JavaScript and AJAX

Software Languages Team
University of Koblenz-Landau
Ralf Lämmel and Andrei Varanovich

JavaScript (sometimes abbreviated JS) is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

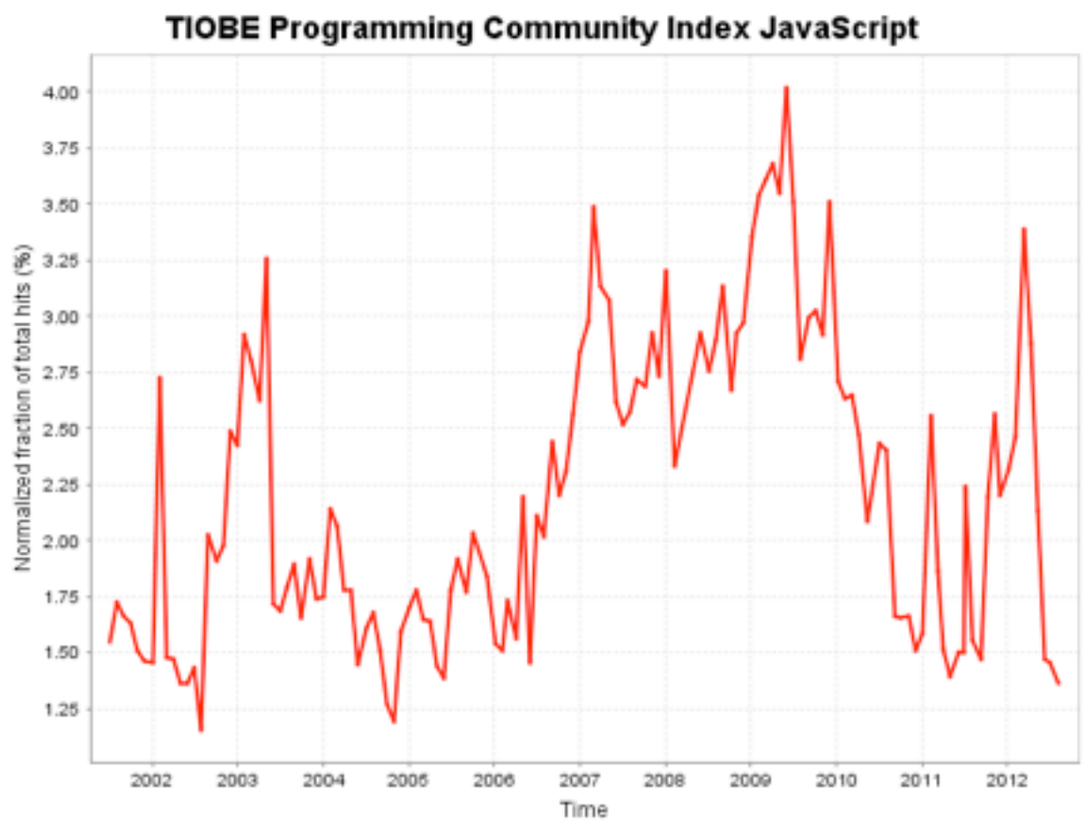
[http://en.wikipedia.org/wiki/JavaScript]

Standardized JavaScript = ECMAScript

http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf

Tiobe popularity

- Highest Rating (since 2001): 4.021%, **8th position**, June 2009
- Lowest Rating (since 2001): 1.154%, **10th position**, July 2002



[http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html]

JavaScript is a very important language :-)

Warning / Disclaimer

- JS might look ***very*** unnatural at first.
- It is essentially ***LISP*** with C-like syntax:
  - Very powerful
  - Very flexible
  - Complicated due to some language design decisions

We will compare JS with Java, erratically.

Basics of JS

Some 'good' part of JS:

```
var add = function (a, b) {
        return a + b;
}
var y = add(2,3)
```

Define and apply a function.

