

Spring/Web サービス (アノテーションによる実装) とクライアント

内容

はじめに	1
1. Web サービスとは	1
2. 開発環境	2
3. プロジェクトの作成	2
4. 作成するコード	3
5. Web サービスの起動	10
6. Web サービスの呼び出し	11
7. その他のクライアントアプリ	14
7.1. React	14
7.2. Excel/VBA による実装例	15

Spring/Web サービス (アノテーションによる実装) とクライアント

はじめに

国産汎用機メーカーが OS、DB/DC 等ミドルウェア(以下、MW)のエンハンスを止めるという話を聞いたのは 20 世紀終わりの頃でした。Unix サーバを中核にしたネットワークシステムに移行し、オンライン処理はクライアント/サーバと TP モニター(Tuxedo、Encina 等)で実現しようとしたのですが日本語の情報が少ない上に当時のクライアント PC は非力で信頼性が低く、全国的なシステムで成功した事例は聞いたことがありません (2000 年近辺に地方銀行でトライした事例は有名です)。全国的なシステムが実現できたのはクライアントを VisualBasic や Delphi (初期 Ver. は 20 世紀末頃の出荷) で開発し、バックエンドに Oracle 等の RDBMS を使う構成ができてからです。ただこれもクライアントが肥大化するにつれて全国の端末の一斉バージョンアップ (以下、VUP) が困難になり、更に VB ランタイムの VUP や当時の OS (Windows NT) のサポート終了でお荷物扱いとなります。

以上のような背景があり、J2EE が発表 (1999/12) されると数年で『Web アプリケーション/MVC フレームワーク』に関心が集中しました。Web アプリケーション (以下、Web アプリ) はクライアントにブラウザだけあればよいので、クライアント側の問題の多くが解決されました。ただ、ブラウザは文章 (html) 表示用のアプリのため、それまでのクライアントアプリに比べると見た目はチープで操作性も劣るうえ、出力を html に編集しなければならないというアプリ開発の負担があります。

日本で Web アプリやフレームワークの Struts の入門書が書店に並び出だした頃、アメリカでは既に Web アプリは下火で Web サービスの開発が中心だというコラムが専門誌に載りました。不特定多数を対象とする消費者向けのシステムではブラウザだけあればよい Web アプリに頼らざるを得ませんが、社内向けのシステムや Web アプリの業務ロジックの実装では Web サービスを使うと簡便で拡張性が高くなります。

1. Web サービスとは

Web アプリはサーブレット/JSP の仕様が JCP (Java Community Process) で JSR (Java specification requests) として明確に定義されていますが、Web サービスの定義は W3C のサイト (<https://www.w3.org/TR/ws-arch/>) に以下のようにやや曖昧に記述されています (W3C ワーキンググループノート 2004 年 2 月 11 日版)。

『Web サービスは、さまざまなプラットフォームやフレームワークで実行される、さまざまなソフトウェアアプリケーション間で相互運用するための標準的な手段を提供します。

--中略--

このアーキテクチャーは、Web サービスの実装方法を指定しようとはせず、Web サービスの組み合わせ方法に制限を課しません。』

Web サービスの実装方法は複数あります。当初は Web サービスと言えば SOAP (Simple Object Access Protocol) でしたが、SOAP は XML や WSDL (Web Services Description Language) の知識が必要で、名前ほど Simple な代物ではありません。

以降では、よりシンプルな RESTful Web サービスを Spring を使って実装する例を説明します。この RESTful Web サービスは RESTful な (http セッション情報を使わない) Web アプリをベースに、MVC モデルの View を html から JSON に変えたものです。

Spring/Web サービス (アノテーションによる実装) とクライアント

2. 開発環境

(1) 使用ツール

- ① 製造からテスト迄を Eclipse ベースの STS(Spring Tool Suite)から JDK11 を使って行います。

<https://spring.io/tools>

- ② データの保存/検索

DBMS に PostgreSQL を使います。

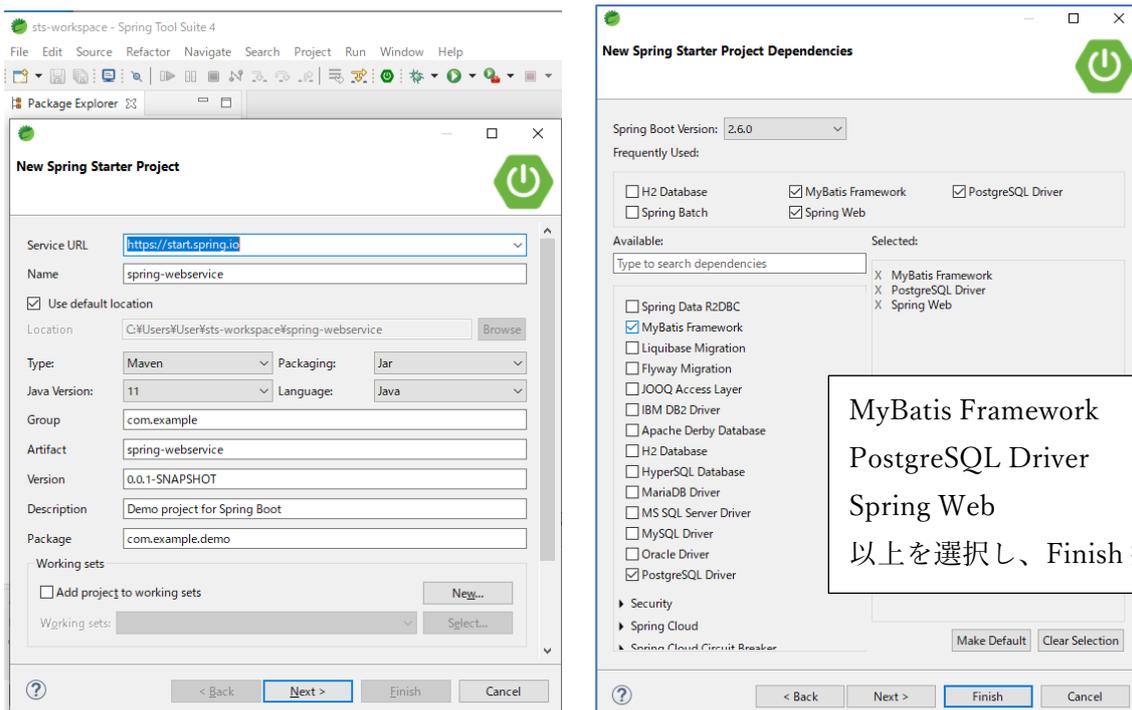
<https://www.postgresql.org/download/>

(2) 使用テーブル DDL

```
CREATE TABLE denpyo(  
    err          bool  
    , trdate     CHAR(8) NOT NULL  
    , denpyono   CHAR(5)  
    , tantou     CHAR(4)  
    , karikamokud CHAR(6) []  
    , karikingaku VARCHAR(13) []  
    , kasikamokud CHAR(6) []  
    , kasikingaku VARCHAR(13) []  
    , tekiyo     VARCHAR(50)  
    , sysdate    CHAR(8) NOT NULL  
    , errmessage VARCHAR(100)  
    , CONSTRAINT pk_denpyo PRIMARY KEY (denpyono, tantou)  
);
```

3. プロジェクトの作成

STS の Project Explorer を右クリックし、New > Spring Starter Project を実行。または、Project Explorer が初期状態で、Create a project... が表示されている場合はこのリンクをクリックする。

