Webプログラミング実習Ⅱ　授業用レジュメ④

■サーブレットの解説

●サーブレットクラスのひな形

```
//必要クラスのインポート
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

//『@WebServlet』アノテーションでURLパターンを設定：「サーバーに接続するためのアドレス/動的Webプロジェクト名+URLパターン」のURLにリクエストを飛ばすことでクラスがインスタンス化され`アプリケーションサーバ上で動作する゜
@WebServlet("/URLパターン")
public class SerbletSample extends HttpServlet
{
  //「serialVersionUID」変数: サーブレットの親クラスであるHttpServletクラスは`
  //「Serializable」というインタフェースを実装している゜
  //「Serializable」インタフェースは
  //「このクラスはそのままバイトの配列(外部ファイルとして保存しやすい形式)に
  //変換(シリアライズ)することが可能ですよ」という目印になるインタフェースだが`
```

```
  //それを実装するためのお約束として↓のフィールドが必要となる。(宣言してないと警告が出る)
  private static final long serialVersionUID = 1L;

  //HTTPにおいて`ブラウザなどのクライアントから送られてくる主なリクエストは
  //「Get(情報の取得)」「Post(情報の送信)」の2種類。
  //サーブレットは`それぞれのリクエストを「doGet」「doPost」というメソッドに振り分けて処理する。
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

          //ここにGetリクエスト時の処理
  }

  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

          //ここにPostリクエスト時の処理
  }
}
```

●サーブレット内で行う主な処理の構文

・リクエストパラメータの取り出し
クライアントから送られてきたリクエストパラメータは、リクエストの中から取り出してやらないと利用できない。

```
//クラス･メソッドの宣言は省略

//送られてきたリクエストの文字コードをutf-8に揃える(文字化け対策)
request.setCharacterEncoding("utf-8");
```

```
//取得したいリクエストパラメータ名を『request.getParameter』メソッドの引数に渡すとそのリクエストパラメータの値がString型で返ってくるので`変数に取る
String 変数名 = request.getParameter("リクエストパラメータ名");

//取り出したリクエストパラメータは必要に応じて値に変換したり`エンティティクラスにまとめたりして利用する
```

・レスポンス内容の書き出し
サーブレットクラスが終了したタイミングでクライアントへレスポンスが返る。
そのレスポンスの中にHTMLやJSONなどを書き出してやることで、クライアント側へデータを送り返すことができる。

```
import java.io.PrintWriter;

//クラス･メソッドの宣言は省略

//返すレスポンスの文字コードをUTF-8に揃える（文字化け対策）
response.setCharacterEncoding("utf-8");

//『response.getWriter()』メソッドを使い`
//レスポンス内に文字列を書き込むためのPrintWriter型インスタンスを変数に取得
PrintWriter out = response.getWriter();
```