Some 'bad' part of JS:

```
[] + {}
"[object Object]"
{} + []
0
```

The notion of prototype

```
Object.create = function (o) {
  var F = function () {};
  F.prototype = o;
}

meganalysis = {
      "name": "Meganalysis"
};
meganalysis2 = Object.create(meganalysis);
```

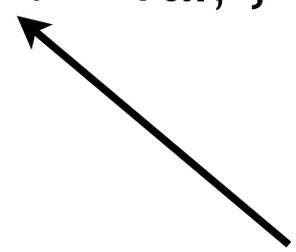
Everything is an Object,
like in Smalltalk :-)

```
meganalysis2 = Object.create(meganalysis);
▼ Object
  ▼ __proto__: Object
    name: "Meganalysis"
    ▼ __proto__: Object
      ▶ __defineGetter__: function __defineGetter__() { [native code] }
      ▶ __defineSetter__: function __defineSetter__() { [native code] }
      ▶ __lookupGetter__: function __lookupGetter__() { [native code] }
      ▶ __lookupSetter__: function __lookupSetter__() { [native code] }
      ▶ constructor: function Object() { [native code] }
      ▶ hasOwnProperty: function hasOwnProperty() { [native code] }
      ▶ isPrototypeOf: function isPrototypeOf() { [native code] }
      ▶ propertyIsEnumerable: function propertyIsEnumerable() { [native code] }
      ▶ toLocaleString: function toLocaleString() { [native code] }
      ▶ toString: function toString() { [native code] }
      ▶ valueOf: function valueOf() { [native code] }
```

The Method Invocation Pattern

```
var company = {
    total: 1000,
    increment: function(val)  { this.total += val; }
}

company.increment(100);
console.log(company.total);
```



'this'/local scope: "company" object

company - object
total - public property
increment - public method

Think in Java: no classes???

The Function Invocation Pattern

```
add = function (a,b) {
    console.log(this);
    return a+b;
}
x = add(2,3);
```

'this'/local scope: a **global** object

```
add = function (a,b) { debugger;
    console.log(this);
    return a+b;
}
x = add(2,3);
}
```

```
Window
  Infinity: Infinity
▶ $: function (selector,context){return ne…
▶ $j: function (selector,context){return n…
▶ Array: function Array() { [native code] }
▶ ArrayBuffer: function ArrayBuffer() { [n…
▶ Attr: function Attr() { [native code] }
▶ Audio: [object Function]
▶ AudioProcessingEvent: function AudioProc…
▶ BeforeLoadEvent: function BeforeLoadEven…
▶ Blob: function Blob() { [native code] }
▶ Boolean: function Boolean() { [native co…
▶ CDATASection: function CDATASection() { …
▶ CSSCharsetRule: function CSSCharsetRule(…
▶ CSSFontFaceRule: function CSSFontFaceRul…
▶ CSSImportRule: function CSSImportRule() …
▶ CSSMediaRule: function CSSMediaRule() { …
▶ CSSPageRule: function CSSPageRule() { [n…
▶ CSSPrimitiveValue: function CSSPrimitive…
```

JS runs in the web browser.

The global object is ***Window***.

Constructor Invocation Pattern

```
// Create a constructor function for employees.
var Employee = function (name) {
    this.name = name;
};

// Give all employees a public method.
Employee.prototype.get_name = function ( ) {
    return this.name;
};

// Make an instance of Employee.
var ralf = new Employee('Ralf');
name = ralf.get_name();
console.log(name);
```

Think in Java:
constructor invocation

```
var Employee = function (name) {
      this.name = name;
};
Employee.prototype.get_name = function (  ) {
      return this.name;
   };
var ralf = new Employee('Ralf');
ralf.name = “Andrei”
name = ralf.get_name();
```

Q: What is the value of the name?

