عنوان: **String.format در جاوا اسکریپت**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۱۹ ۱۱:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, String.format

مقدمه

با اینکه زبان برنامه نویسی جاوا اسکریپت زبانی بسیار قدرتمند و با امکانات زیاد است، اما فقدان برخی متدهای کمکی پرمصرف در آن در برخی موارد باعث دردسرهایی می‌شود. امکانی برای فرمت‌بندی رشته‌ها یکی از این نیازهای نسبتا پرکاربرد است.
متدی که در این مطلب قصد توضیح پیاده‌سازی آنرا داریم، String.format نام دارد که فرایندی مشابه متد متناظر در دات نت را انجام می‌دهد. هم‌چنین سعی شده است تا نحوه پیاده‌سازی این متد کمکی از ابتدایی‌ترین نمونه‌ها تا نسخه‌های پیشرفته‌تر برای درک بهتر مطلب نشان داده شود.

پیاده‌سازی متد String.format

1. در این پیاده‌سازی از اولین فرایندی که ممکن است به ذهن یک برنامه‌نویس خطور کند استفاده شده است. این پیاده‌سازی بسیار ساده به صورت زیر است:

```
String.format = function () {
  var s = arguments[0];
  for (var i = 0; i < arguments.length - 1; i++) {
    s = s.replace("{" + i + "}", arguments[i + 1]);
  }
  return s;
};
```

.

2. پیاده‌سازی مشابهی هم با استفاده از نوع دیگری از حلقه for که تقریبا! مشابه با حلقه foreach در C# است به صورت زیر می‌توان درنظر گرفت:

```
String.format = function () {
  var s = arguments[0];
  for (var arg in arguments) {
    var i = parseInt(arg);
    s = s.replace("{" + i + "}", arguments[i + 1]);
  }
  return s;
};
```

در این متدها ابتدا فرمت واردشده توسط کاربر از لیست آرگومان‌های متد خوانده شده و در متغیر s ذخیره می‌شود. سپس درون یک حلقه به ازای هر توکن موجود در رشته فرمت، یک عملیات replace با مقدار متناظر در لیست آرگومان‌های متد انجام می‌شود.
نحوه استفاده از این متد نیز به صورت زیر است:

```
console.log(String.format("{0} is nice!", "donettips.info"));
```

هر دو متد خروجی یکسانی دارند، به صورت زیر:

```
donettips.info is nice!
```

تا اینجا به نظر می‌رسد که عملیات به‌درستی پیش می‌رود. اما اولین و بزرگ‌ترین مشکل در این دو متد نحوه کارکردن متد replace در جاوا اسکریپت است. این متد با این نحوه فراخوانی تنها اولین توکن موجود را یافته و عملیات جایگزینی را برای آن انجام می‌دهد. برای روشن‌تر شدن موضوع به مثال زیر توجه کنید:

```
console.log(String.format("{0} is {1} nice! {0} is {1} nice!", "donettips.info", "very"));
```

با اجرای این مثال نتیجه زیر حاصل می‌شود:

```
donettips.info is very nice! {0} is {1} nice!
```

همان‌طور که می‌بنید عملیات replace برای سایر توکن‌ها انجام نمی‌شود.

.

3. برای حل مشکل فوق می‌توان از روش ساده زیر استفاده کرد:

```
String.format = function () {
  var original = arguments[0],
      replaced;
  for (var i = 0; i < arguments.length - 1; i++) {
    replaced = '';
    while (replaced != original) {
      original = replaced || original;
      replaced = original.replace("{" + i + "}", arguments[i + 1]);
    }
  }
  return replaced;
};
```

در این روش عملیات replace تا زمانی‌که تغییری در رشته جاری ایجاد نشود ادامه می‌یابد. با استفاده از این متد، خروجی مثال قبل درست و به صورت زیر خواهد بود:

```
donettips.info is very nice! donettips.info is very nice!
```

.

4. راه حل دیگر استفاده از امکانات شی RegExp در دستور replace است. نکته مهم استفاده از modifier کلی یا global (که با حرف g مشخص می‌شود) در شی تولیدی از RegExp است (^ و ^ و ^ ، برای جلوگیری از دورشدن از بحث اصلی، جستجو برای کسب اطلاعات بیشتر در این زمینه به خوانندگان واگذار می‌شود). برای استفاده از این شی متد ما به صورت زیر تغییر می‌کند:

```
String.format = function () {
  var s = arguments[0];
  for (var i = 0; i < arguments.length - 1; i++) {
    s = s.replace(new RegExp("\\{" + i + "\\}", "g"), arguments[i + 1]);
  }
  return s;
};
```

استفاده از این متد هم نتیجه درستی برای مثال آخر ارائه می‌دهد.

