Previous Next

Prototype in JavaScript

JavaScript is a dynamic language. You can attach new properties to an object at any time as shown below.

Example: Attach property to object

```
function Student() {
    this.name = 'John';
    this.gender = 'Male';
}

var studObj1 = new Student();
studObj1.age = 15;
alert(studObj1.age); // 15

var studObj2 = new Student();
alert(studObj2.age); // undefined
```

Try it

As you can see in the above example, age property is attached to studObj1 instance. However, studObj2 instance will not have age property because it is defined only on studObj1 instance.

So what to do if we want to add new properties at later stage to a function which will be shared across all the instances?

The answer is **Prototype**.

The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.

Every function includes prototype object by default.

Prototype in JavaScript

The prototype object is special type of enumerable object to which additional properties can be attached to it which will be shared across all the instances of it's constructor function.

So, use prototype property of a function in the above example in order to have age properties across all the objects as shown below.

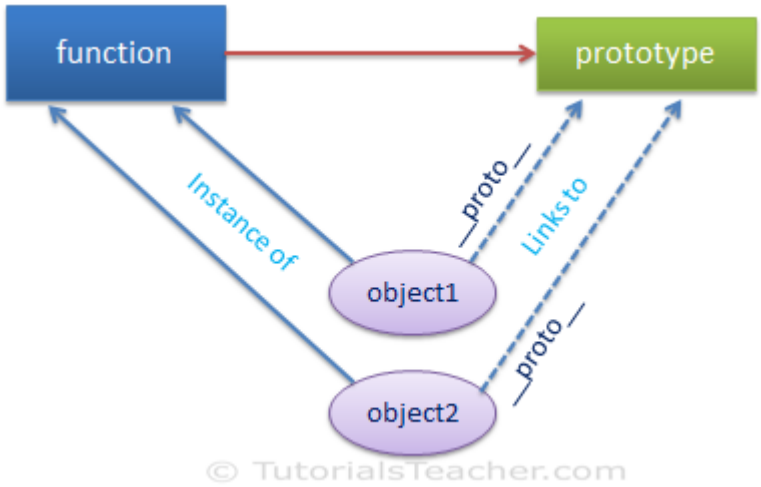
Example: prototype

```
function Student() {
    this.name = 'John';
```

```
    this.gender = 'M';
}

Student.prototype.age = 15;

var studObj1 = new Student();
alert(studObj1.age); // 15

var studObj2 = new Student();
alert(studObj2.age); // 15
```

Try it

Every object which is created using literal syntax or constructor syntax with the new keyword, includes `__proto__` property that points to prototype object of a function that created this object.



Prototype in JavaScript

You can debug and see object's or function's prototype property in chrome or firefox's developer tool. Consider the following example.

Example: prototype

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}

var studObj = new Student();

console.log(Student.prototype); // object
console.log(studObj.prototype); // undefined
console.log(studObj.__proto__); // object

console.log(typeof Student.prototype); // object
console.log(typeof studObj.__proto__); // object

console.log(Student.prototype === studObj.__proto__ ); // true
```

Try it

As you can see in the above example, Function's prototype property can be accessed using <function-name>.prototype. However, an object (instance) does not expose prototype property, instead you can access it using `__proto__`.

> **Note:**
>
> The prototype property is special type of enumerable object which cannot be iterate using for..in or foreach loop.

Object's Prototype

As mentioned before, object's prototype property is invisible. Use `Object.getPrototypeOf(obj)` method instead of `__proto__` to access prototype object.

Example: Object's prototype

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}

var studObj = new Student();

Student.prototype.sayHi= function(){
    alert("Hi");
};

var studObj1 = new Student();
var proto = Object.getPrototypeOf(studObj1);  // returns Student's prototype object

alert(proto.constructor); // returns Student function
```

Try it

The prototype object includes following properties and methods.

Property	Description

Property	Description
constructor	Returns a function that created instance.
__proto__	This is invisible property of an object. It returns prototype object of a function to which it links to.

Method	Description
hasOwnProperty()	Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.
isPrototypeOf()	Returns a boolean indication whether the specified object is in the prototype chain of the object this method is called upon.
propertyIsEnumerable()	Returns a boolean that indicates whether the specified property is enumerable or not.
toLocaleString()	Returns string in local format.
toString()	Returns string.
valueOf	Returns the primitive value of the specified object.

Chrome and Firefox denotes object's prototype as `__proto__` which is public link whereas internally it reference as [[Prototype]]. Internet Explorer does not include `__proto__`. Only IE 11 includes it.

The `getPrototypeOf()` method is standardize since ECMAScript 5 and is available since IE 9.

Changing Prototype

As mentioned above, each object's prototype is linked to function's prototype object. If you change function's prototype then only new objects will be linked to changed prototype. All other existing objects will still link to old prototype of function. The following example demonstrates this scenario.

