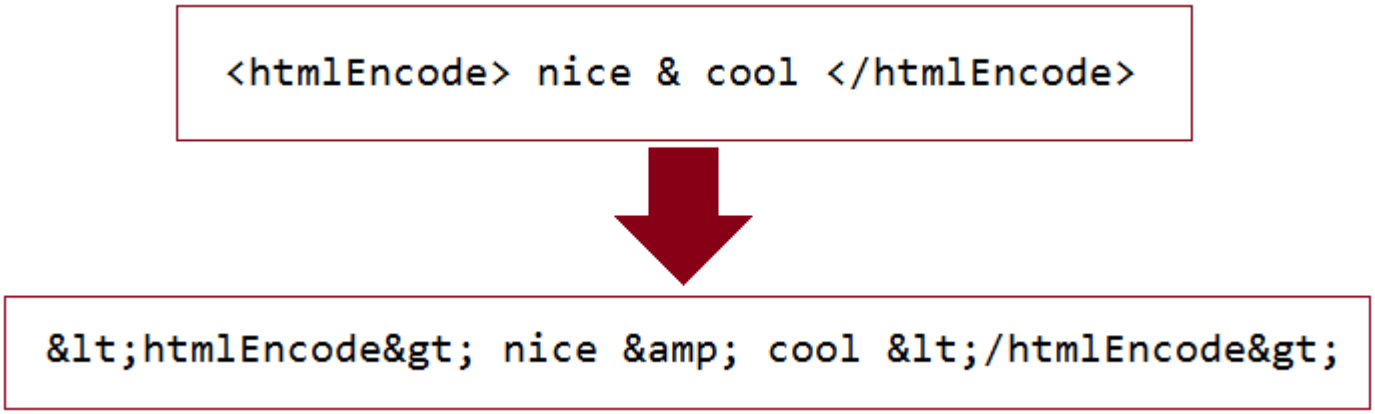
عنوان: **Html Encoding**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۵/۰۷ ۲۲:۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, HTML, Html Encoding, HtmlEncode, HtmlDecode




.

.

## مقدمه

در دنیای وب دو انکدینگ معروف داریم: Url Encoding و Html Encoding. در هر کدام از این انکدینگ‌ها یک عملیات کلی صورت می‌گیرد: *تبدیل کاراکترهای غیرمجاز به عبارات معادل مجاز* .

**Url Encoding** همان‌طور که از نامش پیداست روشی برای کدکردن Url هاست. مثل عبارت کدشده زیر:

```
Hello%20world%20,%20hi
```

درواقع کاراکتر مشخص‌کننده رشته‌ای که Url Encoding احتمالا در آن اعمال شده است، همان کاراکتر **%** است. بحث درباره این نوع انکدینگ کمی مفصل است که خود مطلب جداگانه‌ای می‌طلبد. ( اطلاعات بیشتر )

**Html Encoding** نیز با توجه به نامش برای انکدینگ عبارات HTML استفاده می‌شود. مثلا عبارت زیر را درنظر بگیرید:

```
<html>encoding</html>
```

این عبارت پس از اعمال عملیات Html Encoding به صورت زیر در خواهد آمد:

```
&lt;html&gt;encoding&lt;/html&gt;
```

می‌بینید که در اینجا کاراکترهای > و < به صورت عبارات **<** و **>** در آمده‌اند. شرح کاملی درباره این عبارات معادل (که اصطلاحا به آن‌ها character entity می‌گویند) در اینجا آورده شده است.

در حالت کلی Html Encoding شامل کدکردن 5 کاراکتر زیر است:

.

| کاراکتر | عبارت معادل | توضیحات |
|---|---|---|
| > | > | |
| < | < | |
| " | " | |
| ' | ' | یا ' به غیر از IE |
| & | & | |

**نکته:** در برخی استانداردها (بیشتر برای XML) برای کاراکتر ' از عبارت ' استفاده می‌شود. این عبارت جایگزین به غیر از IE در بقیه مرورگرها درست کار می‌کند.