.

5. روش دیگری که کمی از دو متد قبلی سریع‌تر اجرا می‌شود (به دلیل استفاده از حلقه while) به صورت زیر است:

```
String.format = function () {
  var s = arguments[0],
      i = arguments.length - 1;
  while (i--) {
    s = s.replace(new RegExp('\\{' + i + '\\}', 'g'), arguments[i + 1]);
  }
  return s;
};
```

این متد نیز نتیجه مشابهی ارائه می‌کند. حال به مثال زیر توجه کنید:

```
console.log(String.format("{0}:0 {1}:1 {2}:2", "zero", "{2}", "two"));
```

خروجی صحیح مثال فوق باید به صورت زیر باشد:

```
zero:0 {2}:1 two:2
```

درصورتی‌که رشته‌ای که دو متد از سه متد آخر (3 و 4) به عنوان خروجی ارائه می‌دهند به‌صورت زیر است:

```
zero:0 two:1 two:2
```

برای آخرین متد که ازحلقه while (درواقع با اندیس معکوس) استفاده می‌کند (5) مثالی که خطای مورد بحث را نشان می‌دهد به صورت زیر است:

```
console.log(String.format("{0}:0 {1}:1 {2}:2", "zero", "one", "{1}"));
```

که خروجی اشتباه زیر را برمی‌گرداند:

```
zero:0 one:1 one:2
```

درصورتی‌که باید مقدار زیر را برگشت دهد:

```
zero:0 one:1 {1}:2
```

دلیل رخدادن این خطا اجرای عملیات replace به صورت جداگانه و کامل برای هر توکن، از اول تا آخر برای رشته‌های replace شده جاری است که کار را خراب می‌کند.

.

6. برای حل مشکل بالا نیز می‌توان از یکی دیگر از امکانات دستور replace استفاده کرد که به صورت زیر است:

```
String.format = function () {
  var args = arguments;
  return args[0].replace(/{(\d+)}/g, function (match, number) { return args[parseInt(number) + 1]; });
};
```

در اینجا از قابلیت سفارشی‌سازی عملیات جایگزینی در دستور replace استفاده شده است. با استفاده از این ویژگی عملیات replace برای هر توکن جداگانه انجام می‌شود و بنابراین تغییرات اعمالی در حین عملیات تاثیر مستقیمی برای ادامه روند نخواهد گذاشت.
دقت کنید که برای بکاربردن RegExp درون دستور replace به جای تولید یک نمونه از شی RegExp می‌توان عبارت مربوطه را نیز مستقیما بکار برد. در اینجا از عبارتی کلی برای دریافت تمامی توکن‌های با فرمتی به صورت {عدد} استفاده شده است.
متد سفارشی مربوطه نیز شماره ردیف توکن یافته‌شده به همراه خود عبارت یافته‌شده را به عنوان آرگومان ورودی دریافت کرده و مقدار متناظر را از لیست آرگومان‌های متد اصلی پس از تبدیل شماره ردیف توکن به یک عدد، برگشت می‌دهد (در اینجا نیز برای جلوگیری از دورشدن از بحث اصلی، جستجو برای کسب اطلاعات بیشتر در این زمینه به خوانندگان واگذار می‌شود).
برای جلوگیری از تداخل بین آرگومان‌های متد اصلی و متد تهیه‌شده برای سفارشی‌سازی عملیات جایگزینی، در ایتدای متد اصلی، لیست آرگومان‌های آن درون متغیر جداگانه‌ای (args) ذخیره شده است.
با استفاده از این متد خروجی درست نشان داده می‌شود. حال مثال زیر را درنظر بگیرید:

```
console.log(String.format("{0} is {1} nice!", "donettips.info"));
```

خروجی این مثال به‌صورت زیر است:

```
donettips.info is undefined nice!
```

پیاده‌سازی زیر برای حل این مشکل استفاده می‌شود.

.