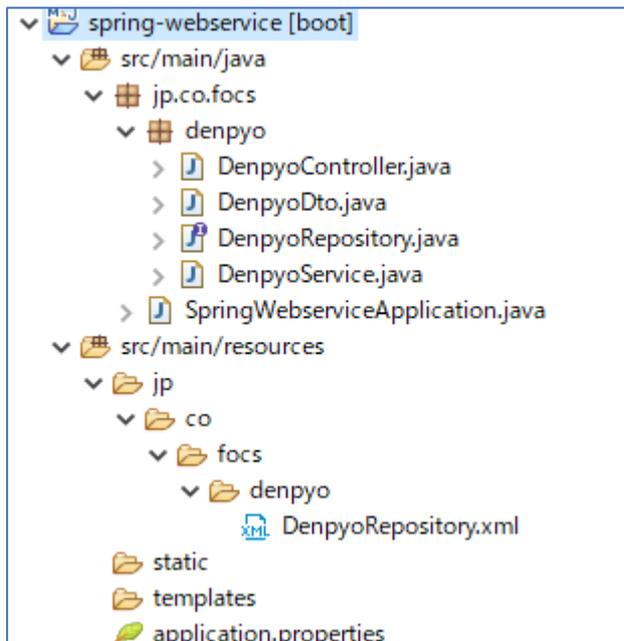


Spring/Web サービス (アノテーションによる実装) とクライアント

4. 作成するコード

STS Var.4.11 を使って前項のデフォルト手順 (Spring Starter Project で Packaging: Jar 指定) でプロジェクトを作ると Tomcat を内包した Java アプリケーション (以下、Java アプリ) として構成されます。Packaging を War にすればそのまま別の Web コンテナにデプロイする形式 (Web アプリ) に変更できます (Java アプリの方が STS を使った試験に便利なのでデフォルトのままにします)。

アノテーションを使って Web サービスを作る場合、最低限作成が必要な資材は以下で全てです。



Web サービスとして必須なのは、

- SpringWebserviceApplication
- DenpyoContorller

の 2 本です。

(1) SpringWebserviceApplication.java¹

Java アプリとして起動するための main メソッドを宣言し、@SpringBootApplication アノテーションを付けています。package 以外は変更をする必要ありません。

```
package jp.co.focs;
```

```
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication  
public class SpringWebserviceApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(SpringWebserviceApplication.class, args);  
    }  
  
}
```

¹ <https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.spring-application>

Spring/Web サービス (アノテーションによる実装) とクライアント

(2) コントローラ (@RestController)

http リクエストを受け、http メソッド(GET,POST...)とパスにより処理を振り分けます。
クラスに @RestController アノテーションを付け、@GetMapping や @PostMapping を付けたメソッドの戻り型に Dto (ゲッターでデータを外部に提供できるクラス) を指定すると JSON 形式の http レスポンスが作られます。

```
package jp.co.focs.denpyo;

import java.util.List;
import javax.inject.Inject;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
/**
 * @author User
 *
 */
@RestController
public class DenpyoController {
    @Inject//⑥
    DenpyoRepository denpyoRep;
    @Inject//⑤-2
    DenpyoService denpyoService;

    //①伝票問合せ param 応答 JSON (引数に@RequestParam で Http リクエストパラメータを処理)
    @GetMapping("/denpyo/query")
    public List<DenpyoDto> query(@RequestParam(value = "denNo", defaultValue = "") String denNo) {
        if (denNo.equals("")) {
            return denpyoRep.findAll();
        } else {
            return denpyoRep.findDenpyo(denNo);
        }
    }

    //②伝票登録 受信、送信とも JSON
    @PostMapping("/denpyo/post")
    @Transactional(rollbackFor=Exception.class) //③なにか例外が発生したら RollBack
    //④
    public ResponseEntity<DenpyoDto> post(@RequestBody DenpyoDto denpyo) {
        DenpyoDto newDenpyo = //⑤-1
            //new DenpyoService().insertDenpyo(denpyo);×
            denpyoService.insertDenpyo(denpyo);
        return new ResponseEntity<DenpyoDto>(newDenpyo
            , newDenpyo.getErr()? HttpStatus.NOT_ACCEPTABLE : HttpStatus.CREATED);
    }
}
```

Spring/Web サービス (アノテーションによる実装) とクライアント

① `@GetMapping(url パス)`を付けることで、`url` パスに対する GET リクエストがこのメソッドで処理されるようになります。また、メソッドの引数に`@RequestParam(value= "denNo", defaultValue = "")`と書くことでリクエストパラメータを引数で受け取ることができ、更に"denNo"が指定されていないときは長さ 0 の文字列が指定されたように取り扱うことができます。

例: `http://localhost:8080/denpyo/query` は `http://localhost:8080/denpyo/query?denNo=""`と同じ

※`@GetMapping` は`@RequestMapping(method = RequestMethod.GET)`と同義です…以下同様

② `@PostMapping(url パス)`を付けることで、`url` パスに対する POST リクエストがこのメソッドで処理されるようになります。また、メソッドの引数に`@RequestBody`と書くことでリクエストボディを引数で受け取ることができ、引数型に指定した Dto (セッターでデータを受け取るクラス) に値が設定された状態になっています。

③ `@Transaction` が付いているメソッドはメソッドからリターンするまでが 1 トランザクションで処理され、例外が発生した場合はロールバックされます。

※【注意】`@Transaction` を付ける対象のメソッドは `public` である必要があります²

④ `HttpStatus(200,400 等の結果ステータス)`を自分で設定したい場合は戻り型を `ResponseEntity` にし、コンストラクタでステータスを設定します(戻り値の Dto の JSON 化は自動で行われます)。

<Spring の DI に関する注意>

⑤ Spring の DI 機能 (`@Inject`、`@Autowired`) を使っているクラス (この例では `denpyoService`) を `new` 演算子でインスタンス化^{⑤-1}すると注入される側のクラス (`denpyoService`) 内の`@Inject`が働かず参照が `Null` になります (Spring が介入できず、Inject されないようです)。

※`@Inject` が記述されているクラスは、`@Inject` により^{⑤-2}使います

⑥ MyBatis を使う場合は `Mapper` (DB アクセス用のメソッドを定義したインターフェース) を Bean 化するために`@MapperScan(basePackages = "Mapper パッケージ名")`を`@SpringBootApplication` (main メソッド) が書かれたクラスに書くか、`Mapper` に`@Mapper` アノテーションを書きます。

² <https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html#transaction-declarative-annotations-method-visibility>

Spring/Web サービス (アノテーションによる実装) とクライアント

(3) サービス (@Service)

コントローラから呼び出されて Http メソッドの業務処理を行います。@GetMapping と同様にコントローラ内で処理を完結しても機能的には変わりありません。

```
package jp.co.focs.denpyo;

import javax.inject.Inject;
import org.springframework.dao.DuplicateKeyException;
import org.springframework.stereotype.Service;
/**
 * @author User
 *
 */
@Service
public class DenpyoService {
    @Inject //①
    DenpyoRepository denpyoRep;

    public DenpyoDto insertDenpyo(DenpyoDto denpyo) {
        //denpyo 内容チェック
        denpyo.setErr(false);
        validateDenpyo(denpyo);

        if (denpyo.getErr()) { ;
        } else {
            try { //登録
                denpyoRep.insertDenpyo(denpyo); //②
            } catch(DuplicateKeyException e) {
                denpyo.setErr(true);
                denpyo.setErrmessage(e.getMessage());
            }
        }
        return denpyo;
    }

    /**
     * リクエスト内容チェック
     * @param denpyo
     */
    private void validateDenpyo(DenpyoDto denpyo) {
        // TODO validate etc.
        if (!denpyo.getDenpyono().matches("^\d{5}$")) {
            denpyo.setErr(true);
            denpyo.setErrmessage("伝票番号は数字 5 桁必須です。");
        }
    }
}
```

① コントローラと同様に MyBatis の Mapper インタフェースを注入します

② このメソッドの呼び元 (コントローラの post メソッド) で

@Transactional(rollbackFor=Exception.class)と注記しているので、全ての例外でロールバックが掛かります (この例では更新対象は 1 行/1 テーブルだけなので結果に変化はできません)。