این کاراکترها درواقع از عناصر اصلی تشکیل‌دهنده ساختار Html هستند، بنابراین وجود آن‌ها درون یک متن می‌تواند در روند رندر صفحات html اختلال ایجاد کند. بنابراین با استفاده از Html Encoding و تبدیل این کاراکترها به معادلشان (عباراتی که مرورگرها آن‌ها را می‌شناسند)، می‌توان از نمایش درست این کاراکترها مطمئن شد. البته یکی دیگر از دلایل مهم اعمال این انکدینگ، افزایش امنیت و جلوگیری از حملات XSS است.

فرمت این عبارات معادل به صورت **&entity_name;** است. به کل این عبارت اصطلاحا Character Entity گفته می‌شود. این عبارات با کاراکتر **&** شروع شده و به یک کاراکتر **;** ختم می‌شوند. کلمه میان این دو کاراکتر نیز عبارت جایگزین (یا همان entity name) هر یک از این کاراکترهاست که در لینک بالا به همراه بسیاری دیگر از کاراکترها اشاره شده است ( ^ ).
روش دیگری نیز برای کدکردن کاراکترها با فرمت **&#entity_number;** وجود دارد. این entity_number درواقع کد کاراکتر مربوطه در جدول کاراکترست جاری مرورگر است. معمولا این کدها منطبق بر جدول ASCII هستند. برای کاراکترهای خارج از جدول اسکی هم از سایر جداول (مثلا یونیکد) استفاده می‌شود. عملیات انکدینگ برای کاراکترهای با کد 160 تا 255 (براساس استاندارد ISO-8859-1) با این روش انجام می‌شود ( ^ ). اطلاعات بیشتر راجع به این کدها در اینجا آورده شده است.

خوشبختانه در سمت سرور، در دات‌نت روش‌های گوناگون و قابل اطمینانی برای اعمال این انکدینگ وجود دارد. اما متاسفانه در سمت کلاینت چنین امکاناتی اصلا فراهم نیست و برنامه نویسان خود باید دست به کار شوند. ازآنجاکه امروزه قسمت‌های بیشتری از اپلیکیشن‌های تحت وب در سمت کلاینت پیاده می‌شوند و کتابخانه‌های سمت کلاینت روز به روز پرطرفدارتر می‌شوند وجود نمونه‌های مشابه از این متدها در سمت کلاینت می‌تواند بسیار مفید باشد.
بنابراین تمرکز اصلی ادامه این مطلب بیشتر بر نحوه اعمال این انکدینگ در سمت کلاینت با استفاده از زبان جاوا اسکریپت است.

.

.

## Html Encoding در دات‌نت

در دات‌نت متدهای متعددی برای اعمال Html Encoding وجود دارد. برخی از آن‌ها صرفا برای اسناد HTML طراحی شده‌اند و برخی دیگر یک پیاده‌سازی کلی دارند و بعضی نیز برای فایل‌های XML ارائه شده‌اند. این متدها عبارتند از:

متد System.Security.SecurityElement.Escape : این متد بیشتر برای اعمال این انکدینگ در XML به‌کار می‌رود. در این متد 5 کاراکتر اشاره شده در بالا به عبارات معادل انکد می‌شوند. البته برای کاراکتر ' از عبارت ' استفاده می‌شود.

متدهای موجود در System.Net.WebUtility : متدهای HtmlEncode و HtmlDecode موجود در این کلاس عملیات انکدینگ را انجام می‌دهند. این کلاس از دات‌نت 4 اضافه شده است.

متدهای کلاس System.Web.HttpUtility : در این کلاس از متدهای موجود در کلاس System.Web.Util.HttpEncoder استفاده می‌شود. در پیاده‌سازی پیش‌فرض، متدهای این کلاس از متدهای موجود در کلاس WebUtility استفاده می‌کنند. البته می‌توان با فراهم کردن یک Encoder سفارشی و تنظیم آن در فایل کانفیگ (خاصیت encoderType در قسمت HttpRuntime ) این رفتار را تغییر داد. دلیل اصلی جابجایی مکان پیاده‌سازی این متدها از دات نت 4 به بعد نیز به همین دلیل است. (اطلاعات بیشتر ^ و ^ ).