7. برای کنترل بیشتر و رفع خطاهای احتمالی در متد بالا، می‌توان ابتدا از وجود آرگومان مربوطه در متغیر args اطمینان حاصل کرد تا از جایگزینی مقدار undefined در رشته نهایی جلوگیری کرد. مانند نمونه زیر:

```
String.format = function () {
  var s = arguments[0],
      args = arguments;
  return s.replace(/{(\d+)}/g, function (match, number) {
    var i = parseInt(number);
    return typeof args[i + 1] != 'undefined' ? args[i + 1] : match;
  });
};
```

با استفاده از این متد جدید خروجی مثال‌های قبل درست خواهد بود.
در فرمت بندی رشته‌ها برای نمایش خود کاراکتر { یا } از تکرار آن‌ها (یعنی {{ یا }}) استفاده می‌شود. اما متد ما تا این لحظه این امکان را ندارد. برای مثال:

```
console.log(String.format("{0}:0 {1}:1 {2}:2, {{0}} {{{1}}}  {{{{2}}}}   {2}", "zero", "{2}", "two"));
```

که خروجی زیر را ارائه می‌دهد:

```
zero:0 {2}:1 two:2, {zero} {{{2}}}  {{{two}}}   two
```

8. برای پیاده‌سازی امکان اشاره‌شده در بالا می‌توان از کد زیر استفاده کرد:

```
String.format = function () {
  var s = arguments[0],
      args = arguments;
  return s.replace(/\{\{|\}\}|\{(\d+)\}/g, function (match, number) {
    if (match == "{{") { return "{"; }
    if (match == "}}") { return "}"; }
    var i = parseInt(number);
    return typeof args[i + 1] != 'undefined'
                              ? args[i + 1]
                              : match;
  });
};
```

در اینجا با استفاده از یک عبارت RegExp پیچیده‌تر و کنترل تکرار کاراکترهای { و } در متد سفارشی جایگزینی در دستور replace، پیاده‌سازی اولیه این ویژگی ارائه شده است.
این متد خروجی صحیح زیر را برای مثال آخر ارائه می‌دهد:

```
zero:0 {2}:1 two:2, {0} {{2}}  {{2}}   two
```

.

پیاده‌سازی به‌صورت یک خاصیت prototype

تمامی متدهای نشان داده‌شده تا اینجا به‌صورت مستقیم از طریق String.format در دسترس خواهند بود (تعریفی شبیه به

متدهای استاتیک در دات نت). درصورتی‌که بخواهیم از این متدها به صورت یک خاصیت prototype شی string استفاده کنیم (چیزی شبیه به متدهای instance در اشیای دات نت) می‌توانیم از تعریف زیر استفاده کنیم:

```
String.prototype.format = function () {
   ...
}
```

تنها فرق مهم این پیاده‌سازی این است که رشته مربوط به فرمت وارده در این متد از طریق شی this در دسترس است و بنابراین شماره اندیس آرگومان‌های متد یکی کمتر از متدهای قبلی است که باید مدنظر قرار گیرد. مثلا برای متد آخر خواهیم داشت:

```
String.prototype.format = function () {
  var s = this.toString(),
      args = arguments;
  return s.replace(/\{\{|\}\}|\{(\d+)\}/g, function (match, number) {
    if (match == "{{") { return "{"; }
    if (match == "}}") { return "}"; }
    return typeof args[number] != 'undefined'
                               ? args[number]
                               : match;
  });
};
```

.

نکته: در تمامی خواص prototype هر شی در جاوا اسکریپت، متغیر this از نوع object است. بنابراین برای جلوگیری از وقوع هر خطا بهتر است ابتدا آن‌را به نوع مناسب تبدیل کرد. مثل استفاده از متد toString در متد فوق که موجب تبدیل آن به رشته می‌شود.

.

ازآنجاکه نیاز به تغییر اندیس در متد سفارشی عملیات replace وجود ندارد، بنابراین خط مربوط به تبدیل آرگومان number به یک مقدار عددی (با دستور parseInt) حذف شده است و از این متغیر به صورت مستقیم استفاده شده است. در این حالت عملیات تبدیل توسط خود جاوا اسکریپت مدیریت می‌شود که کار را راحت‌تر می‌سازد. بنابراین متد ما به صورت زیر قابل استفاده است:

```
console.log("{0}:0 {1}:1 {2}:2, {{0}} {{{1}}}  {{{{2}}}}   {2}".format("zero", "{2}", "two"));
```

.

