Client side presentation

PA 165, Lecture 12

Martin Kuba

Outline

- JavaScript basics
- TypeScript
- JSON, REST API
- jQuery
- AngularJS
- browser APIs

SaaS architecture

- Software-as-a-Service can divide presentation layer to two locations – server and client
- traditional approach
  - thin HTML client, HTML generated on server
  - use SpringMVC and JSPs
- recent approach
  - server provides REST API
  - fat client in HTML5+JavaScript or native app
  - use SpringMVC for REST API
  - use some JavaScript framework for browser

JavaScript

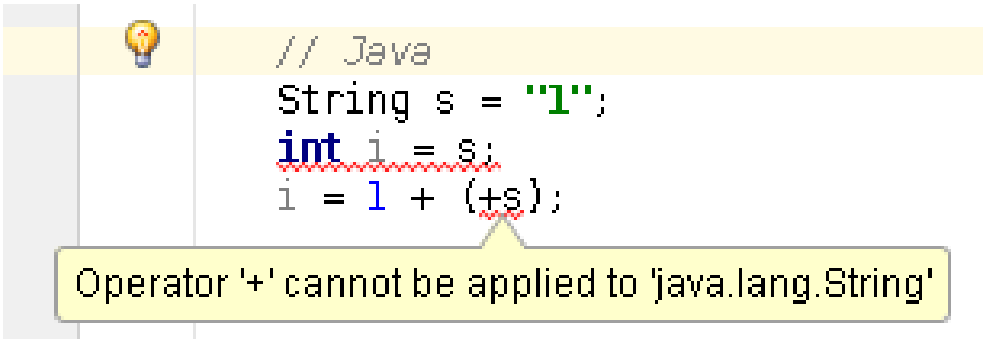
JavaScript for Java programmers

- syntax of both is based on C

```
for (i = 0; i < 10; i++) { }

if (i > 0) { } else { }

while (i<10) { }

do {  } while (i > 0);

switch (s) {
    case "A":
        break;
    case "B":
        break;
    default:
}

i = (s == null) ? 0 : 1;
```

Java / JavaScript differences

- types
  - Java is **strongly statically** typed
  - JavaScript is **weakly dynamically** typed




```javascript
//JavaScript
var s = "1";
var i = 1;
i = s;
i= 1 + (+s); // result is i==2
```

Java / JavaScript differences

- types in JS: Boolean, Number, String, Object, Null
- literals

```java
//Java
String s = "SSS\u20AC";
char ch = '\n';
Object o = null;
boolean b = true;
int i = 1;
long l = 0xFFFF_FFFF_FFFF_FFFFL;
byte bl = 0b0111111;
float f = 0.000_001f;
double d = 1.0e-6d;
String[] arr = new String[]{"A", "B"};
```

```javascript
//JavaScript
var s1 = 'SSS';
var s2 = "SSS\u20AC";
var o1 = null;
var o2 = undefined;
var b = true;
var num1 = 1;
var num2 = 0xFFFF;
var num3 = 1.0E-6;
var num4 = Number.NaN;
var arr = [ 'A', 'B' ];
var regex =/[a-z]\d+/gi;
var obj = {
    a: 'a',
    '!@#$%^': 1,
    m : function(x) {}
};
```

Java / JavaScript differences

- Object orientation
  - Java is based on classes
  - JavaScript has no classes, only instances
  - JavaScript is prototype-based language
  - in JavaScript functions are first-class objects and can be passed as arguments

```
var myobj = { };
myobj.a = 3;
myobj["b"] = true;
myobj.m = function(x,y) { return x + y + this.a; };
myobj.m(1,2);
```

