

目次

1. 導入 ～ 前回のおさらい ～
 - 1-1 前回やった事を振り返ってみましょう
2. PHPでメールを送信すること
 - 2-1 メールを送信するために必要なこと
3. セキュリティに関する色々な知識
 - 3-1 スクリプト・インサレーション
 - 3-2 XSS (クロスサイト・スクリプティング)
 - 3-3 CSRF (クロスサイト・リクエストフォージェリー)
 - 3-4 メールヘッダインジェクション
4. メール送信プログラムの作成
 - 4-1 入力フォームから作成してみよう
 - 4-2 入力チェックを行う
 - 4-3 メールを送信してみよう ～ ライブラリの使用 ～
5. 発展 ～ 確認画面を作りたい ～
6. 補足

導入 ～ 前回のおさらいと今回のプログラム

前回の勉強会にご参加いただいた方、それからPHPには触れてますでしょうか？
今回が初めてのご参加の方、PHPには興味をお持ちでしょうか？

前回の講義では文字の出力、日時の取得から条件分岐までをやりました。これらは基本的な処理でありながらも、PHPではかなり頻繁に使われる処理であり、また、それらを知るだけでアイデア次第で幅広い動作をさせることが可能となりました。

では、いったいどれだけ幅広くなったのでしょうか？

それは、やはり実際にプログラムを組んでみる事が一番です。何事も試してみないとわかりませんよね。

今回は前回学んだことを生かして、メールを送信するプログラムを組んでみようと思います。「お問い合わせフォーム」と言うにはまだまだですが、これをベースにしてお問い合わせフォームは作られます。

尚、今回も時間の関係上、「確認画面」までの作成は難しいと思います。ご期待くださった方は申し訳ございません。今回作成するのは単純にメールを送信するプログラムです。

(確認画面については『5. 発展』をご参照ください)

しかしながら、単純に「メールを送信する」といえど、それには少し処理が必要なので、その処理をセキュリティ面を含めてお話を交えながら作成して頂きます。

さて、実際にプログラミングを行ううえで、やはりまた新しい処理、構文などが次々と出てきます。でもご安心ください。

やはりそれは、「知らない」だけなのです。
知ってしまえばどうということはありません。

それでは、実際にプログラミングを進める前に、前回の復習も兼ねて少しだけ振り返ってみましょう。

1-1 前回やった事を振り返ってみましょう

今回は文字の出力と条件分岐が主なテーマでした。それらを簡単に振り返ってみましょう。

まずは、文字の出力。これには「echo 構文」を使います。単純に文字列を出力するだけならば、

```
<?php
    $string = 'こんにちは';
    echo $string;
?>
```

このように書くだけです。この場合は変数 \$string に「こんにちは」という文字列を格納して、echo 構文で出力しています。変数 \$string は文字列型ですね。これをブラウザで出力すると、『こんにちは』という文字が表示されます。

このように、直接文字列を、若しくは変数に格納したものを echo するだけで表示は完了です。今回のプログラムでは変数に格納する手法を頻繁に用いますので、この形式は覚えておいてください。

次に、前回の肝となった「条件分岐」ですね。これは少しばかり複雑でした。

```
<?php
    $a = 3;
    if ( $a > 0 )
    {
        echo 'a は 0 より大きいです。';
    }
    else
    {
        echo 'a は 0 より小さいです。';
    }
?>
```

このサンプルでは、変数 \$a に 3 を格納して、その値が 0 より大きいか、そうでないかを判定しています。「if (~)」の部分が条件判定です。この中の式にマッチしていれば、true(正しい)が返り、その下の処理が行われます。逆にマッチしなければ false(正しくない)が返り、else{ ~ } の処理が行われます。もっと条件を増やしたければ「elseif」を使うんですね。

以上が前回で学んだ処理です。今から見れば簡単でしょうか？それは皆さんが「処理を知った」からであり、その積み重ねによってPHPの理解につながると私は思います。少しずつ知っていきましょう。

では、次は今回のプログラムについてです。

2. PHPでメールを送信する、ということ

2-1 メールを送信するために必要なこと

冒頭でも述べたように、今回はメールを送信するプログラムを作成します。…と言ってもどのように作っていけば良いか、想像しにくいかと思います。そこで、まずは今回のプログラム作成に当たって必要な機能、処理、処理の流れをイメージしてみましょう。実際のアプリケーション開発でもこの事前の構想を練ることは非常に重要です。この段階での準備が完成したアプリケーションの品質に大きく関わってくるほど、重要な準備段階なのです。

アプリケーションの規模に関わらず、こうして事前にイメージを作るのはとても大事なので、今回の簡単なプログラムでも同じようにイメージを作ってみましょう。

- ・ どんな機能にしようかな？

まずは機能について考えてみましょう。今回は「メールを送信する」機能が欲しいですね（それしかないですよ）。そこで、機能は「メール送信」としましょう。終了です。

- ・ どんな処理が必要かな？

ではその機能を実装するに当たって、どんな処理が必要になるのでしょうか？これには、「最低限必要な処理」と「こうしたら面白いんじゃない処理」の二つをいつも私は考えます。