متدهای موجود در System.Web.HttpServerUtility : متدهای HtmlEncode و HtmlDecode موجود در این کلاس مستقیما از متدهای موجود در کلاس HttpUtility استفاده می‌کنند. خاصیت Server موجود در HttpContext یا در کلاس Page از نوع این کلاس است.

متدهای موجود در کلاس System.Web.Security.AntiXss.AntiXssEncoder : این کلاس از دات نت 4.5 اضافه شده است. همانطور که از نام این کلاس بر می‌آید، از HttpEncoder مشتق شده است که در متدهای مرتبط با html encoding تغییراتی در آن اعمال شده است. متدهای این کلاس برای امنیت بیشتر به جای استفاده از Black List از یک White List استفاده می‌کنند.

درحال حاضر بهترین گزینه موجود برای عملیات انکدینگ، متدهای موجود در کلاس WebUtility هستند. ازآنجاکه این کلاس در فضای System.Net و در کتابخانه System.dll قرار دارد (کتابخانه‌ای که معمولا برای تمام برنامه‌های دات‌نتی نیاز است)، بنابراین بارگذاری آن در برنامه نیز بار اضافی بر حافظه تحمیل نمی‌کند.
پیاده‌سازی عملیات HtmlEncode کار سختی نیست. مثلا می‌توان برای سادگی از متد Replace استفاده کرد. اما برای رشته‌های طولانی این متد کارایی مناسبی ندارد. به همین دلیل در تمام پیاده‌سازی‌ها، معمولا از یک حلقه بر روی تمام کاراکترهای رشته موردنظر برای یافتن کاراکترهای غیرمجاز استفاده می‌شود. در کدهای متدهای موجود، برای افزایش سرعت حتی از اشاره‌گر و کدهای unsafe نیز استفاده شده است.
برای افزایش کارایی در تولید رشته نهایی تبدیل‌شده، بهتر است از یک StringBuilder استفاده شود. در پیاده‌سازی‌های متدهای بالا برای اینکار معمولا از یک TextWriter استفاده می‌شود. TextWriterهای موجود از کلاس StrigBuilder برای دستکاری رشته‌ها استفاده می‌کنند.

صرفا جهت آشنایی بیشتر، پیاده‌سازی خلاصه‌شده متد HtmlEncode در کلاس WebUtility در زیر آورده شده است:

```
public static unsafe void HtmlEncode(string value, TextWriter output)
{
  int index = IndexOfHtmlEncodingChars(value, 0);
  if (index == -1)
  {
    output.Write(value);
    return;
  }
  int cch = value.Length - index;
  fixed (char* str = value)
  {
    char* pch = str;
    while (index-- > 0)
    {
      output.Write(*pch++);
    }
    while (cch-- > 0)
    {
      char ch = *pch++;
      if (ch <= '>')
      {
        switch (ch)
        {
```

```
          case '<':
            output.Write("&lt;");
            break;
          case '>':
            output.Write("&gt;");
            break;
          case '"':
            output.Write("&quot;");
            break;
          case '\'':
            output.Write("&#39;");
            break;
          case '&':
            output.Write("&amp;");
            break;
          default:
            output.Write(ch);
            break;
        }
      }
      else if (ch >= 160 && ch < 256)
      {
        // The seemingly arbitrary 160 comes from RFC
        output.Write("&#");
        output.Write(((int)ch).ToString(NumberFormatInfo.InvariantInfo));
        output.Write(';');
      }
      else
      {
        output.Write(ch);
      }
    }
  }
}
private static unsafe int IndexOfHtmlEncodingChars(string s, int startPos)
{
  int cch = s.Length - startPos;
  fixed (char* str = s)
  {
    for (char* pch = &str[startPos]; cch > 0; pch++, cch--)
    {
      char ch = *pch;
      if (ch <= '>')
      {
        switch (ch)
        {
          case '<':
          case '>':
          case '"':
          case '\'':
          case '&':
            return s.Length - cch;
        }
      }
      else if (ch >= 160 && ch < 256)
      {
        return s.Length - cch;
      }
    }
  }
  return -1;
}
```

