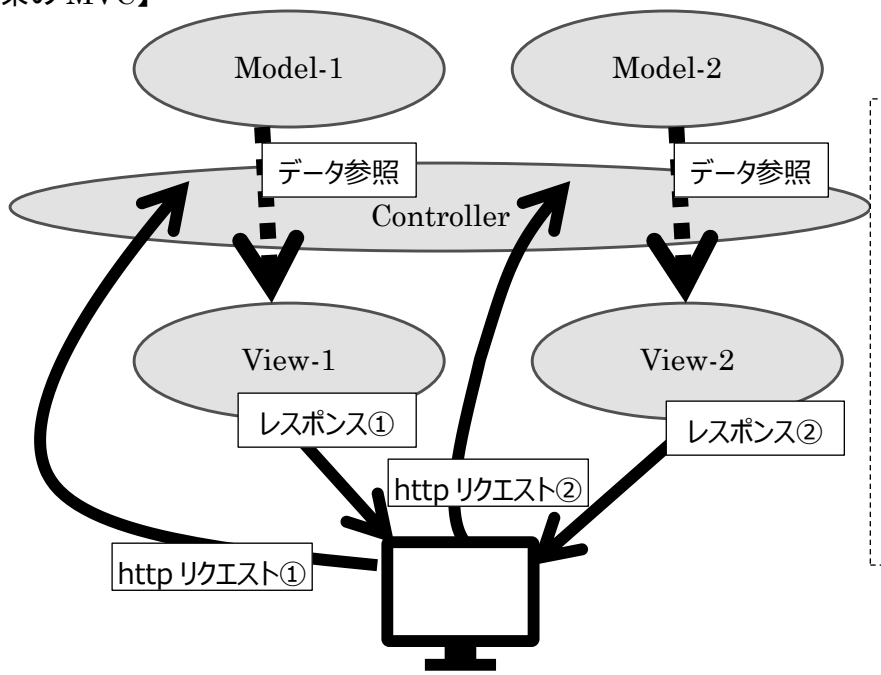
・ブラウザ間の差異の吸収のため jQuery 等のライブラリが普及したが、廃止・変更 API への対処が困難になってきた（AP、ブラウザ、ライブラリのバージョン合わせが困難）。

・JavaScript が ESMAScript として標準化が進み、主要なブラウザ（IE を除く）が同一コードで動作するようになった（特に Ver.2016 以降の ajax 用 API 等）。