「最低限必要な処理」とは、サーバーサイドスクリプトでは必須のセキュリティ周りの処理（後述）など、機能を実装する上で必要なものを指します。

「こうしたら面白いんじゃない処理」とは（私が勝手に考えたものですが）、よりリッチな体験をユーザーに提供するために考えるものです。これは必須というわけでは勿論ありません。今回は無視しても構いません。

それでは、「最低限必要な処理」を考えてみましょう。今回の機能は「メール送信」です。それにあたって必要な処理とは何でしょうか？

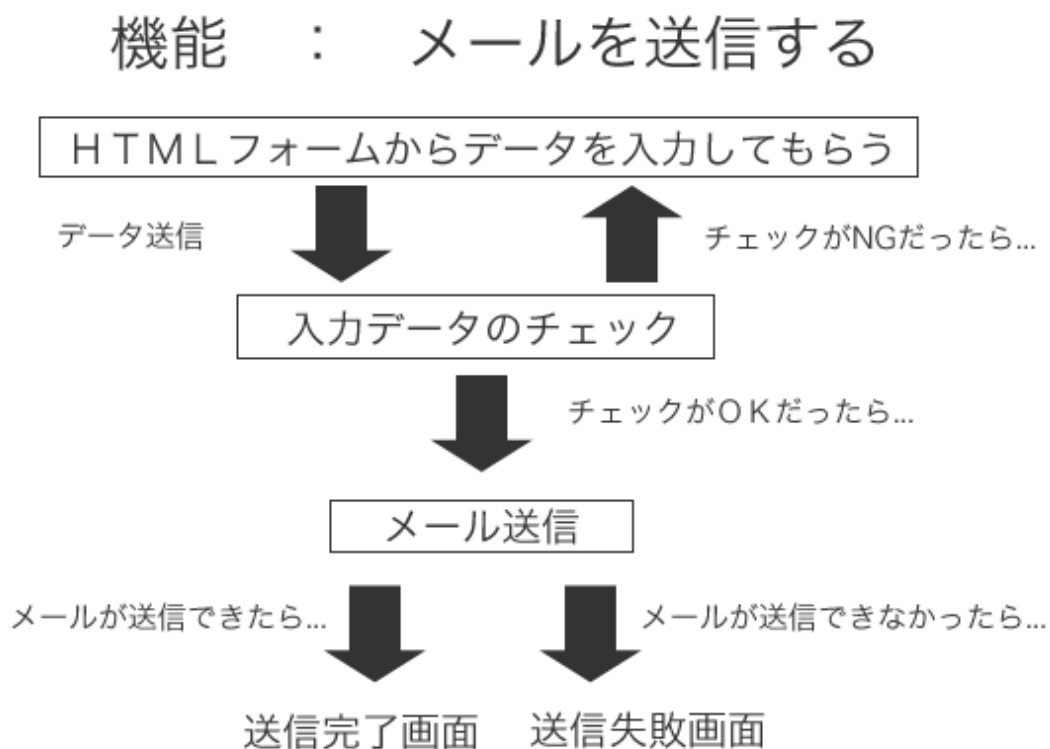
まずはメールの内容を入力してもらうフォームが必要ですよね。これはHTMLで作成できますので問題ありません。

次に必要なのは、セキュリティ関連の処理です。これはサーバーサイドスクリプトでは必ず必要な処理で、フォームとは一対一の関係になっていると言えます。セキュリティについては後述いたしますので、ここでは「セキュリティ関連の処理」としておきましょう。あと必要なのは、「メールを送信する処理」ですね。

さて、これで欲しい機能に対して必要な処理が見えてきました。あとはそれらの処理の流れを考えながら、形にしていきましょう。

- ・ どんな処理の流れになるのかな？

今までのイメージを図にしてみましょう。今回はこんな感じになりました。



図にしてみるとよくわかります。見た感じ、処理は矢印の数だけ必要です。あとは矢印と添え字のテキスト分の処理を記述すればOKです。

ここで、添え字のテキストに注目してください。「～たら…」という文字がありますね。これはどこかで聞いたことがあると思います。そう、「if」ですね。つまり、この部分はif文を使うことになります。

さらに、ここで役割分担を行うファイルを決めてしまいましょう。

「入力してもらおうファイル」は「input.php」、「入力チェックを行うファイル」は「check.php」としましょうか。メールを送信する処理はcheck.php内で行います。

さて、ここで「入力フォームはHTMLなのにわざわざphpにファイルにしなくてもいいんじゃない？」と思われた方もいるかと思います。それには少しばかり理由がありますので、とりあえずはphpファイルにしておきましょう。

これで機能のイメージと役割分担するファイル、記述しなければいけない処理が見えました。これでようやくプログラムを書き始め・・・る前に、セキュリティ関連のお話をしておこうと思います。これはとっても大事な事なので、必ず覚えていただきたいと思いません。