در ابتدا بررسی می‌شود که آیا اصلا متن ورودی حاوی کاراکترهای غیرمجاز است یا خیر. درصورت عدم وجود چنین کاراکترهایی، کار متد با برگشت خود متن ورودی پایان می‌یابد. درغیراینصورت عملیات انکدینگ آغاز می‌شود. همان‌طور که می‌بینید عملیات انکدینگ برای 5 کاراکتر اشاره شده به صورت جداگانه انجام می‌شود و برای کاراکترهای با کد 160 تا 255 (با توجه به توضیحات موجود در مقدمه) نیز با استاندارد &#code; عملیات تبدیل انجام می‌شود.

در سمت دیگر، پیاده‌سازی بهینه متد HtmlDecode چندان ساده نیست. چون به جای یافتن یک کاراکتر غیرمجاز باید به دنبال عبارات چند کاراکتری معادل گشت که کاری نسبتا پیچیده است.

اطلاعات و پیاده‌سازی نسبتا کاملی درباره Html Encoding در سمت سرور در اینجا قابل مشاهده است.

**نکته:** درصورت نیاز به کدکردن سایر کاراکترها (مثلا کاراکترهای یونیکد) پیاده‌سازی‌های موجود کارا نخواهند بود. بنابراین باید encoder سفارشی خود را تهیه کنید. مثلا می‌توانید شرط دوم در بررسی کد کاراکترها را بردارید (منظور قسمت ch < 256) که در این‌صورت متد شما محدوده وسیعی را پوشش می‌دهد. اما دقت کنید که با این تغییر متدی سفارشی برای عملیات decode نیز باید تهیه کنید!

.

.

## Html Encoding در جاوا اسکریپت

برای انجام عملیات Url Encoding در جاوا اسکریپت چند متد توکار وجود دارد، که فرایند کلی عملیات همه آن‌ها تقریبا یکسان است. اما متاسفانه برای انجام عملیات Html Encoding متدی در جاوا اسکریپت وجود ندارد. بنابراین متدهای مربوطه باید توسط خود برنامه‌نویسان پیاده‌سازی شوند.

یک روش برای اینکار استفاده از لیست اشاره‌شده در بالا و انجام عملیات replace برای تمام این کاراکترهاست (5 کاراکتر اصلی و درصورت نیاز سایر کاراکترها). این کار می‌تواند کمی سخت باشد و درواقع پیاده‌سازی چنین متدی نسبتا مشکل نیز هست (مخصوصا عملیات decode).
اما خوشبختانه امکانی در اسناد html وجود دارد که این کار (مخصوصا Decode کردن) را آسان می‌کند.

این روش جالب برای انجام عملیات Html Encoding در جاوا اسکریپت، استفاده از یک قابلیت توکار در مرورگرهاست. عناصر DOM (مانند div) دو خاصیت **innerText** و **innerHTML** دارند که مرورگرها با توجه به مقادیر تنظیم‌شده برای هر یک، عملیات coding و decoding مربوطه را به صورت کاملا خودکار انجام داده و مقدار خاصیت دیگر را به‌روزرسانی می‌کنند (دقت کنید که در این دو پراپرتی، کلمه HTML کاملا با حروف بزرگ است، برخلاف Text که تنها حرف اول آن بزرگ است).

برای روشن‌تر شدن موضوع به مثال زیر برای عملیات encode توجه کنید:

```
<div id="log"></div>
<script type="text/javascript">
  var element = document.getElementById('log');
  element.innerText = '<html> encoding </html>';
  console.log(element.innerHTML);
</script>
```

که خروجی زیر را خواهد داشت:

```
&lt;html&gt; encoding &lt;/html&gt;
```

عکس این عملیات یعنی decoding نیز با استفاده از کدی مثل زیر امکان‌پذیر است:

```
<div id="log">
</div>
<script type="text/javascript">
  var element = document.getElementById('log');
  element.innerHTML = "&lt;html&gt; encoding &lt;/html&gt;";
  console.log(element.innerText);
</script>
```

خروجی کد بالا به صورت زیر است:

```
<html> encoding </html>
```

می‌بینید که با استفاده از این ویژگی جالب، می‌توان عملیات Html Encoding را انجام داد. در ادامه پیاده‌سازی مناسب این دو متد آورده شد است.

