オブジェクトのカタチ

MORITA Hajime <omo@dodgson.org>
2008/11/16

あらすじ

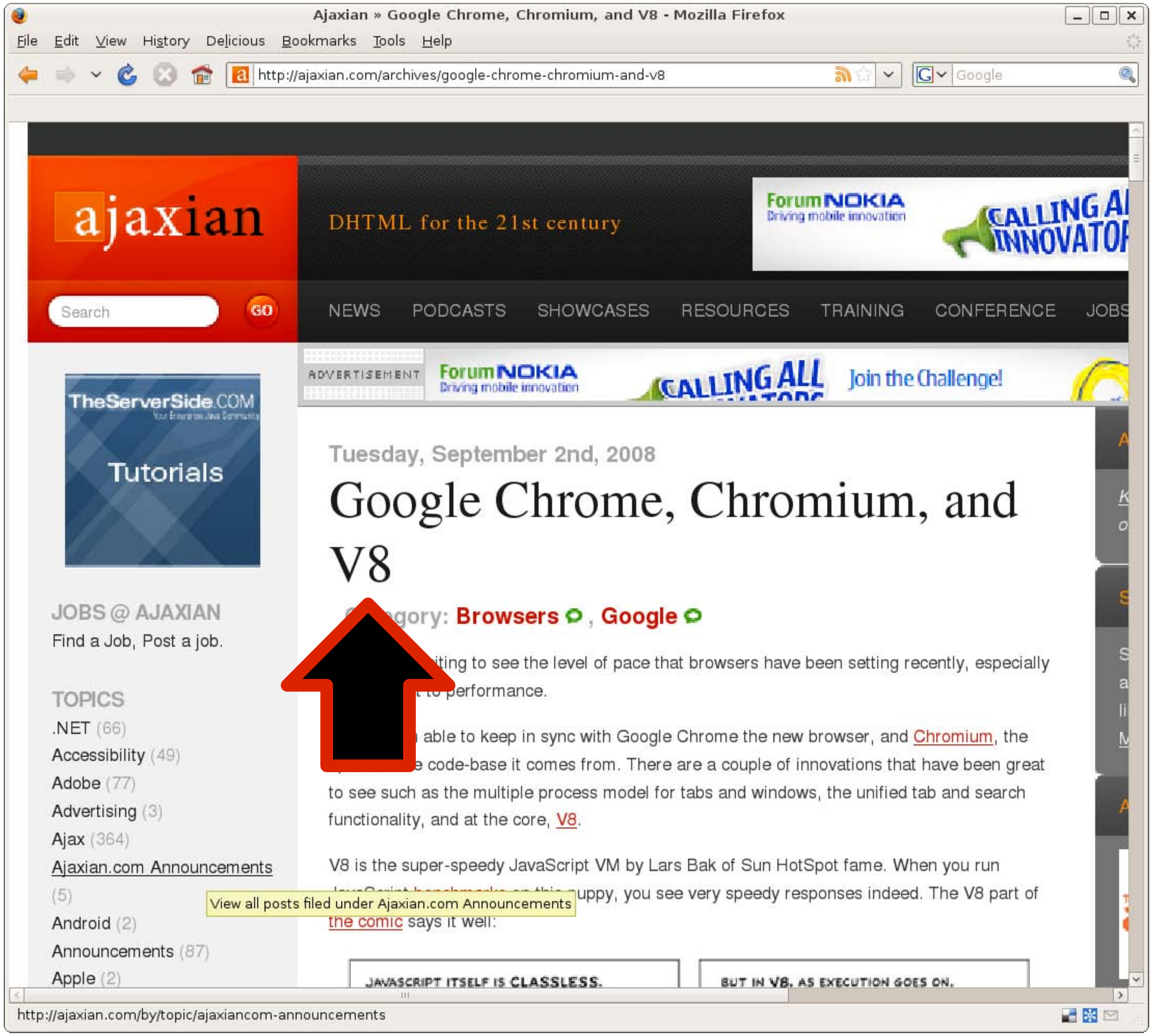
TraceMonkey の
プロパティアクセスは
爆速だぜ

http://www.flickr.com/photos/dnorman/2726761770/

Property Accesses

```
// read
var a = foo.x;
// write
foo.y = b;
// call
foo.z();
```