A: Think in Java: We need to ‘hide’ properties.

```
var ralf = (function () {
      var name = “Ralf”;
      return {
         getName: function (  ) {
            return name;
         }
}; }());
```

```
ralf.getName:
function () { return name; }
```

versus

```
ralf.getName():
“Ralf”
```

“name” is hidden

```
> ralf
  ▼ Object
    ▼ getName: function ( ) {
        arguments: null
        caller: null
        length: 0
        name: ""
      ▶ prototype: Object
      ▶ __proto__: function Empty() {}
    ▼ __proto__: Object
      ▶ __defineGetter__: function __defineGetter__() { [native code] }
      ▶ __defineSetter__: function __defineSetter__() { [native code] }
      ▶ __lookupGetter__: function __lookupGetter__() { [native code] }
      ▶ __lookupSetter__: function __lookupSetter__() { [native code] }
      ▶ constructor: function Object() { [native code] }
      ▶ hasOwnProperty: function hasOwnProperty() { [native code] }
      ▶ isPrototypeOf: function isPrototypeOf() { [native code] }
      ▶ propertyIsEnumerable: function propertyIsEnumerable() { [native code] }
      ▶ toLocaleString: function toLocaleString() { [native code] }
      ▶ toString: function toString() { [native code] }
      ▶ valueOf: function valueOf() { [native code] }
```

Inheritance

```
var Person = function (name) {
      this.name = name;
      this.isHuman = true;
};
var Employee = function (name) {
      this.name = name;
};
Person.prototype.isHuman = function(){
    return this.isHuman;
};
Person.prototype.toString = function(){
  return '[Person "'+this.name+'"]';
};

// Here's where the inheritance occurs
Employee.prototype = new Person();

// Otherwise instances of Employee
would have a constructor of Person
Employee.prototype.constructor = Employee;

Employee.prototype.toString = function(){
  return '[Employee "'+this.name+'"]';
};
```

Think in Java:
toString is overridden.

JS is not the 'best' OO language.
Why should I care?

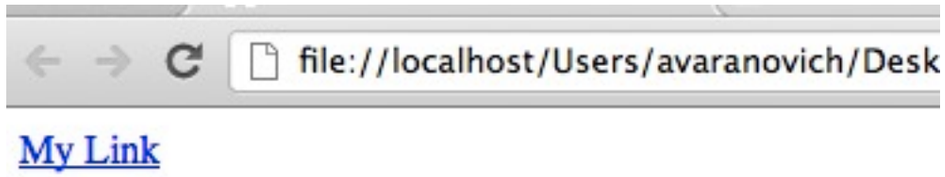
Because it's ***the*** language in the Web browser:

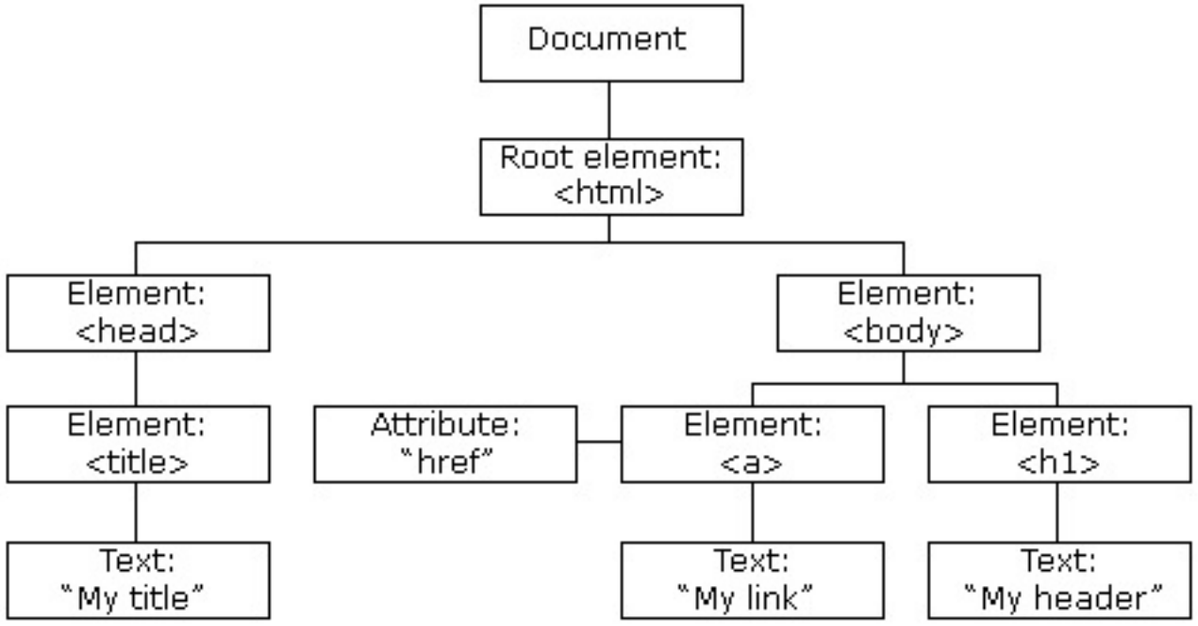
Client-side scripting
Front-end development
Interactive web applications