3. セキュリティに関する色々な知識

PHPのようにサーバー上で動作するプログラムで処理を行う時、「セキュリティ」について常に考える必要があります。

「ハッキング」や「クラッキング」という言葉を耳にしたことはありますか？これらはそのセキュリティの際を突いて色々悪意のあるプログラムをサーバー上で動作させてしまうことを言います。

ユーザーが入力してきたデータが正しいものなのかはスクリプトは判断できません。そのため、正しいかどうかを検証する必要があります、それを行うのは私達なのです。

JavaScript（実際、悪さをするのは主にこれだったりします）など、ブラウザ上で動作するスクリプトならまだ被害は軽いです。最悪、お使いのブラウザが壊れる程度（これも問題ですが）で済みます。しかし、サーバーサイドでの被害は際限なく大きくなるものです。

少し極端な例を挙げましょう。例えば、皆さんが使っているサーバーに、お客様の個人情報ファイルやデータベースに存在しているとします。もし入力されてきたデータに、それらの個人情報を全て抽出し、誰かにメールを送信するプログラムが混じっていたとしたら・・・大変なことになるのはお分かりですね。これは「私のPCが壊れた！」で済むレベルではありません。

Webアプリケーションにおいては、顧客情報を管理したり、といったことはごく当たり前に行います。それ故にセキュリティについてはしっかりと対策を施しておかないといけません。

今回はメールを送信するプログラムです。データベースなどの連携は行いませんが、一口にメール送信といっても、それに対するセキュリティ対策は結構あります。

そして、サーバーサイドのプログラミングで常に念頭において頂きたい言葉があります。

『 ユーザーからの入力には絶対に信用しない！ 』

これは私が尊敬する方から聞かされた言葉なのですが、あらゆる入力データには不正データが存在すると思った方が良く、という考えだと私は解釈しています。

ともすれば人間不信になってしまいそうですが、信用してはいけないのは「データ」であって、人間ではありません。

話が脱線しましたが、実際にはセキュリティについてどんな攻撃があるのか、ご紹介します。今まで「知らないだけなのです」と言ってましたが、今回については言い方を変えます。

「知らないでは済まされない」

といった感じでしょうか。それでは紹介に移りたいと思います。

※ 以下の紹介で記述している JavaScript はあくまでサンプル用のコードです。
絶対に実際に使用しないでください！

3-1 スクリプト・インサレーション

簡単に言えば、入力データに JavaScript などを埋め込む攻撃法です。
掲示板があったとして、入力フォームに、

```
<script>location.href = 'http://yahoo.co.jp' ; </script>
```

と入力したとします。一般ユーザーならこんな入力は普通しないのですが、悪意のあるユーザーは当たり前のように入力してきます。

では、これが投稿されて、投稿一覧を見ると・・・この JavaScript が実行されて、一覧ページを開くと必ず yahoo のサイトにジャンプしてしまいます。これで攻撃完了です。

つまり、少し JavaScript の知識があれば、攻撃は成立してしまうわけです。

上記のスクリプト程度ならまだ良いですが、これなら JavaScript は何でも実行可能なわけで、上級者ならクッキーを表示させて盗んだり、といったこと、つまり JavaScript で出来ることは何でも出来てしまいます。

対策として、スクリプトタグや「JavaScript」の記述を無害化する（サニタイズと呼ばれることが多いです）必要があります。

3-2 XSS (クロスサイト・スクリプティング)

セキュリティ関連の話題で最もよく耳にするのがXSSだと思います。
スクリプト・インサレーションでは投稿画面で攻撃しましたが、XSSの場合は外部から、
しかも任意に作成したHTMLファイルから攻撃してしまいます。
例えば、こんな風に記述したHTMLファイルがあるとします。

```
<html>
<body>
  <form action="http://www.hogehoge.com/check.php" method="post">
    <input type="text" name="attack" value="" />
    <input type="submit" value="attack!" />
  </form>
</body>
</html>
```

これをローカルで開き、テキストフォームに先ほどのスクリプトを記述してボタンを押すと、やはり先ほどと同じことができます。

<input>タグの name 属性によって PHP は処理を行います。この name 属性は実際の投稿フォームのソースを見れば簡単に分かってしまいます。

この攻撃で危険なのは、「自分のローカルマシンから攻撃できる」という点です。つまり、どこからでも攻撃ができてしまうのです。インターネットならではの攻撃法ですね。

実際、多くの攻撃はこのXSSを基点に始まる、と言われているようで、また、XSS対策が完璧に施されているサイトは少ないのも現状のようです。

サンプルは事前に準備したテストファイルをご参考ください。
こちら対策として、JavaScriptの無害化などが考えられます。

3-3 CSRF (クロスサイト・リクエストフォージェリー)

かなり特殊であり、対策が難しいのがこのCSRFです。

CSRFについて簡単に説明すると、悪意のあるデータを含むHTML等を登録ユーザー、若しくは管理者にメール送信などの方法で送りつけます。管理者は「何だろう？」と思ってリンクをクリックしたが最後、そのデータが送信されて、悪意のあるデータが実行されてしまいます。酷いものであればページを開いた瞬間にデータを送信してしまうものもある

