

互联网应用开发技术

Web Application Development

第3课

WEB前端-脚本与JAVASCRIPT

Episode Three

Scripting & JavaScript

陈昊鹏

chen-hp@sjtu.edu.cn



Web Application
Development

- Scripting
- JavaScript
 - Syntax overview
 - HTML DOM Objects
 - ECMAScript & JavaScript
 - JavaScript Compatibility Techniques
 - JavaScript Closure Compiler
- Rich Client JS Framework
 - Angular
 - React
 - jQuery
 - Bootstrap

- A script is program code that doesn't need pre-processing (e.g. compiling) before being run.
 - In the context of a Web browser, scripting usually refers to program code written in **JavaScript** that is executed by the **browser** when a page is downloaded, or in response to an event triggered by the user.
- Scripting can make Web pages more **dynamic**.
 - For example, without reloading a new version of a page it may allow modifications to the content of that page, or allow content to be added to or sent from that page.
 - The former has been called **DHTML** (Dynamic HTML), and the latter **AJAX** (Asynchronous JavaScript and XML).

- **Strings**

- **Strings** are values made up of *text* and can contain letters, numbers, symbols, punctuation, and **even emoji**.
- Strings are contained within a pair of either single quotation marks `' '` or double quotation marks `" "`.

```
'This is a string. 🙌';
```

```
"This is the 2nd string. 🙋";
```

- **Enclosing quotation marks**

```
"It's six o'clock.";
```

```
'Remember to say "please" and "thank you."';
```

```
'It\'s six o\'clock.';
```

```
"Remember to say \"please\" and \"thank you.\"";
```

from: <https://www.javascript.com/learn/javascript/strings>

- **Strings: Properties and methods**

- **length**

- EXAMPLE

- ```
"caterpillar".length;
```

- OUTPUT

- ```
11
```

- **toLowerCase**

- EXAMPLE

- ```
"THE KIDS".toLowerCase();
```

- OUTPUT

- ```
"the kids"
```

from: <https://www.javascript.com/learn/javascript/strings>

- **Strings: Properties and methods**

- **toUpperCase**

- EXAMPLE

- ```
"I wish I were big.".toUpperCase();
```

- OUTPUT

- ```
"I WISH I WERE BIG."
```

- **trim**

- EXAMPLE

- ```
" but keep the middle spaces ".trim();
```

- OUTPUT

- ```
"but keep the middle spaces"
```

from: <https://www.javascript.com/learn/javascript/strings>

- **Numbers**

- **Numbers** are values that can be used in mathematical operations.
- You don't need any special syntax for numbers — **just write them straight** into JavaScript.

12345

- **Decimals and fractions**

10 + 3.14159;

13.14159

1 / 3;

0.3333333333333333

1 + (4 / 3);

2.3333333333333333

from: <https://www.javascript.com/learn/javascript/numbers>

- **Booleans**

- **Booleans** are values that can be only one of two things: true or false.

- EXAMPLE

```
var kitchenLights = false;  
kitchenLights = true;  
kitchenLights;
```

- OUTPUT

True

from: <https://www.javascript.com/learn/javascript/booleans>

- **Operators**

- **Operators** are the symbols between values that allow different *operations* like addition, subtraction, multiplication, and more.
- **Arithmetic**
 - The **+** operator adds two numbers.
 - The **-** operator subtracts one number from another.
 - The ***** operator multiplies two numbers.
 - The **/** operator divides one number by another.
- **Grouping**
 - **()** operator groups other values and operations.
- **Concatenation**
 - The **+** operator can also concatenate strings, which is another way of saying it can add them together.
- **Assignment**
 - The **=** operator assigns values. It's used for setting the value of variables.

from: <https://www.javascript.com/learn/javascript/operators>

- **Variables**

- **Variables** are *named values* and can store any type of JavaScript value.

```
var x = 100;
```

- `var` is the *keyword* that tells JavaScript you're declaring a variable.
- `x` is the *name* of that variable.
- `=` is the **operator** that tells JavaScript a value is coming up next.
- `100` is the *value* for the variable to store

```
var x = 100;
```

```
x + 102;
```

```
var y = x + 102;
```

```
var weather = "rainy";
```

```
weather = "sunny";
```

```
weather;
```

from: <https://www.javascript.com/learn/javascript/variables>

- **Variables**

- **Naming variables**

- Start them with a letter, underscore `_`, or dollar sign `$`.
 - After the first letter, you can use numbers, as well as letters, underscores, or dollar signs.
 - Don't use any of JavaScript's **reserved** keywords.