پیاده‌سازی با استفاده از توکن‌های غیرعددی

برای استفاده از توکن‌های غیرعددی می‌توانیم به صورت زیر عمل کنیم:

```
String.format = function () {
  var s = arguments[0],
      args = arguments[1];
  for (var arg in args) {
    s = s.replace(new RegExp("{" + arg + "}", "g"), args[arg]);
  }
  return s;
};
```

برای حالت prototype نیز داریم:

```
String.prototype.format = function () {
  var s = this.toString(),
      args = arguments[0];
  for (var arg in args) {
    s = s.replace(new RegExp("{" + arg + "}", "g"), args[arg]);
  }
```

```
  return s;
};
```

با استفاده از این دو متد داریم:

```
console.log(String.format("{site} is {adj}! {site} is {adj}!", { site: "donettips.info", adj: "nice"
}));
console.log("{site} is {adj}! {site} is {adj}!".format({ site: "donettips.info", adj: "nice" }));
```

تا اینجا متدهایی نسبتا کامل برای نیازهای عادی برنامه‌نویسی تهیه شده است. البته کار توسعه این متد برای پشتیبانی از امکانات پیشرفته‌تر فرمت‌بندی رشته‌ها می‌تواند ادامه پیدا کند.

.

کتابخانه‌های موجود

یکی از کامل‌ترین کتابخانه‌های کار با رشته‌ها همان کتابخانه معروف Microsoft Ajax Client Libray است که بیشتر امکانات موجود کار با رشته‌ها در دات نت را در خود دارد. صرفا جهت آشنایی، پیاده‌سازی متد String.format در این کتابخانه در زیر آورده شده است:

```
String.format = function String$format(format, args) {
  /// <summary locid="M:J#String.format" />
  /// <param name="format" type="String"></param>
  /// <param name="args" parameterArray="true" mayBeNull="true"></param>
  /// <returns type="String"></returns>
//  var e = Function._validateParams(arguments, [
//    { name: "format", type: String },
//    { name: "args", mayBeNull: true, parameterArray: true }
//  ]);
//  if (e) throw e;
  return String._toFormattedString(false, arguments);
};
String._toFormattedString = function String$_toFormattedString(useLocale, args) {
  var result = '';
  var format = args[0];
  for (var i = 0; ; ) {
    var open = format.indexOf('{', i);
    var close = format.indexOf('}', i);
    if ((open < 0) && (close < 0)) {
      result += format.slice(i);
      break;
    }
    if ((close > 0) && ((close < open) || (open < 0))) {
      if (format.charAt(close + 1) !== '}') {
        throw Error.argument('format', Sys.Res.stringFormatBraceMismatch);
      }
      result += format.slice(i, close + 1);
      i = close + 2;
      continue;
    }
    result += format.slice(i, open);
    i = open + 1;
    if (format.charAt(i) === '{') {
      result += '{';
      i++;
      continue;
    }
    if (close < 0) throw Error.argument('format', Sys.Res.stringFormatBraceMismatch);
    var brace = format.substring(i, close);
    var colonIndex = brace.indexOf(':');
    var argNumber = parseInt((colonIndex < 0) ? brace : brace.substring(0, colonIndex), 10) + 1;
    if (isNaN(argNumber)) throw Error.argument('format', Sys.Res.stringFormatInvalid);
    var argFormat = (colonIndex < 0) ? '' : brace.substring(colonIndex + 1);
    var arg = args[argNumber];
    if (typeof (arg) === "undefined" || arg === null) {
      arg = '';
    }
    if (arg.toFormattedString) {
      result += arg.toFormattedString(argFormat);
    }
```

```
    else if (useLocale && arg.localeFormat) {
      result += arg.localeFormat(argFormat);
    }
    else if (arg.format) {
      result += arg.format(argFormat);
    }
    else
      result += arg.toString();
    i = close + 1;
  }
  return result;
}
```

دقت کنید قسمت ابتدایی این متد که برای بررسی اعتبار آرگومان‌های ورودی است، برای سادگی عملیات کامنت شده است. همان‌طور که می‌بینید این متد پیاده‌سازی نسبتا مفصلی دارد و امکانات بیشتری نیز در اختیار برنامه نویسان قرار می‌دهد. البته سایر متدهای مربوطه بدلیل طولانی بودن در اینجا آورده نشده است. برای مثال امکانات پیشرفته‌تری مثل زیر با استفاده از این کتابخانه در دسترس هستند:

```
console.log(String.format("{0:n}, {0:c}, {0:p}, {0:d}", 100.0001));
// result:   100.00, ¤100.00, 10,000.01 %, 100.0001

console.log(String.format("{0:d}, {0:t}", new Date(2015, 1, 1, 10, 45)));
// result:   02/01/2015, 10:45
```

آخرین نسخه این کتابخانه از اینجا قابل دریافت است (این متدها درون فایل MicrosoftAjax.debug.js قرار دارند). این کتابخانه دیگر به این صورت و با این نام توسعه داده نمیشود و چند سالی است که تصمیم به توسعه ویژگی‌های جدید آن به صورت پلاگین‌های jQuery گرفته شده است.

.