## 4. React と Servlet の JSON を使った連携

V と C を React[1]に置き換えることで、MVC の塊を疎な関係に作り替えることができる。

【従来の MVC】

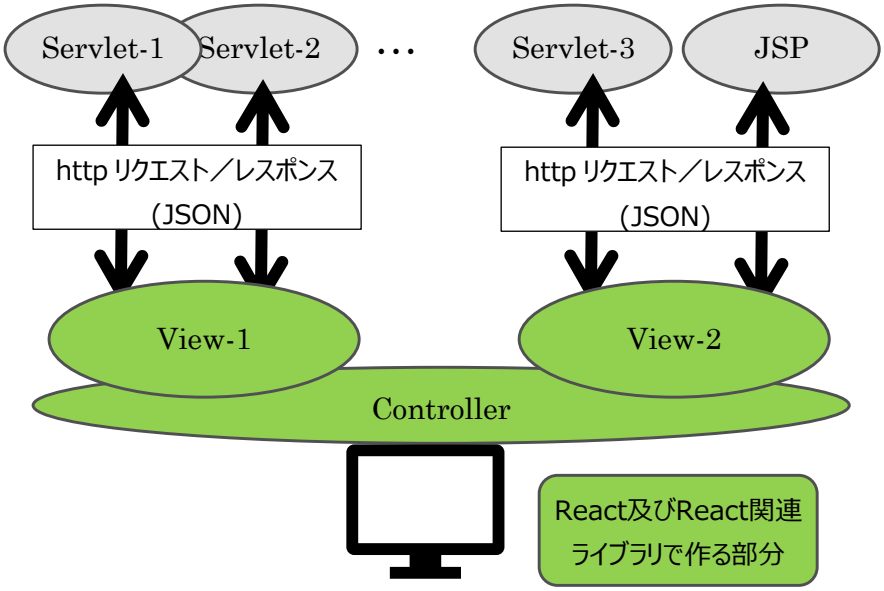



・リクエストで POST されたデータの割当先、レスポンスを作成する View は Controller が決める。

※*Controller* は通常、*Servlet* として実装されている

・View は Model の処理結果を編集してレスポンスを作る（View と Model は取扱うデータの関係で通常 1:1 になる）

【React を使用】




・View は React のコンポーネントを使って作る

・View 及び Controller は、クライアントとして動作する

・Servlet/JSP は MVC の Model を使いやすい粒度に分割し、ajax で呼び出す

・Servlet 側は Http/JSON をインタフェースに使える相手なら全てクライアントになり得る（Excel 等の View も可能）

---

[1] Facebook 社が開発し MIT ライセンスで公開されている「ユーザインターフェースを作成する為の JavaScript のライブラリ」https://ja.reactjs.org/docs/getting-started.html

類似のライブラリ Vue, Angular と比較し下記 URL (2020/11/01)の人気度から React を選択した。
https://trends.google.co.jp/trends/explore?q=%2Fm%2F012l1vxv,%2Fm%2F0j45p7w,%2Fg%2F11c0vmgx5d&geo=,,&date=today%205-y,today%205-y,today%205-y&cat=13#TIMESERIES

# 5. React、JSON を使う利点・難点

## 5.1. 利点

| ポイント | 利点 |
|---|---|
| 開発 | デザイナーが従来 HTML で作っていた画面を React で作った場合、後工程で AP 開発者が弄る必要がない |
| | 画面の入力と出力の処理を別個の Servlet として実装可能 |
| | インタフェース（JSON）の変更があっても、Servlet への影響は少ない<br>…MVC の場合はインタフェースが Model の構造に影響する |
| テスト | 画面が直ぐ動く（確認が楽） |
| | バックエンド（Servlet）は curl コマンド等の簡易なツールで動作やインタフェースの確認ができる |
| その他 | インタフェース（JSON）の構造が決まれば**画面の開発とサーバ側の開発が平行して行える**。画面の外観凍結を待つ必要がない |
| | http と JSON が扱えるツールであれば、ブラウザ以外をクライアントとしたシステム構成が可能 |

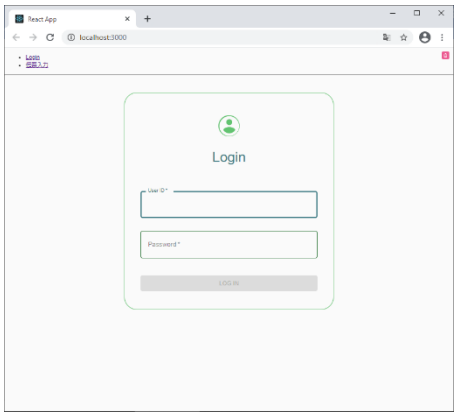
## 5.2. 難点

Servlet は Http リクエストを受け取り Http レスポンスを返すが、想定しているリクエスト／レスポンスの内容は JSON 形式ではない。このため、Servlet 側で JSON を取り扱うための処理が必要になる。