- some *valid* variable names:

```
var camelCase = "lowercase word, then uppercase";  
var dinner2Go = "pizza";  
var I_AM_HUNGRY = true;  
var _Hello_ = "what a nice greeting"  
var $_$ = "money eyes";
```

- some *invalid* variable names :

```
var total% = 78;  
var 2fast2catch = "bold claim";  
var function = false;  
var class = "easy";
```

- Variable names are case-sensitive, so `myVar`, `MyVar`, and `myvar` are all different variables.

from: <https://www.javascript.com/learn/javascript/variables>

- Variables - let

```
<!DOCTYPE html>
<html>
<body>

<h1>使用 var 声明变量</h1>

<p id="demo"></p>

<script>
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

使用 var 声明变量

2

from: https://www.w3school.com.cn/tiy/t.asp?f=js_es6_var

- **Variables - let**

```
<!DOCTYPE html>
<html>
<body>

<h1>使用 let 声明变量</h1>

<p id="demo"></p>

<script>
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

使用 let 声明变量

10

from: https://www.w3school.com.cn/ty/t.asp?f=js_es6_let

- **Functions**

- **Functions** are blocks of code that can be **named** and **reused**.

```
function addTwoNumbers(x, y) {  
  return x + y;  
}
```

- `function` is the **keyword** that starts declaring a function.
- `addTwoNumbers` is the function's **name**, which is customizable — just like **variable** names.
- `(x, y)` are **parameters**, variable names for the inputs a function will accept.
- `return` is the **keyword** that exits the function and shares an optional value outside.

from: <https://www.javascript.com/learn/javascript/functions>

- **Using functions**

- Once a function is defined, you can use it by **referencing its name** with parentheses **()** right after.
- Note that a function *doesn't have to* have parameters.

```
function greetThePlanet() {  
    return "Hello world!";  
}  
greetThePlanet();
```

- If a function does have parameters, you'll need to provide values inside the parentheses when using the function.

```
function square(number) {  
    return number * number;  
}  
square(16);
```

from: <https://www.javascript.com/learn/javascript/functions>

- Function

```
var x = function (a, b) {return a * b};  
var z = x(4, 3);
```

- 箭头函数(lambda expression)

```
// ES5
```

```
var x = function(x, y) {  
  return x * y;  
}
```

```
// ES6
```

```
const x = (x, y) => x * y;
```

from: https://www.w3school.com.cn/js/js_function_definition.asp


```
var obj = {  
  birth: 1990,  
  getAge: function () {  
    var b = this.birth; // 1990  
    var fn = function () {  
      return new Date().getFullYear() - this.birth;  
      // this指向window或undefined, 而不是obj  
    };  
    return fn();  
  }  
};
```

from: <https://www.liaoxuefeng.com/wiki/1022910821149312/1031549578462080>

```
var obj = {  
  birth: 1990,  
  getAge: function () {  
    var b = this.birth; // 1990  
    var fn = () => new Date().getFullYear() - this.birth;  
    // this指向obj对象  
    return fn();  
  }  
};  
obj.getAge(); // 25
```

from: <https://www.liaoxuefeng.com/wiki/1022910821149312/1031549578462080>

- **Conditionals**

- **Conditionals** control *behavior* in JavaScript and determine whether or not pieces of code can run.

```
if (10 > 5) {  
    var outcome = "if block";  
}
```

```
if ("cat" === "dog") {  
    var outcome = "if block";  
} else {  
    var outcome = "else block";  
}
```

```
if (false) {  
    var outcome = "if block";  
} else if (true) {  
    var outcome = "first else if block";  
} else if (true) {  
    var outcome = "second else if block";  
} else {  
    var outcome = "else block";  
}
```

from: <https://www.javascript.com/learn/javascript/conditionals>

- **Conditionals**

- **Conditionals** control *behavior* in JavaScript and determine whether or not pieces of code can run.

```
var d=new Date().getDay();
switch (d)
{
  case 0: x="Today it's Sunday";    break;
  case 1: x="Today it's Monday";    break;
  case 2: x="Today it's Tuesday";   break;
  case 3: x="Today it's Wednesday"; break;
  case 4: x="Today it's Thursday";  break;
  case 5: x="Today it's Friday";     break;
  case 6: x="Today it's Saturday";  break;
  default: x="OMG!";
}
```

from: <https://www.javascript.com/learn/javascript/conditionals>

- **Loops**

- **for**

```
for (var i=0; i<5; i++) {  
  x=x + "The number is " + i + "<br>";  
}
```

