عنوان: **ارث بری prototype ای در جاوا اسکریپت به زبان خیلی ساده**
نویسنده: فرید بکران
تاریخ: ۱۳۹۳/۰۸/۲۵ ۲۳:۰
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, AngularJS

انگیزه اصلی این نوشته شروع کار با AngularJs و استفاده از scope در این کتابخانه است. بیشتر دوستانی که کار با این کتابخانه را شروع می‌کنند و تجربه زیادی با جاوا اسکریپت ندارند، با مفهوم ارث بری scope مشکل پیدا می‌کنند.

ارث بری در scope های AngularJs موضوع پیچیده و عجیب و غریبی نیست. در واقع همان ارث بری prototype ای است که جاوا اسکریپت پشتیبانی می‌کند.

این روش توضیح خیلی ساده ای دارد.

در هنگام دسترسی به مقدار یک خصوصیت روی یک شی اگر آن خصوصیت در شی مورد نظر وجود نداشته باشد جاوا اسکریپت یک سطح در زنجیره‌ی prototype ها بالا رفته و به شی پدر دسترسی پیدا کرده و در آن به دنبال مقدار خصوصیت می‌گردد. این کار را آن قدر ادامه می‌دهد تا به بالاترین سطح برسد و دیگر چیزی پیدا نکند.

این بالا رفتن در زنجیره‌ی prototype ها عملا با دسترسی به خصوصیت prototype انجام می‌شود.

فرض کنید دو شی (دقت کنید که می‌گویم شی) به نام‌های employee و person داریم. این دو شی را به صورت زیر تعریف می‌کنیم.

```
var person = { type: '', name:'No Name' };
var employee = {  };
```

شی employee الان هیچ خصوصیت ای ندارد. و دسترسی به هر خصوصیت ای از آن هیچ نتیجه‌ای در بر نخواهد داشت.

```
console.log('Before Inheritance -> employee.name = ' + employee.name);
```

با مقدار دهی کردن خصوصیت prototype مربوط به employee به person این شی را از person ارث بری می‌کنیم.

```
employee.__proto__ = person;
```

بعد از اجرا شدن این خط از برنامه هنگام دسترسی پیدا کردن به مقدار name، مقدار اصلی آن که در شی person وارد شده بود را خواهیم دید.

ملاحظه کردید که وقتی خصوصیت name در شی مورد نظر وجود نداشت به شی پدر رجوع شد و مقدار خصوصیت مربوطه از آن بدست آمد.

الان فرض کنید که در قسمتی از برنامه خواستیم مقدار name در شی employee را به مقدار مشخصی تغییر دهیم. به طور مثال:

```
employee.name = 'farid';
console.log('After Assiginig -> employee.name = ' + employee.name);
console.log('After Assiginig -> person.name = ' + person.name);
```

با چاپ کردن مقادیر person.name و employee.name انتظار دارید چه نتیجه ای ببینید؟

اگر از زبان‌های شی گرایی مانند #C آمده باشید احتمالا خواهید گفت مقادیر یکسان خواهند بود. ولی در واقع این گونه نیست. مقدار person.name همان مقدار اولیه ما خواهد بود و مقدار employee.name نیز ‘farid’.

دلیل این رفتار یک نکته ساده و اساسی است.

جاوا اسکریپت فقط در زمان دسترسی به یک خصوصیت در صورت پیدا نکردن آن در شی مورد نظر ما به سطوح بالاتر prototype ای رفته و دنبال آن خصوصیت می‌گردد.

اگر ما قصد مقدار دهی به یک خصوصیت را داشته باشیم و خصوصیت مورد نظر ما در شی وجود نداشته باشد جاوا اسکریپت یک نسخه محلی از خصوصیت برای آن شی می‌سازد و مقدار ما را به آن می‌دهد.

در واقع در مثال ما هنگام مقدار دهی به employee.name آن خصوصیت در شی موجود نبود و یک نسخه محلی به نام name در شی ایجاد شد و دفعه بعدی که دسترسی به مقدار این خصوصیت اتفاق افتد این خصوصیت به صورت محلی وجود خواهد داشت و جاوا اسکریپت به سطوح بالاتر نخواهد رفت .

تمام کدهای بالا در bin زیر موجود هستند.

Prototypal Inheritance in Javascript

الان فرض کنید شیءهای ما به این صورت هستند:

```
var person = {
  info : { name: 'No Name', type: '' }
};
var employee = {};
```

به همان صورت بالا ارث بری می‌کنیم.

```
employee.__proto__ = person;
```

و سپس name را مقداردهی می‌کنیم.

```
employee.info.name = 'farid';
```

و مقادیر را چاپ می‌کنیم.

```
console.log('After Assiginig -> employee.name = ' + employee.info.name);
console.log('After Assiginig -> person.name = ' + person.info.name);
```

ملاحظه خواهید کرد که مقادیر مساوی هستند.
دلیل این امر به زبان ساده این است که وقتی اقدام به مقدار دهی name در شی employee کردیم در واقع قبل از مقدار دهی اصلی name یک دسترسی به شی info نیاز بود و دسترسی به شیء با استفاده از همان قانونی که مطرح کردیم انجام شده و شیء مربوط به person برگردانده شده است. چون name یک خصوصیت از info است نه employee.
سوالی که می‌توان مطرح کرد این است که در صورت نوشتن این خط کد چه اتفاقی خواهد افتاد؟

```
employee.info = {
  name: 'farhad'
};
```

Prototypal Inheritance with objects

با توجه به مطالب گفته شده باید قادر به حدس زدن نتیجه خواهید بود. **نکته: روش‌های کار با prototype در این نوشته فقط جنبه آموزشی و توضیحی دارد و روش درست استفاده از prototype این نیست.**