# 6. 実装例

## 6.1. 画面構成

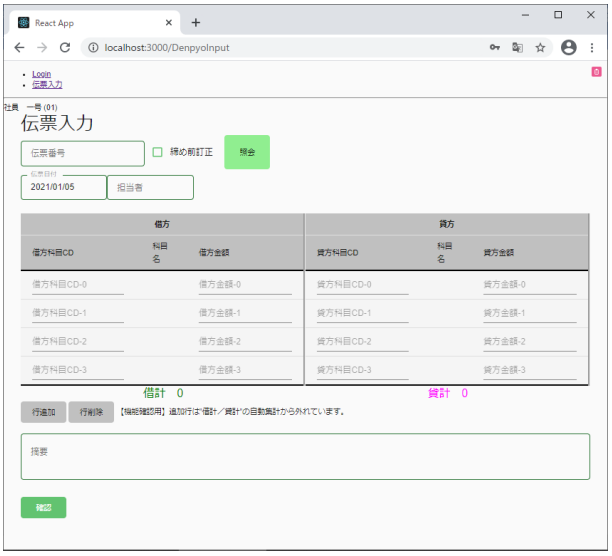
### ① ログイン画面




●ユーザ ID とパスワードで認証を行う

認証はサーバ側で行い、成否を画面側に返している。

### ② 伝票入力画面




●伝票の内容を入力する

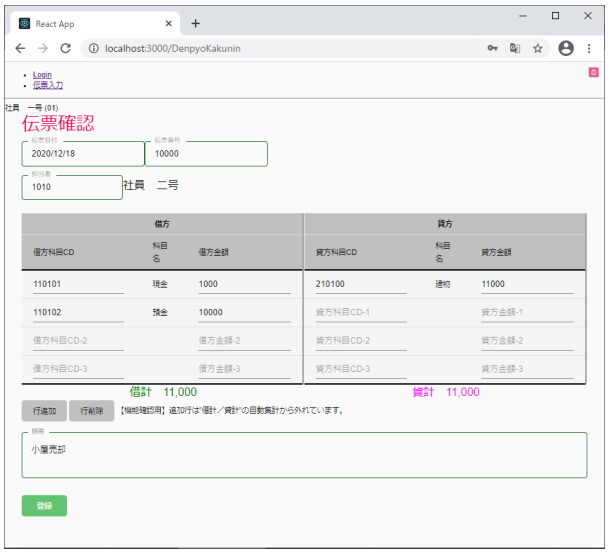
・「照会」ボタンで登録済の伝票を表示する

・「担当者」と「科目」は入力値のキーインと並行して ajax で名称を取得する

・「確認」ボタンでサーバ側にデータを送りエラーがなければ伝票確認画面に遷移する

※項目の必須チェック、属性チェック等は画面内でも行っておりエラーがあると「確認」ボタンで送信できない

### ③ 伝票確認画面




●伝票の登録依頼をサーバに送信する

・伝票入力画面で入力した値を読取り専用で表示し、「登録」ボタン押下でサーバ側にデータを送って DB への登録を行う

## 6.2. React 関連ライブラリと共通機能

### （１）React 関連ライブラリ

実装には、React 本体の他に以下のライブラリを利用（主なもの）している。

| No. | ライブラリ名 | 用途 |
|---|---|---|
| 1 | react-hook-form | ・HTML の form に書かれる項目の値／状態を管理する<br>・項目単位のチェック（必須、属性...）とエラー処理 |
| 2 | react-router-dom | 画面遷移を管理する(Controller) |
| 3 | react-table | Table（伝票の明細行部分）を管理する |
| 4 | material-ui | 画面部品（コンポーネント）のスタイルを管理する |

### （２）画面共通機能

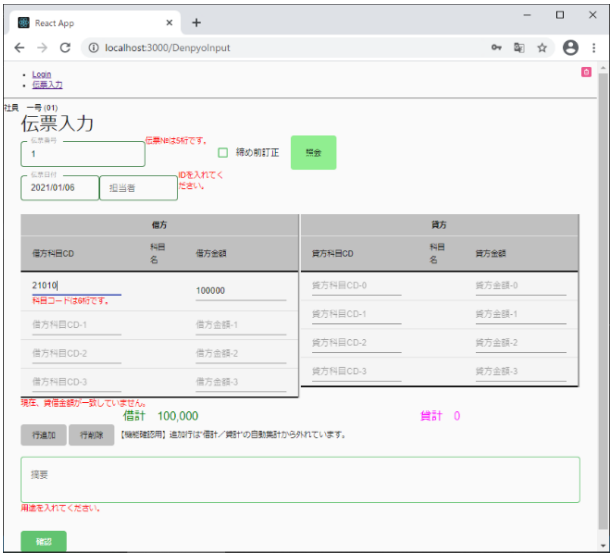
① 各画面の上部にはリンクを表示しており、認証済であれば直接 リンク先画面に遷移できる
② 認証済であれば一般のブラウザ操作（alt+矢印等）で画面遷移ができる

