[/home/s1211436/www_local/wp/bbs_js.rb]

```
#!/usr/bin/env ruby
# encoding: utf-8
require 'cgi'
cgi = CGI.new
print cgi.header("text/html; charset=utf-8")
data = open("bbsdata.txt", "r:utf-8")
messages = CGI.escapeHTML(data.read)
data.close

print <<-EOF
<!doctype html>
<html lang="ja">
<head>
    <meta charset="utf-8" />
    <title>1行掲示板</title>
</head>
<body>

    <h1>1行掲示板</h1>
    <form action="update.rb" method="post" name="post">
        <div>メッセージ：<input type="text" name="message"></div>
        <div>お名前：<input type="text" name="name"></div>
        <div>
            <input type="submit" value="書き込む">
            <input type="reset" value="クリア">
        </div>
    </form>
    <pre>#{messages}</pre>

<script>

    String.prototype.h = function() {
        var s = this;
        s = s.replace(/&/g, '&amp;');
        s = s.replace(/</g, '&lt;');
        s = s.replace(/>/g, '&gt;');
        s = s.replace(/"/g, '&quot;');
        return s;
    }

    function pre_check() {
        var err = [];
        empty_check(err);
        html_check();
        console.log("ここ来る？");
        if (err.length > 0) {
            alert(err.join("\\n"));
            return false;
        }
        return true;
    }
```

せっかくだからJavaScriptの勉強をしてみようと思って作ったが、このメソッドは不要になった。その理由はhtml_check()で述べる。
JavaScriptのオブジェクト指向言語としての機能を今まで勉強してこなかった。良い機会なので少し調べた。
個々のインスタンスにメソッドを追加することは分かったが、クラスそのものにメソッドを追加するにはprototype属性というものを使うらしい。JavaScriptは、RubyやJavaで慣れ親しんだクラスベースではなくプロトタイプベースのオブジェクト指向らしい。一応メソッドを追加することはできたものの、「インスタンス化と継承を区別しない」という概念がまだ理解できていない。

どうしたらエラーチェックをごちゃごちゃさせずに書けるかを考えたらこのようになった。たぶんエラーチェック項目が多岐に及んでも対応できる方法だと思う。
htmlタグが含まれているだけで投稿できなくするのは可哀想なので、html_check()にはerr配列を渡さないことにしている（「こういうHTMLを書いたんだけどうまく動かないんだ！」って掲示板に相談したいじゃないですか)。

```
    function html_check() {
        var message = document.forms["post"]["message"].value;
        // document.forms["post"]["message"].value = message.h();
        if (message.match(/<\\S+?>/)) {
            alert("htmlタグはそのまま表示されます。")
        }
    }

    function empty_check(err) {
        tmp = [];
        if(document.forms["post"]["name"].value.length == 0) {
            tmp.push("名前");
        }
        if(document.forms["post"]["message"].value.length == 0) {
            tmp.push("メッセージ");
        }
        if (tmp.length > 0) {
            err.push(tmp.join("・") + "を入力してください");
        }
    }

    function init() {
        document.forms["post"].onsubmit = function() { return pre_check(); };
    }

    document.addEventListener("DOMContentLoaded", init, false);

</script>
</body>
</html>
EOF
```

h()メソッドが不要になった理由を述べる。投稿データをテキストファイルから読み込むときにRubyでエスケープをしているからである。二重にエスケープをしてしまうと、例えば、<script>は、<script>に変換されて格納され、読み込むときに<script>になってしまう。このように表示がおかしくなっているWebページを何度か見たことがあるが、なるほどこういうことだったのかと思った。

では、RubyとJavaScriptどちらでエスケープすべきかという問題になるが、Rubyでエスケープした方が良い。理由は、このフォームを使わなくても投稿できてしまうためJavaScriptによるチェックはスルーされてしまうからである。

最近はインラインにJavaScriptを書くべきでないと言われているらしい。

はじめは以下のように書いていた。

```
window.onload = function() {
    document.forms["post"].onsubmit = function() { return pre_check(); };
}
```

だが調べていると、onloadイベントは、DOMツリーの取得だけでなく、画像ファイルなども全て読み込み終わってから発生するイベントらしいので重いページで使うには好ましくない。
jQueryの$(document).readyは、DOMツリーを取得した段階で呼び出されるらしいことを知った。次に、以下のページでDOMContentLoadedイベントを使えば良いと書いてあるのを見つけた。
http://hisasann.com/housetect/2008/01/jqueryreadyonload.html
このページにDOMContentLoadedイベントをサポートしていないIEであっても、全ての要素が読み込み終わる前に$(document).readyが呼ばれていると書いてあったので、jQueryのソースに当たってみると、IEではonreadystatechangeイベントが使われていた。IE以外ではDOMContentLoadedイベントが使われていた。

次ページにjQuery-1.10.2を引用する。
IE用のコードの行数が凄い。今までjQueryのソースコードなんて見なかったのだけれど、IE用の行数が想像以上だった。

String.matchメソッド内で例外処理が使われているようだ。以前JavaScriptの挙動が不思議だなと思っていた原因はこれらしい。今は例外処理を理解したので挙動が理解できる。
html_check()で、if (message.match(/<\\S+?>/).length > 0) としてしまうと、マッチするものがなかった場合matchメソッドが例外を発生するようだ。するとpre_check()はfalseもtrueも返さない。そのとき、formはsubmitされてしまうようだ。

[http://code.jquery.com/jquery-1.10.2.jsから引用]

```
jQuery.ready.promise = function( obj ) {
    if ( !readyList ) {

        readyList = jQuery.Deferred();

        if ( document.readyState === "complete" ) {
            // Handle it asynchronously to allow scripts the opportunity to delay ready
            setTimeout( jQuery.ready );

        // Standards-based browsers support DOMContentLoaded
        } else if ( document.addEventListener ) {
            // Use the handy event callback
            document.addEventListener( "DOMContentLoaded", completed, false );

            // A fallback to window.onload, that will always work
            window.addEventListener( "load", completed, false );

        // If IE event model is used
        } else {
            // Ensure firing before onload, maybe late but safe also for iframes
            document.attachEvent( "onreadystatechange", completed );

            // A fallback to window.onload, that will always work
            window.attachEvent( "onload", completed );

            // If IE and not a frame
            // continually check to see if the document is ready
            var top = false;

            try {
                top = window.frameElement == null && document.documentElement;
            } catch(e) {}

            if ( top && top.doScroll ) {
                (function doScrollCheck() {
                    if ( !jQuery.isReady ) {

                        try {
                            // Use the trick by Diego Perini
                            // http://javascript.nwbox.com/IEContentLoaded/
                            top.doScroll("left");
                        } catch(e) {
                            return setTimeout( doScrollCheck, 50 );
                        }

                        // detach all dom ready events
                        detach();

                        // and execute any waiting functions
                        jQuery.ready();
                    }
                })();
            }
        }
    }
    return readyList.promise( obj );
};
```