Spring/Web サービス (アノテーションによる実装) とクライアント

(4) Mapper インタフェース (@Mapper) と Mapper xml

MyBatis を使った RDB アクセスはインターフェースを作成し、@Mapper アノテーションを付けます (@Mapper は @MapperScan(basePackages = “Mapper パッケージ名”) と互換)。具象クラスは MyBatis が動的に作成するため、コーディングする必要はありません。

Mapper xml は resources フォルダ配下に Mapper インタフェースのパッケージ名と同様のフォルダ階層を作って格納します。また、配列項目は typeHandler=org.apache.ibatis.type.ArrayTypeHandler を指定します。

【Mapper インタフェース】

```
package jp.co.focs.denpyo;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface DenpyoRepository {
    //Dto×n 件で返す場合は、戻す型を List<Dto 型>にするだけで OK…1 件で戻す型は Dto (xml の方は変更は不要)
    List<DenpyoDto> findAll();
    List<DenpyoDto> findDenpyo(String denNo);
    int insertDenpyo(DenpyoDto denpyo);
}
```

【Mapper xml】

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.co.focs.denpyo.DenpyoRepository">
    <resultMap id="mapDenpyo" type="jp.co.focs.denpyo.DenpyoDto">
        <!-- 配列は TypeHandler の設定が必要 -->
        <result column="karikamokucd" property="karikamokucd" typeHandler="org.apache.ibatis.type.ArrayTypeHandler"
            jdbcType="CHAR" javaType="java.lang.String" />
        <result column="karikingaku" property="karikingaku" typeHandler="org.apache.ibatis.type.ArrayTypeHandler"
            jdbcType="VARCHAR" javaType="java.lang.String" />
        <result column="kasikamokucd" property="kasikamokucd" typeHandler="org.apache.ibatis.type.ArrayTypeHandler"
            jdbcType="CHAR" javaType="java.lang.String" />
        <result column="kasikingaku" property="kasikingaku" typeHandler="org.apache.ibatis.type.ArrayTypeHandler"
            jdbcType="VARCHAR" javaType="java.lang.String" />
    </resultMap>

    <!-- select id="cursor" resultType="jp.co.focs.denpyo.DenpyoDto" -->
    <select id="findAll" resultMap="mapDenpyo">
        SELECT
            err
            , trdate
            , denpyono
            , tantou
            , karikamokucd
            , karikingaku
            , kasikamokucd
            , kasikingaku
            , tekiyo
            , sysdate
            , errmessage
        FROM
            denpyo
    </select>
```

Spring/Web サービス (アノテーションによる実装) とクライアント

```
<select id="findDenpyo" resultMap="mapDenpyo">
    SELECT
        err
        , trdate
        , denpyono
        , tantou
        , karikamokucd
        , karikingaku
        , kasikamokucd
        , kasikingaku
        , tekiyo
        , sysdate
        , errmessage
    FROM
        denpyo
    WHERE
        denpyono = #{denNo}
</select>

<insert id="insertDenpyo" parameterType="jp.co.focs.denpyo.DenpyoDto">
    INSERT INTO denpyo(
        err
        , trdate
        , denpyono
        , tantou
        , karikamokucd
        , karikingaku
        , kasikamokucd
        , kasikingaku
        , tekiyo
        , sysdate
        , errmessage
    )
    VALUES(
        #{err}
        , #{trdate}
        , #{denpyono}
        , #{tantou}
        , #{karikamokucd, typeHandler=org.apache.ibatis.type.ArrayTypeHandler}
        , #{karikingaku, typeHandler=org.apache.ibatis.type.ArrayTypeHandler}
        , #{kasikamokucd, typeHandler=org.apache.ibatis.type.ArrayTypeHandler}
        , #{kasikingaku, typeHandler=org.apache.ibatis.type.ArrayTypeHandler}
        , #{tekiyo}
        , #{sysdate}
        , #{errmessage}
    )
</insert>

<update id="updateDenpyo" parameterType="jp.co.focs.denpyo.DenpyoDto">
    UPDATE
        denpyo
    SET
        err = #{err}
        , errmessage = #{errmessage}
    WHERE
        denpyono = #{denpyono}
</update>
</mapper>
```

Spring/Web サービス (アノテーションによる実装) とクライアント

(5) Dto

データ保持用のクラスです。メソッドにセッターとゲッターがあればよく、所謂 POJO です。
http リクエスト/レスポンスや RDB とのデータ受け渡しはフレームワークが行います。

RDB とやり取りする項目は全てセッターとゲッターを宣言する必要がありますが、それ以外の
(例えば帳票出力用に編集方法を変えた) ゲッター等が宣言されていても問題ありません。

```
package jp.co.focs.denpyo;
```

```
import java.util.Arrays;
```

```
/**
```

```
 * @author User
```

```
 *
```

```
 */
```

```
public class DenpyoDto {
```

```
    private Boolean err;
```

```
    private String trdate;
```

```
    private String denpyono;
```

```
    private String tantou;
```

```
    private String[] karikamokucd;
```

```
    private String[] karikingaku;
```

```
    private String[] kasikamokucd;
```

```
    private String[] kasikingaku;
```

```
    private String tekiyo;
```

```
    private String sysdate;
```

```
    private String errmessage;
```

```
    public Boolean getErr() {
```

```
        return err;
```

```
    }
```

```
    public void setErr(Boolean err) {
```

```
        this.err = err;
```

```
    }
```

(中略)

```
    public String[] getKarikamokucd() { //配列項目ゲッター
```

```
        return karikamokucd;
```

```
    }
```

```
    public void setKarikamokucd(String[] karikamokucd) { //配列項目セッター
```

```
        this.karikamokucd = karikamokucd;
```

```
    }
```

(以下略)

```
}
```



Spring/Web サービス (アノテーションによる実装) とクライアント

(6) application.properties

環境定義を行います。殆どはデフォルトとアノテーションだけで環境設定が可能ですが、RDB等の外部リソースを使う場合は指定が必要です。

#①DataSource

```
spring.application.name=focs-demo
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=postgres
```

#②listening port

```
server.port=9000
server.tomcat.accesslog.enabled=true
```

#③log

```
logging.group.tomcat=org.apache.catalina, org.apache.coyote, org.apache.tomcat
logging.level.tomcat=ERROR
logging.level.root=INFO
logging.level.org.springframework.web=DEBUG
```

① データベースの接続先は、Web サービスと同一 OS 下でデフォルトのポートを使って動作している PostgreSQL を指定しています。username と password はインストール時のものです

② Web サービスが使うポートと、アクセスログを取得するための指定です。アクセスログはデフォルトのままだと、以下の場所に作られます。

C:\Users\ユーザー\AppData\Local\Temp\tomcat.<ポート番号等>

③ ログの出力レベルです

この例では、全体(root)は INFO 以上、但し、tomcat グループのパッケージは ERROR 以上、springframework のパッケージは DEBUG 以上のレベルのログを出力します。

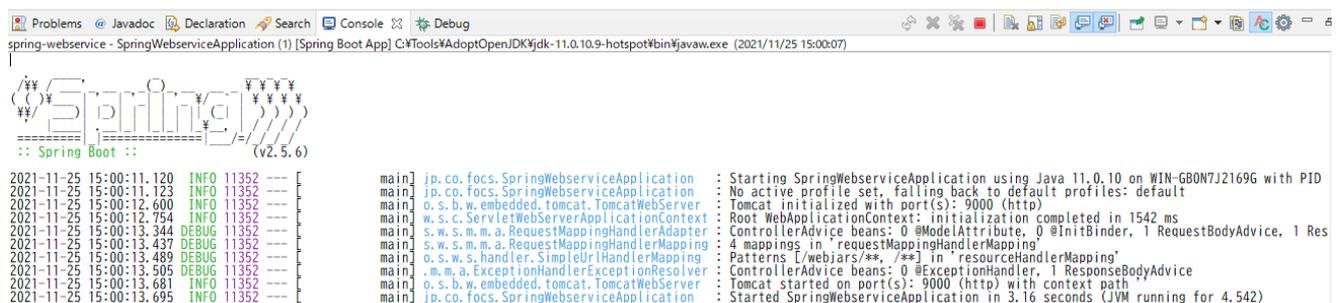
※FATAL < ERROR < WARN < INFO < DEBUG < TRACE の順で詳細になっていきます

5. Web サービスの起動

STS の Package Explorer から起動するプロジェクトを選択し、右クリックで以下の操作をします。

Run As > Spring Boot App

Console ビューに「Started SpringWebServiceApplication in ...」と表示されたら起動完了です。