ライバルのニュース



Ajaxian » Google Chrome, Chromium, and V8 - Mozilla Firefox
File Edit View History Delicious Bookmarks Tools Help
http://ajaxian.com/archives/google-chrome-chromium-and-v8
ajaxian
DHTML for the 21st century
Search
GO
NEWS PODCASTS SHOWCASES RESOURCES TRAINING CONFERENCE JOBS
TheServerSide.COM
Tutorials
JOBS @ AJAXIAN
Find a Job, Post a job.
TOPICS
.NET (66)
Accessibility (49)
Adobe (77)
Advertising (3)
Ajax (364)
Ajaxian.com Announcements (5)
Android (2)
Announcements (87)
Apple (2)
View all posts filed under Ajaxian.com Announcements
http://ajaxian.com/by/topic/ajaxiancom-announcements

Tuesday, September 2nd, 2008

Google Chrome, Chromium, and V8

gory: **Browsers**, **Google**

iting to see the level of pace that browsers have been setting recently, especially t to performance.

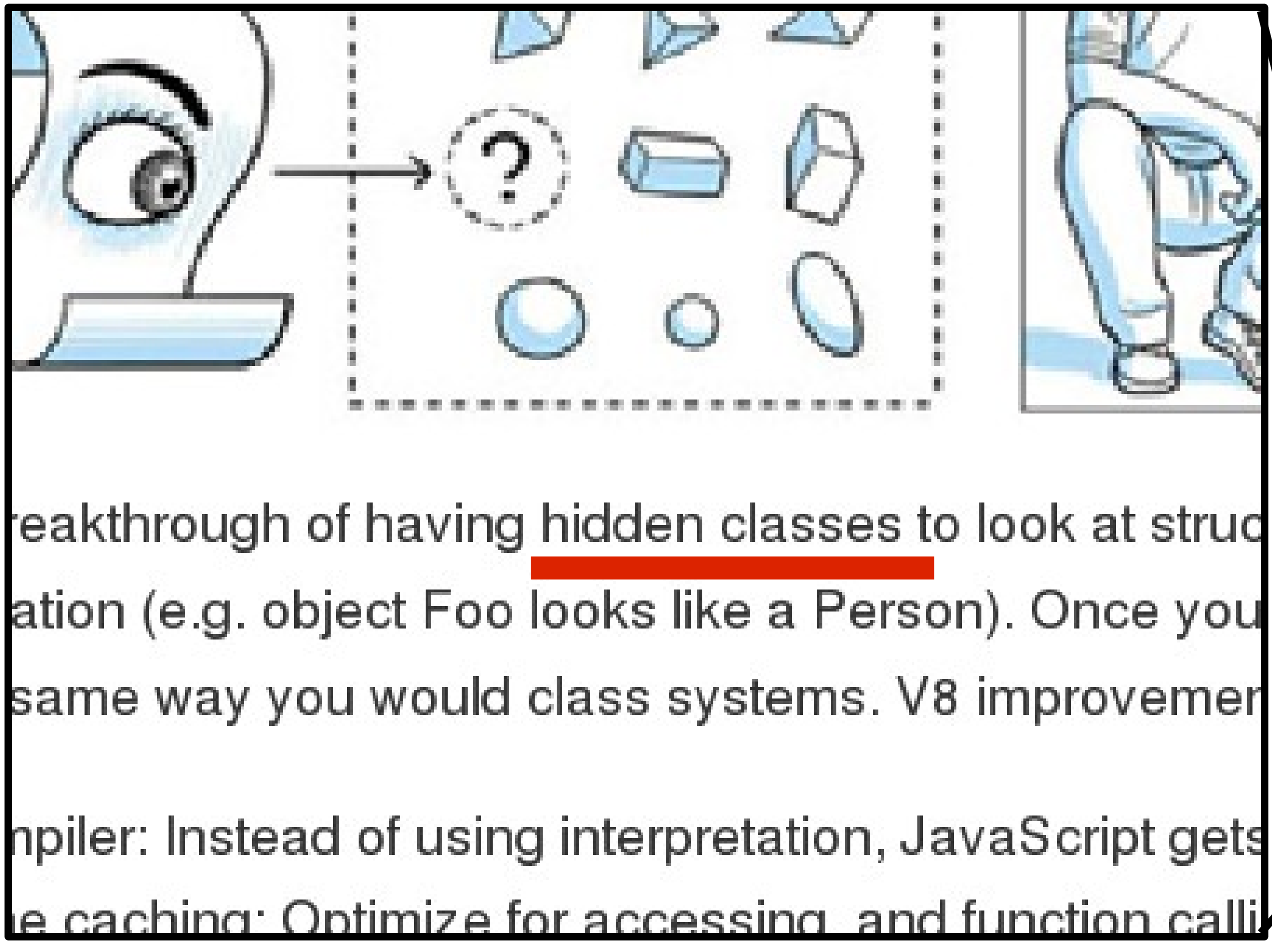
able to keep in sync with Google Chrome the new browser, and Chromium, the e code-base it comes from. There are a couple of innovations that have been great to see such as the multiple process model for tabs and windows, the unified tab and search functionality, and at the core, V8.

