عنوان: **CoffeeScript #14**
نویسنده: وحید محمّدطاهری
تاریخ: ۱۳۹۴/۰۷/۱۷ ۱۰:۲۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, CoffeeScript

قسمت‌های اصلاح نشده CoffeeScript در حال رفع برخی از معایب طراحی جاوااسکریپت است و این راه، بس طولانی است. همانطور که قبلا گفته شد، CoffeeScript به شدت به تجزیه و تحلیل استاتیک در زمان طراحی محدود شده است و هیچ بررسی در زمان اجرایی را برای بهبود کارآیی آن انجام نمی‌دهد.
CoffeeScript از یک کامپایلر مستقیم منبع به منبع استفاده می‌کند. با این دیدگاه که هر دستور در CoffeeScript در نتیجه به یک دستور معادل در جاوااسکریپت تبدیل می‌شود.
CoffeeScript برای همه‌ی کلمات کلیدی جاوااسکریپت، کلمه‌ی معادلی ایجاد نمی‌کند، مانند typeof؛ و همچنین برخی از معایب طراحی جاوااسکریپت، به CoffeeScript نیز اعمال می‌شود.
در دو قسمت قبل + و + بر روی معایب طراحی در جاوااسکریپت که توسط CoffeeScript اصلاح شده بود، توضیح دادیم. حال می‌خواهیم درباره برخی از معایب جاوااسکریپت که CoffeeScript تا به حال نتوانسته است آنها را اصلاح کند صحبت کنیم.

استفاده از eval

در حالیکه CoffeeScript برخی از نقاط ضعف جاوااسکریپت را اصلاح کرده است، اما همچنان معایب دیگری نیز وجود دارند، که شما تنها باید از این نقاط ضعف آگاه باشید. یکی از این موارد، تابع **eval** است. برای استفاده از آن، باید با اشکالاتی که در حین کار با آن مواجه می‌شوید، آگاهی کامل داشته باشید و در صورت امکان از استفاده از آن اجتناب کنید.

تابع **eval** یک رشته از کد جاوااسکریپت را در حوزه‌ی محلی اجرا می‌کند و توابعی مانند **setTimeout** و **setInterval** نیز می‌توانند در آرگومان اولشان یک رشته از کد جاوااسکریپت را دریافت و ارزیابی کنند.

با این حال، مانند with، eval نیز ردیابی کامپایلر را از کار می‌اندازد و این امر تاثیر بسیار زیادی بر روی کارآیی آن دارد. کامپایلر هیچ ایده ای درباره کدی که درون eval قرار داده شده است، ندارد تا زمانی که آن را اجرا کند. به همین دلیل نمی‌تواند هیچ عمل بهینه سازی را بر روی انجام دهد. یکی دیگر از نگرانی‌های استفاده‌ی از eval، **امنیت** است. در صورتیکه شما ورودی را به eval ارسال کنید، eval باعث می‌شود که کد شما به راحتی در معرض حملات تزریق کد قرار می‌گیرد. در 99% از مواقع، وقتی شما می‌خواهید از eval استفاده کنید، راه‌های بهتر و امن‌تری وجود دارند (مانند استفاده از براکت).

```
# Don't do this
model = eval(modelName)

# Use square brackets instead
model = window[modelName]
```

استفاده از typeof

اپراتور typeof احتمالا بزرگترین نقص طراحی جاوااسکریپت است؛ تنها به این دلیل که اساسا به طور کامل شکست خورده است. در واقع از آن فقط یک استفاده می‌شود تا تشخیص داده شود که یک مقدار undefined است یا نه.

```
typeof undefinedVar is "undefined"
```

برای چک کردن type همه types، متاسفانه **typeof** نمی تواند به درستی این کار را انجام دهد و مقدار بازگشتی آن وابسته به مرورگر و چگونگی نمونه سازی آن نمونه است. در این رابطه CoffeeScript هیچ کمکی به شما نمی‌تواند بکند، چرا که قبلا نیز گفته شد، CoffeeScript یک زبان با تجزیه و تحلیل استاتیک است و هیچ بررسی در زمان اجرایی بر روی نوع آن ندارد.
در اینجا لیستی از مشکلات، هنگام استفاده از typeof را مشاهده می‌کنید:

```
Value               Class      Type
```

```
-----------------------------------------------
"foo"               String     string
new String("foo")   String     object
1.2                 Number     number
new Number(1.2)     Number     object
true                Boolean    boolean
new Boolean(true)   Boolean    object
new Date()          Date       object
new Error()         Error      object
[1,2,3]             Array      object
new Array(1, 2, 3)  Array      object
new Function("")    Function   function
/abc/g              RegExp     object
new RegExp("meow")  RegExp     object
{}                  Object     object
new Object()        Object     object
```

همانطور که مشاهده می‌کنید تعریف یک رشته در داخل **""** و یا با کلاس **String** ، در نتیجه‌ی استفاده از typeof تاثیر گذار است. به طور منطقی typeof باید "string" را به عنوان خروجی در هر دو حالت نشان دهد، اما برای دومی به صورت "object" باز می‌گرداند.