ります。

ここで厄介なのが、「管理者権限で送信してしまう」ということです。例えば自分のサイトにログインしていて、しばらくログイン状態を保持した上でリンクをクリックしてしまうと、管理者しか実行できないこと（例えば、投稿を全て削除してしまう、等）が実行できてしまいます。これは対策が難しい傾向にあります。

先述した「ユーザーからの入力には絶対に信用しない」という言葉の「ユーザー」には、管理者も含まれます。つまり、たとえサイトの管理者の入力であっても、セキュリティ対策は必要となるのです。

この対策としては、ワンタイムトークン（後述します）の使用、リファラーチェック等の処理が必要となります。

3-4 メールヘッダ・インジェクション

実際にはHTTPヘッダインジェクションとほぼ同じなのですが、メール送信プログラム、ということでこれを取り上げました。

普通にネットサーフィンをしていると、HTTPヘッダやメールヘッダにどのような内容が入っているか知ることはありません。しかし、そんな普通は見えないような部分にも悪意のあるデータは埋め込まれます。

メール送信について説明します。メールにはヘッダがあり、ここに件名や差出人、送信先メールアドレスなどが入っているのですが、ここに改行コードが入っていると、そこでヘッダ領域は終了、次からがメール送信内容だ、と認識してしまいます（何故改行が入るとそうなるのかは専門書に任せます。ここではそういうものなのだ、とご認識ください）。

そのため、対策として、件名も入力してもらった場合等、メールヘッダに該当する部分には改行コードを削除する必要があります。

この他にもセキュリティについての問題は沢山あります。その詳細は専門書に任せるとして、今回関連があるものをピックアップしてみました。

さて、色々とあんな攻撃法、こんな攻撃法と説明ばかりで、「そんなに危険なことばかりあるんだったらやらないほうが安全だ」と思われるかもしれません。しかし、本来 Web アプリケーションはもっと色んなことが出来て楽しいもので、セキュリティが怖いから、といって開発の幅を狭めてしまうのは勿体無いことです。きちんとセキュリティの知識を持って、正しく処理をしてさえいれば、そこばかりに気を使う必要はなくて、もっと面白い処理を考える方に注力しましょう。

それでは、以上を踏まえた上で、実際にメール送信プログラムを記述してみましょう。セキュリティ対策を行っている部分は随時注釈を入れます。

4. メール送信プログラムの作成

4-1 入力フォームから作成してみよう

まずは入力フォームの作成です。このフォームについては普通のHTMLでできますので、詳細はサンプルファイル（input.php）を参照してください。拡張子を.phpにする以外は何の変哲もないHTMLソースですね。入力項目は名前と内容です。

さて、まずは入力フォームを完成させましょう。実は、普通にHTMLフォームを作るだけでは完成ではありません。ここでもセキュリティ対策を盛り込む必要があります。

- ・ ワンタイムトークンの生成

3-3 で書いたとおり、CSRFの対策を行います。

この対策の詳細としては、「入力データがいったいどこから来たのか？」を明確にするものです。先述の通り、悪意のあるデータがメールからジャンプされてきたらマズイので、ちゃんと「この入力フォームからの送信ですよ」ということをPHPに伝える必要があります。

実際の流れとして、「ワンタイムトークン」とは、「ワンタイムチケット」と呼ばれることもあり、ページ表示の際に「チケット」の半券を渡して、データを送信した際にそのチケットの半券を持ってきているかどうかを判定します。コンサートの入場にはチケットが必要ですよね。それと同じ感覚です。

では、そのチケットの半券を生成して、入力フォームに渡しましょう。以下のコードをソースの先頭に記述してください。

```
<?php
    session_start();
    $ticket = md5( uniqid( mt_rand(), true ) );
    $_SESSION[ ' ticket ' ] = $ticket ;
?>
```

何をやっているのか説明します。

- ・ session_start()

「セッション」というものを開始しています。ブラウザでページを移動する際に、基本的にデータは保持できない（送信データなどは別）仕組みなので、ここで生成したチケット

の半券も保持できないのが通常です。それを解決するために、「セッション」という機構を使います。このセッションを開始することにより、小さなデータ（大きいデータも格納できますが、推奨しません）をページ間でやりとりできるようになります。

注意点として、この「`session_start()`」という記述は必ずソースの「一番最初に」記述するようにしてください。

```
・ $ticket = md5( uniqid( mt_rand(), true ) )
```

なんだか意味の分からない記述が出てきましたね。分かるのは何かを変数に格納していることくらいでしょうか。

式の右側は、

「ランダムな数値を MD5 でハッシュしたものを生成している」

のですが、詳しく説明していると時間が足りないので、`mt_rand()`関数（前回のテキストの発展参照）、`uniqid()`関数、`md5()`関数を使ってこのように作ります。これは魔法の言葉だと思っていて結構です。

では、何故ランダムにするのか。その理由は簡単です。

もし表示する度に同じチケットが発行されたとしたら、攻撃者はその毎回発行される同じチケットをデータに仕込んでしまうからです。そのため、毎回ランダムなものにする必要があるのです。

```
・ $_SESSION[ ' ticket ' ] = $ticket
```