V8 is the super-speedy JavaScript VM by Lars Bak of Sun HotSpot fame. When you run uppy, you see very speedy responses indeed. The V8 part of the comic says it well:

JAVASCRIPT ITSELF IS CLASSLESS.

BUT IN V8, AS EXECUTION GOES ON,

ライバルのひみつ



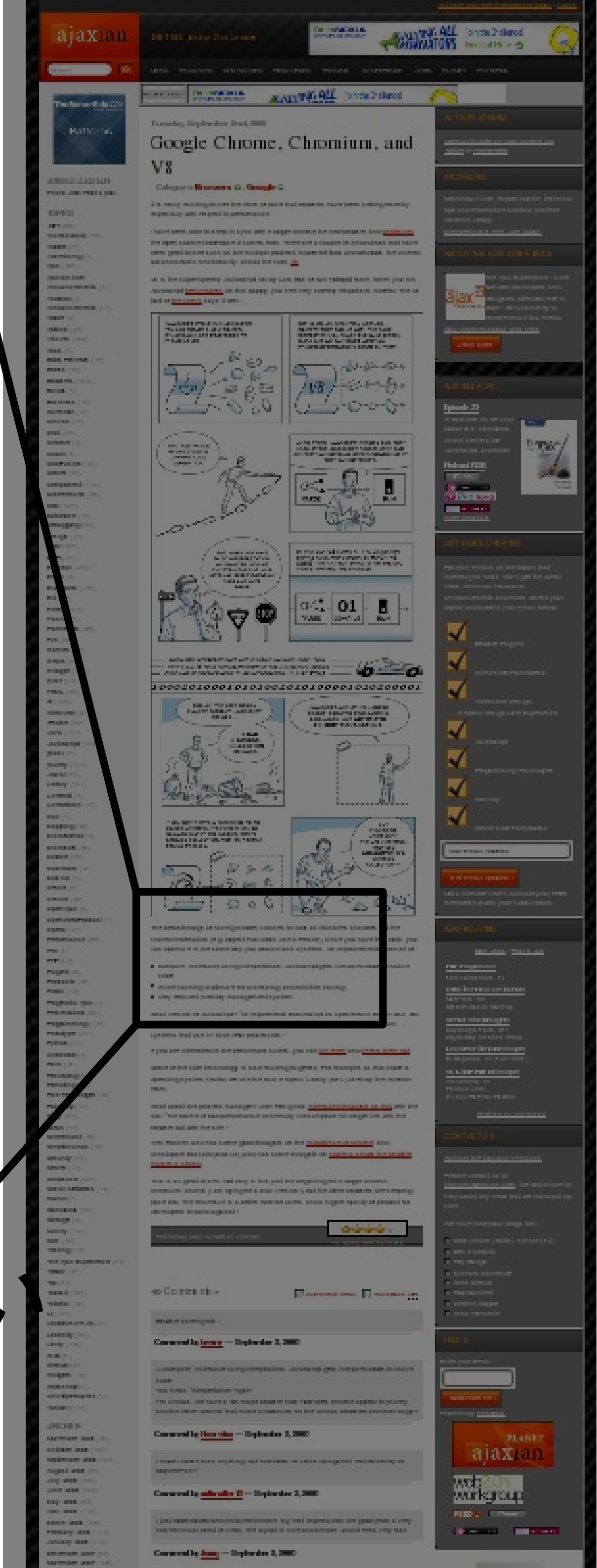
reakthrough of having hidden classes to look at struc