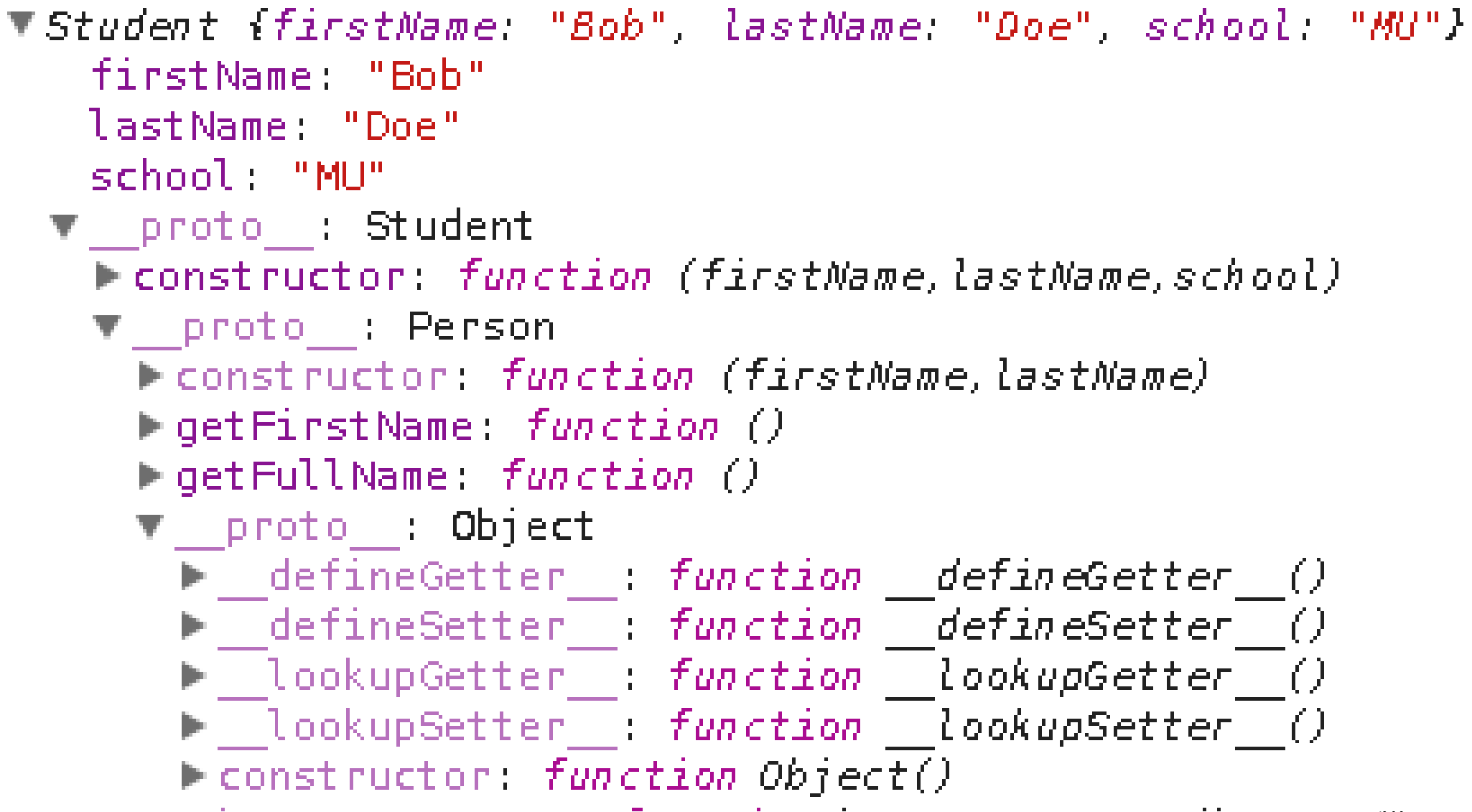
```
var f1 = function(x) { return x + 1 };
var higher_order_func = function(f, y) { return f(5) + y };
```

Class-based vs Prototype-based

```java
// Java
class Person {
    String firstName;
    String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    String getFirstName() {
        return firstName;
    }

    String getFullName() {
        return firstName + " " + lastName;
    }
}

class Student extends Person {
    String school;

    Student(String firstName, String lastName, String school) {
        super(firstName, lastName);
        this.school = school;
    }
}
```

```javascript
//JavaScript

//constructor assigning properties
var Person = function (firstName,lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
};
//methods shared by all instances of Person
Person.prototype.getFirstName = function() {
    return this.firstName;
};
Person.prototype.getFullName = function() {
    return this.firstName +' '+this.lastName;
};

//Student inherits from Person
var Student = function(firstName,lastName,school) {
    Person.call(this,firstName,lastName);
    this.school = school;
};
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

//creating new instances
var person1 = new Person("Alice","Smith");
var student1 = new Student("Bob","Doe","MU");
console.log(student1.getFullName());
```

Console in Chrome (F12)



TypeScript

- http://www.typescriptlang.org/
- strict superset of JavaScript which adds **optional static typing** and **class-based object-oriented programming**
- existing JavaScript programs are also valid TypeScript programs
- transcompiled to JavaScript source during development
- definition files for existing JavaScript libraries

TypeScript example

```
//TypeScript
class Person {
    constructor(private firstName: string, private lastName: string) {
    }
    getFirstName():string {
        return this.firstName;
    }
    getFullName():string {
        return this.firstName+' '+this.lastName;
    }
}

class Student extends Person {
    constructor(firstName: string,lastName: string,private school: string) {
        super(firstName,lastName);
    }
}

var s:Student = new Student("John","Doe","MU");
console.log(s.getFullName());
```

Variable scopes in JavaScript

- JS does not have block-level scope ! Only
  - global scope
    - outside of function (with or without **var**)
    - inside of function without **var**
  - function-level scope
    - inside of function with **var**
    - function parameters

```
var firstName = 'Richard';
for(var i=0;i<10;i++) {
    var firstName = 'Bob';  // not block-level scope !
}
console.log('firstname='+firstName); //prints Bob
```

Closure in JavaScript

- a closure is a function having access to the parent scope, even after the parent function has closed
- in other words, the function defined in the closure remembers the environment in which it was created

```
function adder(a){
    return function(b) { return a + b; };
}
var add5 = adder(5);
console.log( add5(10) ); //prints 15
```

Exceptions in JavaScript

```
<script>
    function MyException(message) {
        this.message = message;
        this.name = "MyException";
    }

    try {
        nonexistentfunction();
        throw "some string";
        throw new MyException("we have a problem");
    } catch (ex) {
        // prints: ReferenceError: nonexistentfunction is not defined
        console.log(ex);
    } finally {
        console.log("finally is the same as in Java");
    }

</script>
```

Threads in JavaScript

- a single execution thread
- no multi-threading
- can use setTimeout(func,timeout) for scheduling in the single thread
- can use WebWorkers API for executing a separate JS file in a separate thread (requires MSIE 10+, Android browser 4.4+, Chrome 4+, Firefox 3.5+) - still a W3C Draft as of 2015-11