セッションを開始すると、「`$_SESSION`」という特殊な変数を使用可能になります。この変数に格納されたデータはページを移動しても取り出すことができます。

[' ticket ']の「ticket」という名前は任意で結構です。私はいつもこの名称にします。

これは配列のキーの名前に指定しているので、覚えて置いてください。

よって、ここでは`$_SESSION[' ticket ']`という変数に先ほど生成したランダムな値を格納しています。これでページを移動してもチケットの半券が失われることはありません。

これでチケットの作成が完了しました。あとは、フォーム送信時にチケットを渡してやればOKですね。記述はこうなります。

※ <form>~</form>の間に記述してください

```
<input type="hidden" name="ticket" value="<?php echo $ticket ; ?>" />
```

普段「見せる」ページを作成している方は、この type="hidden" という属性は先ず使わないでしょう。しかし、データを受け渡しする際に隠しデータ（ブラウザには表示されないもの）として必要な場合、この属性を使います。そして name 属性には先ほどセッションで指定した配列のキーを指定します（これは違って良いのですが、分かりやすくするためにこのように記述しています）。あとは value の値の内容に先ほど生成したチケットの半券データを echo で出力しれやればOKです。

以上でワンタイムトークンの生成、入力フォームの作成は完了です。
次はいよいよ肝の部分、「入力チェック」を行っていきましょう。

4-2 入力チェックを行う

さて、先ほど作成した入力フォームから送信されたデータを受け取って、正しい入力データなのかチェックしてみましょう。記述するのは、check.php です。

今回入力されるデータは名前と内容なので、それぞれについてチェックを行きましょう。今回は名前、内容ともに必須入力、ということにしましょう。

と、チェックを始める前に最初に行うべき処理があります。

そうです。「本当に入力フォームから送信されてきたデータなのか」調べる必要がありますね。これは先ほど仕込んだチケットがあり、かつその内容がセッションに入っている内容と同じであるかチェックします。コードは以下のようになります。

```

<?php
    session_start();
    $ticket = $_SESSION[ ' ticket ' ];
    if ( ! isset($_POST[ ' ticket ' ]) || $_POST[ ' ticket ' ] != $ticket )
    {
        echo ' チケットの半券が無い、内容が一致しません！ ';
        exit;
    }
?>

```

先ず\$_SESSION 変数ができるように、session_start()を宣言します。

そして、変数\$ticket の値は新たに生成するのではなく、入力ページで作ったデータを格納します。\$_SESSION 変数はページを移動しても保持できているので、\$_SESSION['ticket ']の中にチケットデータが入っているはずですね。これを格納します。

さて、\$_POST[' ticket ']という記述があります。これは入力ページで<input>タグの name 属性に指定した値が配列で入っています。なので、name="ticket"で指定したチケットデータを読み出しています。ということは、名前のデータは\$_POST [' name ']、内容は\$_POST[' comment ']に入っている訳ですね。

話を戻します。このチケットデータが存在するか、もし存在したら入力ページで作ったチケットの内容が一致するか、を if 文で判定しています。

- ・ isset 関数

if 文の第一条件に「 !isset 」という記述があります。この関数は「その変数が存在するかどうか」を判定し、存在したら true、存在しなければ false を返すものです。そして isset 関数の頭に否定を表す「 ! 」が付いています。よってこの場合、「チケットを持ってきているかどうか」を判定しているわけです。もしこれが false だったら、チケットが無い、すなわち入力ページからのデータではない、ということになります。

次に、「 || 」で「または」という追加条件を指定しています。これでどちらか一方の条件が true ならば、処理を実行することが出来ます。二番目の追加条件は、「チケットの内容が入力ページで作ったものと同じかどうか」をチェックしてます。これがダメだったら、違う入力者であると判断でき、処理を実行させます。

その場合、直接 echo で文字列を出力し、exit (PHP の処理を終わらせる命令) で処理を終了させます。そのとき、特に HTML で何か出力させる必要はありません。なぜなら、この処理が実行されるのは、悪意のある人たちに向けてなのですから。

さて、チケットの判定が終わったらいよいよ入力データのチェックです。まずは、名前の入力チェックを行きましょう。

```
if ( ! isset( $_POST[ 'name' ] ))
{
    $error_name = '名前が入力されていません！';
}
```

これはチケットと同じで、「名前データが存在するかどうか」をチェックしています。もし存在しなければ、エラー表示用の変数にエラーメッセージを格納しています（まだ出力しません）。内容も同じように存在するかどうかチェックしましょう。

```
if ( ! isset( $_POST[ 'comment' ] ))
{
    $error_comment = '内容が入力されていません！';
}
```

これで、それぞれのデータがちゃんと送信されてきたかチェックができました。実際の場合、「文字数が何文字まで」とか、メールアドレスならば「アドレスの形式が正しいか」などをチェックする必要がありますが、その方法については補足をご参照ください。

チェックがOKだったかNGだったかを判定してみましょう。

```
if ( ! isset( $error_name ) && ! isset( $error_comment ))
{
    //チェックがOKだったときの処理
}
else
{
    //チェックがNGだったときの処理
}
```