```
spring-web-service - SpringWebServiceApplication (1) [Spring Boot App] C:\Tools\AdoptOpenJDK\jdk-11.0.10.9-hotspot\bin\javaw.exe (2021/11/25 15:00:07)
:: Spring Boot ::
2021-11-25 15:00:11.120 INFO 11352 --- [main] jp.co.focs.SpringWebServiceApplication : Starting SpringWebServiceApplication using Java 11.0.10 on WIN-GB0N7J2169G with PID
2021-11-25 15:00:11.123 INFO 11352 --- [main] jp.co.focs.SpringWebServiceApplication : No active profile set, falling back to default profiles: default
2021-11-25 15:00:12.600 INFO 11352 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9000 (http)
2021-11-25 15:00:12.754 INFO 11352 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1542 ms
2021-11-25 15:00:13.344 DEBUG 11352 --- [main] s.w.s.m.m.a.RequestMappingHandlerAdapter : ControllerAdvice beans: 0 @ModelAttribute, 0 @InitBinder, 1 RequestBodyAdvice, 1 Res
2021-11-25 15:00:13.437 DEBUG 11352 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : 4 mappings in 'requestMappingHandlerMapping'
2021-11-25 15:00:13.489 DEBUG 11352 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Patterns [/webjars/**, /**] in 'resourceHandlerMapping'
2021-11-25 15:00:13.505 DEBUG 11352 --- [main] .m.m.a.ExceptionHandlerExceptionResolver : ControllerAdvice beans: 0 @ExceptionHandler, 1 ResponseBodyAdvice
2021-11-25 15:00:13.681 INFO 11352 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9000 (http) with context path
2021-11-25 15:00:13.695 INFO 11352 --- [main] jp.co.focs.SpringWebServiceApplication : Started SpringWebServiceApplication in 3.16 seconds (JVM running for 4.542)
```

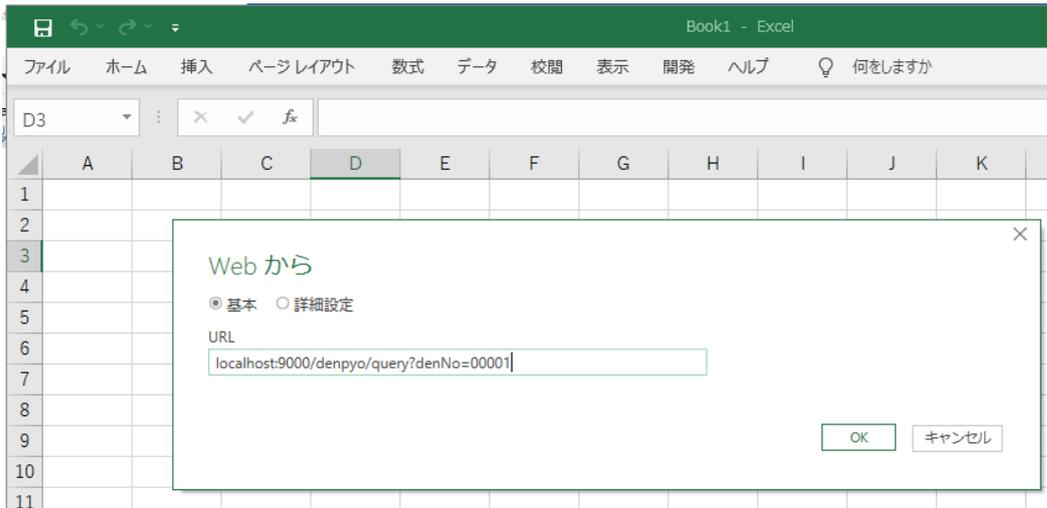
Spring/Web サービス (アノテーションによる実装) とクライアント

6. Web サービスの呼び出し

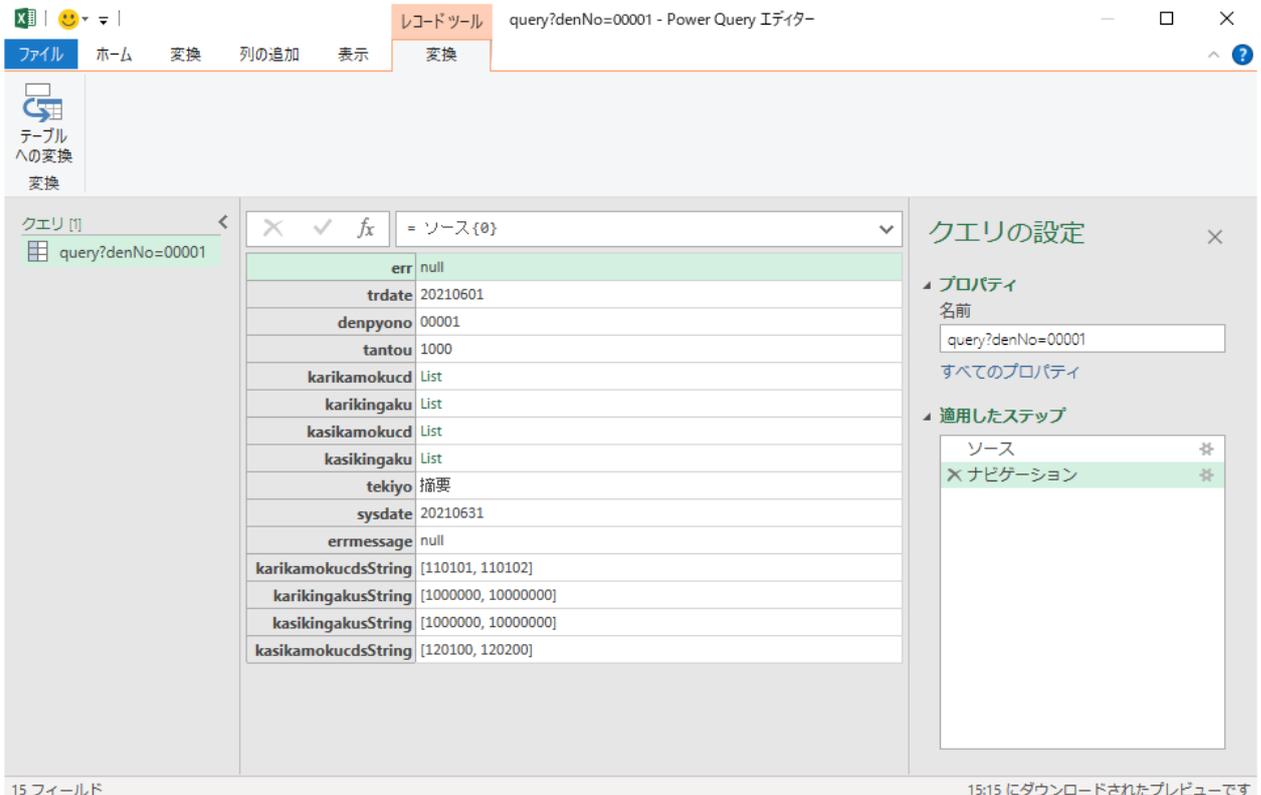
RESTful Web サービスは http のプロトコルを使って JSON データをやり取りするものなので、http と JSON が使えるツール/コマンドから呼び出せます (認証/暗号化が必要な場合を除く)。

(1) Excel

メニューバーの データ > Web から を選択し、Web サービスの起動 URL とコントローラで指定した@GetMapping(/denpyo/query)のパス、@RequestParam(value = "denNo")の value を入力して OK 押下



RequestParam で指定した伝票No.のデータが表示されます
パラメータを指定しなければ、テーブル全件が抽出されます



フィールド名	値
err	null
trdate	20210601
denpyono	00001
tantou	1000
karikamokucd	List
karikingaku	List
kasikamokucd	List
kasikingaku	List
tekiyo	摘要
sysdate	20210631
errmessage	null
karikamokucdsString	[110101, 110102]
karikingakusString	[1000000, 10000000]
kasikingakusString	[1000000, 10000000]
kasikamokucdsString	[120100, 120200]

Spring/Web サービス (アノテーションによる実装) とクライアント

(1) curl コマンド

POST メソッドは Excel シートから送信できない (VBA マクロが必要) ので、curl コマンドから実行する例をあげます。

curl コマンドはターミナルやコマンドプロンプトから http 他各種の通信プロトコルを送受信することができるコマンド³です。コマンドプロンプトからは少し使い勝手が異なりますが、コマンドは同じです。

【コマンド基本形式】