. .

.

**متد htmlEncode**

برای پیاده‌سازی این متد برای حالت استفاده مستقیم داریم:

```
String.htmlEncode = function (s) {
  var el = document.createElement("div");
  el.innerText = s || '';
  return el.innerHTML;
};
```

در اینجا با استفاده از متد createElement شی document یک المان DOM (در اینجا div) ایجاد شده و سپس با توجه به توضیحات بالا خاصیت innerText آن به مقدار ورودی تنظیم می‌شود. استفاده از عبارت '' || s در اینجا برای جلوگیری از برگشت عبارات ناخواسته (مثل undefined یا null) برای ورودی‌های غیرمجاز است. درنهایت خاصیت innerHTML این المان به عنوان رشته انکدشده برگشت داده می‌شود.

نحوه استفاده از این متد به صورت زیر است:

```
console.log(String.htmlEncode("<html>"));
//result:   &lt;html&gt;
```

و برای حالت استفاده از خاصیت prototype داریم:

```
String.prototype.htmlEncode = function () {
  var el = document.createElement("div");
  el.innerText = this.toString();
  return el.innerHTML;
};
```

نحوه استفاده از این متد نیز به صورت زیر است:

```
console.log("<html>".htmlEncode());
//result:    &lt;html&gt;
```

.

.

**متد htmlDecode**

با استفاده از مطالب اشاره‌شده در بالا، پیاده‌سازی این متد به صورت زیر است:

```
String.htmlDecode = function (s) {
  var el = document.createElement("div");
  el.innerHTML = s || '';
  return el.innerText;
};
```

و به‌صورت خاصیتی از prototype شی String داریم:

```
String.prototype.htmlDecode = function () {
  var el = document.createElement("div");
```

```
  el.innerHTML = this.toString();
  return el.innerText;
};
```

نحوه استفاده از این متدها هم به صورت زیر است:

```
console.log(String.htmlDecode("&lt;html&gt;"));
console.log("&lt;html&gt;".htmlDecode());
```

.

.

## پیاده‌سازی با استفاده از jQuery

درصورت در دسترس بودن کتابخانه jQuery، کار پیاده‌سازی این متدها بسیار ساده‌تر خواهد شد. برای این‌کار می‌توان از متدهای زیر استفاده کرد:

.

- متد **htmlEncode** :

```
String.htmlEncode = function (s) {
  return $('<div/>').text(value).html();
};

String.prototype.htmlEncode = function () {
  return $('<div/>').text(this.toString()).html();
};
```

.

- متد **htmlDecode** :

```
String.htmlDecode = function (s) {
  return $('<div/>').html(s).text();
};

String.prototype.htmlDecode = function () {
  return $('<div/>').html(this.toString()).text();
};
```

.

.

## نکات پایانی

**1.** با اینکه به نظر می‌رسد در متدهای ارائه شده در بالا، بین نسخه‌های معمولی و نسخه مخصوص jQuery تفاوتی وجود ندارد اما تست زیر نشان می‌دهد که نکات ریزی باعث به‌وجود آمدن برخی تفاوت‌ها می‌شود. رشته زیر را درنظر بگیرید:

```
var value = "a \n b";
```

با استفاده از متد htmlEncode معمولی نشان داده شده در بالا، عبارت انکدشده رشته فوق به صورت زیر خواهد بود:

"a <br> b"

می‌بینید که به صورت هوشمندانه‌ای! مقدار \n به تگ <br> انکد شده است. اما اگر با استفاده از متد نوشته شده با jQuery سعی

به انکدکردن این رشته کنیم، می‌بینیم که مقدار \n بدین صورت انکد نمی‌شود! حال کدام روش درست و استاندارد است؟

در ابتدای این مطلب هم اشاره شده بود که Html Encoding برای کدکردن یکسری کاراکتر غیرمجاز در متون موجود در صفحات HTML بکار می‌رود و معمولا همان 5 کاراکتر اشاره‌شده در بالا به عنوان کاراکترهای اصلی غیرمجاز به حساب می‌آیند. کاراکتر \n از این نوع کاراکترها محسوب نمی‌شود. هم‌چنین ازآنجاکه عملیات عکس این تبدیل در Decode مربوطه صورت نمی‌گیرد، تبدیل این کاراکتر به معادلش در html اصلا کاری منطقی نیست و باعث خراب شدن متن موردنظر می‌شود.