■DAOの解説

●DAOクラスのひな形

```
//必要クラスのインポート
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class SampleDAO{
    //データベースへ接続する際に使う情報は `
    //final修飾子付きのフィールド定数として記述しておくと便利
    //(このコードでは `実際に授業で使ったDBのパス・DB名・文字コード変換用パラメータ・ユーザー名・パスワードを仮入力しています)
    private final String ROOT_PATH = "jdbc:mysql://localhost:3306/";
    private final String DB_NAME  = "nejiko_online";
    private final String DB_CHARACTER_ENCODE = "?useUnicode=true&characterEncoding=utf8";
    private final String DB_USER = "root";
    private final String DB_PASS = "root";

    //お約束①：コンストラクタでJDBCドライバを読み込む
    public SampleDAO(){
        try{
            Class.forName("com.mysql.jdbc.Driver");
        //例外「JDBCドライバ見つからなくない？」など
        }catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public 任意の戻り値 sampleConnect(任意の引数)
```

```
    {

        //お約束②：try with Resources文の中で｀
        //接続の情報を保持するConnection型の変数を宣言゜
        //『DriverManager.getConnection』メソッドを使い｀使いたいDBとの
        //接続を確立する
        try(Connection conn = DriverManager.getConnection(ROOT_PATH + DB_NAME + DB_CHARACTER_ENCODE, DB_USER, DB_PASS)) {


            //お約束⑤：DBへ飛ばしたいクエリを文字列変数として用意
            String sql = ”任意のSQLクエリ”

            //お約束⑥：「用意された命令文」を意味するPreparedStatement型変数を宣言゜
            //Connection型インスタンスの『Connection.prepareStatement("飛ばしたいクエリ")』
            //メソッドを呼び出し｀飛ばしたいクエリを実行できる形にprepare(準備)する
            PreparedStatement pStmt = conn.prepareStatement(sql);

            //お約束⑦：以上を済ませた状態でPreparedStatement型インスタンスの
            //『executeQuery()』『executeUpdate()』メソッドを実行すると｀
            //お約束⑥で準備したクエリの内容に沿ってDBから取得されたデータが
            //ResultSet型のインスタンスとしてResultSet型変数へ代入される
            ResultSet rs = pStmt.executeQuery();
```

```
            //↑(データの取得でなく`テーブル内の情報を書き換えるクエリを飛ばす時は
            //『ResultSet rs = pStmt.executeUpdate();』
            //となることに注意)



            //お約束⑧：情報をResultSetインスタンスから取り出す
            //↓if「ResultSetインスタンスの中に取得結果がある場合は」
            if(rs.next()){
                //この中で情報の取り出し処理゜
                //結果が複数あって配列に格納したい場合は`上の「if」を「while」にして
                //while「ResultSetインスタンスの中に取得結果がある限り繰り返し」とする

                //ResultSetインスタンスの中のデータは`
                //『ResultSet.getXX("カラム名")系メソッドで変数に取り出すことができる

                //取り出したい対象が文字列なら「getString("カラム名")」
                String 任意の変数名A = rs.getString("カラム名");
                //intなら「getInt("カラム名")」など゜
                //他にもあるのでJavaEEのリファレンスを見てみよう
                int 任意の変数名B = rs.getInt("カラム名");

                //必要なら`取り出したデータをさらに
                //クラス型のインスタンスなどに入れておく
            }
            //返したい形に整えた取得結果を戻り値として返す
            return 任意の戻り値;
```

```
        //お約束②のcatch節
        //例外「飛ばしてるクエリがおかしいぞ！」「DBがクエリを受け取ってくれないぞ！」など
        }catch(SQLException e){
            e.printStackTrace();

        }
        //例外が起こった際に返す値が全て同じの場合は｀
        //何かしらの例外が起きたらnullが返ってくる
        return null;
    }
}
```

■Unity側の解説

●UnityでのWeb接続

今回の授業では、Unity標準で用意された**「UnityWebRequest」クラス**のインスタンスを用いている。

●UnityからGetリクエストを行うための骨組み

```
public class WebConnectSample(){
        //アクセスするURLを定数にしておくと変更が楽
        const string ROOTPATH = "http://localhost:8080/nejikoRunOnlineServer/";
        //Web接続はコルーチンで行うが｀コルーチンはIEnumerator型以外の戻り値を返せないため｀
        //取得結果をフィールドに入れたりしておく必要がある
```

```
        string getText = "";

        //取得結果を他のインスタンスから引き出すためのプロパティ
        public string GetText
        {
            get { return getText; }
        }

        //お約束①  Webとの通信はコルーチンの中で行う
        //(サーバーからのレスポンスを待たなければいけないため)
        IEnumerator SampleWebConnectionGet()
        {
            //お約束②UnityWebRequest型の変数を宣言 `
            //そのコルーチンでアクセスしたいURLの文字列を引数で渡した
            //UnityWebRequestインスタンスをnewして代入
            UnityWebRequest request = new UnityWebRequest(ROOT
PASS + "getranking");

            //お約束③UnityWebRequest内のフィールド「downloadHandle
r」に `
            //DownloadHandler型にキャストしたDownloadHandlerBuffe
r型インスタンスを
            //newして代入(受け取ったレスポンスの情報はこのインスタンスに
入る)
            request.downloadHandler = (DownloadHandler)new Dow
nloadHandlerBuffer();

            //お約束④：UnityWebRequest.SetRequestHeader(設定項
目,設定したい値)メソッドを
            //呼び出し `レスポンスで受け取りたいファイル形式に合わせたリク
エストヘッダを設定
```

```
            request.SetRequestHeader("Content-Type","application/json");
            //↑「受け取りたいファイル形式」の設定項目に「JSON」を設定しますよ

            //お約束⑤：レスポンスが返ってくるまで実行を待つ
            yield return request.SendWebRequest();

            //お約束⑥：エラーが起きた場合はログを返す
            //何かエラーが発生した際は `
            //UnityWebRequest型インスタンスの「isXXError」系フィールドにtrueが入る °
            //それを条件式に使うことでエラーを検出する
            if (request.isNetworkError || request.isHttpError)
            {
                Debug.Log(request.error);
                Debug.Log(request.responseCode);
            }
            //正常に接続できた場合
            else
            {
              //お約束⑦：UnityWebRequest.downloadHandlerから取得したレスポンスを取り出す
              //今回受け取っているのは文字列なので `「text」フィールドから文字列型で取り出せる
                getText = request.downloadHandler.text;
            }
        }
}
```

