The Prototype Framework Part II: General JavaScript Support

(Prototype 1.6 Version)

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html

Topics in This Section

- **Element**
  - Helper methods for exploring DOM
  - Helper methods for updating DOM
- **Array**
  - Helper methods that take simple arguments
- **Enumerable**
  - Helper methods that take functions as arguments
- **Function**
  - Helper functions that operate on other functions
    - Usually to build new functions that are based on old functions
- **Number**
  - Methods called *on* numbers

Element

Element Methods: Overview

- **Can be called on an Element**
  - $("some-id").update("some-html")
  - $("some-id").hide()
- **Can be used as static method with element name as first argument**
  - Element.update("some-id", "some-html")
  - Element.hide("some-id")
- **First version will be shown on next slide**
  - E.g., I will say it has methods "hide()" and "update(html)"
  - But you can always use either version
- **Methods that return elements**
  - Always return Prototype extended Element objects

Element Methods: Exploring DOM Tree

- **childElements()**
  - Returns array of children (directly nested elements).
- **cleanWhitespace()**
  - Removes all text nodes which contain only whitespace.
- **firstDescendant()**
  - Returns the first child that is an element. This is opposed to firstChild DOM property which will return any node (often text node containing whitespace).
- **identify()**
  - Returns element id if it exists, otherwise sets unique one and returns it
- **readAttribute(attributeName)**
  - Returns attribute with given name (null if none)
  - Also see writeAttribute method

Element Methods: Modifying DOM

- **addClassName(cssName)**
  - Acts as if element had <... class="cssName">.
- **getStyle(styleName), setStyle(name, value)**
  - Gets/sets value of CSS style (e.g. "font-size", "border")
- **hide(), show()**
  - Makes the given element invisible or visible
- **insert(html, {position: xxxx)**
  - Inserts content: before, after, top, or bottom
- **remove()**
  - Removes element from DOM
- **scrollTo()**
  - Scrolls page so element is visible. Often used with update.
- **update(html)**
  - Inserts html into innerHTML (replacing anything there)

Element Methods: Other

- **Many methods for finding elements that match CSS names**
  - adjacent, down, up, match, next, etc.
- **Many methods for positioning**
  - absolutize, clonePosition, cumulativeOffset, getHeight, getWidth, makePositioned, relativize, etc.
- **Boolean tests**
  - descendantOf, empty, hasClassName, match, visible

Array and Enumerable

New Array Methods

- **Array access**
  - first, last (returns first/last entry in array)
  - indexOf
    - Index of entry that is == to test (-1 if no match)
- **Array modification**
  - clear (makes array empty)
  - compact (removes null and undefined entries)
  - without (removes all specified values)
  - flatten (returns non-nested version of array)
  - reduce (returns entry for 1-element arrays, else returns array unchanged)
  - uniq (removes duplicates)
  - toJSON (returns JSON string version of array)

Enumerable Methods

- **Idea**
  - Arrays support Enumerable, so these are methods you can call on arrays. But other classes (e.g Hash, ObjectRange) do too, so it is worth looking at these methods separately.
- **Main method: each**
  - Calls function on each element
  - Function can take one arg (the element) or two args (the element and the index)
  - Returns the array (Enumerable), so you can chain calls
  - Examples:

```
[1, 2, 3].each(alert);
["a", "b", "c"].each(
  function(entry, index) {
    console.log("Entry %s is %s.", index, entry);
  });
```

Enumerable Methods (Continued)

- **all(funct), any(funct)**
  - Determines if all/any entries pass given function
    - [1,2,3].all(function(entry) { return(entry>0); }) → true
- **find(funct), findall(funct)**
  - Returns first/all entries that pass function.
    - if no match, undefined (find), empty array (findAll)
- **map(funct)**
  - Returns result of applying function to each entry
    - [1,2,3].map(function(n) { return(n*n); }); → [1,4,9]
- **member(element)**
  - Same as (indexOf(element) > -1)
- **partition(funct)**
  - Returns two subarrays: first with all elements that passed function, second with all elements that failed function