```
curl -v -X POST localhost:9000/denpyo/post -H 'Content-Type:application/json; charset=sjis'  
-d '{JSON}'
```

[オプション]

-v 詳細メッセージを表示

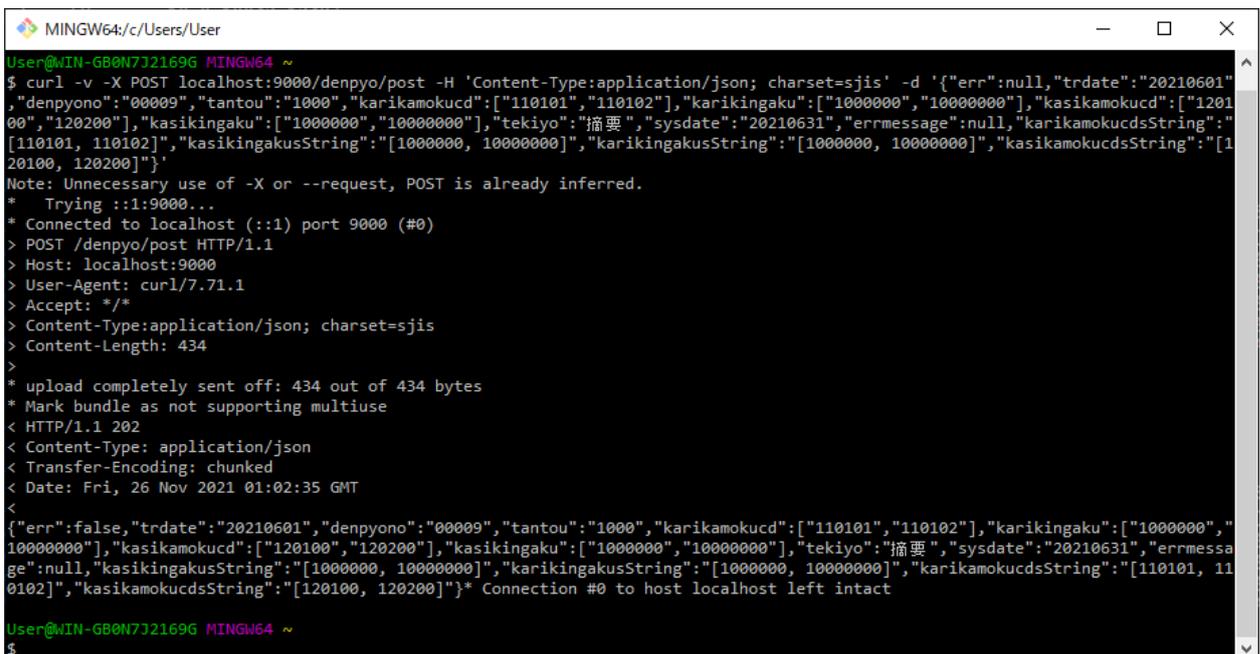
-X Http リクエストメソッド (POST,GET 等) + URL

-H Http リクエストヘッダ…'Content-Type:application/json; charset=sjis' の charset だけが変わる場合があり、Windows で実行し-d 以降に直接データを入力する場合は sjis にする

-d 送信するデータ…オプションに続けてデータを入力するか、@ファイルパス の形式で指定します

① ターミナル (Linux/MingW) からの実行例

Linux のターミナルから実行 (実際は Windows 版 Git に添付の MingW) した例。



```
MINGW64:/c/Users/User
User@WIN-GB0N7J2169G MINGW64 ~
$ curl -v -X POST localhost:9000/denpyo/post -H 'Content-Type:application/json; charset=sjis' -d '{"err":null,"trdate":"20210601",
,"denpyono":"00009","tantou":"1000","karikamokucd":["110101","110102"],"karikingaku":["1000000","10000000"],"kasikamokucd":["1201
00","120200"],"kasikingaku":["1000000","10000000"],"tekiyo":"摘要","sysdate":"20210631","errmessage":null,"karikamokucdsString":["1
110101, 110102"],"kasikingakusString":["1000000, 10000000"],"karikingakusString":["1000000, 10000000"],"kasikamokucdsString":["1
20100, 120200"]}'
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying ::1:9000...
* Connected to localhost (::1) port 9000 (#0)
> POST /denpyo/post HTTP/1.1
> Host: localhost:9000
> User-Agent: curl/7.71.1
> Accept: */*
> Content-Type:application/json; charset=sjis
> Content-Length: 434
>
* upload completely sent off: 434 out of 434 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 202
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Fri, 26 Nov 2021 01:02:35 GMT
<
{"err":false,"trdate":"20210601","denpyono":"00009","tantou":"1000","karikamokucd":["110101","110102"],"karikingaku":["1000000",
"10000000"],"kasikamokucd":["120100","120200"],"kasikingaku":["1000000","10000000"],"tekiyo":"摘要","sysdate":"20210631","errmessag
e":null,"kasikingakusString":["1000000, 10000000"],"karikingakusString":["1000000, 10000000"],"karikamokucdsString":["110101, 11
0102"],"kasikamokucdsString":["120100, 120200"]}
* Connection #0 to host localhost left intact

User@WIN-GB0N7J2169G MINGW64 ~
$
```

- ・最初のコマンドから4行が、コマンド+送信データ
- ・">"が送信した http リクエストヘッダで、"<"が http レスポンスヘッダと受信データです
- ・受信データの"err"が false になっていることでエラーが無かったと判ります

³ 公式サイト <https://curl.se/docs/>

Spring/Web サービス (アノテーションによる実装) とクライアント

② コマンドプロンプト (Windows) からの実行例

以下の点がターミナルと異なります。

- ・文字列は「”」で囲む …JSON の文字リテラルとバッティングしてしまう (エスケープは可能)
- ・文字コードは Shift-JIS(CP932、MS932) …CHCP コマンドで確認できます

※文字コードはクライアント、AP サーバ(Java)、DBMS がそれぞれ独自に管理しており、相互の文字コード変換が自動で行われます。コマンドプロンプトの文字コードは CHCP コマンドで変更できますが、現在のバージョンでは送信データに合わせると受信データが文字化けを起こします