با استفاده از متدهای HtmlEncode موجود در کلاس‌های دات نت (WebUtility و HtmlUtility که در بالا به آن‌ها اشاره شده بود) عملیات انکدینگ برای این رشته تکرار شد و نتیجه حاصله نشان داد که عبارت \n در خروجی این متدها نیز انکد نمی‌شود. بنابراین متد نوشته شده با استفاده از jQuery خروجی‌های استانداردتری ارائه می‌دهد.

با کمی تحقیق و بررسی کدهای jQuery مشخص شد که دلیل این تفاوت، در استفاده از متد createTextNode از شی document در متد ()text است. بنابراین برای بهبود متد htmlEncode اولیه داریم:

```
String.htmlEncode = function (s) {
  var el = document.createElement("div");
  var txt = document.createTextNode(s);
  el.appendChild(txt);
  return el.innerHTML;
};
```

با استفاده از این متد نتایج مشابه متد نوشته شده با jQuery حاصل خواهد شد.

.

.

**2.** نکته مهم دیگری که باید بدان توجه داشت برقراری اصل مهم زیر در عملیات انکدینگ است:

```
String.htmlDecode(String.htmlEncode(myString)) === myString;
```

حال سعی می‌کنیم که برقراری این شرط را در یک مثال بررسی کنیم:

```
var myString = "<HTML>";
String.htmlDecode(String.htmlEncode(myString)) === myString;
// result:   true
// ----------------------------------------------------------------------------
myString = "<اچ تی ام ال>";
String.htmlDecode(String.htmlEncode(myString)) === myString;
// result:   true
```

تا اینجا همه چیز ظاهرا درست پیش رفته است. اما حالا مثال زیر را درنظر بگیرید:

```
myString = "a \r b";
String.htmlDecode(String.htmlEncode(myString)) === myString;
// result:   false
```

می‌بینید که با وارد شدن کاراکتر \r کار خراب می‌شود. این نتیجه برای تمامی متدهای جاوا اسکریپتی نشان داده شده صادق است. اما متدهای دات نتی اشاره شده در ابتدای این مطلب با این کاراکتر مشکلی ندارند و نتیجه درستی برمی‌گردانند. بنابراین یک جای کار می‌لنگد!
پس از کمی تحقیق و بررسی بیشتر مشخص شد که مرورگرها در تبدیل کاراکترها، کاراکتر carriage return (یا CR یا همان \r با کد اسکی 13 یا 0D) را تبدیل به کاراکتر line feed (یا LF یا \n با کد اسکی 10 یا 0A) می‌کنند. برای آزمایش این نکته می‌توانید از سه خط زیر استفاده کنید:

```
console.log(escape(String.htmlDecode('\r'))); // result:    %0A  :  it is url encode of character '\n'
```

```
console.log(escape(String.htmlDecode('\n'))); // result:    %0A
console.log(escape(String.htmlDecode('\r\n'))); // result:    %0A
```

با بررسی بیشتر مشخص شد که این تبدیل به محض مقداردهی به یکی از خاصیت‌های یک عنصر DOM صورت می‌گیرد. برای مثال کد زیر را در مرورگرهای مختلف امتحان کنید:

```
var el = document.createElement('div');
el.innerText = '\r';
console.log(escape(el.innerText)); // result:    %0A
el.innerHTML = '\r';
console.log(escape(el.innerHTML)); // result:    %0A
console.log(escape('\r')); // result:    %0D
```

با بررسی هایی که من کردم دلیل و یا راه‌حلی برای این مشکل پیدا نکردم!
بنابراین در استفاده از این متدها باید این نکته را مدنظر قرار داد. ازآنجاکه این مشکل حالتی به خصوص دارد نمی‌توان راه‌حلی کلی برای آن ارائه داد. پس برای موقعیت‌های گوناگون با توجه به زوایای روشن‌شده از این مشکل باید به دنبال راه‌حل مناسب بود.