Functions

New Functions for Functions

- **bind**
  - Associates function with object so "this" works
- **methodize**
  - Turns func(obj, arg1, arg2) into obj.func(arg1, arg2)
- **curry**
  - Prefills arguments
- **delay**
  - Call a function after some time delay
- **defer**
  - Call a function after a 0.01 second delay
- **wrap**
  - Make a new function that is a composition of an old one
- **argumentNames**
  - Get an array of Strings representing args to a function

bind

- **Problem**
  - When you return a method outside of the bounds of the object, it does not remember what "this" referred to.
    - It now refers to "window"!
- **Solution**
  - Call someFunction.bind(someObject)
  - Now, whenever you call someFunction(...), references to "this" within someFunction refer to someObject

bind: Example

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;

  this.fullName = function() {
    return(this.firstName + " " + this.lastName);
  }
}

var sam = new Person("Sam", "Stephenson");
sam.fullName();                   → "Sam Stephenson"
var getSamName1 = sam.fullName;
getSamName1();                    → "undefined undefined"
window.firstName = "Somebody";
window.lastName = "Else";
getSamName1();                    → "Somebody Else"
var getSamName2 = sam.fullName.bind(sam);
getSamName2();                    → "Sam Stephenson"
```

methodize

- **Idea**
  - Given a function that takes arg1, arg2, ..., argN, returns a function that when you call someObj.method(arg2, ..., argN), calls function(someObject, arg2, ..., argN)
- **Example**

```
function showLocation(person, city, state) {
  return(person.fullName() + " lives in " +
         city + ", " + state + ".");
}
showLocation(sam, "JavaScript Heaven", "California");
  → "Sam Stephenson lives in JavaScript Heaven, California."
Person.prototype.showLocation = showLocation.methodize();
sam.showLocation("Lambdaville", "Ohio");
  → "Sam Stephenson lives in Lambdaville, Ohio."
```

methodize: Examples

- Note: Math.cos(x) is the same as Math["cos"](x)

```
function addNumberMethods() {
  var functionNames = [ 'cos', 'sin', 'sqrt', 'pow' ];
  for(var i=0; i<functionNames.length; i++) {
    var functionName = functionNames[i];
    Number.prototype[functionName] =
      Math[functionName].methodize();
  }
}
addNumberMethods();
3.14159265.cos();     → -1
Math.PI.sin();        → 0 (actually 1E-16)
2.sqrt();             → 1.4142135623730951
10.0.pow(3);          → 1000
```

curry

- **Idea**
  – Builds a function that calls original function with first N arguments already filled in
- **Examples**
  – var twoToThe = Math.pow.curry(2);
  – twoToThe(2); → 4
  – twoToThe(3); → 8
  – String.prototype.splitOnColons =
      String.prototype.split.curry(":");
  – "a:b:c:d:e".splitOnColons(); → ["a", "b", "c", "d", "e"]

delay and defer

- **delay**
  – Idea: calls function after specified delay (delay starts after current function call completes)
  – Examples:
    - Math.cos.delay(0.5, Math.PI);
    - alert(1.0, "Warning!");
- **defer**
  – Idea: calls function after 0.01 second delay
  – Examples:
    - Math.cos.defer(Math.PI);
    - alert.defer("Warning!");
  – How defer was built
    - Function.prototype.defer =
        Function.prototype.delay.curry(0.01)

wrap

- **Idea**
  - Returns a function that gets passed original function plus arguments
  - Lets you make a new function that does everything old function did, plus side effects
  - Similar to Aspect-Oriented Programming (AOP)
- **Example**
  - Make version of alert that logs first

```
var newAlert = alert.wrap(
  function(alert, text) {
    console.log("About to call alert on '%s'.",
                text);
    alert(text);
  });
newAlert("Test");
```

  - Better yet, just override old version of alert

```
alert = alert.wrap(...);
```

argumentNames

- **Idea**
  - Returns an array of the argument names to the function
- **Example**

```
function hypotenuse(leg1, leg2) {
  return(Math.sqrt((leg1*leg1) + (leg2*leg2)));
}
hypotenuse.argumentNames();  → ["leg1", "leg2"]
function bigRandom() {
  return(Math.random() * 10);
}
bigRandom.argumentNames();  → [ ]
```

Number

Methods on Numbers

- **Idea**
  - JavaScript already has toString and toFixed that you can call on numbers
    - Math.PI.toString(); → "3.141592653589793"
    - Math.sqrt(2).toFixed(3); → "1.414"
  - Prototype adds more
    - abs, ceil, floor, round, times, toColorPart, toPaddedString
    - (5).times(function() { ... })
- **Note**
  - Most of these are added to Number.prototype using methodize in the exact same manner as the cos, sin, sqrt, and pow examples shown earlier
  - times is a useful addition for simple loops

Methods on Numbers: Examples

```
(-2).abs();        → 2
Math.PI.ceil();    → 4
Math.PI.floor();   → 3
Math.PI.round();   → 3
(3).times(function() {
            alert("Called three times");
          });
```

Wrapup

Summary

- **Element**
  - childElements, hide, show, update, scrollTo
- **Array**
  - first, last, indexOf, compact, without, uniq
- **Enumerable**
  - all, any, each, find, findAll, map, member, partition
- **Function**
  - bind, methodize, curry, delay, defer, wrap, argumentNames
- **Number**
  - abs, ceil, floor, round
  - times

Questions?