```
コマンドプロンプト
C:\Users\User>curl -v -X POST localhost:9000/denpyo/post -H "Content-Type:application/json; charset=utf-8" -d @C:\Users\User\Desktop\POSTデータ.txt
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 9000 (#0)
> POST /denpyo/post HTTP/1.1
> Host: localhost:9000
> User-Agent: curl/7.55.1
> Accept: */*
> Content-Type:application/json; charset=utf-8
> Content-Length: 436
>
* upload completely sent off: 436 out of 436 bytes
< HTTP/1.1 400
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Fri, 26 Nov 2021 01:37:02 GMT
< Connection: close
<
{"err":true,"trdate":"20210601","denpyono":"00009","tantou":"1000","karikamokucd":["110101","110102"],"karikingaku":["1000000","10000000"],"kasikamokucd":["120100","120200"],"kasikingaku":["1000000","10000000"],"tekiyo":"摘要","sysdate":"20210631","errmessage":"\r\n### Error updating database. Cause: org.postgresql.util.PSQLException: ERROR: 重複したキー値は一意的制約 pk denpyo 違反となります\r\n 詳細: キー (denpyono, tantou)=(00009, 1000) はすでに存在します。"}\r\n### The error may exist in jp.co.focs.denpyo/DenpyoRepository.xml\r\n### The error may involve jp.co.focs.denpyo.DenpyoRepository.insertDenpyo-Inline\r\n### The error occurred while setting parameters\r\n### SQL: INSERT INTO denpyo(err, trdate, denpyono, tantou, karikamokucd, karikingaku, kasikamokucd, kasikingaku, tekiyo, sysdate, errmessage) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)\r\n### Cause: org.postgresql.util.PSQLException: ERROR: 重複したキー値は一意的制約 pk denpyo 違反となります\r\n 詳細: キー (denpyono, tantou)=(00009, 1000) はすでに存在します。; nested exception is org.postgresql.util.PSQLException: ERROR: 重複したキー値は一意的制約 pk denpyo 違反となります\r\n 詳細: キー (denpyono, tantou)=(00009, 1000) はすでに存在します。","kasikingakusString":["1000000","10000000"],"karikingakusString":["1000000","10000000"],"karikamokucdsString":["110101","110102"],"kasikamokucdsString":["120100","120200"]})* Closing connection 0
C:\Users\User>
```

- ・送信データを utf-8 の文字コードで作成し、ファイルに保存してあります
- ・-H 'Content-Type:application/json; charset=utf-8' の charset は送信データの文字コードです
- ・この例では、DB 登録時に重複キーエラーが発生していることが受信データの errmessage で判ります

Spring/Web サービス (アノテーションによる実装) とクライアント

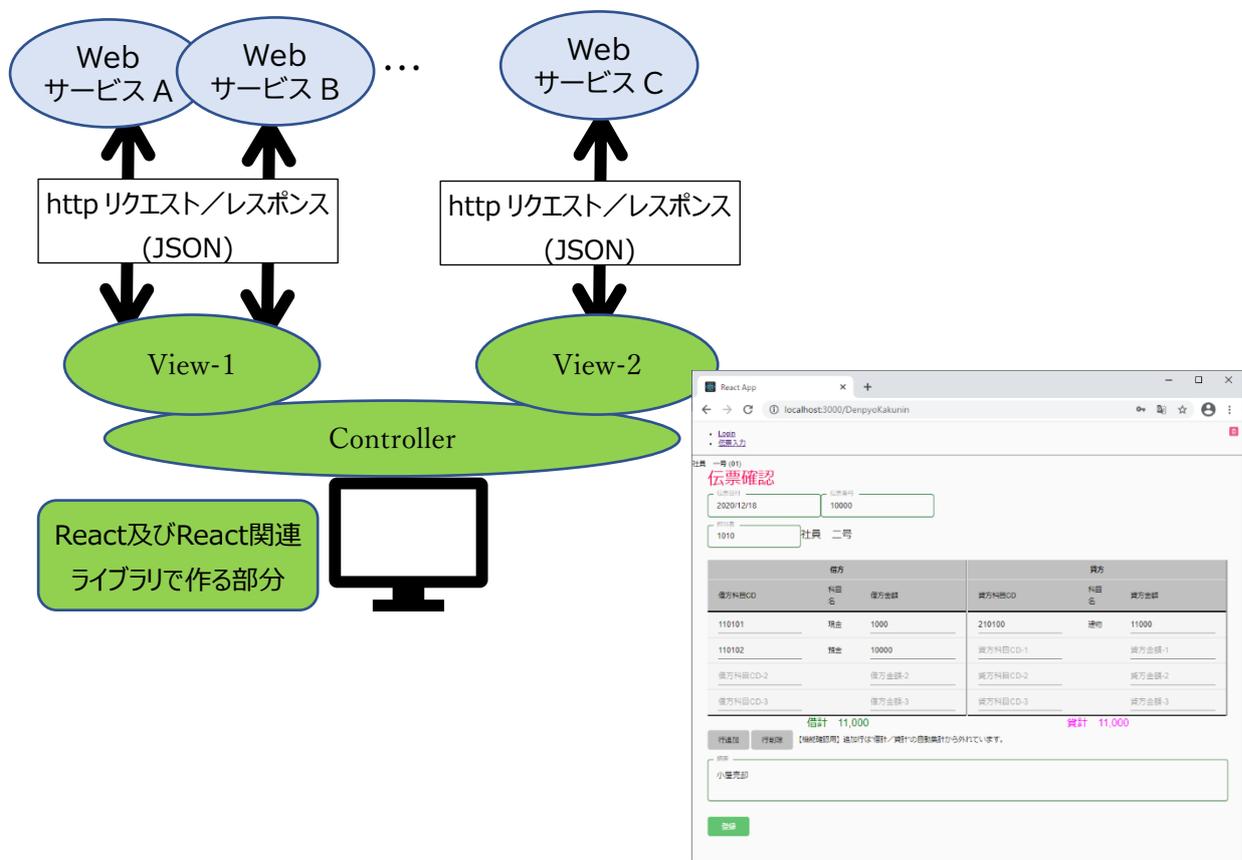
7. その他のクライアントアプリ

JSON オブジェクトは {キー(key):バリュー(value)} の形式なので参照系は汎用ツールで機能するのですが、登録/更新は Web サービスが期待している項目を編集して送信するアプリが必要になります。エディターと curl コマンドでも可能ですが、定常作業で使用するには専用のアプリケーションが必要になります。

7.1. React

React は SPA(Single Page Application)と呼ばれるものの一種で、Web アプリの View と Controller をこれで置き換えることができます (Model に Web サービスを使う)。類似のものとして Angular、Vue.js 等が有名です。特徴は JavaScript でブラウザ内に DOM (Document Object Model) を構築することです。一般的な Web アプリは html を使って DOM を記述しますが、SPA では html を使わずに動的に DOM を作り出します。

アプリ開発上の利点は、全く別のスキルを必要とする画面デザインとサーバ処理を切り離して開発できる点です。サーブレット/JSP で画面を作る一般的な Web アプリは業務ロジックと画面編集の処理が一体になっているため画面デザインが終わらないとサーバ側の実装に着手できません。一方、SPA を使った方式では送受信インタフェースの JSON 以外は独立・平行して開発ができ、また、JavaScript で記述する SPA は JSON(JavaScript Object Notation)が扱いやすくなっています。



Spring/Web サービス (アノテーションによる実装) とクライアント

7.2. Excel/VBA による実装例

Web アプリのクライアントはブラウザです。過去にリッチクライアントという触れ込みで Curl 言語、JavaApplet、Flash (サポートは 2020 年で終了) が使われてきましたが、これらもブラウザを土台にしています (SPA も同様です)。しかし、Web サービスは http さえ使えばクライアントの制限はありません。以下に Excel VBA マクロの例を紹介します。

(1) 実行例

① Excel で作った画面と、サーバとの通信(POST)結果

The screenshot shows an Excel spreadsheet with a form for '振替伝票' (Transfer Slip). The form includes input fields for '伝票日付' (20211201), '伝票番号' (10000), and '担当' (0001). Below the form is a table with columns for '借方' (Debit) and '貸方' (Credit), each with sub-columns for '科目' (Account) and '金額' (Amount). The table contains two rows of data: one for account 333 with an amount of 10,000, and another for account 9999 with an amount of 10,000,000. A '確認' (Confirm) button is at the bottom left. A dialog box is open in the center, displaying the message: 'サーバの処理が正常に終わりました。 Status=201' (Server processing completed normally. Status=201) with an 'OK' button.

② STS から起動した Web サービスで正常終了(Completed 201 CREATED)を確認できる