※上記は react-router-dom の機能を使用している

### （３）エラー処理

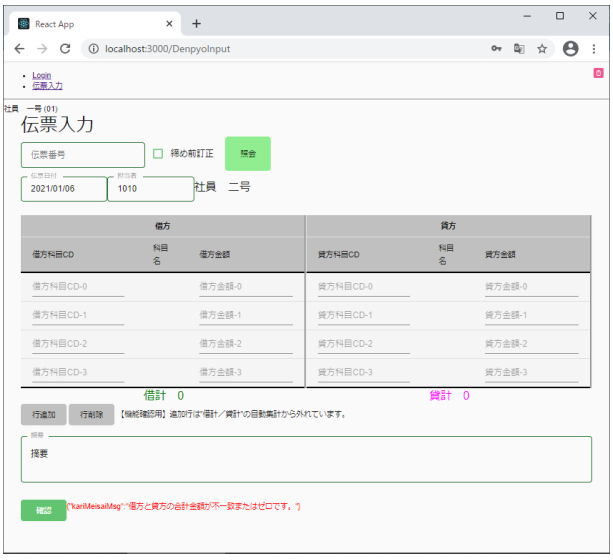
エラーチェックは、画面とサーバで行っている。

①画面でのチェック




必須、形式等のチェックを行いエラーメッセージを項目傍に出力

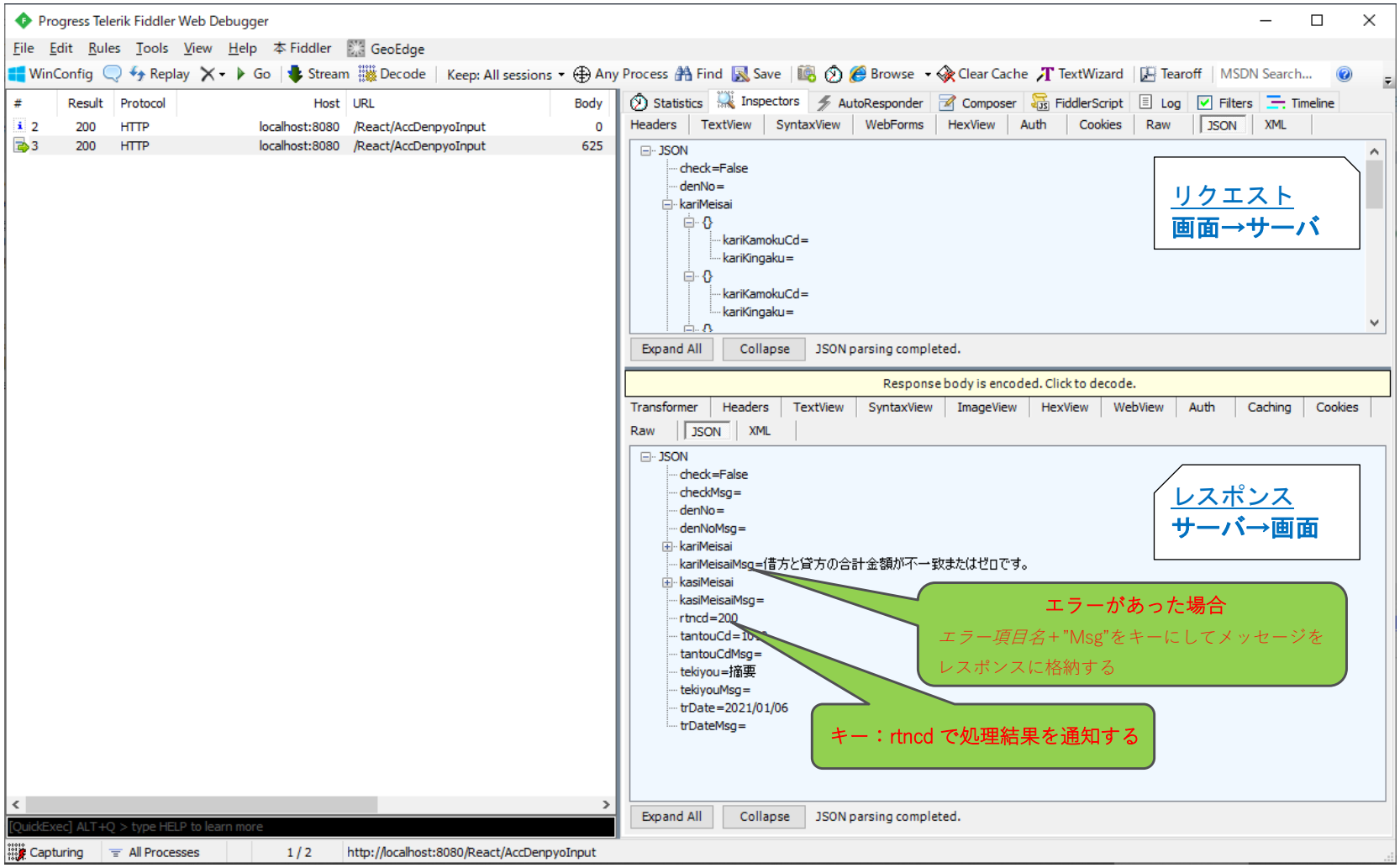
②サーバでのチェック




サーバから返ってきたエラーメッセージを、契機になったボタン傍に表示

## 6.3. Http リクエスト／レスポンス

画面とサーバーで交換する電文のイメージ




※レスポンスにリクエストに入っている項目を全て含めているが、画面側で使用しているのは rtncd と項目名～Msg のみ。

## 6.4. サーバ側の構成

サーバ側は JavaEE の AP サーバ（TomEE v8）上に JDK 1.8 で作成した。メッセージを処理している Servlet と JSP は以下のものである。他に DB アクセスや認証を管理するクラス等があるが割愛する。また、Http リクエスト／レスポンスの処理や JSON の加工を行う基底クラスについては別で説明する。