今度は逆に、それぞれのエラーメッセージが存在しなければOKなので、このような変定式になりました。では先に、もしチェックがNGだった場合について考えて見ましょう。

チェックがNGだったときは「本当にお問い合わせをしてくれてるんだけど、入力してもらったデータに誤りがある人」である可能性が高いですよ。その場合、もう一度入力フォームに戻って再入力をしてもらわないといけません。よって、上記コードのチェックがNGだったときの処理は、「エラーメッセージを出力して、再度入力フォームを表示」させる処理が必要です。

その処理は簡単で、入力ページのHTMLソースをそのまま入れて、あとはエラーメッセージを任意の場所に echo すればよいだけです。

```
        else
        {
?>
//入力ページのHTML
<?php
        }
?>
```

ここでの注意点は、「HTML出力の部分は PHP コードの中に含めない」事です。そのため、HTMLソースを書く前に一度 ?> で PHP の処理に含めないようにしています。

あとは、エラーメッセージの出力ですね。これは、任意の場所に echo すればOKです。

```
<?php if ( isset($error_name) ) echo $error_name ; ?>
<?php if ( isset($error_comment) ) echo $error_comment ; ?>
```

ここではエラーメッセージを出力する前に、そのメッセージが存在したなら出力するようにしています。何故なら、「名前のチェックはOKだったけど、内容のチェックがダメだった」という場合もあるからです。勿論、その逆もありえます。

さらに、ユーザビリティを考える上で、入力フォームに戻った際にさっき入力したデータが入っていると嬉しいですよ。その処理も追記しましょう。

<input>、<textarea>タグの value 属性に、

```
<input type="text" name="name"
  value="<?php echo htmlspecialchars($_POST[ ' name ' ], ENT_QUOTES);?> />
<textarea name="comment">
  <?php echo htmlspecialchars($_POST[ ' comment ' ], ENT_QUOTES);?>
</textarea>
```

このように記述します。<textarea>の場合は記述する場所に注意してください。

さて、echo の際に何か関数を使っていますね。これもセキュリティ対策なのです。

入力チェックがエラーだったとき、入力データはそのままフォームに戻ります。

もしそのエラーだった入力データにスクリプトが混じっていたら、入力フォームに戻ったときにスクリプトが実行されてしまいます、これはマズイです。

そのため、「htmlspecialchars()関数」を用いてサニタイズを行っています。

この関数の動作は、「HTMLエンティティ変換」するものです・・・といっても分かりにくいので、例を挙げます。

例えば、

```
<script>alert( ' attacked! ' ); </script>
```

と入力されたとします。これがエラーで入力画面に戻ったとき、そのまま表示させると当然スクリプトは実行されます。これを htmlspecialchars()関数にかけると、

```
&gt;script&lt;alert( ' attacked! ' );&gt;/script&lt;
```

このようになり、スクリプトは実行されません。<script>タグではなくなっているからです。勿論見た目はちゃんと表示されているので大丈夫です。

「ENT_QUOTES」の部分はオプションです。詳細は省きますが、必ず指定するようにしましょう。

このように、入力データを再度HTML上表示させる際には必ず htmlspecialchars()を行うようにしましょう。うっかり忘れるとそこが攻撃者の糸口になってしまいますので。

では、チェックがOKだった場合のメール送信の処理に入りましょう。

4-3 メールを送信してみよう

メールを送信するにあたって、何か特殊なことをする必要はありません。PHP にはちゃんとメール送信用の関数が用意されています。

- ・ mail 関数、mb_send_mail 関数

メール送信関数には2種類あって、どちらもメール送信が可能なのですが、日本語のメールを送信するには mb_send_mail 関数のほうが良いので、今回はこちらを用います。もし興味があれば mail 関数でもやってみてください。

mb_send_mail 関数には、以下のように値を渡します。

```
mb_send_mail ( '送信先', '件名', '内容', [ 追加パラメータ ] )
```

送信先、件名、内容は必須で、追加パラメータは任意です。追加パラメータとは、例えば差出人の名前などです。今回は必須事項のみ設定しましょう。

```
mb_language( 'ja' );  
$name      = $_POST[ 'name' ];  
$comment   = $_POST[ 'comment' ];  
$mail_to   = ' test@hogehoge.com ';  
$body = <<<END
```

テストメール送信です。ちゃんと入力データが入ってるかな？

お名前 : { \$name }

内容 : { \$comment }

END;

```
$subject   = mb_encode_mimeheader( 'メール送信プログラム' );  
$result = mb_send_mail ( $mail_to , $subject , $comment );
```

一つずつ見ていきましょう。

まず始めに、「mb_language('ja')」という記述があります。これはメール送信時のお

まじないのようなもので、ja というのは日本語を表します。よって、「メールに送る文字列は日本語ですよ」と指定しているようなものです。

そして次に、分かりやすくするため、各入力内容を変数に格納しています。

そして、送信先のメールアドレス、件名を設定していますが、件名のときに何やら長い関数を使っていますね。

```
mb_encode_mimeheader( 文字列 )
```