Google Web Toolkit (GWT)

- allows writing JavaScript applications in Java
- provides
  - cross-compiler from Java to JavaScript
  - JavaScript implementation of classes in the java.lang and java.util packages
  - Web UI class library for creating widgets
  - development mode browser plugin

JSON

- **J**ava**S**cript **O**bject **N**otation http://json.org/
- in essence JavaScript literals without functions
- combination of objects, arrays and primitive values (string, boolean, number, null)
- lightweight data-interchange format
- easy to parse in any programming language
- in Java, use the **Jackson JSON processor**
- in JS, use **JSON.parse()** and **JSON.stringify()**

Toggle Collapsed | use cfdump format:

Query: - JSONQuery


```
{
    id: 25,
    name: "Strawberries",
    description: "Very tasty and exceptionally red strawberries.",
    color: "RED",
    addedDate: "2015-09-19 00:00",
  - categories: [
      - {
            id: 1,
            name: "Food"
        }
    ],
  - priceHistory: [ … ],
  - currentPrice: {
        id: 100,
        value: 80,
        currency: "CZK",
        priceStart: "2015-11-19 00:00"
    }
}
```

REST and AJAX

- REST – **Re**presentational **S**tate **T**ransfer
  - architectural style for highly scalable APIs
  - today uses JSON messages carried by HTTP to resources identified by URLs
- AJAX – **A**synchronous **J**avascript **A**nd **X**ML
  - catchy name, but in fact uses JSON instead of XML
  - Ajax is a Greek mythology hero from Trojan war
  - XMLHttpRequest asynchronously communicates with a REST API

Hypertext Application Language

- format for JSON messages in REST APIs
- HATEOAS (Hypertext As The Engine Of Application State)
- Richardson maturity model Level 3
  - Level 3 introduces discoverability, providing a way of making a protocol more self-documenting
- every object has _**links** property
- collections are wrapped in _**embedded**

HAL example


```
{
  - _embedded: {
      - categories: [
          - {
                id: 1,
                name: "Food",
              - _links: {
                  - self: {
                        href: "http://localhost:8080/eshop/api/v1/categories/1"
                    },
                  - products: {
                        href: "http://localhost:8080/eshop/api/v1/categories/1/products"
                    }
                }
            },
          - {
                id: 2,
                name: "Office",
              - _links: {
                  - self: {
                        href: "http://localhost:8080/eshop/api/v1/categories/2"
                    },
                  - products: {
                        href: "http://localhost:8080/eshop/api/v1/categories/2/products"
                    }
```

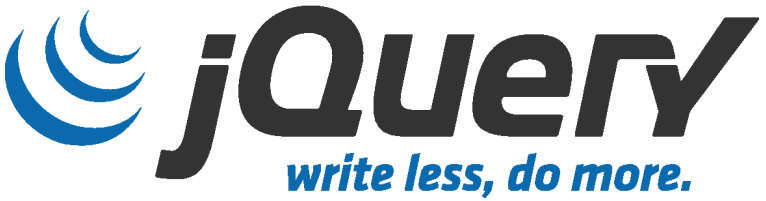
Cross-Origin Resource Sharing (CORS)

- the *same-origin security policy* in browsers disallows scripts to access data from URLs with different protocol, host or TCP port
- thus AJAX requests to other sites are disallowed
- CORS (http://www.w3.org/TR/cors/) allows cross-origin requests
- based on special HTTP headers
- browser asks server using `Origin:` HTTP header
- server may allow with `Access-Control-Allow-Origin:` response HTTP header

JavaScript Frameworks

jQuery

- cross-platform JavaScript library designed to simplify the client-side scripting of HTML
- the most popular JavaScript library in use today
- DOM element selections, traversal and manipulation
- DOM (Document Object Model) is a tree-structure representation of all the elements of a web-page
- jQuery invented “selector engine” which led to the standardization of “Selectors API” by W3C
- can be hosted locally or used from a CDN (Content Delivery Network)



jQuery AJAX

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Product AJAX</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
    <script>
        function loadProduct(id) {
            $.ajax({
                type: 'GET',
                url: '/eshop-rest/products/'+id,
                success: function (data) {
                    $('#name').text(data.name);
                    $('#price').html('<i>'+data.currentPrice.value+' '+data.currentPrice.currency+'</i>');
                }
            });
        }
    </script>
</head>
<body>

<button onclick="loadProduct( $('#prod_id').val() )">Load </button><input id="prod_id" value="2">