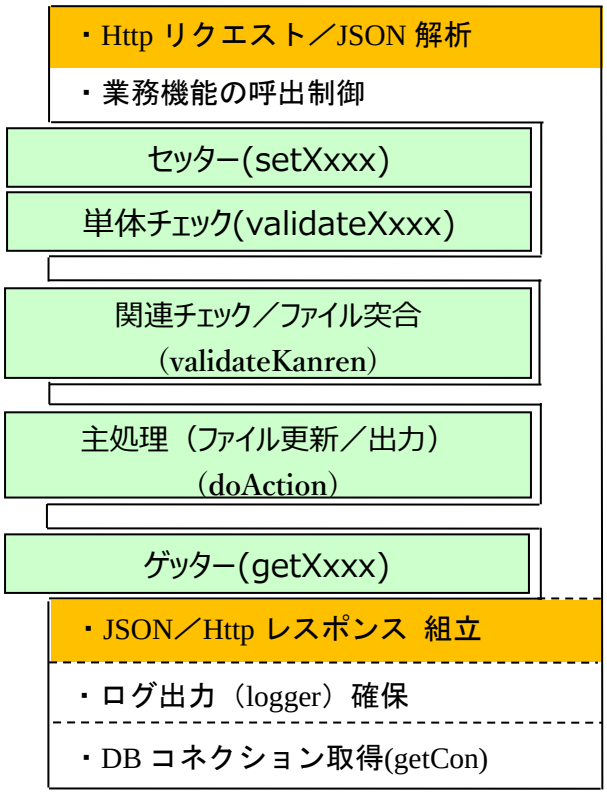
| No. | 資材名 | 内容 |
|---|---|---|
| 1 | LoginBean.java | 「ログイン」サーブレット<br>セッション情報を作成し、認証情報（担当者名、役職）を保存する。認証情報を画面側に返す |
| 2 | AccDenpyoInquiryBean.java | 「照会」サーブレット<br>伝票番号を受け取り、伝票の内容を画面に返す |
| 3 | AccDenpyoInputBean.java | 「伝票入力」サーブレット<br>画面入力内容を受け取り、内容のチェックを行う。エラーの有無はリターンコードとエラーメッセージで返す |
| 4 | AccDenpyoKakuninBean.java | 「確認」サーブレット<br>伝票入力画面と同一のチェック後、エラーが無ければ DB に登録する |
| 5 | AccGetTantouName.jsp | 「担当者名取得」JSP<br>担当者コードを受け取り(http/GET)、担当者名を JSON 形式で返す |
| 6 | AccGetKamokuName.jsp | 「勘定科目名取得」JSP<br>科目コードを受け取り(http/GET)、科目名を JSON 形式で返す |

※資材名に”Bean”とついているものは実際は Servlet で JavaBean ではない。
（MVC／JSF2 フレームワーク用に作成したものを改造し、対比させるため名前を残した）
※資材名の拡張子が jsp になっているものは通常では HTML でレスポンスを作成するが、この実装例では JSON 形式で返すように実装している。

## 6.5. 基底クラス

MVC フレームワーク用に作成したクラスで不足するのが JSON を扱う機能である。この不足している機能を Servlet の基底クラスに作りこんだ例を説明する。

（１）構成イメージ




・セッターはメソッド名=「set 項目名」、単体チェックはメソッド名=「validate~項目名」とし、http リクエストから取り出した項目名に一致するメソッドを呼出す
・単体チェックは戻り値として、エラーがなければ０、エラーは１を返すこと（全項目実行）

・関連チェックは、メソッド名=「validateKanren」で引数を持たない。戻り値は単体チェックと同（前段でエラーがあったら実行しない）

・ファイル更新は、メソッド名=「doAction」で引数なし（前段でエラーがあったら実行しない）

・logger は業務機能からは既に使用可能な状態になっている（環境設定は env.properties ファイルを参照して行うのでパス、サイクル等を指定しておく）

