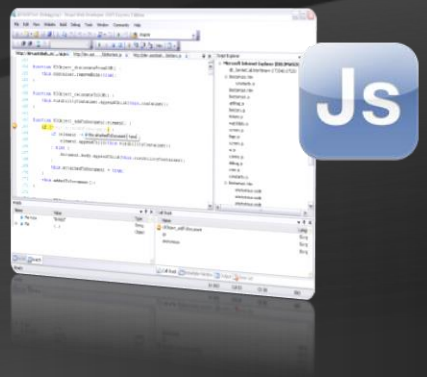
Telerik Academy


# Object and Object.prototype methods



JavaScript OOP

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

# Table of Contents

- Object methods
  - .defineProperty() and .defineProperties()
  - .create()
  - .keys()
  - .preventExtensions() and .isExtensible()
  - .seal() and .isSealed()
  - .freeze() and .isFrozen()
  - .assign()
  - .is()

Telerik Academy

# Table of Contents

- Object.prototype methods
  - .hasOwnProperty()
  - .isPrototypeOf()
  - .propertyIsEnumerable()
  - .valueOf()

Telerik Academy

# Object Methods

Telerik Academy

# Object.defineProperty()

Telerik Academy

# Object.defineProperty()

- Defines a new property directly on an object, or modifies an existing property on an object, and returns the object
- Syntax: Object.defineProperty(obj, prop, desc)
  - obj – the object on which to define the property
  - prop – the name of the property to be defined or modified
  - descriptor – the descriptor for the property being defined or modified

Telerik Academy

# Object.defineProperty()

- Descriptors:
  - data descriptor – property that has a value
  - accessor descriptor – property described by a getter-setter pair of functions
- Both are objects and have required keys
  - configurable – true if property can be changed
  - enumerable – if property shows up during enumeration of the properties on the corresponding object
    - Both default to false

Telerik Academy

# Object.defineProperty()

- Data descriptor has optional keys:
  - value – associated value with the prop
    - Defaults to undefined
  - writable – true if the value can be changed
    - Defaults to false
- Accessor descriptor has optional keys:
  - get – getter function
  - set – setter function
    - Both default to undefined

Telerik Academy

# Object.defineProperties()

Telerik Academy

# Object.defineProperties()

- Defines a new property directly on an object, or modifies an existing property on an object, and returns the object
- Syntax: Object.defineProperties(obj, props)
  - obj – the object on which to define the property
  - props – an object whose own enumerable properties constitute descriptors for the properties to be defined or modified

Telerik Academy

# Object.defineProperties()

- Properties have the following optional keys:
  - configurable, enumerable, value,writable, get and set

```
var obj = {};
Object.defineProperties(obj, {
  'property1': {
    value: true,
    writable: true,
    enumerable: true
  },
  'property2': {
    value: 'Hello',
    writable: false
  }
});
```

Telerik Academy

# Object.create()

# Object.create()

- Creates a new object with the specified prototype object and properties
- Syntax: Object.create(proto[, propertiesObject])
  - proto – the object which should be the prototype of the newly created object
  - propertiesObject – specify property descriptions to be added to the newly-created object

```
function Shape() { … }
function Rectangle() { Shape.call(this); }

Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
```

Telerik Academy

# Object.keys()

# Object.keys()

- Returns an array of a given object's own enumerable properties
  - No properties form the prototype chain
- Syntax: Object.keys(obj)
  - obj – the object whose enumerable own properties are to be returned

```
var obj = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.keys(obj)); // ['0', '1', '2']

var an_obj = { 100: 'a', 2: 'b', 7: 'c' };
console.log(Object.keys(an_obj)); // ['2', '7', '100']
```

Telerik Academy

# Object.preventExtensions() and Object.isExtensible()

Telerik Academy

# Object.preventExtensions()

- Prevents new properties from ever being added to an object
  - the properties may still be deleted
- Syntax: Object.preventExtensions(obj)
  - obj – the object that should be made non-extensible

```
var obj = { removable: true };
Object.preventExtensions(obj);
assert(Object.isExtensible(obj) === true);

Object.defineProperty(obj, 'new', { value: 8675309 });
// throws a TypeError
```