<div id="product">
    <div id="name"></div>
    <div id="price"></div>
</div>
```

Node.js

- open-source, cross-platform runtime environment for developing **server-side** web applications in JavaScript
- uses Google V8 JavaScript engine
- operates on a single thread, using non-blocking I/O calls
- has a package manager **npm** for publishing and sharing Node.js libraries
- apps can be written in JavaScript, TypeScript, CoffeeScript, Dart or any other language transcompiled to JavaScript

Bower

- package management system for **client-side** programming on the World Wide Web
- depends on Node.js and npm

AngularJS

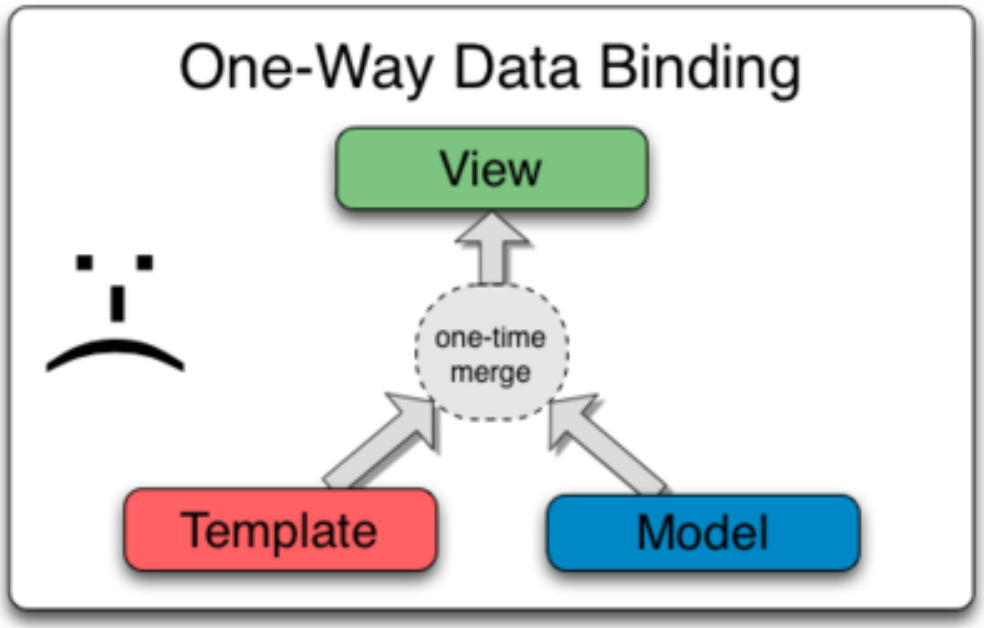
- open-source web application framework for single-page applications

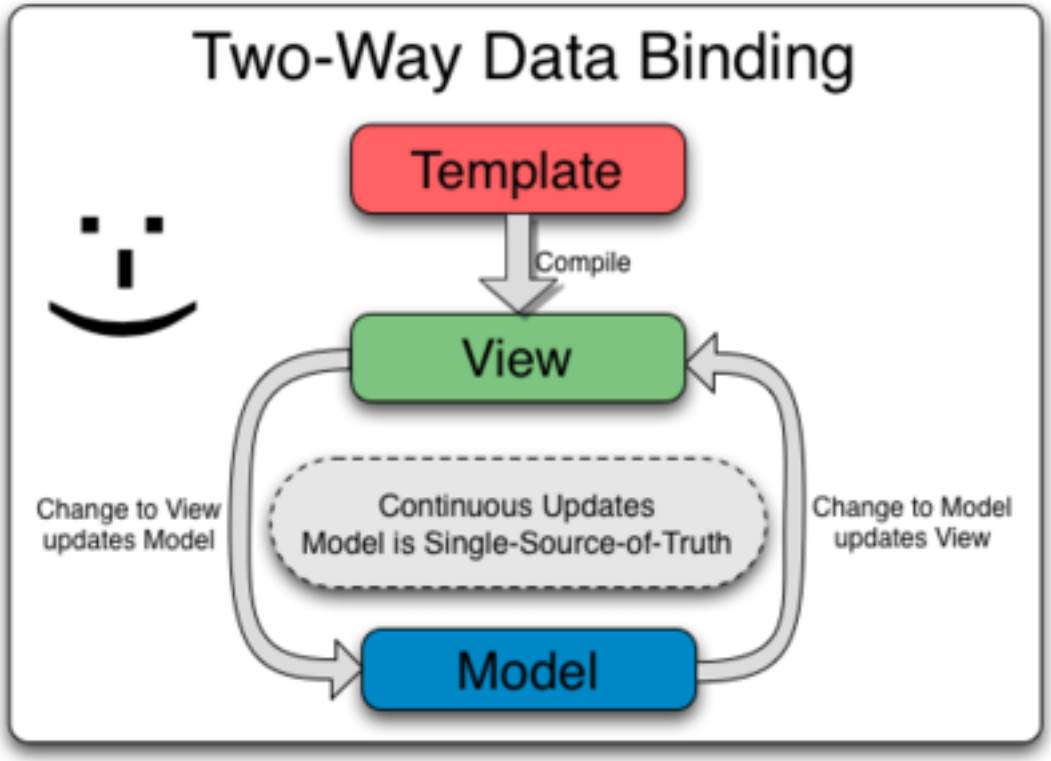
AngularJS terminology

- a **template** is HTML with Angular-specific elements and attributes
- a template is processed by the **compiler** and rendered into a **view** that a user sees
- a **directive** is a marker on a DOM element (attribute, element, comment or CSS class) that attaches a specified behavior
  - e.g. the attribute `ng-repeat` repeats the HTML element
- **markup** in {{expression|filter}} is replaced by its evaluated value
- **model** are values in variables used in expressions
- a **filter** formats the value of an expression for display to the user

Two-Way Data Binding

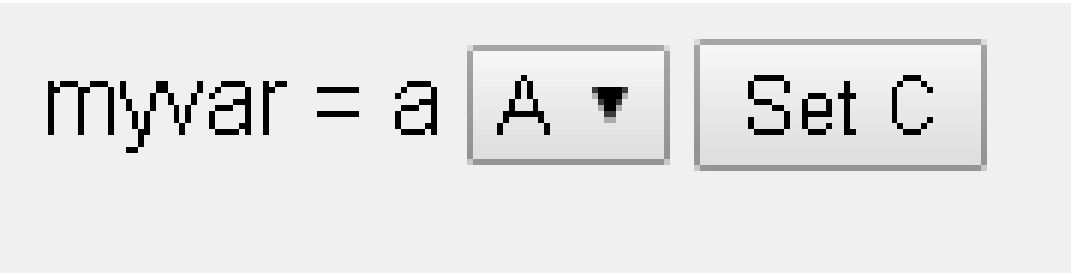
- automatic synchronization of data between the **model** and **view** components





Two-way binding in AngularJS

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular.min.js"></script>
</head>
<body>
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myCtrl', function ($scope) {
        $scope.myvar = 'a';
        $scope.setMyVar = function (x) { $scope.myvar = x; };
    });
</script>
<div ng-app="myApp" ng-controller="myCtrl">
    myvar = {{myvar}}
    <select ng-model="myvar">
        <option value="a">A</option>
        <option value="b">B</option>
        <option value="c">C</option>
    </select>
    <button ng-click="setMyVar('c')">Set C</button>
</div>
</body>
</html>
```