・DB コネクションは基底クラスの getCon(DB 識別)メソッドを呼出せばよい（env.properties ファイルに DB 識別の情報を記述しておくこと）

凡例  各 Servlet で実装が必要な部分（処理が不要なら宣言を省略してもよい）
※メソッド名で使う項目名は先頭を大文字にして使う ⇒ setKariMeisai

（２）動作概要

基底クラスはHttpリクエストを受信（HttpServlet::doPostを実装）し、以下を行う。

① リクエストを JSON-P(外部 Lib: javax.json-1.1.4)を使って JSON オブジェクトに変換
② 上記の JSON オブジェクトの階層／配列を Map の階層／List に変換する
③ Map の最上位の KeySet を順次取り出し、自インスタンス(this)に該当する名前のメソッドが存在したらセッターと単体チェックを呼び出す
④ 上記でエラーがなければ関連チェックと主処理のメソッドを順に呼び出す
⑤ 自インスタンスの public で宣言された全てのゲッターを呼び出し、レスポンス用 Map に格納する。合わせて認証成否、チェック結果・実行結果を rtncd として追加格納する
⑥ レスポンス用 Map の階層／List を JSON 化し、Servlet のレスポンスとして送出する

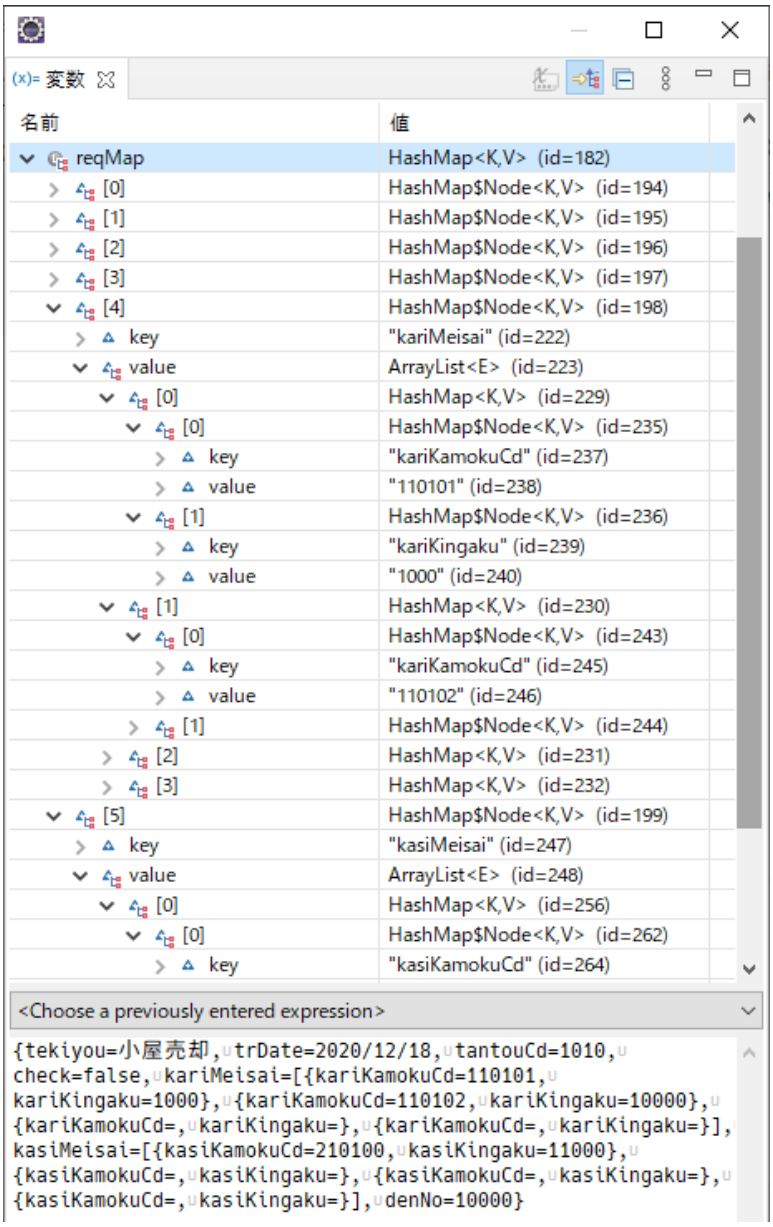
## 6.6. セッター、単体チェックの引数（JSON / Map）

画面側から送った JSON（Http リクエスト・ボディ）

＜JSON＞


```
{"denNo":"10000"
,"check":false
,"trDate":"2020/12/18"
,"tantouCd":"1010"
,"kariMeisai":[
    {"kariKamokuCd":"110101"
    ,"kariKingaku":"1000"}
    ,{"kariKamokuCd":"110102"
    ,"kariKingaku":"10000"}
    ,{"kariKamokuCd":""
    ,"kariKingaku":""}
    ,{"kariKamokuCd":""
    ,"kariKingaku":""}
]
,"kasiMeisai":[
    {"kasiKamokuCd":"210100"
    ,"kasiKingaku":"11000"}
    ,{"kasiKamokuCd":""
    ,"kasiKingaku":""}
    ,{"kasiKamokuCd":""
    ,"kasiKingaku":""}
    ,{"kasiKamokuCd":""
    ,"kasiKingaku":""}
]
,"tekiyou":"小屋売却"}
```


＜Map＞




※セッター、単体チェックの実装単位は Map の最上位の KeySet なので、以下のメソッドの引数は Map の List となる。

- setKariMeisai
- validateKariMeisai
- setKasiMeisai
- validateKasiMeisai

## 6.7. 単体チェック／関連チェックの実装例

Map の List を引数とする単体チェック及び関連チェックの例を以下に示す。

### （１） 単体チェック例

```
/**
