JavaScript Object Layout (perspective 2) [Hursh Jain/**mollypages.org**]

(my own simplified assumption…implementations will differ)

Interpreter Startup

1. create *JS environment objects and functions* (in lower level interpreter code) and also make them "visible" to JS code (as JS Objects)

interpreter: function
JS name: "Object"
dictionary: […]

interpreter: function
JS name: "Function"
dictionary: […]

Interpreter Internals

JS interpreter code of Object, Function, etc

All objects can contain arbitrary key/value pairs (maps/dictionary)

parse/run code:

```
function myfoo(){
  …
  }

var f1 = new foo()
```

objects

contains function code

2. parse a webpage for all scripts and script fragments (also create JS objects representing the HTML DOM internally)

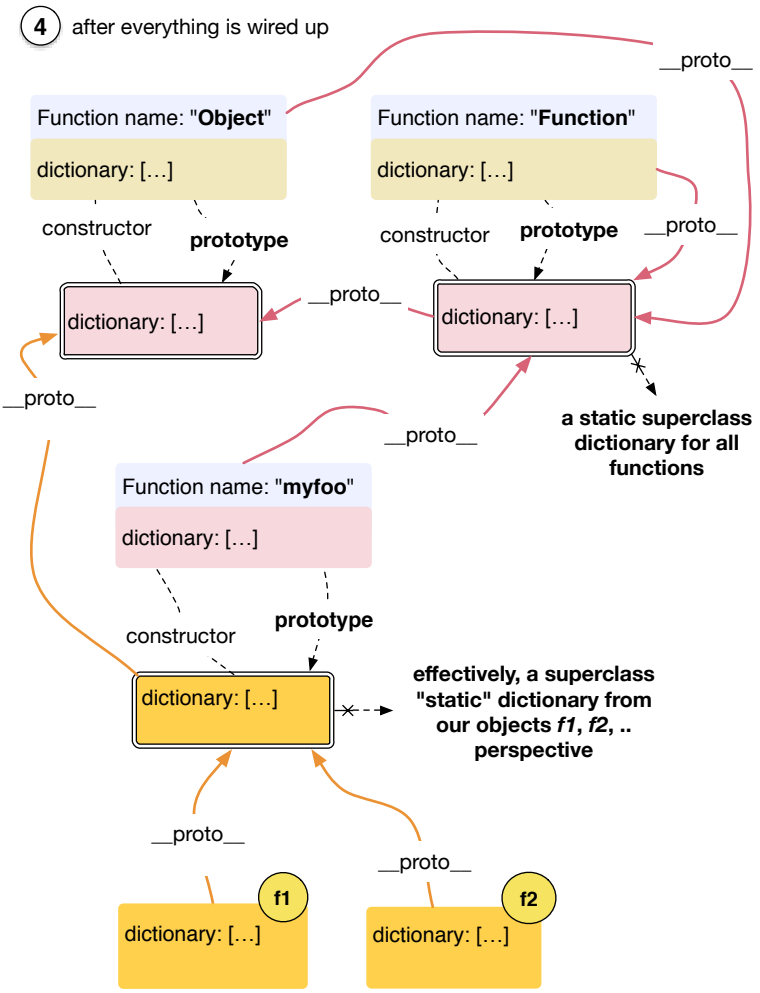
interpreter: function
JS name: "myfoo"
dictionary: […]

3.

JSObject (dictionary)
dictionary: […]

If (for example) using Java to implement the JS interpreter, we could have a *JSObject* class that internally has a Map to store arbitrary data. We could subclass JSObject to create JSFunction, which then additionally could have interpreted code associated with it.

4. after everything is wired up



So, basically:

1. At JS level, any **function**, say for example:
   function myfoo() { }

has a *prototype* property (in it's dictionary) that points to a shared **object**. This object is in variable search scope of all objects created via the **new** operator, like:

var **f1 = new** foo()

The prototype/shared object is similar to a superclass **static** class variable in say, Java (of type *Map*). All instances (f1, f2, etc) have access to the same superclass Map automatically.

This scope is shown in yellow/orange in the above diagram.

---

2. function themselves are *also* presented as "objects" in JS. These function objects are created by a JS "Function" creator , so functions also have their own shared static superclass Map (common to all functions)

This scope is shown in red in the above diagram.

Note 1 : the red and orange chains are **separate**.

Note 2: both scopes (whether or functions or simple objects) end at the prototype for "Object", so **anything set in Object.prototype is visible to everything in the system** (unless over-ridden/shadowed by the same property in a particular object).

As always, the 2 best sources of information are:

1. Mozilla developer documentation (*mozilla.org*)
2. JavaScript: The Definitive Guide, by *David Flanagan*