ation (e.g. object Foo looks like a Person). Once you

same way you would class systems. V8 improvemen

npiler: Instead of using interpretation, JavaScript gets

e caching: Optimize for accessing, and function calli

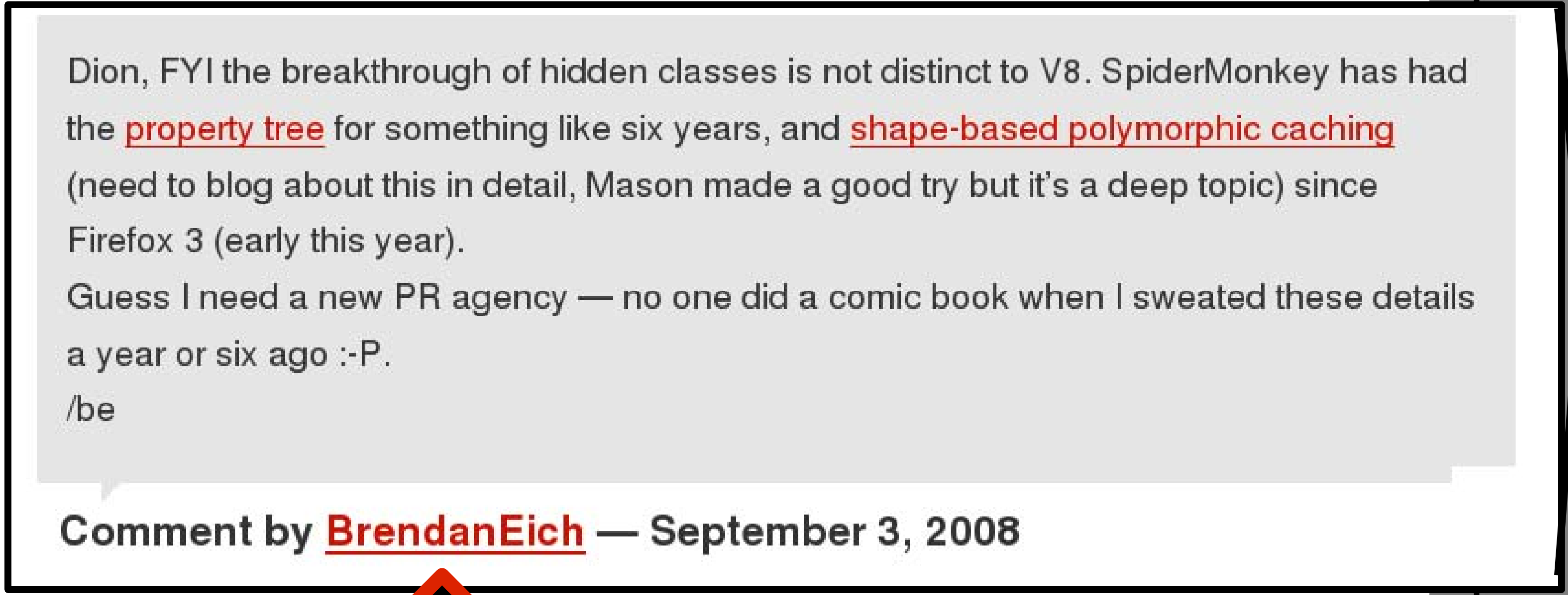


ajaxian

Google Chrome, Chromium, and V8

•**Hidden class** というのがあるらしい

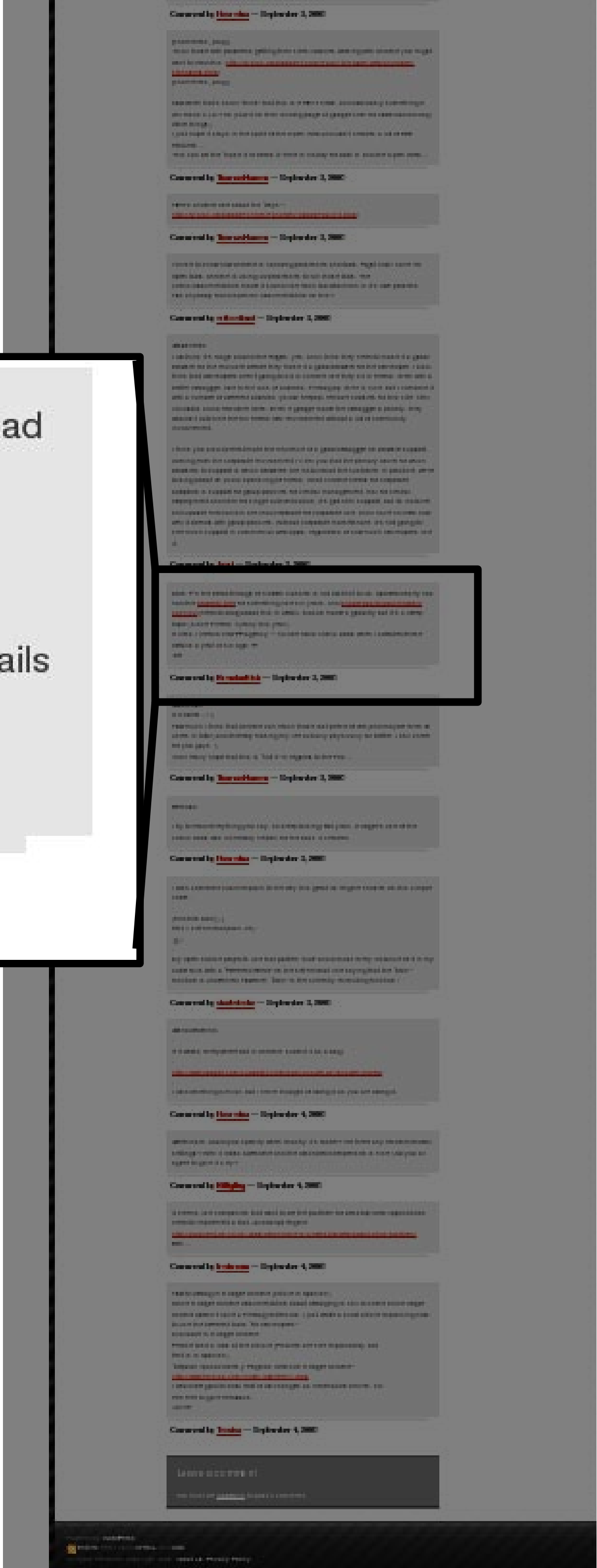
ライバルのコメント欄



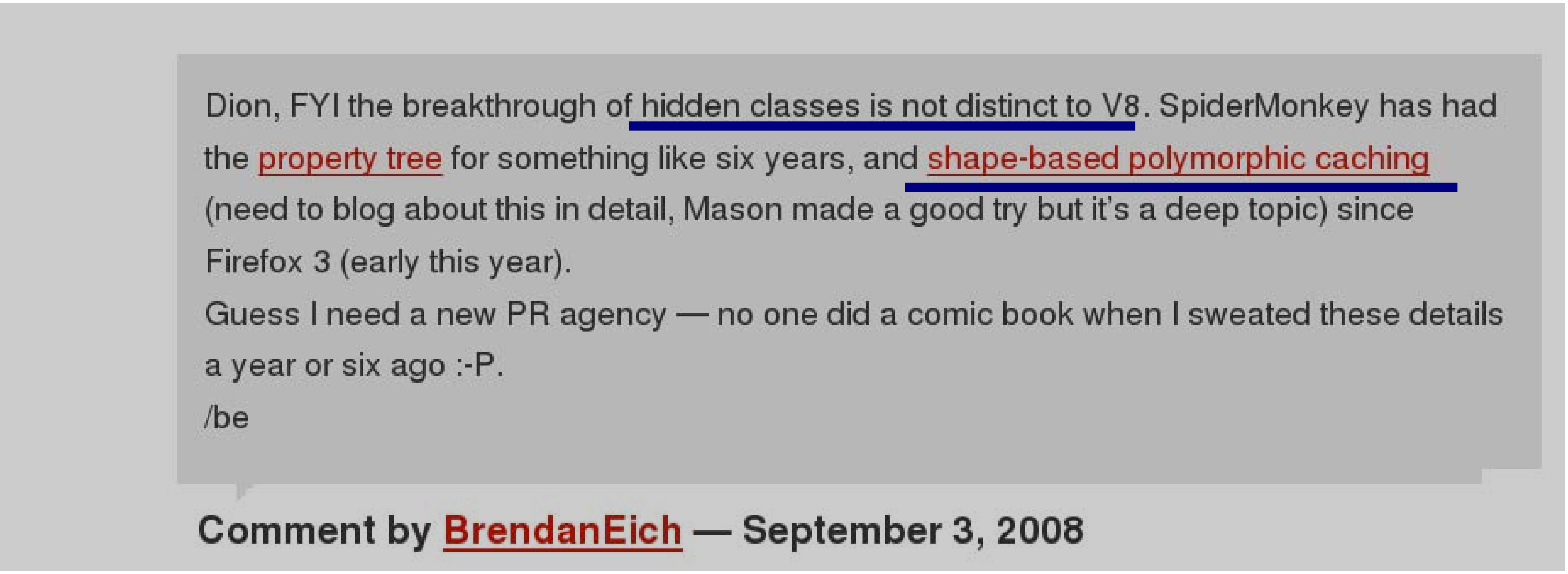
Dion, FYI the breakthrough of hidden classes is not distinct to V8. SpiderMonkey has had the property tree for something like six years, and shape-based polymorphic caching (need to blog about this in detail, Mason made a good try but it's a deep topic) since Firefox 3 (early this year).
Guess I need a new PR agency — no one did a comic book when I sweated these details a year or six ago :-P.
/be