} = JavaScript

HTML Document Object Model

```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="#">My Link</a>
    <h1>My header</h1>
  </body>
</html>
```





DEMO

```html
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="#">My Link</a>
    <h1>My header</h1>
      <button id ="createButton">Click
me</button>
  </body>
</html>
```

```javascript
var button =
document.getElementById("createButton");
button.addEventListener("click", function() {
    alert("Click!");
}, false);
```

CSS

Result

My Link

My header

Click me

HTML DOM Event Handling

http://jsfiddle.net/DrGigabit/aQctY/1/

```html
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="#">My Link</a>
    <h1>My header</h1>
      <button id ="createButton">Click me</button>
  </body>
</html>
```

```javascript
var button = document.getElementById("createButton");
button.addEventListener("click", function() {
    alert("Click!");
}, false);
```

asynchronously = interactive UI

Function as arguments (callbacks)

```
// Define a function on two number args and a function arg.
function randomBlock(arg1, arg2, callback) {
 // Generate a random number between arg1 and arg2.
 var rnd = Math.ceil(Math.random() * (arg2 - arg1) + arg1);
 // Pass the result to the function argument.
 callback(rnd);
}
// Apply randomBlock to an anonymous function.
randomBlock(5, 15, function(arg) {
 // This anonymous function will be applied later.
 console.log("Callback called with arg = " + arg);
});
```

Motivating scenario: Asynchronous input/output

Make a request ***synchronously***

```
request = prepare_the_request(...);
response = send_request_synchronously(request);
zzzzZZZZZZZzzz <--- Waiting time
display(response);
```

Make a request ***asynchronously***

```
request = prepare_the_request(....);
send_request_asynchronously(request, function (response) {
    display(response);        <--- When ready
});
doSomethingElse();
```

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

jQuery

```
var button = $('#createButton');
button.click(function(){
     alert('clicked');
});
```

plain JS

```
var button = document.getElementById("createButton");
button.addEventListener("click", function() {
    alert("Click!");
}, false);
```

```
$('#createButton') == document.getElementById("createButton");
```

Another DOM Manipulation

```
h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

+

```
$('.inner').append('<p>Test</p>');
```

=

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    Hello
    <p>Test</p>
  </div>
  <div class="inner">
    Goodbye
    <p>Test</p>
  </div>
</div>
```

Asynchronous JavaScript and XML (AJAX)

Motivation

We know how to do client-side programming in JavaScript.

How do we interact with the server?

What's AJAX?

- AJAX = Asynchronous JavaScript and XML
- Make asynchronous requests to the server.
- Receive response eventually through callback.
- Support based on ***XMLHttpRequest*** object.
- 'No page refresh'

AJAX example: *loading company data from the server*

```
var company = {};

company.response;

company.loadData = function() {
   var xhr = new XMLHttpRequest();
   xhr.open('GET', 'company.xml', true);

   xhr.onload = function(e) {
      if (this.status == 200) {
         company.response = xhr.responseXML;
         controller.loadInner();
      }
   };

   xhr.send();
}
```

Prepare request object

Point to resource

Register response handler

Send actual request

DEMO

101implementation:html5XMLHttpRequest

Show XHR (XmlHttpRequest)
in a 101implementation.

Summary

You learned ...

- why JavaScript is important for the Web,
- how to handle HTML events in JavaScript,
- how jQuery helps to simplify your client-side code,
- the basic principles of AJAX,
- how to utilize AJAX in client-server applications,
- how to use the AJAX API on the client side.

Resources

- https://developer.mozilla.org/en-US/docs/AJAX