Dependency injection in AngularJS

- controller functions get parameters injected by their names

```
var myApp = angular.module('myApp', []);
myApp.controller('myCtrl',
        function ($scope, $rootScope, $http, $location) {
    // variable scoped to current controller
    $scope.myvar = 'a';
    // variable in root scope shared by all controllers
    $rootScope.myGlobalVar = 0;
    // AJAX service
    $http.get('/eshop/api/v1/product/5')
            .then(function (res) { $scope.p5name = res.data.name});
    // change URL location
    $location.path('/shopping');
});
```

URL routing in AngularJS

- AngularJS is intended for single-page apps
- links are not to complete URLs, but to URL fragments, e.g. **#/product/1**
- main HTML page has just an empty with **ng-view** attribute
- each view has
  - identifying URL fragment
  - a JavaScript controller
  - an HTML template in a separate file
- module ngRoute provides the routing

AngularJS routing example

```
<!DOCTYPE html>
<html>
<head>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular.min.js"></script>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular-route.min.js"></script>
</head>
<body>
<script>
    var myApp = angular.module('myApp', ['ngRoute']);
    myApp.config(['$routeProvider', function ($routeProvider) {
        $routeProvider.
        when('/url1', {templateUrl: 'view1.html', controller: 'myCtrl1'}).
        when('/url2', {templateUrl: 'view2.html', controller: 'myCtrl2'}).
        otherwise({redirectTo: '/url1'});
    }]);
    myApp.controller('myCtrl1', function ($scope) {  });
    myApp.controller('myCtrl2', function ($scope) {  });
</script>
<div ng-app="myApp">
    <div ng-view></div>
</div>
</body>
</html>
```

ng-repeat directive

- repeats the enclosing HTML element

```
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myCtrl', function ($scope) {
        $scope.fruits = ['apples', 'oranges', 'bananas'];
        $scope.peopleAges = {'Alice': 10, 'Bob': 15, 'Eve': 47};
        $scope.products = [
            {name: 'TV', price: 10000, inStock: true},
            {name: 'car', price: 200000, inStock: false},
            {name: 'lollipop', price: 10, inStock: true}
        ];
    });
</script>
<div ng-app="myApp" ng-controller="myCtrl">
    <div ng-repeat="f in fruits">
        {{$index+1}}. {{f}}
    </div>
    <div ng-repeat="(person,age) in peopleAges">
        {{person}} has age {{age}} years
    </div>
    <div ng-repeat="p in products | filter:{inStock:true} | orderBy:'-price'">
        {{p.name}} {{p.price}}
    </div>
</div>
```

ng-click directive

- evaluates an expression when clicked
- expressions are evaluated in current scope

```
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myCtrl', function ($scope) {
        $scope.a = 1;
        $scope.fruits = ['apples', 'oranges', 'bananas', 'kiwis', 'grapes'];
        $scope.deleteFruit = function(idx){ $scope.fruits.splice(idx,1); };
    });
</script>
<div ng-app="myApp" ng-controller="myCtrl">
    a={{a}} <button ng-click="a = a+1">increment a</button>
    <hr>
    <div ng-repeat="f in fruits">
        {{$index+1}}. {{f}} <button ng-click="deleteFruit($index)">Delete</button>
    </div>
</div>
```

Forms in AngularJS

```
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myCtrl', function ($scope) {
        $scope.n3 = '';
        $scope.doSomething = function () { console.log('n3=' + $scope.n3); };
    });
</script>
<div ng-app="myApp" ng-controller="myCtrl">

    <form name="f1" novalidate>
        <label for="n1">Name</label>
        <input id="n1" type="text" ng-minlength="3" name="n2" ng-model="n3" required/>

        <p ng-show="f1.n2.$error.required">name is required</p>
        <p ng-show="f1.n2.$error.minlength">name must be at least 3 characters</p>

        <button ng-show="!f1.$valid" type="submit" disabled>Submit</button>
        <button ng-show=" f1.$valid" type="submit" ng-click="doSomething()">Submit</button>
    </form>
</div>
```