Comment by BrendanEich — September 3, 2008

CTO of Mozilla Corp.



コメント欄の CTO 曰く …



Dion, FYI the breakthrough of hidden classes is not distinct to V8. SpiderMonkey has had the property tree for something like six years, and shape-based polymorphic caching (need to blog about this in detail, Mason made a good try but it's a deep topic) since Firefox 3 (early this year).

Guess I need a new PR agency — no one did a comic book when I sweated these details a year or six ago :-P.

/be

Comment by BrendanEich — September 3, 2008

- Hidden class は別にすごくないぜ
- SpiderMonkey には **Shape based polymorphic caching** てのがあるんだぜ

Shape based polymorphic caching

ある JS コードの **JIT 結果** (機械語) が

- プロパティの**検索結果をキャッシュ**
  - 次回以降のアクセスで使う
- オブジェクトの **shape** が同じなら
キャッシュにヒット
- ヒットすると**高速**にプロパティアクセス

ぎもん

- 検索結果ってなに？
- Shape ってなに？

実行イメージ(キャッシュ前)

JavaScript

```
var a = point.x;
```

初回実行(JIT前)

```
// 検索処理：遅い
ScopeProperty* point_x_sp
    = point->scope->findScopeProperty("x");
// 配列アクセス：速い
Object* a = point->dslot[point_x_sp->slot];
// 検索結果 = ScopeProperty(slot)をキャッシュ
// shapeがキー
ctx->update_cache(point->scope->shape,
                  point_x_sp,
                  __FILE__, __LINE__);
```

実行イメージ(キャッシュ後)

JIT されたコード

```
// キャッシュした shape と一致するか？
if (POINT_SHAPE != point->scope->shape)
  goto cache_miss; // 一致しなかった：ハズレ
~~ScopeProperty* sprop~~
~~  = obj->scope->findScopeProperty("x");~~
~~Object* a = obj->dslot[sprop->slot];~~
// shape が一致した：
// キャッシュした添字で配列アクセス．検索ナシ．
//                          → 爆速！
Object* a = point->dslot[POINT_X_SP_SLOT];
```

Shape?

オブジェクトのもつ**プロパティの種類**のこと

- オブジェクトＡとＢが
  - 同じ**名前**で
  - 同じ**属性**のプロパティを
  - 同じ**順番**に
  - 同じ**数**だけ持っていたら
- ＡとＢは同じ **shape** をもつ

- 細かい面倒は色々あるけど割愛

OK: 同じ

```
function Point(x, y) {
  this.x = x; this.y = y;
}
function Location(x, y) {
  this.x = x; this.y = y;
}
var a = new Point(10, 20);
var b = new Location(30, 40);
```

NG: 違う数

```
function Point(x, y) {
  this.x = x; this.y = y;
}
function Point3D(x, y, z) {
  this.x = x; this.y = y; this.z = z;
}
var a = new Point(10, 20);
var b = new Point3D(30, 40, 50);
```