 * kariMeisaiの単体チェックを行います。
 * @return チェック結果を戻します（0:正常、1:エラー有）。
 */
@SuppressWarnings("unchecked")
    public int validate1KariMeisai(List<Object> karimeisai) {
    kariMeisaiMsg = "";
    logger.log(Level.INFO, "validate1kariMeisai:" + kariMeisai.toString());
    //金額の10進数チェック
    for (Object row: karimeisai) {
            try {
                    new BigDecimal( ((Map<String, String>)row).get("kariKingaku") );
            }catch(NumberFormatException e){
                    kariMeisaiMsg = "借方金額に数値以外が指定されています。";
                    return 1;
            }
    }
    return 0;
}
```

### （２） 関連チェック例

```
/**
 * 関連チェックを行います。
 * @return チェック結果を戻します（0:正常、1:エラー有）。
 */
public int validateKanren() {
    logger.log(Level.INFO, "AccDenpyoInputBean validateKanren");
    /* 合計金額チェック */
    //借明細の各行(Map)から借金額(kariKingaku)を取り出し[stream().map()]て10進数に変換後、集計する
    BigDecimal karikei = kariMeisai.stream()
        .map(p->( new BigDecimal( ((Map<String, String>)p).get("kariKingaku") ) ) )
        .reduce(BigDecimal.ZERO, BigDecimal::add);
    //貸明細
    BigDecimal kasikei = kasiMeisai.stream()     //各行に対して
        .map(p->( new BigDecimal( ((Map<String, String>)p).get("kasiKingaku") ) ) )//貸金額を10進数
        .reduce(BigDecimal.ZERO, BigDecimal::add);        //初期値0から各明細を集計
    if (karikei.compareTo(kasikei)!=0 || karikei==BigDecimal.ZERO) {
        kariMeisaiMsg = "借方と貸方の合計金額が不一致またはゼロです。";
        return 1;
    }
    logger.log(Level.INFO, "伝票番号："+denNo+",借方合計金額："+karikei+",貸方合計金額："+kasikei);
     ＜以下略＞
```