سوالی که اینجا مطرح می‌شود این است که ما چطور می‌توانیم یک نوع را در جاوااسکریپت چک کنیم؟

خوب، خوشبختانه Object.prototype.toString() ما را نجات داده است. اگر ما این تابع را بر روی یک شیء خاص فراخوانی کنیم، مقدار صحیح را بر می‌گرداند.

در اینجا مثالی از نحوه‌ی پیاده سازی jQuery.type را مشاهده می‌کنید:

```
type = do ->
  classToType = {}
  for name in "Boolean Number String Function Array Date RegExp Undefined Null".split(" ")
    classToType["[object " + name + "]"] = name.toLowerCase()

  (obj) ->
    strType = Object::toString.call(obj)
    classToType[strType] or "object"

# Returns the sort of types we'd expect:
type("")          # "string"
type(new String) # "string"
type([])          # "array"
type(/\d/)        # "regexp"
type(new Date)    # "date"
type(true)        # "boolean"
type(null)        # "null"
type({})          # "object"
```

در صورتیکه بخواهید تشخیص دهید یک متغیر تعریف شده است یا نه، باید از typeof استفاده کنید؛ در غیر این صورت پیام خطای ReferenceError را دریافت خواهید کرد.

```
if typeof aVar isnt "undefined"
  objectType = type(aVar)
```

و یا به طور خلاصه‌تر با استفاده از اپراتور وجودی:

```
objectType = type(aVar?)
```

راه دیگری برای چک کردن نوع، استفاده از اپراتور وجودی CoffeeScript است. برای مثال: می‌خواهیم یک مقدار را در یک آرایه اضافه کنیم. می‌توان گفت تا زمانیکه تابع push پیاده سازی شده باشد ما باید با آن مانند یک آرایه رفتار کنیم.

```
anArray?.push? aValue
```

اگر anArray یک شیء به غیر از آرایه باشد، اپراتور وجودی تضمین خواهد کرد که هیچگاه تابع push فراخوانی نخواهد شد.

استفاده از instanceof

کلمه‌ی کلیدی **instanceof** نیز تقریبا همانند **typeof** شکست خورده است. در حالت ایده آل، instanceof، سازنده‌ی دو شیء را با هم مقایسه می‌کند، در صورتیکه یک شیء نمونه‌ای از شیء دیگر باشد، یک مقدار boolean را باز می‌گرداند. در واقع instanceof موقعی کار مقایسه را انجام می‌دهد که اشیاء، سفارشی سازی شده باشند. وقتی عمل مقایسه می‌خواهد بر روی این نوع اشیاء سفارشی سازی شده، انجام شود، استفاده از typeof بی‌فایده است.

```
new String("foo") instanceof String # true
"foo" instanceof String             # false
```

علاوه بر این، instanceof همچنین بر روی اشیاء در فریم‌های مختلف مرورگر عمل مقایسه را نمی‌تواند انجام دهد. در واقع instanceof فقط نتیجه‌ی صحیح مقایسه را در اشیاء سفارشی سازی شده برمی‌گرداند؛ مانند کلاس‌های CoffeeScript.

```
class Parent
class Child extends Parent

child = new Child
child instanceof Child  # true
child instanceof Parent # true
```

مواقعی از instanceof استفاده کنید که مطمئن هستید بر روی اشیای ساخته شده توسط شما بکار گرفته می‌شود و یا هرگز از آن استفاده نکنید.

استفاده از delete از کلمه کلیدی **delete** برای حذف خصوصیات موجود در اشیاء به صورت کاملا مطمئن، می‌توان استفاده کرد.

```
anObject = {one: 1, two: 2}
delete anObject.one
anObject.hasOwnProperty("one") # false
```

هر نوع استفاده دیگر، از قبیل حذف متغیرها و یا توابع کار نخواهد کرد.

```
aVar = 1
delete aVar
typeof Var # "integer"
```

در صورتیکه می‌خواهید یک اشاره گر به یک متغیر را حذف کنید فقط کافیست مقدار **null** را به آن انتساب دهید.

```
aVar = 1
aVar = null
```