AJAX in AngularJS

```
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myCtrl', function ($scope, $http) {
        var createCategoryDAO = {
            name: 'Electronics'
        };
        $http.post('/eshop/api/v1/categories/create', createCategoryDAO)
            .then(
             function success(response) { $scope.category = response.data; },
             function error(response) { $scope.problem = response.statusText; }
            );
    });
</script>
```

Own directives

```
<script>
    var myApp = angular.module('myApp', []);
    myApp.directive('myAlert', function () {
        return {
            restrict: 'E',
            transclude: true,
            scope: { type: '@type' },
            template: '<div class="alert alert-{{type}}" ng-transclude></div>'
        };
    });
</script>
<div ng-app="myApp" >

    <my-alert type="warning">
        <strong>Warning!</strong> Something is wrong !
    </my-alert>

</div>
```

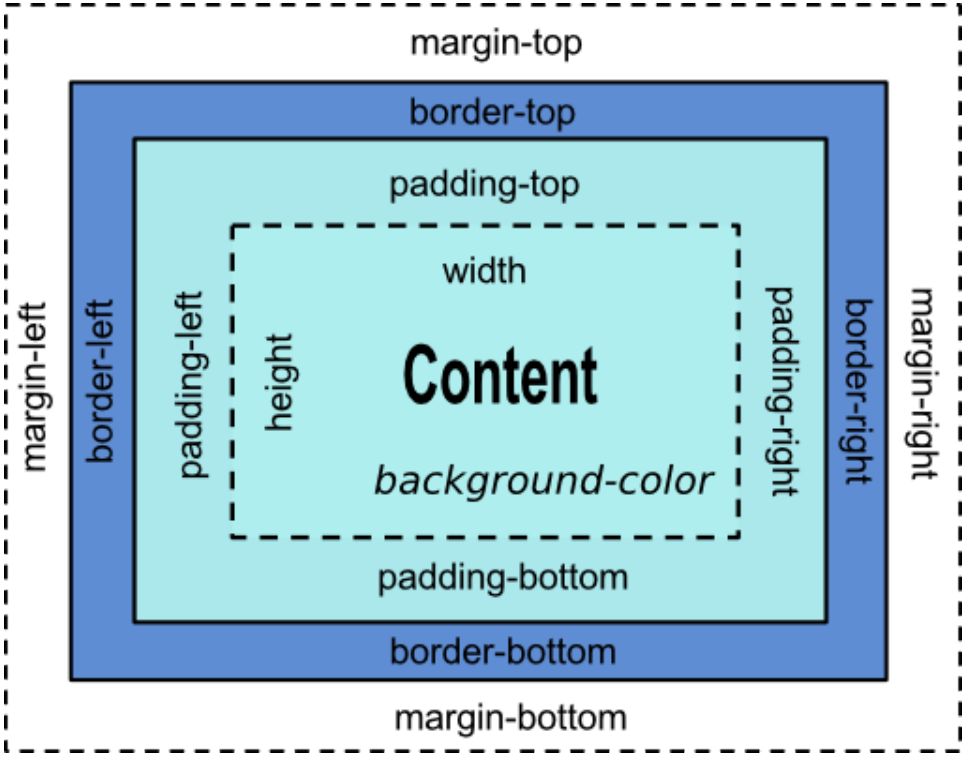
Summary of AngularJS

- AngularJS has
  - ability to extend HTML with custom directives used as elements, attributes, comments, CSS classes
  - two-way data binding between model and view
  - dependency injection
  - URL routing
  - AJAX service
  - directives (ng-app, ng-repeat, ng-show, …)
  - much more ...

Browser APIs

HTML parsing

- document begins with version declaration (`<! DOCTYPE html>`)
- browsers mostly ignore it and treat HTML as **tag soup**, but affects box model



