CNU 국립 충남대학교 CHUNGNAM NATIONAL UNIVERSITY

8. JS Objects & Functions

웹프로그래밍

2016년 1학기

충남대학교 컴퓨터공학과

목차

- **JS Objects**
  - Object Definitions
  - Object Properties
  - Object Methods
  - Object Prototypes
- **JS Functions**
  - Function Definitions
  - Function Parameters
  - Function Invocation
  - Function Closures

Object Definitions

- **In JavaScript, almost "everything" is an object.**
  - Booleans can be objects. (or primitive data treated as objects)
  - Numbers can be objects. (or primitive data treated as objects)
  - Strings can be objects. (or primitive data treated as objects)
  - Dates are always objects.
  - Maths are always objects.
  - Regular expressions are always objects.
  - Arrays are always objects.
  - Functions are always objects.
  - Objects are objects.
- **Primitive values are: string("John Doe"), number(3.14), true, false, null, and undefined.**

Object Definitions (cont'd)

- **Objects are Variables Containing Variables.** *Try it!*
  - A JavaScript object is an unordered collection of variables called **named values**.
- **Object Properties**
  - The named values are called **properties.**

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Object Definitions (cont'd)

- **Object Methods**
  - Object properties can be both primitive values, other objects, and functions.
  - An object method is an object property containing a **function definition**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Object Definitions (cont'd)

- **Creating a JavaScript Object**
  - Different ways to create new objects:
    - Define and create a single object, using an object literal. *Try it!*
    - Define and create a single object, with the keyword **new**. *Try it!*
    - Define an object constructor, and then create objects of the constructed type. *Try it!*
- **The *this* Keyword**
  - The thing called ***this***, is the object that "owns" the JavaScript code.
- **Built-in JavaScript Constructors** ***Try it!***

Object Definitions (cont'd)

- **Did You Know?**
  - JavaScript has object versions of the primitive data types String, Number, and Boolean.
  - There is no reason to create complex objects. Primitive values execute much faster. *Try it!*
  - And there is no reason to use new Array(), new RegExp(), new Function(), and new Object(). Use array literals: **[]**, pattern literals: **/()/**, function expressions: **function () {}**, and object literals: **{}**, instead.
- **JavaScript Objects are Mutable.** *Try it!*

Object Properties

Accessing JavaScript Properties

```
objectName.property          // person.age
```

Try it!

or

```
objectName["property"]       // person["age"]
```

Try it!

or

```
objectName[expression]       // x = "age"; person[x]
```

Object Properties (cont'd)

- **JavaScript for … in Loop** *Try it!*
  - Loops through the properties of an objects.
  - Syntax

```
for (variable in object) {
    code to be executed
}
```

- **Adding New Properties** *Try it!*
- **Deleting Properties** *Try it!*
  - **delete** keyword deletes a property from an object.

Object Properties (cont'd)

- **Property Attributes**
  - All properties have a name. In addition they also have a value.
  - The value is one of the property's attributes.
  - Other attributes are: enumerable, configurable, and writable.
  - These attributes define how the property can be accessed.
  - In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).
- **Prototype Properties**
  - JavaScript objects inherit the properties of their prototype.
  - The delete keyword does not delete inherited properties, but if you delete a prototype property, it will affect all objects inherited from the prototype.

Object Methods

- **Accessing Object Methods** *Try it!* *Try it!*
  - Create an object method with the following syntax:

```
methodName : function( ) { code lines }
```

  - Access an object method with the following syntax:

```
objectName.methodName( )
```

- **Using Built-In Methods**
  - Example

```
var message = "Hello world!";
var x = message.toUpperCase( );
```

- **Adding New Methods** *Try it!*

Object Prototypes

- **JavaScript Prototypes**
  - All JavaScript objects inherit the properties and method from their prototype.
  - Objects created using an object literal, or with new Object( ), inherit from a prototype called **Object.prototype**.
- **Creating a Prototype** *Try it!*
- **Adding a Property to an Object** *Try it!*
- **Adding a Method to an Object** *Try it!*