```
Problems @ Javadoc Declaration Search Console Debug
spring-webservice - SpringWebserviceApplication (1) [Spring Boot App] C:\Tools\AdoptOpenJDK\jdk-11.0.10.9-hotspot\bin\javaw.exe (2021/12/08 11:45:58)
2021-12-08 11:53:35.784 DEBUG 2632 --- [nio-9000-exec-4] o.s.web.servlet.DispatcherServlet : POST "/denpyo/post", parameters={}
2021-12-08 11:53:35.785 DEBUG 2632 --- [nio-9000-exec-4] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to jp.co.foccs.denpyo.DenpyoController#post(
2021-12-08 11:53:35.787 DEBUG 2632 --- [nio-9000-exec-4] m.m.a.ResponseBodyMethodProcessor : Read "application/json;charset=UTF-8" to [jp.co.fo
2021-12-08 11:53:35.796 DEBUG 2632 --- [nio-9000-exec-4] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Using "application/json", given [*/] and supporte
2021-12-08 11:53:35.796 DEBUG 2632 --- [nio-9000-exec-4] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Writing [jp.co.foccs.denpyo.DenpyoDto@2bbf15dd]
2021-12-08 11:53:35.797 DEBUG 2632 --- [nio-9000-exec-4] o.s.web.servlet.DispatcherServlet : Completed 201 CREATED
```

Spring/Web サービス (アノテーションによる実装) とクライアント

③ Excel からのリクエスト (上段) とサーバからのレスポンス (下段) の内容

The screenshot shows the Fiddler interface with the 'Raw' tab selected. The top section displays the request details for a POST to `http://localhost:9000/denpyo/post`. The request body is a JSON object:

```
{
  "denpyono": "10000",
  "karikamokucd": ["333", "", "", "", ""],
  "karikingaku": ["10,000", "", "", "", ""],
  "kasikamokucd": ["", "", "", ""],
  "kasikingaku": ["", "", "", "10,000,000"],
  "tantou": "0001",
  "tekiyo": "",
  "trdate": "20211201"
}
```

The bottom section shows the response details for an HTTP 201. The response body is a JSON object:

```
{
  "err": false,
  "trdate": "20211201",
  "denpyono": "10000",
  "tantou": "0001",
  "karikamokucd": ["333", "", "", "", ""],
  "karikingaku": ["10,000", "", "", "", ""],
  "kasikamokucd": ["", "", "", "9999"],
  "kasikingaku": ["", "", "", "10,000,000"],
  "tekiyo": "",
  "sysdate": "20211208",
  "errmessage": null,
  "karikamokucdsString": ["333", , , ],
  "karikingakusString": ["10,000", , , ],
  "kasikamokucdsString": [ , , , 9999],
  "kasikingakusString": [ , , , 10,000,000]
}
```

④ ③のリクエストボディを JSON 形式で表示したもの …形式に問題がなければ表示できる ※レスポンスには DenpyoDto のゲッター(getXxxx メソッド)の値が全て含まれています

The screenshot shows the Fiddler interface with the 'JSON' tab selected. The response body is displayed in a tree view:

- JSON
 - denpyono=10000
 - karikamokucd
 - 333
 - karikingaku
 - 10,000
 - kasikamokucd

The bottom section shows the response body in a tree view with error handling details:

- JSON
 - denpyono=10000
 - err=False
 - errmessage=(null)
 - karikamokucd
 - 333
 - karikamokucdsString=[333, , ,]
 - karikingaku
 - 10,000

Spring/Web サービス (アノテーションによる実装) とクライアント

(2) Excel VBA マクロの内容

JSON オブジェクトに含まれるキー値がサーバ側の実装と完全に一致していないと処理漏れが発生するため、定義体から画面を作るようにしました (Excel シートを動的に編集)。添付の実装例は文字コードが SJIS (または ANSI)、改行コードは LF で動作するようになっています。



denpyo.def

① 画面定義体の内容

```
1 URL=http://localhost:9000/denpyo/post
2 UIF=denpyo
3 #行-繰返 列 桁数 入出力タイプ 項目名 編集 初期値 背景色 配置
4 item=10-5 5 10 i karikamokucd 16777215 C
5 item=10-5 15 13 i karikingaku 16777215 R
6 item=10-5 28 10 i kasikamokucd 16777215 C
7 item=10-5 38 13 i kasikingaku 16777215 R
8 item=16~5 5 46 i tekiyo 16777215 S
9 item=4 11 10 i trdate 16777215 L
10 item=5 11 10 i denpyono 16777215 L
11 item=6 11 10 i tantou 16777215 L
12 item=22 5 10 b action 確認 -2147483648 C
13 item=2 25 1 t 振替伝票 16777215 S
14 item=4 5 1 t 伝票日付 16777215 S
15 item=5 5 1 t 伝票番号 16777215 S
16 item=6 5 1 t 担当 16777215 S
17 item=8 5 23 t 借方 13285804 C
18 item=8 28 23 t 貸方 13285804 C
19 item=9 5 10 t 科目 13285804 C
20 item=9 15 13 t 金額 13285804 C
21 item=9 28 10 t 科目 13285804 C
22 item=9 38 13 t 金額 13285804 C
23 item=10 5 10 t 333 16777215 C
24 item=10 15 13 t 10,000 16777215 R
25 item=14 28 10 t 9999 16777215 C
26 item=14 38 13 t 10,000,000 16777215 R
27 item=15 5 1 t 摘要 16777215 S
28 item=21 50 1 t #end 16777215 S
```

<内容> ※左端の行番号は説明用につけたものでデータではありません

- ① 1 行目 URL=は Web サービスへの送信先
- ② 2 行目 UIF=は画面名
- ③ 1 桁目"#"はコメント (3 行目の内容は以降の列の項目見出し)
- ④ 4 行目以降の item=は入出力タイプごとに、i:入力項目/b:ボタン/t:固定出力

Spring/Web サービス (アノテーションによる実装) とクライアント

② VBA

Excel のメニュー 開発 > Visual Basic から添付のファイル (ファイル名末尾の“.text”は削除して拡張子は .bas にしてください) をインポートし、マクロの auto_open を実行して①の定義書を読み込ませると項番(1)の画面がでます。



com.bas.text

```
,
' action 呼出
' <btn> 押されたボタンの名前
,
Public Sub gos(btn As String)
    Dim reqUrl As String
    Dim N As name
    Dim R As Range
    Dim C As Range
    Dim INm As String
    Dim i As Integer
    Dim para(), p() As String
    Dim paraStr, items As Variant

    ' JSON 組立
    Set uif = Worksheets("UIF")
    ReDim para(0)
    For Each N In uif.Names
        INm = Mid(N.name, InStr(N.name, "!") + 1)
        If N.RefersToRange.Count > 1 Then
            ' 名前が複数の領域を指していたら配列…Join 関数を使いたいで Range を配列にコピーする
            ' Range と配列は開始インデックスが異なる(1 と 0)ので合わせる(Option Base で 1 を指定すれば一致するが、他モジュールにも波及する)
            ReDim Preserve p(N.RefersToRange.Count - 1)
            For i = 1 To N.RefersToRange.Count
                p(i - 1) = N.RefersToRange(i)
            Next
            para(UBound(para)) = "" & INm & "" & ":" & "[" & Join(p, ",") & "]" & ""
        Else
            para(UBound(para)) = "" & INm & "" & ":" & IIf(N.RefersToRange.Value <> "", "" & N.RefersToRange.Value & "", "")
        End If
        If UBound(para) + 1 < uif.Names.Count Then ReDim Preserve para(UBound(para) + 1)
    Next
    If para(UBound(para)) = "" Then MsgBox "送信データ無し": End
    paraStr = "{" & Join(para, ",") & "}"