●UnityからPostリクエストを行うための骨組み

```
public class WebConnectSample(){
        //アクセスするURLを定数にしておくと変更が楽
        const string ROOTPATH = "http://localhost:8080/nejikoRunOnlineServer/";
        //Web接続はコルーチンで行うが `コルーチンはIEnumerator型以外の戻り値を返せないため `
        //取得結果をフィールド変数に入れたりしておく必要がある
        string getText = "";

        //取得結果を他のインスタンスから引き出すためのプロパティ
        public string GetText
        {
            get { return getText; }
        }

        //お約束①  Webとの通信はコルーチンの中で行う
        //(サーバーからのレスポンスを待たなければいけないため)
        IEnumerator SampleWebConnectionPost()
        {
            //【Getとの相違点①】postしたデータをサーバ側から引き出すには `
            //「リクエストパラメータ」の設定が必要となる °
            //Unityでは `WWWform型インスタンスを新規作成し `
            //その「AddField(”その情報を引き出すためのキー”,Postしたい情報)」メソッドを
            //使うことでリクエストパラメータを設定できる °

            int sampleNumber = 1;
```

```
            WWWForm form = new WWWForm();
            form.AddField("number", sampleNumber);

            //お約束②UnityWebRequest型の変数を宣言 。
            //【Getとの相違点②】
            //newしたUnityWebRequestインスタンスから
            //「Post(データをPostしたいURL,それに使いたいWWWform)」メソッドを呼び出し `
            //そのコルーチンでアクセスしたいURLの文字列と `
            //先ほど作ったWWWform型インスタンスを引数として渡す
            UnityWebRequest request = new UnityWebRequest.Post(ROOTPASS + "getranking",form);

            //お約束③UnityWebRequest内のフィールド「downloadHandler」に `
            //DownloadHandler型にキャストしたDownloadHandlerBuffer型インスタンスを
            //newして代入(受け取ったレスポンスの情報はこのインスタンスに入る)
            request.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();

            //お約束④：（サンプルプログラムではPostメソッドで特定の形式のファイルを
            //受け取る必要がないため `リクエストヘッダは定義しない)

            //お約束⑤：レスポンスが返ってくるまで実行を待つ
            yield return request.SendWebRequest();

            //お約束⑥：エラーが起きた場合はログを返す
            //何かエラーが発生した際は `
```

```
            //UnityWebRequest型インスタンスの「isXXError」系フィールドにtrueが入る。
            //それを条件式に使うことでエラーを検出する
            if (request.isNetworkError || request.isHttpError)
            {
                Debug.Log(request.error);
                Debug.Log(request.responseCode);
            }
            //正常に接続できた場合
            else
            {
              //お約束⑦：UnityWebRequest.downloadHandlerから取得したレスポンスを取り出す
              //今回受け取っているのは文字列なので、「text」フィールドから文字列型で取り出せる
                getText = request.downloadHandler.text;
            }
        }
}
```

※これ以外にもC#標準のメソッドを介する方法などがあり、それぞれに得意分野があるようです。
　授業では触れませんが、興味のある方は調べてみてください。

●どうしてコルーチンを使うの？

通信が終わってからその後の処理をしないと、
まだ結果が代入されていない変数を参照してしまうため。
『NejikoRunOnline』では、
「Web接続→DBへの接続→DBからのデータ取り出し→JSONの受け取り→クラスへの書き出し」
までを順序立てて行いたいため、このサマリーの骨組みにあたる処理をさらにコルーチンで囲んで実行している。

※（おまけ）データベース内のデータにID番号を付けたい時のカラム定義
【ID番号しか持たないテーブル『example』を作るときのＳＱＬクエリ】

```
CREATE TABLE example (id INT PRIMARY KEY AUTO_INCREMENT NOT NU
LL,INDEX(id));
```