この関数により、メールヘッダの部分は mime エンコードしています。mime エンコードについては省略しますが、この指定をしないと日本語は文字化けしてしまうので注意してください。あとは、mb_send_mail 関数に渡すだけです。

さて、ここで変数\$result に入っている値は何でしょうか？mb_send_mail 関数はメールを送信する関数なのですが・・・。

実は、mb_send_mail 関数は、メールが送信できたら true を、送信に失敗したら false を返します。これはとても便利なことで、これが true か false かで判定すれば、メール送信の成否が分かるようになっています。

よって、チェックがOKだった場合の処理は以下ようになります。

```
if ( ! isset( $error_name ) && ! isset( $error_comment ) )
{
    $name      =  $_POST[ ' name ' ];
    $comment   =  $_POST[ ' comment ' ];
    $mail_to   =  ' test@hogehoge.com ';
    $body      =  <<<END
```

テストメール送信です。ちゃんと入力データが入ってるかな？

お名前 : { \$name }

内容 : { \$comment }

END;

```
    $subject = mb_encode_mimeheader( ' メール送信プログラム ' );
    $result  = mb_send_mail ( $mail_to , $subject , $comment );
    if ( $result == true )
    {
```

```
        //メール送信成功時の処理
    }
    else
    {
        //メール送信失敗時の処理
    }

}
else
{
    //チェックがNGだったときの処理
}
}
```

あとは好きな送信完了画面とエラー画面を作成して完成です。
今回は私の方で完了画面も作成しておきました。

このような流れで、プログラムは処理されていきます。HTMLだけの場合とは違って少し複雑ですね。特に「check.php」などは<!DOCTYPE>の宣言が二回あるのには違和感を覚えるかもしれません。しかしながら、これはPHPで処理した結果、どちらかしか出力されませんので、結果として1つのHTMLとなる訳です。

さらに言えば、実際にエラーだった場合は記述したHTMLがそのまま出力されます。ということは当然、そのHTMLに対してCSSで装飾することが可能となります。今までフリーCGI等で出力の形式が決まっていたなかなか触れなかった部分も、皆さんで作成されたのですから、好きなように装飾することができますね。サイトのイメージに合わせることもできます。

さて、今回は簡単なメール送信プログラムを作ってみました。時間のため、駆け足になってしまった部分もあり、疑問に思われた点が多々あると思います。申し訳ございません。今回の内容で難しいと思われたり、疑問に思われた点がございましたら何でもご相談ください。

少しでもPHPの処理の流れや動きを理解していただけたなら光栄です。

本日は「デザイナーのためのプログラミング入門」にご参加頂き、誠にありがとうございました。ご感想、改善点、ご指摘をいただくと嬉しいです。また次回の開催の際にもご参加頂ければ感激です。

5. 発展 ～確認画面を作りたい～

今回のメール送信プログラムには確認画面が足りません。外国のサイトでは確認画面を設けていない場合が多いらしいのですが、日本のサイトでは確認画面は今や必須と言っても良いですね。日本ならではの、なのでしょう。

と言うわけで、今回のプログラムに確認画面を作る場合について少しだけ述べます。

同じように処理の流れをイメージしてみましょう。今回は確認画面を作りますから、確認表示用にHTMLファイルがもう一つ必要ですね。さらにメール送信部分のプログラムを「send.php」として分離し、check.phpは確認表示用にしましょう。

基本的な流れは変わりません。今回作成したプログラムに、入力チェックがOKだった場合、「一度確認画面を表示し、改めてメール送信を行う」ようにするだけです。

では、入力チェックがOKだった場合について考えましょう。

OKだった場合、メール送信のプログラムが書いてある部分を切り取り、「send.php」として別ファイルに保存しましょう。そして、確認画面用のHTMLをここに記述します。

<body>内に以下のように記述しましょう。

```
<p>入力内容確認</p>
<p>名前 : <?php echo htmlspecialchars($_POST[' name ' ], ENT_QUOTES);?></p>
<p>内容 : <?php echo
    nl2br(htmlspecialchars($_POST[' comment ' ], ENT_QUOTES);?></p>
<form action="send.php" method="post">
    <input type="hidden" name="name"
value="<?php echo htmlspecialchars($_POST[' name ' ], ENT_QUOTES);?> />
    <input type="hidden" name="comment"
value="<?php echo htmlspecialchars($_POST[' comment ' ], ENT_QUOTES);?> />
    <input type="hidden" name="ticket"
value="<?php echo $ticket ;?> />
```

```
<input type="submit" value="送信する" />
</form>
```

まずは表示画面に入力内容を出力します。ここで大事なものは…そう、htmlspecialchars です。忘れずに記述しましょう。

そして下の部分にまた<form>タグを書き、その中に全て hidden 属性にした<input>タグをおきます。これでブラウザ上では表示されませんが、実際にはデータとして送信するようになります。また画面遷移する必要がありますので、ワンタイムトークンを入れるのも忘れないでください。