- **for-in**

```
var person={fname:"John",lname:"Doe",age:25};  
for (x in person) {  
  txt=txt + person[x];  
}
```

from: http://www.w3school.com.cn/js/js_loop_for.asp

- **Loops**

- **while**

```
while (i<5) {  
    x=x + "The number is " + i + "<br>";  
    i++;  
}
```

- **do-while**

```
do {  
    x=x + "The number is " + i + "<br>"; i++;  
} while (i<5);
```

from: http://www.w3school.com.cn/js/js_loop_while.asp

- **Arrays**

- **Arrays** are container-like values that can hold other values. The values inside an array are called **elements**.

```
var breakfast = ["coffee", "croissant"];  
var hodgepodge = [100, "paint", [200, "brush"], false];
```

```
var sisters = ["Tia", "Tamera"];  
sisters[0];
```

```
var actors = ["Felicia", "Nathan", "Neil"];  
actors[actors.length - 1];
```

from: <https://www.javascript.com/learn/javascript/arrays>

- **Arrays: Properties and methods**

- **length**

- EXAMPLE

- ```
["a", "b", "c", 1, 2, 3].length;
```

- OUTPUT

- 6

- **concat**

- EXAMPLE

- ```
["tortilla chips"].concat(["salsa", "queso", "guacamole"]);"
```

- OUTPUT

- ```
["tortilla chips", "salsa", "queso", "guacamole"]
```

from: <https://www.javascript.com/learn/javascript/arrays>



- **Arrays: Properties and methods**

- **pop**

- EXAMPLE

- ```
["Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"].pop();
```

- OUTPUT

- ```
"Pluto"
```

- **push**

- EXAMPLE

- ```
["John", "Kate"].push(8);
```

- OUTPUT

- ```
3
```

- **reverse**

- EXAMPLE

- ```
["a", "b", "c"].reverse();
```

- OUTPUT

- ```
["c", "b", "a"]
```

from: <https://www.javascript.com/learn/javascript/arrays>

- **Objects**

- **Objects** are values that can contain other values. They use **keys** to name values, which are a lot like **variables**.

```
var course = {
 name: "GRA 2032",
 start: 8,
 end: 10
};
```

```
course.name;
course["name"];
```

```
course.name = "SE 228";
course["name"] = "SE 228";
```

from: <https://www.javascript.com/learn/javascript/objects>

- This specification uses the term **document** to refer to any use of HTML,
  - ranging from short static documents to long essays or reports with rich multimedia, as well as to fully-fledged interactive applications.
  - The term is used to refer both to **Document** objects and their descendant **DOM trees**.
- The **root element** of a Document object is that Document's first element child, if any.
  - If it does not have one then the Document has no root element.
  - A node's **home subtree** is the **subtree** rooted at that node's root element. When a node is in a **Document**, its home subtree is that Document's tree.

- Every XML and HTML document in an HTML user agent is represented by a **Document** object.
  - Some DOM tree accessors
    - *document.head*
      - Returns the **<head>** element.
    - *document.title [ = value ]*
      - Returns the document's title, as given by the **<title>** element for HTML
    - *document.body [ = value ]*
      - Returns the **<body>** element.
    - *collection = document.getElementsByTagName(name)*
      - Returns a **NodeList** of elements in the **Document** that have a name attribute with the value *name*.
    - *node = document.getElementById(id)*
      - Returns a **Node** in the **Document** that have a id attribute with the value *id*.