البته ممکن است این اشکال درمورد کاراکترهای دیگری هم وجود داشته باشد که من به آن برخورد نکرده باشم (با درنظر گرفتن تفاوت میان مرورگرهای مختلف ممکن است پیچیده‌تر هم باشد).

**نکته:** ازآنجاکه برای رفع این مشکل، پیاده‌سازی متد htmlDecode به این کاملی، با عدم استفاده از ویژگی پراپرتی‌های innerHTML و innerText، کاری نسبتا سخت و پیچیده و طولانی است، بنابراین در بیشتر حالات می‌توان از این مشکل صرف‌نظر کرد! به همین دلیل در اینجا نیز متد دیگری برای رفع این مشکل ارائه نمی‌شود!
.

.

**3.** یک مشکل دیگر که این متدها دارند این است که متاسفانه در متد htmlEncode، از 5 کاراکتر معروف بالا، کاراکترهای ' و " در این متدها اصلا تبدیل نمی‌شوند. همچنین سایر کاراکترهای عنوان‌دار یا کاراکترهای خارج از جدول ASCII (مثلا کاراکترهای با کد 160 تا 255 یا کاراکترهای یونیکد) نیز که معمولا انکد می‌شوند در این متد تغییری نمی‌کنند و به همان صورت برگشت داده می‌شوند.
هرچند متد htmlDecode نشان داده شده در این مطلب، به‌درستی تمامی عبارات معادل (حتی عبارات معادل غیر از 5 کاراکتر نشان داده شده در بالا با هر دو استاندارد &character-entity; و &#code;) را تبدیل کرده و کاراکتر درست را برمی‌گرداند.

برای اصلاح این مشکل می‌توان متد htmlEncode را کاملا به صورت دستی و مستقیم نوشت و اعمال انکدینگ‌های موردنیاز را با استفاده یک حلقه روی تمام کاراکترها متن موردنظر انجام داد. چیزی شبیه به کد زیر:

```
String.htmlEncode = function (text) {
  text = text || '';
  var encoded = '';
  for (var i = 0; i < text.length; i++) {
    var c = text[i];
    switch (c) {
      case '<':
        encoded += '&lt;';
        break;
      case '>':
        encoded += '&gt;';
        break;
      case '&':
        encoded += '&amp;';
        break;
      case '"':
        encoded += '&quot;';
        break;
      case "'":
```

```
        encoded += '&#39;';
        break;
      default:
        // the upper limit can be removed to support more chars...
        var code = c.charCodeAt();
        if (code >= 160 & code < 256)
          encoded += '&#' + code + ';';
        else
          encoded += c;
    }
  }
  return encoded;
};
```

روش استفاده شده در متد بالا همانند متد HtmlEncode در کلاس WebUtility است.

.

.

## کتابخانه‌های موجود

هرچند توضیحات ارائه شده در این مطلب کافی هستند، اما صرفا برای آشنایی با سایر کتابخانه‌های موجود، روش‌های استفاده‌شده در آن‌ها و نقایص و مزایای آن‌ها این قسمت اضافه شده است.

.

Prototype

: این کتابخانه شامل مجموعه‌ای از متدهای کمکی برای راحتی کار در سمت کلاینت است. برای عملیات html encoding دو متد escapeHTML و unescapeHTML دارد که به صورت زیر پیاده شده‌اند:

```
function escapeHTML() {
  return this.replace(/&/g, '&amp;').replace(/</g, '&lt;').replace(/>/g, '&gt;');
}

function unescapeHTML() {
  return this.stripTags().replace(/&lt;/g, '<').replace(/&gt;/g, '>').replace(/&amp;/g, '&');
}
```

همان‌طور که می‌بینید در این متدها از replace استفاده شده است که برای متن‌های طولانی کندتر از روش‌های نشان داده‌شده در این مطلب است. هم‌چنین عملیات انکد و دیکد را تنها برای 3 کاراکتر < و > و & انجام می‌دهد که نقص بزرگی محسوب می‌شود.