    ' On Error GoTo error_trap

    ' 通信先設定
    ServerURL = Worksheets("sys").Cells(1, 1).Value

    ' HTTP 接続開始
    If req Is Nothing Then
        Set req = CreateObject("MSXML2.XMLHTTP")
    End If

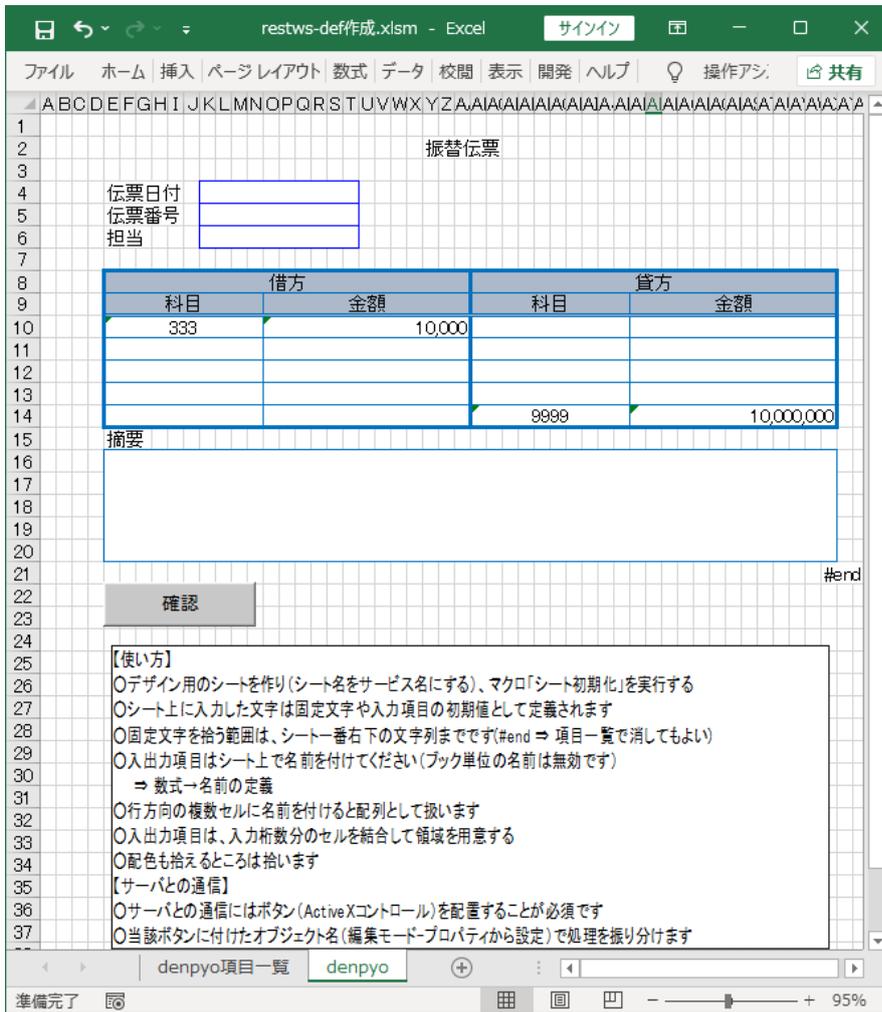
    reqUrl = ServerURL & ";jsessionid=" & SessionID
    req.Open "POST", reqUrl, False
    req.setRequestHeader "Content-Type", "application/json"
    req.send paraStr

    If req.Status >= 200 _
    And req.Status <= 299 Then
        MsgBox "サーバの処理が正常に終わりました。" & vbCrLf _
        & "Status=" & req.Status, vbNormal, "OK"
    ElseIf req.Status = HTTPSTATUS_NOTACCEPTABLE Then
        MsgBox "サーバのデータ処理が失敗しました。" & vbCrLf _
        & "Status=" & req.Status & req.responseText, vbError, "処理失敗"
    Else
        MsgBox "action(POST)で続行不可能なエラーが発生しました" & vbCrLf _
        & "Status=" & req.Status, vbCritical, "通信障害"
    End If
    Exit Sub
error_trap:
    MsgBox btn & "で続行不可能なエラーが発生しました" & vbCrLf _
    & "err_no=" & Err.Number & ", err_desc=" & Err.Description & vbCrLf _
    & "ServiceID=" & ServiceID, _
    vbCritical, "通信障害"
End Sub
```

Spring/Web サービス (アノテーションによる実装) とクライアント

【補足】画面定義体の作成

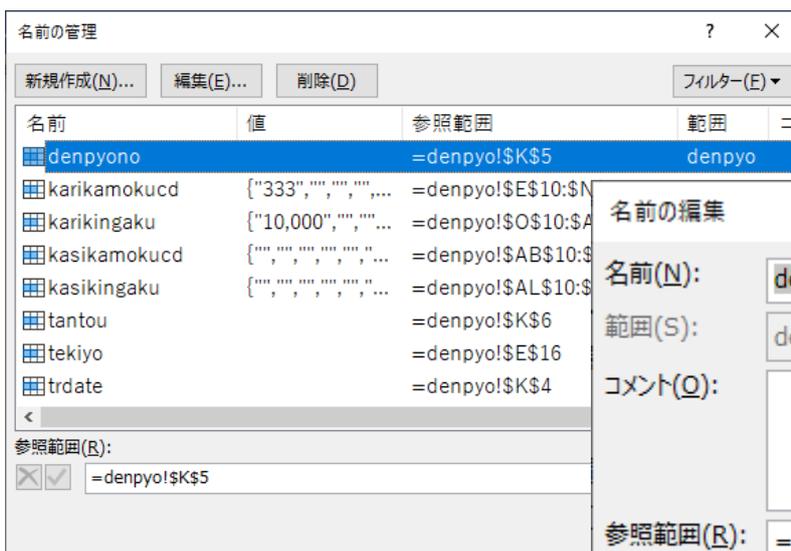
画面のイメージ作成と画面定義体の作成も Excel で行うことができます。



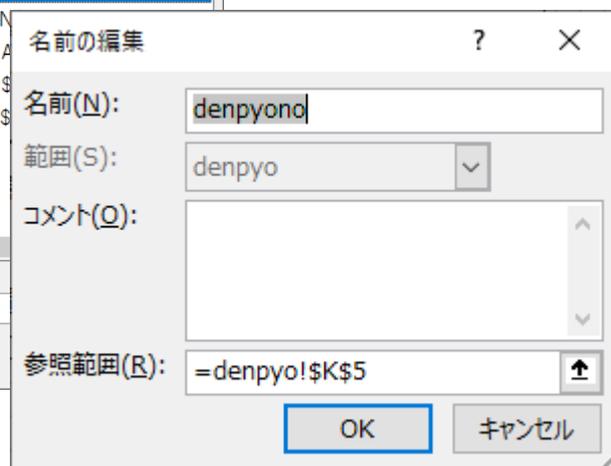
def.bas.text

新しい Workbook を作り...

- ① def.bas をインポート
- ② シートを画面名にする
- ③ [Z_シート初期化]マクロ実行
- ④ 画面のデザインを行う
⇒数式 > 名前の管理
- ⑤ [A_項目抽出]マクロ実行
- ⑥ [B_画面定義体作成]マクロ実行



※名前を付ける範囲はシートにしてください



以上