کتابخانه دیگری که می‌توان برای عملیات فرمت‌بندی رشته‌ها در جاوا اسکریپت از آن استفاده کرد، کتابخانه معروف jQuery Validation است. این کتابخانه یک متد نسبتا خوب با نام format برای فرمت کردن رشته‌ها دارد. نحوه استفاده از این متد به صورت زیر است:

```
var template = jQuery.validator.format("{0} is not a valid value");
console.log(template("abc"));
// result: 'abc is not a valid value'
```

.

کتابخانه نسبتا کامل دیگری که وجود دارد، با عنوان Stringformat از اینجا قابل دریافت است. برای استفاده از این کتابخانه باید به صورت زیر عمل کرد:

```
String.format([full format string], [arguments...]);
// or:
[date|number].format([partial format string]);
```

همان‌طور که می‌بینید این کتابخانه امکانات کامل‌تری نیز دارد. مثال‌های مربوط به این کتابخانه به صورت زیر هستند که توانایی‌های نسبتا کامل آن‌را نشان می‌دهد:

```
// Object path
String.format("Welcome back, {username}!",
{ id: 3, username: "JohnDoe" });
// Result: "Welcome back, JohnDoe!"

// Date/time formatting
String.format("The time is now {0:t}.",
new Date(2009, 5, 1, 13, 22));
// Result: "The time is now 01:22 PM."
```

```
// Date/time formatting (without using a full format string)
var d = new Date();
d.format("hh:mm:ss tt");
// Result: "02:28:06 PM"

// Custom number format string
String.format("Please call me at {0:+##0 (0) 000-00 00}.", 4601111111);
// Result: "Please call me at +46 (0) 111-11 11."

// Another custom number format string
String.format("The last year result was {0:+$#,0.00;-$#,0.00;0}.", -5543.346);
// Result: "The last year result was -$5,543.35."

// Alignment
String.format("|{0,10:PI=0.00}|", Math.PI);
// Result: "|   PI=3.14|"

// Rounding
String.format("1/3 ~ {0:0.00}", 1/3);
// Result: "1/3 ~ 0.33"

// Boolean values
String.format("{0:true;;false}", 0);
// Result: "false"

// Explicitly specified localization
// (note that you have to include the .js file for used cultures)
msf.setCulture("en-US");
String.format("{0:#,0.0}", 3641.667);
// Result: "3,641.7"

msf.setCulture("sv-SE");
String.format("{0:#,0.0}", 3641.667);
// Result: "3 641,7"
```

.

یک کتابخانه دیگر نیز از این آدرس قابل دریافت است. این کتابخانه با عنوان String.format نام‌گذاری شده است. نحوه استفاده از این کتابخانه نیز به صورت زیر است:

```
//inline arguments
String.format("some string with {0} and {1} injected using argument {{number}}", 'first value', 'second
value');
//returns: 'some string with first value and second value injected argument {number}'

//single array
String.format("some string with {0} and {1} injected using array {{number}}", [ 'first value', 'second
value' ]);
//returns: 'some string with first value and second value injected using array {number}'

//single object
String.format("some string with {first} and {second} value injected using
{{propertyName}}",{first:'first value',second:'second value'});
//returns: 'some string with first value and second value injected using {propertyName}'
```

.

کتابخانه نسبتا معروف و کامل sprintf نیز در اینجا وجود دارد. این کتابخانه امکانات بسیاری همچون متدهای متناظر در زبان C دارد.

.

منابع

http://www.w3schools.com/jsref/jsref_replace.asp

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace

http://www.w3schools.com/jsref/jsref_obj_regexp.asp

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

http://stackoverflow.com/questions/610406/javascript-equivalent-to-printf-string-format

http://stackoverflow.com/questions/1038746/equivalent-of-string-format-in-jquery

http://stackoverflow.com/questions/2534803/string-format-in-javascript

http://jqueryvalidation.org/jQuery.validator.format

http://www.masterdata.se/r/string_format_for_javascript

https://github.com/tracker1/core-js/blob/master/js-extensions/100-String.format.js

http://www.diveintojavascript.com/projects/javascript-sprintf

نظرات خوانندگان

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۰۳/۲۰ ۰:۲۸

با تشکر از شما بابت ارسال این پست خیلی خیلی خوب!
از چند جهت عالی بود
اول معرفی چند کتابخانه
دوم ارائه‌ی بخشی از کد کتابخانه و توضیح پیاده سازی آن !
سوم آموزش اضافه کردن توابع جدید به صورت استاتیک و همچنین به قول خودتان `instance method`
عالی بود! ممنونم