- **Elements** in the DOM **represent** things
  - that is, they have intrinsic *meaning*, also known as semantics.

```
<html>
```

```
<head>
```

```
<script type="text/javascript">
```

```
function getValue() {
```

```
 var x=document.getElementById("myHeader")
```

```
 alert(x.innerHTML)
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<h1 id="myHeader" onclick="getValue()">This is a header</h1>
```

```
<p>Click on the header to alert its value</p>
```

```
</body>
```

```
</html>
```

- **Elements** in the DOM **represent** things
  - that is, they have intrinsic *meaning*, also known as semantics.

```
<html>
<body>
 <ul id="myList">CoffeeTea
 <p id="demo">Insert a new Item</p>
 <button onclick="myFunction()">试一下</button>
 <script>
 function myFunction(){
 var newItem=document.createElement("LI")
 var textnode=document.createTextNode("Water")
 newItem.appendChild(textnode)
 var list=document.getElementById("myList")
 list.insertBefore(newItem,list.childNodes[0]);
 }
 </script>
</body>
</html>
```

- Except where otherwise specified, **attributes** on html elements may have any string value, including the empty string.

```
<html>
<body>
 <h1 style="color:red;">Hello World</h1>
 <p id="demo">Change style attribute</p>
 <button onclick="myFunction()">Try it</button>
 <script>
 function myFunction(){
 var h=document.getElementsByTagName("H1")[0];
 h.getAttributeNode("style").value="color:green";
 }
 </script>
</body>
</html>
```

- The semantics of a document are therefore based on the document's state at a particular instance in time, but may also change in response to external **events**.

```
<html>
<head>
 <script type="text/javascript">
 function show_coords(event){
 let x=event.clientX
 let y=event.clientY
 alert("X: " + x + ", Y: " + y)
 }
 </script>
</head>
<body onmousedown="show_coords(event)">
 <button>Click the button and a message box will be popped up</button>
</body>
</html>
```



- ECMAScript was originally designed to be a Web scripting language
  - providing a mechanism to enliven Web pages in browsers and to perform server computation as part of a Web-based client-server architecture.
  - ECMAScript is now used to provide **core scripting capabilities for a variety of host environments**
- In November 1996
  - **Netscape** submitted JavaScript to Ecma International to carve out a standard specification, which other browser vendors could then implement based on the work done at Netscape.
  - This led to the official release of the language specification ECMAScript published in the first edition of the ECMA-262 standard in June 1997, with **JavaScript** being the most well known of the implementations.
  - **ActionScript** and **JScript** are other well-known implementations of ECMAScript, with extensions.

- JavaScript
  - like Java, is one of a new breed of **platform-independent languages**.
  - That is, you can develop a program in JavaScript and expect to run it unchanged in a JavaScript-enabled web browser running on any type of computer with any type of operating system.
- There are,
  - and probably always will be, **compatibility problems** that JavaScript programmers must bear in mind.
  - The one fact that we must always remember is that it is a **heterogeneous network** out there.
    - Your JavaScript programs may run on three or more operating systems, using three or more versions of browsers from at least two different vendors.
- The compatibility issues fall into two broad categories:
  - platform-specific, browser-specific, and version-specific;
  - and bugs and language-level incompatibilities, including the incompatibility of JavaScript with non-JavaScript browsers.

- Knowledge of existing incompatibilities is crucial to writing compatible code.
  - Unfortunately, producing a definitive listing of all known vendor, version, and platform incompatibilities would be an enormous task.
- For example
  - `document.formName.item("itemName")`
  - IE:
    - `document.formName.item("itemName")`
    - `document.formName.elements ["elementName"]`
  - Firefox :
    - `document.formName.elements["elementName"]`
  - Solution: **The Least-Common-Denominator Approach**
    - `document.formName.elements["elementName"]`

- For example
  - the `split()` method of the String object exists only for JavaScript 1.1 implementations
  - Solution: **Feature Testing**

```
if (s.split) // Check if the method exists, without invoking it
 a = s.split(":"); // If it does exist, it is safe to invoke it
else // Otherwise:
 a = mysplit(s, ":"); // use our alternative implementation
```

- For example
  - `innerText` vs. `textContent`
  - IE:
    - `innerText`
  - Firefox :
    - `textContent`
  - Solution: **Platform-Specific Workarounds**

```
if (navigator.appName.indexOf("Explorer") > -1) {
 document.getElementById('element').innerText = "my text";
} else {
 document.getElementById('element').textContent = "my text";
}
```

- Other Solutions:
  - **Defensive Coding**
    - You write code that contains platform-independent workarounds for platform-specific incompatibilities.
  - **Compatibility Through Server-Side Scripts**
    - A program on the server side can generate customized JavaScript code that is known to work correctly on that browser.
  - **Ignore the Problem**
    - If the incompatibility is minor or cosmetic, you might simply decide to ignore the problem and let the users affected by it cope with it on their own.
  - **Fail Gracefully**
    - Failing gracefully means recognizing that the required features are not available and informing the user that he will not be able to use your JavaScript program.