Example: Changing Prototype

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}

Student.prototype.age = 15;

var studObj1 = new Student();
alert('studObj1.age = ' + studObj1.age); // 15

var studObj2 = new Student();
alert('studObj2.age = ' + studObj2.age); // 15

Student.prototype = { age : 20 };

var studObj3 = new Student();
alert('studObj3.age = ' + studObj3.age); // 20

alert('studObj1.age = ' + studObj1.age); // 15
alert('studObj2.age = ' + studObj2.age); // 15
```

Try it

Use of Prototype

The prototype object is being used by JavaScript engine in two things, 1) to find properties and methods of an object 2) to implement inheritance in JavaScript.

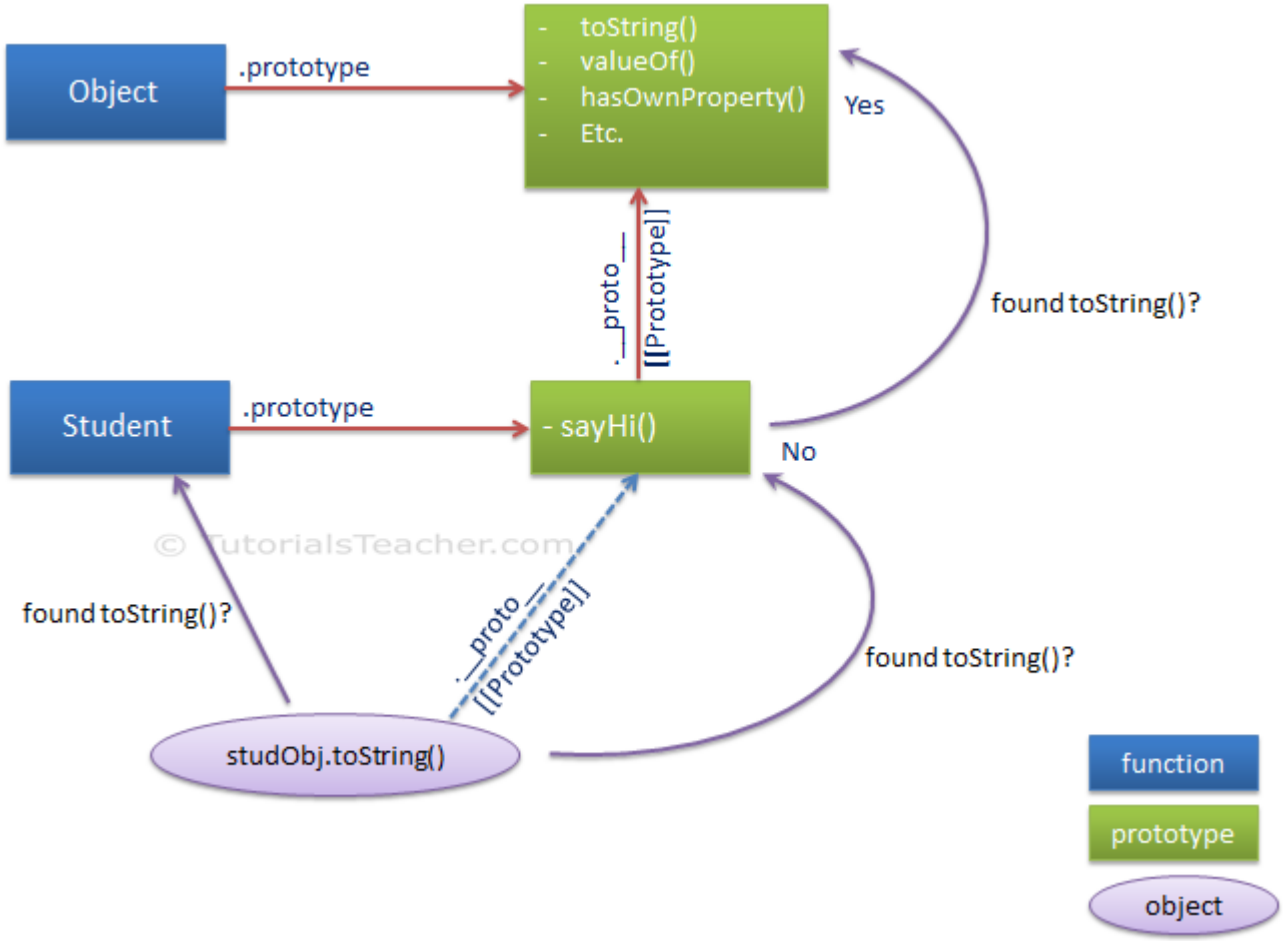
```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}

Student.prototype.sayHi = function(){
    alert("Hi");
};

var studObj = new Student();
studObj.toString();
```

In the above example, toString() method is not defined in Student, so how and from where it finds toString()?

Here, prototype comes into picture. First of all, JavaScript engine checks whether toString() method is attached to studObj? (It is possible to attach a new function to a instance in JavaScript). If it does not find there then it uses studObj's `__proto__` link which points to the prototype object of Student function. If it still cannot find it there then it goes up in the heirarchy and check prototype object of Object function because all the objects are derived from Object in JavaScript, and look for toString() method. Thus, it finds toString() method in the prototype object of Object function and so we can call studObj.toString().

This way, prototype is useful in keeping only one copy of functions for all the objects (instances).

The following figure illustrates the above scenario.



Prototype in JavaScript

Learn how we can achieve inheritance using prototype in the next section.

Previous Next