これで入力画面から check.php に送信して、入力チェックがOKなら、確認画面が表示されます。あとは、もう一度この確認画面からデータを送信して、メールを送信します。

さて、この確認画面から送られたデータは send.php で受け取るわけですが、このデータについてどう思われますでしょうか？「一度 check.php で入力チェックしてるし、htmlspecialchars もやってるから正しいデータである」と思われる方もいらっしゃると思います・・・が、実はこのデータも「本当に正しいデータである」保証は残念ながらありません。何故でしょうか？

その理由は簡単です。例えば、check.php を表示せずに、いきなり send.php にデータを送信してくる可能性があるからです。人間の心理として、「send.php は check.php からのデータを受け取るものだ」と思ってしまいがちですが、そういった所に後し穴があるのでご注意ください。

よって、対策として、send.php でも同じようにもう一度ワンタイムチケットの確認、入力チェックを行う必要があります。

```
<?php
    session_start();
    $ticket = $_SESSION[ ' ticket ' ];
    if ( ! isset($_POST[ ' ticket ' ]) || $_POST[ ' ticket ' ] != $ticket )
    {
        echo ' チケットの半券が無いが、内容が一致しません！ ';
        exit;
    }
```

```

if ( ! isset( $_POST[ 'name' ] ))
{
    $error_name = '名前が入力されていません！';
}
if ( ! isset( $_POST[ 'comment' ] ))
{
    $error_comment = '内容が入力されていません！';
}

if ( ! isset( $error_name ) && ! isset( $error_comment ))
{
    $name      = $_POST[ ' name ' ];
    $comment   = $_POST[ ' comment ' ];
    $mail_to   = ' test@hogehoge.com ';
    $body      = <<<END

```

テストメール送信です。ちゃんと入力データが入ってるかな？

お名前 : { \$name }

内容 : { \$comment }

END;

```

    $subject = mb_encode_mimeheader( ' メール送信プログラム ' );
    $result = mb_send_mail ( $mail_to , $subject , $comment );
    if ( $result == true )
    {
        //メール送信完了画面
    }
    else
    {
        //メール送信失敗画面
    }
}

```

```
else
{
    //チェックがNGだったときの処理
}
```

このようになります。

check.php で入力チェックを行い、そこから送られたデータを再度チェックして、OKだったなら初めてメール送信、という流れです。

つまり、チェックの手間がもう一つ増えるわけですね。勿論、画面遷移が一つ増える、と言うことはその分セキュリティのリスクも増えますので気をつけてください・・・と言いたいところですが、確認画面が無いとお話にならない、と言ったケースが殆どなので、その分セキュリティ周りはしっかりと処理しましょう。

6. 補足

・ 色々な入力チェック

今回は名前と内容のみでしたが、実際はメールアドレスや文字数のチェックも必要かと思えます。そこで、これらのチェックの仕方を簡単にご説明します。

・ メールアドレスのチェック

メールアドレスの入力はほぼ必須といっても良いでしょう。

チェックの方法として、「メールアドレスの形式が正しいかどうか」をチェックする必要があります。他にも「入力されたメールアドレスが本当に存在するか」などのチェックもありますが、高度なため、ここでは割愛させていただきます。チェックするコードは以下のようになります。

```
if(!preg_match("/^[a-z0-9\+\_\-]+\.[a-z0-9\+\_\-]+@[a-z0-9\-\+\.]+[a-z]{2,6}$/ix", $_POST['email']))
{
    $error = 'メールアドレスの形式が正しくありません！'
}
```

とってもややこしいですね。簡単に説明すると、

〇〇〇〇@〇〇〇〇.com 等の形式であるかを「正規表現」という手法を用いてチェックし、その形式でなければ false を返すようにしてあります。詳しく知りたい方は「正規表現」で

検索してみてください。トピックが沢山あります。

- ・ 文字数のチェック

よくパスワードなどでは「○文字以上、○文字以内」という表現があります。これを実装するには、文字数のチェックが必要となります。

PHP にはちゃんと文字数をカウントしてくれる関数があります、なので、これを使ってチェックしてみましょう。

- ・ strlen()関数、mb_strlen()関数

文字数をカウントするには2種類の関数があります。strlen 関数、そしてその頭に「mb_」をついた mb_strlen 関数です。この mb_ というのは「マルチバイト」という意味で、ここでは日本語のことを指すと思っていただいて構いません。

通常の半角英数をチェックするのは strlen 関数ですが、ここでは内容などの日本語入力に対してチェックを行いますので、mb_strlen 関数を使います。

```
if ( mb_strlen( $_POST[ ' comment ' ] ) > 300 )
{
    $error = ' 文字数は 3 0 0 文字以内で入力してください。 ';
}
```

「mb_strlen 関数でのカウント数が 300 より大きいなら～」という条件でチェックしています。文字数の制限によっては 300 の数字は変わるでしょう。

このように、入力チェックには様々な関数を用いて、厳密にチェックする必要があります。他にもラジオボタンのチェックや、セレクトボックスのチェックなどもありますが、基本的には今回のような方法でチェックが出来ますので、是非実装してみてください。