NG: 違う順序

```
function Point(x, y) {
  this.x = x; this.y = y;
}
function PointYX(x, y) {
  this.y = y; this.x = x;
}
var a = new Point(10, 20);
var b = new PointYX(30, 40);
```

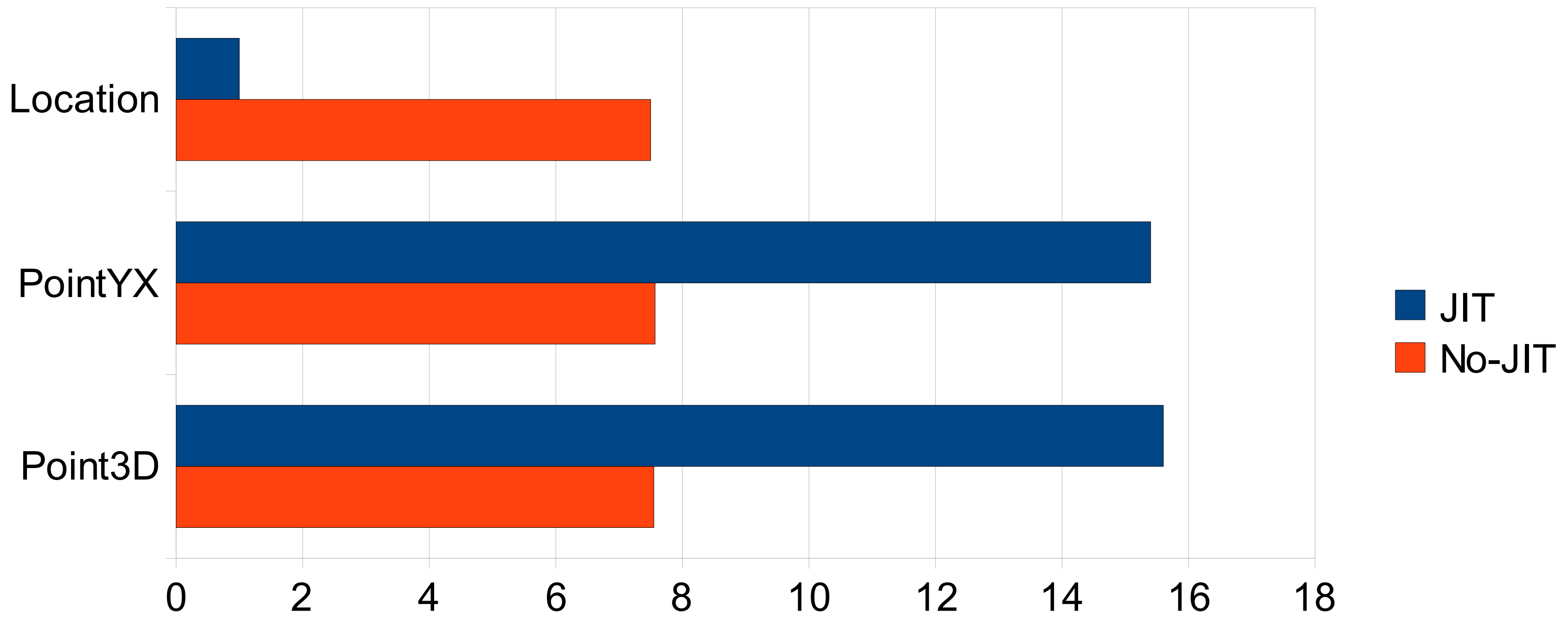
ベンチマーク

```
...
function manhattan_length(p) { return p.x + p.y; }

var P0 = Point;
var P1 = Location; // Point, PointYX, Point3D, ..
function hello() {
  var arr = [new P0(10,20), new P1(30,40)];
  for (var i=0; i<10000000; ++i) {
    for (var j=0; j<2; ++j) {
      manhattan_length(arr[j]);
    }
  }
}

hello();
```

結果 (shorter is faster)



- Shape が同じだと**速い**
- Shape が違うと**遅い**
- JIT なしだと大差ない
- ベンチマーク結果はブランチのものです
  - (Shape 違いで JIT の方が遅いのはたぶんバグ)