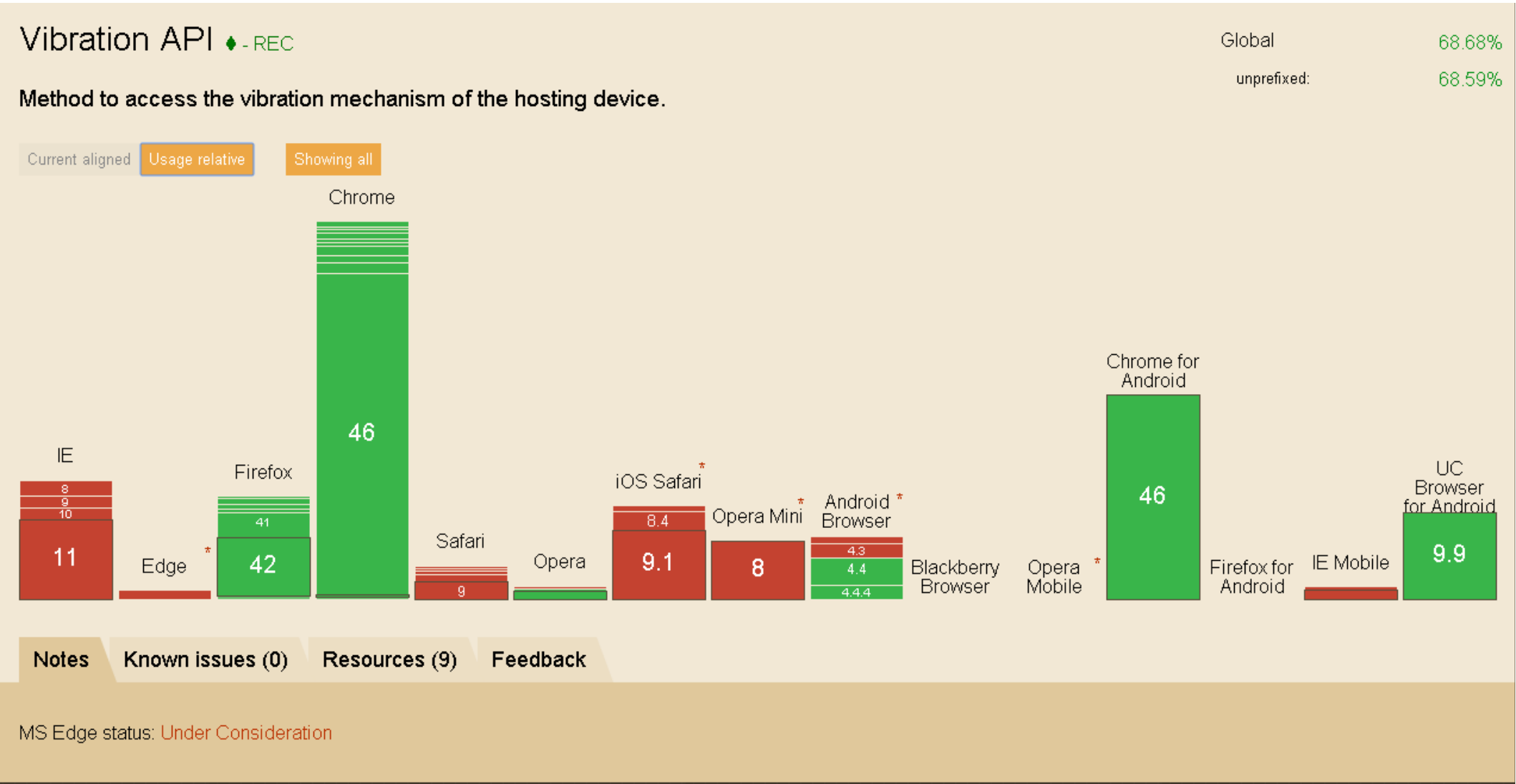
CSS selectors

- used in CSS rules, in jQuery, in standardised Selectors API
- `#x` selects the element with `id="x"`
- `.x` selects all elements with `class="x"`
- anchor `<a>` can have `a:link, a:hover, a:active, a:visited`
- multiple selectors in a rule separated by
  - **space** → descendants
  - **,** → multiple rules
  - **>** → direct child
  - **+** → siblings

JavaScript API in browser

- Document Object Model
- events (onload, onclick, onmousedown, onmouseover, onkeypress, onsubmit, …)
- many APIs described collectively as WebIDL (Interface Definition Language)
  - see http://www.w3.org/wiki/Web_IDL
  - XMLHttpRequest API, HTML5 Canvas, , Web storage API, File API, Indexed Database API, Progress Events API, Selectors API, Screen Orientation API, Device Orientation Event, Web workers API, Web Sockets API, Geolocation API, HTML Media Capture API, Vibration API, Battery Status API, WebRTC API, ...
  - always check usability on caniuse.com

e.g. Vibration API (W3C Recommendation)



Browser update policies

- Chrome updates automatically every 6 weeks
- Firefox updates automatically every 6 weeks
- Opera updates automatically every 6 weeks
- MSIE new version released together with a new Windows version
- Safari new version released with a new OS X

Usage share of all browsers for February 2015

Source	Chrome + Android	Internet Explorer	Firefox	Safari	Opera	Others
StatCounter	50.15%	13.75%	11.56%	13.8%	3.79%[††]	6.95%
Wikimedia	47.07%	11.06%	15.43%	20.47%	2.17%	3.8%[†]