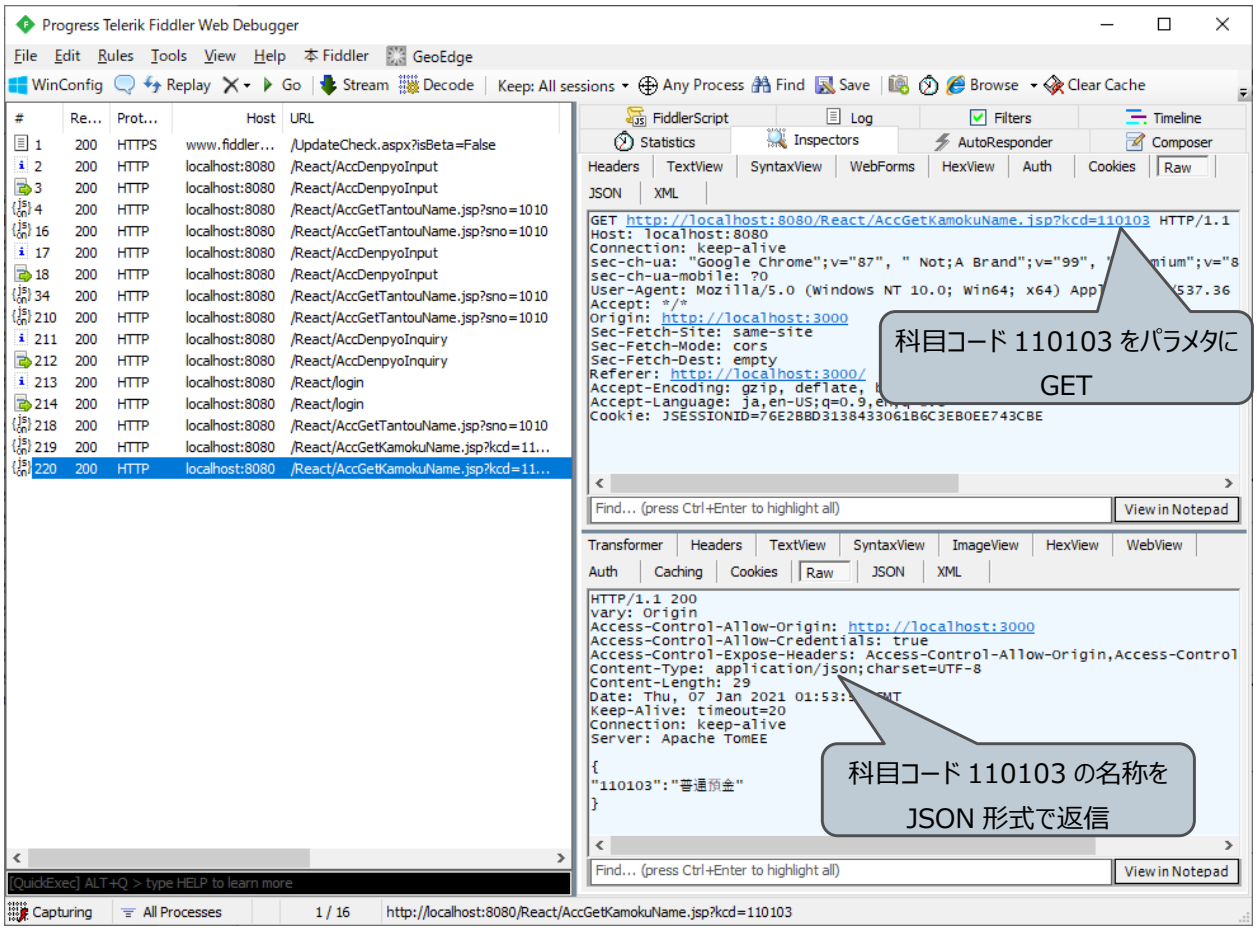
## 6.8. JSP の実装例

JSP を使って JSON を返信する例を以下に示す。

勘定科目名取得<AccGetKamokuName.jsp>

```
<%@ page contentType='application/json; charset=UTF-8' pageEncoding='UTF-8' %>
<%@ page trimDirectiveWhitespaces="true"%>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/sql' prefix='sql' %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/functions' prefix="fn"%>

<c:set var="kcd" value="${param['kcd']}" scope="session"/>
<c:if test="${fn:length(kcd) > 0}">
<%-- データソース（データベース接続先）定義 --%>
<sql:setDataSource dataSource="jdbc/postgres" var="ds" />
<%-- 勘定科目取得 この先で余計な改行コード・タブコード・空白が混ざるとJSONが壊れるためツメツメで書く --%>
<sql:query var="rs" dataSource="${ds}">
SELECT K.KAMOKUNM as knm
        ,K.KAMOKUCD as kcd
        FROM M_KAMOKU K
        WHERE K.KAMOKUCD = ?
        <sql:param value="${kcd}"/>
</sql:query>{
<c:forEach var="row" items="${rs.rows}" varStatus="st">"${row.kcd}":"${row.knm}"</c:forEach>
}</c:if>
```



## 7. 留意点

① 本資料の事例は React とサーバ間のインタフェースがどの程度適合するかを目的にしており、運用面の確認は済んでいない

② React 及び関連ライブラリのエンハンスは続いており、仕様変更があるかもしれない

③ 事例が少ないため、性能や機能が開発対象のシステムに適合するかを小規模な開発から確かめたほうがよい

以上