عنوان: **CoffeeScript #13**
نویسنده: وحید محمّدطاهری
تاریخ: ۱۳۹۴/۰۶/۱۷ ۲۳:۳۰
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, CoffeeScript

بخش‌های بد

در ادامه‌ی قسمت قبل ، به مواردی که توسط CoffeeScript اصلاح شده‌اند، می‌پردازیم.

Reserved words

کلمات کلیدی خاصی در جاوااسکریپت وجود دارد مانند **enum** ، **class** و **const** که برای نسخه‌های بعدی جاوااسکریپت در آینده رزرو شده‌اند. استفاده از این کلمات در برنامه‌های جاوااسکریپت می‌تواند نتایج غیرقابل پیش بینی داشته باشد. برخی از مرورگرهای به خوبی از عهده‌ی این کار برمی‌آیند و بعضی دیگر به طور کامل جلوی استفاده از این‌ها را گرفته‌اند. CoffeeScript بعد از تشخیص استفاده از یک کلمه‌ی کلیدی، با یک راه کار خاص، از این موضوع می‌گریزد.

به عنوان مثال، فرض کنید می‌خواهیم از کلمه کلیدی class به عنوان یک خصوصیت در یک شیء استفاده کنیم:

```
myObj = {
  delete: "I am a keyword!"
}
myObj.class = ->
```

پس از کامپایل، پارسر CoffeeScript متوجه استفاده شما از کلمه کلیدی رزرو شده می‌شود و آنها را در بین "" قرار می‌دهد.

```
var myObj;
myObj = {
  "delete": "I am a keyword!"
};
myObj["class"] = function() {};
```

Equality comparisons

مقایسه برابری ضعف دیگری است که در جاوااسکریپت باعث ایجاد رفتاری گیج کننده و اغلب باعث ایجاد اشکالاتی در کد نوشته شده می‌شود. به مثال زیر توجه کنید:

```
""           ==   "0"// false
0            ==   ""// true
0            ==   "0"// true
false        ==   "false"// false
false        ==   "0"// true
false        ==   undefined// false
false        ==   null// false
null         ==   undefined// true
" \t\r\n"    ==   0// true
```

مطمئنم که شما هم با من موافقید که همه‌ی مقایسه‌های بالا بسیار مبهم هستند و استفاده از آن‌های می‌توانند منجر به نتایج غیر منتظره شوند و همچنین مشکلاتی را پیش بیاورند.
راه حل این کار استفاده از عملگر برابری سختگیرانه است، که از 3 مساوی تشکیل شده است: **===** عملگر برابر سخت گیرانه دقیقا مانند عملگر برابری عادی عمل می‌کند و تنها نوع داده‌ها را بررسی می‌کند که با هم برابر باشند.
توصیه می‌شود که همیشه از عملگر برابری سختگیرانه استفاده کنید و هرجا لازم بود قبل مقایسه عمل تبدیل نوع داده‌ها را انجام دهید.
CoffeeScript این مشکل را به صورت کامل حل کرده است؛ یعنی هر جایی که عمل مقایسه == انجام شود به === تبدیل می‌شود. شما باید به صورت صریح نوع داده‌ها را قبل از مقایسه تبدیل کرده باشید.

نکته: در مقایسه‌ها رشته خالی ""، undefined، null و عدد 0 همگی false برمی گردانند.

```
alert "Empty Array"  unless [].length
alert "Empty String" unless ""
alert "Number 0"      unless 0
```

که پس از کامپایل می‌شود:

```
if (![].length) {
  alert("Empty Array");
}

if (!"") {
  alert("Empty String");
}

if (!0) {
  alert("Number 0");
}
```

در صورتیکه می‌خواهید به صورت صریح null و یا undefined را بررسی کنید، می‌توانید از عملگر ? CoffeeScript استفاده کنید:

```
alert "This is not called" unless ""?
```

پس از کامپایل می‌شود:

```
if ("" == null) {
  alert("This is not called");
}
```

با اجرای مثال بالا alert اجرای نمی‌شود چون رشته خالی با null برابر نیست.

Function definition

خیلی جالب است که در جاوااسکریپت می‌توانید تابعی را بعد از اینکه فراخوانی کردید، تعریف کنید. به عنوان مثال، کد زیر به صورت کامل اجرا می‌شود:

```
wem();
function wem() {alert("hi");}
```

این به دلیل دامنه (scope) تابع است. تمام توابع قبل از اجرای برنامه، به بالا برده می‌شوند و در همه جا در دامنه‌ای که در آن تعریف شده‌اند، قابل دسترسی می‌باشند؛ حتی اگر قبل از تعریف واقعی در منبع، فراخوانی شده باشد. مشکل اینجاست که عمل بالابردن توابع در مرورگرها با یکدیگر متفاوت است. به مثال زیر توجه کنید:

```
if (true) {
  function declaration() {
    return "first";
  }
} else {
  function declaration() {
    return "second";
  }
}
declaration();
```

در بعضی از مرورگرها مانند Firefox ، تابع declaration() مقدار " **first** " را برگشت خواهد داد و در دیگر مرورگرها مانند Chrome، مقدار " **second** " برگشت داده خواهد شد. در حالیکه به نظر می‌رسد که قسمت else هیچگاه اجرا نخواهد شد.

در صورتیکه علاقمند به کسب اطلاعات بیشتری درباره‌ی نحوه تعریف توابع، هستید باید راهنمای آقای Juriy Zaytsev را مطالعه کنید. به صورت خلاصه، رفتار نسبتا مبهم مرورگرها می‌تواند منجر به ایجاد مشکلاتی در مسیر نوشتن یک پروژه شوند.
همه چیز در CoffeeScript در نظر گرفته شده است و بهترین روش برای حل این مشکل، حذف کلمه function و به جای آن استفاده از عبارت (expression) تابع است.

Number property lookups نقصی که در پارسر جاوااسکریپت در مواجه با نماد نقطه (*dot notation*) بر روی اعداد وجود دارد، تفسیر آن به ممیز شناور، بجای مراجعه به ویژگی‌های آن است. برای مثال کد جاوااسکریپت زیر باعث ایجاد خطای نحوی می‌شود:

```
5.toString();
```

پارسر جاوااسکریپت بعد از نقطه به دنبال یک عدد دیگر می‌گردد و با برخورد با ()toString، باعث ایجاد یک Unexpected token می‌شود. راه حل این مشکل، استفاده از **پرانتز** یا اضافه کردن یک نقطه دیگر است.

```
(5).toString();
5..toString();
```

خوشبختانه پارسر CoffeeScript به اندازه‌ی کافی هوشمندانه با این مسئله برخورد می‌کند و هر زمانی که شما دسترسی به ویژگی‌های اعداد را داشته باشید، به صورت خودکار با اضافه کردن دوتا نقطه (همانند مثال بالا) جلوی ایجاد خطا را می‌گیرد.