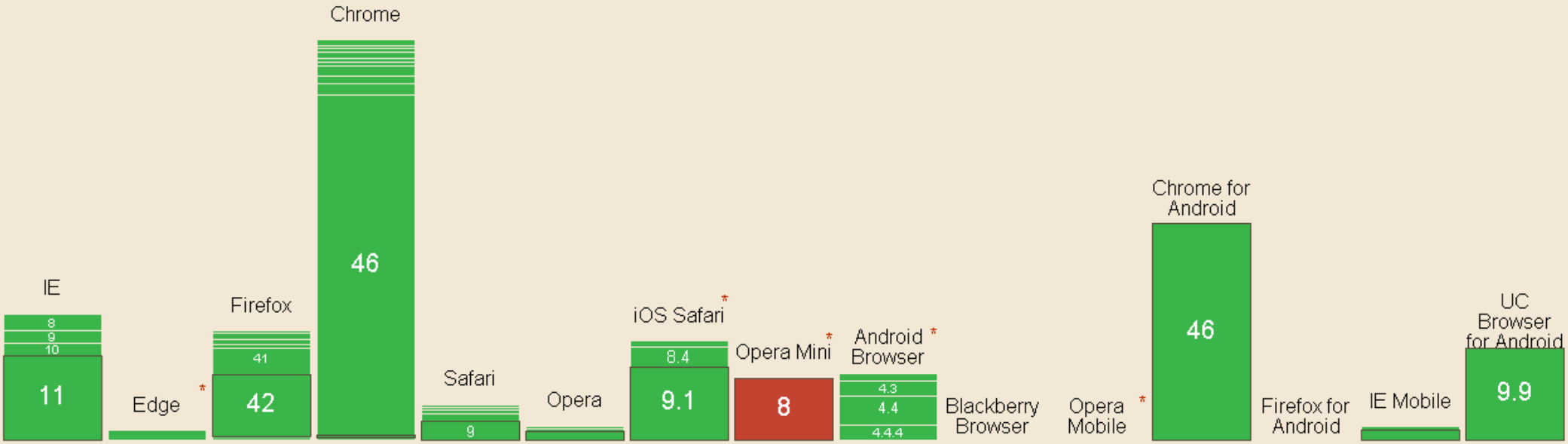
Web Storage API

Web Storage - name/value pairs - REC

Global 92.73% + 0.04% = 92.77%

Method of storing data locally like cookies, but for larger amounts of data (sessionStorage and localStorage, used to fall under HTML5).

Current aligned | Usage relative | Showing all



Notes | **Known issues (8)** | **Resources (8)** | **Feedback**

No notes

Web Storage API Example

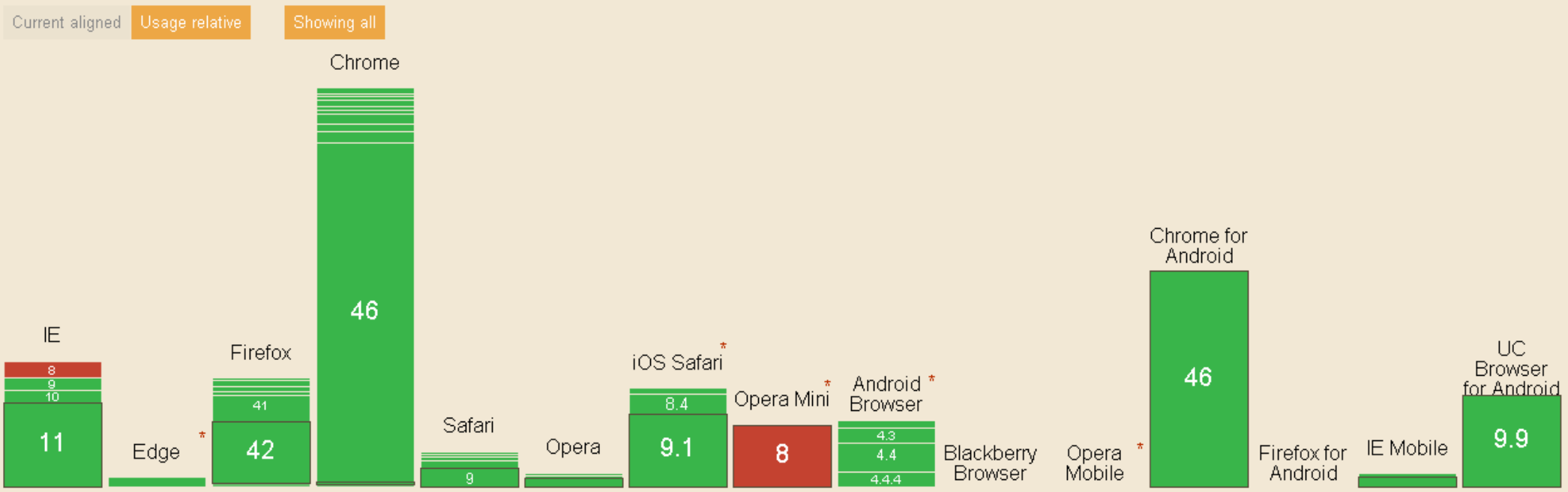
```
<script>
    if (typeof(Storage) !== "undefined") {

        localStorage.setItem('key1','value1');
        localStorage.setItem('key2','value2');

        for(var i=0;i<localStorage.length;i++) {
            var key = localStorage.key(i);
            var value = localStorage.getItem(key);
            console.log('i='+i+' '+ key+': '+value);
        }
    } else {
        console.log('Web Storage not supported in this browser');
    }
</script>
```

Geolocation API

Geolocation - REC

Global 91.47% + 0.02% = 91.49%

Method of informing a website of the user's geographical location



Notes | **Known issues (4)** | **Resources (6)** | **Feedback**

1. IE9 appears to have some issues in correctly determining longitude/latitude.
2. iOS6 has problems with returning high accuracy data.
3. Safari 5 & 6 seem to not provide geolocation data when using a wired connection.
4. Firefox sometimes fails to call the success or error callbacks see bug

GeoLocation API Example

```
<script>
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(
                function success(position) {
                    console.log(position.coords.longitude + ',' + position.coords.latitude);
                },
                function error(error) {
                    switch (error.code) {
                        case error.PERMISSION_DENIED:
                            console.log('User denied the request for Geolocation.');
                            break;
                        case error.POSITION_UNAVAILABLE:
                            console.log('Location information is unavailable.');
                            break;
                        case error.TIMEOUT:
                            console.log('The request to get user location timed out.');
                            break;
                        case error.UNKNOWN_ERROR:
                            console.log('An unknown error occurred.');
                            break;
                    }
                }
        );
    } else {
        console.log('GeoLocation not supported in this browser');
    }
</script>
```

Summary of browser APIs

- W3C releases some recommendations
- browser developers implement what they deem useful
- Chrome and Firefox are the most advanced
- MSIE is the most backward
- use what works for your users

Thank you for your attention