Object Prototypes (cont'd)

- **Adding Properties to a Prototype** *Try it!* *Try it!*
- **Adding Methods to a Prototype** *Try it!*
- **Using the prototype Property**
  - JavaScript **prototype** property allows you to add new properties to an existing prototype. *Try it!*
  - JavaScript **prototype** property allows you to add new methods to an existing prototype. *Try it!*

Function Definitions

- **Function Declarations** *Try it!*
  - Syntax

```
function functionName(parameters) {
    code to be executed
}
```

- **Function Expressions** *Try it!*
  - The function above is actually an anonymous function (a function without a name).

Function Definitions (cont'd)

- **Function Hoisting**

```
myFunction (5);

function  myFunction (y) {
      return  y*y;
}
```

  - Hoisting is JavaScript's default behavior of moving declarations to the top.

- **Self-Invoking Function** *Try it!*

Function Definitions (cont'd)

- **Functions Can Be Used as Values.** *Try it!*
- **Functions are Objects.**
  - The **typeof** operator in JavaScript returns "function" for functions.
  - JavaScript functions have both properties and methods. *Try it!* *Try it!*

Function Parameters

- **Function Parameters and Arguments**

```
functionName(parameter1, parameter2, parameter3) {
    code to be executed
}
```

  - Function **parameters** are the **names** listed in the function definition.
  - Function **arguments** are the real **values** passed to (and received by) the function.

- **Parameter Rules**
  - JavaScript function definitions do not specify data types for parameters.
  - JavaScript functions do not perform type checking on the passed arguments.
  - JavaScript functions do not check the number of arguments received.

Function Parameters (cont'd)

- **Parameter Defaults**
  - If a function is called with **missing arguments** (less than declared), the missing values are set to: **undefined**. *Try it!*
- If a function is called with **too many arguments** (more than declared), these arguments can be reached using **the arguments object**.

- **The Arguments Object** ***Try it!***
  - JavaScript functions have a built-in object called the arguments object.
  - The argument object contains an array of the arguments used when the function was called (invoked).

Function Parameters (cont'd)

- **Arguments are Passed by Value**
  - The parameters, in a function call, are the function's arguments.
  - JavaScript arguments are passed by **value**: The function only gets to know the values, not the argument's locations.
  - If a function changes an argument's value, it does not change the parameter's original value.
  - **Changes to arguments are not visible (reflected) outside the function.**
- **Objects are Passed by Reference**

Function Invocation

- **JavaScript functions can be invoked in 4 different ways.**
- **Invoking a Function as a Function.** *Try it!* *Try it!*
- **The Global Object** *Try it!*
  - When a function is called without an owner object, the value of **this** becomes the global object.
  - In a web browser the global object is the browser window.
- **Invoking a Function as a Method** *Try it!* *Try it!*

Function Invocation (cont'd)

- **Invoking a Function with a Function Constructor** *Try it!*
  - If a function invocation is preceded with the **new** keyword, it is a constructor invocation.
- **Invoking a Function with a Function Method**
  - Both methods takes an owner object as the first argument.
    - **call( )** takes the function arguments separately.
    - **apply( )** takes the function arguments in an array.

```
function myFunction(a, b) {
    return a * b;
}
myFunction.call(myObject, 10, 2);       // Will return 20
```

```
function myFunction(a, b) {
    return a * b;
}
myArray = [10,2];
myFunction.apply(myObject, myArray);    // Will also return 20
```

Function Closures

- **Global Variables**
  - Local variables can only be used inside the function where it is defined. *Try it!*
  - Global variables can be used (and changed) by all scripts in the page (and in the window). *Try it!*
- **Variable Lifetime**
  - Global variables live as long as your application lives.
  - Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

Function Closures (cont'd)

- **JavaScript Nested Functions**
  - Nested functions have access to the scope "above" them. *Try it!*
- **JavaScript Closures** *Try it!*
  - A closure is a function having access to the parent scope, even after the parent function has closed.