- Object.isExtensible(obj) – determines if an object is extensible

Telerik Academy

# Object.seal()
# and
# Object.isSealed()

Telerik Academy

# Object.seal() and .isSealed()

- Preventing new properties from being added
- Marking all existing properties as non-configurable
- Values of present properties can still be changed as long as they are writable.
- Syntax: Object.seal(obj)
  - obj – the object to be sealed

```
var obj = { prop: function() {}, foo: 'bar' };
var o = Object.seal(obj);
assert(Object.isSealed(obj) === true);

// Changing property values on a sealed object still works
obj.foo = 'quux';
```

- Object.isSealed(obj) – determines if an object is sealed

Telerik Academy

# Object.freeze() and Object.isFrozen()

Telerik Academy

# Object.freeze() and .isFrozen()

- Prevents:
  - new properties from being added
  - existing properties from being removed or changed
- The object is made effectively immutable
- Syntax: Object.freeze(obj)
  - obj – the object to freeze

```
var obj = { prop: function() {}, foo: 'bar' };
var o = Object.freeze(obj);
assert(Object.isFrozen(obj) === true);
obj.foo = 'quux'; // silently does nothing
```

- Object.isFrozen(obj) – determines if an object is frozen

Telerik Academy

# Object.assign()

# Object.assign()

- Copy the values of all enumerable own properties from one or more source objects to a target object
- Syntax: Object.assign(target, ...sources)
  - target – the target object
  - sources – the source object(s)

```
var obj = { a: 1 };
var copy = Object.assign({}, obj);
console.log(copy); // { a: 1 }
```

```
var o1 = { a: 1 }, o2 = { b: 2 }, o3 = { c: 3 };

var obj = Object.assign(o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
console.log(o1);  // { a: 1, b: 2, c: 3 }
```

Telerik Academy

# Object.is()

# Object.is()

- Determines whether two values are the same value
- Syntax: Object.is(value1, value2)
  - value1 and value2 – values to compare

```
Object.is('foo', 'foo');     // true
Object.is(window, window);   // true

Object.is('foo', 'bar');     // false
Object.is([], []);           // false

var test = { a: 1 };
Object.is(test, test);       // true

Object.is(0, -0);            // false
Object.is(NaN, 0/0);         // true
```

Telerik Academy

# Object.prototype Methods

Telerik Academy

# Object.prototype .hasOwnProperty()

Telerik Academy

# .hasOwnProperty()

- Returns a boolean indicating whether the object has the specified property
- Syntax: obj.hasOwnProperty(prop)
  - prop – the name of the property to test

```
o = new Object();
o.prop = 'exists';
o.hasOwnProperty('prop');             // returns true
o.hasOwnProperty('toString');         // returns false
o.hasOwnProperty('hasOwnProperty');   // returns false
```

Telerik Academy

# Object.prototype .isPrototypeOf()

Telerik Academy

# .isPrototypeOf()

- Tests for an object in another object's prototype chain
- Syntax: prototypeObj.isPrototypeOf(obj)
  - prototypeObj – an object to be tested against each link in the prototype chain of the object argument
  - obj – the object whose prototype chain will be searched
- Note: isPrototypeOf differs from the instanceof operator.
  - In the expression "object instanceof AFunction", the object prototype chain is checked against AFunction.prototype, not against AFunction itself

Telerik Academy

# Object.prototype .propertyIsEnumerable()

Telerik Academy

# .propertyIsEnumerable()

- Returns a boolean indicating whether the specified property is enumerable
- Syntax: obj.propertyIsEnumerable(prop)
  - prop – the name of the property to test

```
var o = {};
var a = [];
o.prop = 'is enumerable';
a[0] = 'is enumerable';

o.propertyIsEnumerable('prop');   // true
a.propertyIsEnumerable(0);        // true

Math.propertyIsEnumerable('random');   // false
this.propertyIsEnumerable('Math');     // false
```

Telerik Academy

# Object.prototype .valueOf()

# .valueOf()

- Returns the primitive value of the specified object
- Syntax: object.valueOf()
  - JavaScript automatically invokes it when encountering an object where a primitive value is expected
  - If an object has no primitive value, valueOf returns the object itself

```
var o = new Object();
myVar = o.valueOf();      // [object Object]
```

- You can create a function to be called in place of the default valueOf method (override valueOf)

Telerik Academy

# Object and Object.prototype methods


http://academy.telerik.com