.

jQuery.string

: این پلاگین حاوی چند متد برای کار با رشته‌هاست که یکی از این متدها با نام htmlspecialchars مخصوص عملیات انکدینگ است. در این متد تنها همان 5 کاراکتر اصلی تبدیل می‌شوند. متاسفانه متدی برای decode در این پلاگین وجود ندارد. پیاده‌سازی خلاصه‌شده این کتابخانه تنها برای نمایش نحوه عملکرد متد فوق به صورت زیر است:

```
var andExp = /&/g,
    htmlExp = [/(<|>|")/g, /(<|>|')/g, /(<|>|'|")/g],
    htmlCharMap = { '<': '&lt;', '>': '&gt;', "'": '&#039;', '"': '&quot;' },
    htmlReplace = function (all, $1) {
  return htmlCharMap[$1];
};
$.extend({
  // convert special html characters
  htmlspecialchars: function (string, quot) {
    return string.replace(andExp, '&amp;').replace(htmlExp[quot || 0], htmlReplace);
  }
});
```

نحوه استفاده از این متد هم به صورت زیر است:

```
$.htmlspecialchars("<div>");
```

.

$.string

: پلاگین دیگری برای jQuery که عملیات مربوط به رشته‌ها را دربر دارد. در این پلاگین برای عملیات انکدینگ دو متد escapeHTML و unescapeHTML به صورت زیر تعریف شده‌اند:

```
this.escapeHTML = function (s) {
  this.str = this.s(s)
      .split('&').join('&amp;')
      .split('<').join('&lt;')
      .split('>').join('&gt;');
  return this;
};

this.unescapeHTML = function (s) {
  this.str = this.stripTags(this.s(s)).str.replace(/&amp;/g, '&').replace(/&lt;/g,
'<').replace(/&gt;/g, '>');
  return this;
};
```

همان‌طور که می‌بنید در متد encode این پلاگین از یک روش جالب اما به نسبت ناکارآمد در رشته‌های طولانی، برای استخراج کاراکترهای غیرمجاز استفاده شده است. در این متدها هم تنها 3 کاراکتر & و < و > انکد و دیکد می‌شوند.

.

encoder.js

: کتابخانه نسبتا کاملی برای عملیات انکدینگ رشته‌ها در سمت کلاینت. این کتابخانه علاوه بر encode و decode رشته‌ها متدهایی برای تبدیل html entityها به فرمت عددی‌شان و برعکس، حذف کاراکترهای یونیکد، بررسی اینکه رشته ورودی شامل کاراکترهای انکد شده است، جلوگیری از انکدینگ مجدد یک رشته و ... نیز دارد.

. .

htmlEncode

: این متد پیاده‌سازی کاملی برای اجرای عملیات Html Encode دارد و محدوده وسیعی از کاراکترها را نیز تبدیل می‌کند. مشاهده عملیات موجود در این متد برای آشنایی با مطالب ظریف‌تر پیشنهاد می‌شود.

.

.

**منابع برای مطالعه بیشتر**

http://en.wikipedia.org/wiki/Character_encodings_in_HTML
http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references
http://stackoverflow.com/questions/1812473/difference-between-url-encode-and-html-encode
http://stackoverflow.com/questions/1219860/javascript-jquery-html-encoding
http://d802.nexlink.net/Gabriel/archive/2008/10/15/howto-html-encode-a-string-in-javascript.aspx
http://www.jardinesoftware.com/Documents/ASPNET_HTML_Encoding.pdf
http://www.west-wind.com/weblog/posts/2009/Feb/05/Html-and-Uri-String-Encoding-without-SystemWeb
http://blog.binarymist.net/tag/infosec
http://www.w3.org/TR/REC-html40/sgml/entities.html http://www.w3schools.com/html/html_entities.asp
http://www.w3schools.com/tags/ref_entities.asp
http://www.strictly-software.com/htmlencode