- Question:
  - How to use new features of the JavaScript language in a way that **does not cause errors** on browsers that **do not support those features**.
- Our goals are simple:
  - We need to prevent JavaScript code from being interpreted by browsers that don't understand it, and we need to display **special messages** on those browsers that inform users that their browsers cannot run the scripts.
- Solutions:
  - The language Attribute

```
<script language="JavaScript1.1">
 // JavaScript 1.1 code goes here
</script>
```

- Solutions:
  - Explicit Version Testing

```
<!-- Set a variable to determine what version of JavaScript we support -->
<!-- This technique can be extended to any number of language versions -->
<script language="JavaScript"> var _version = 1.0; </script>
<script language="JavaScript1.1"> _version = 1.1; </script>
<script language="JavaScript1.2"> _version = 1.2; </script>

<!-- Run this code on any JavaScript-enabled browser -->
<!-- If the version is not high enough, display a message -->
<script language="JavaScript">
 if (_version < 1.1) {
 document.write('<hr><h1>This Page Requires JavaScript 1.1</h1>');
 document.write('Your JavaScript 1.0 browser cannot run this page.<hr>');
 }
</script>

<!-- Now run the actual program only on JavaScript 1.1 browsers -->
<script language="JavaScript1.1">
 // The actual JavaScript 1.1 code goes here
</script>
```



- Solutions:
  - Suppressing Version-Related Errors

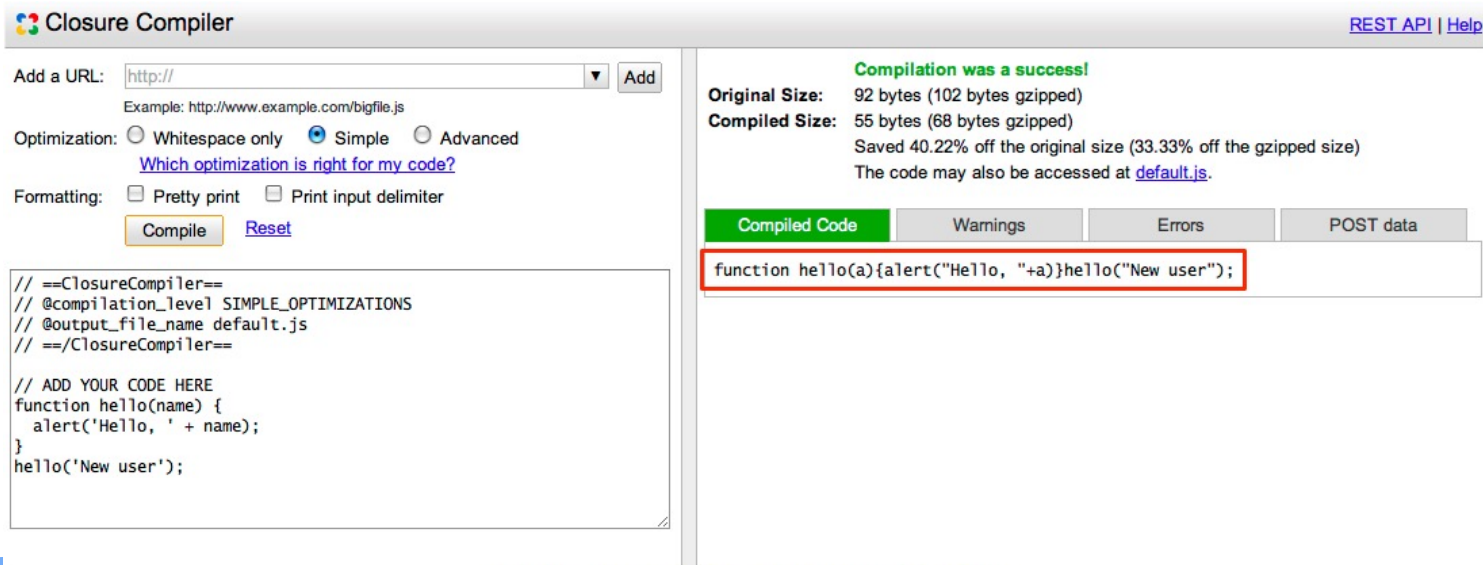
```
<!-- Check whether JavaScript 1.2 is supported -->
<script language="JavaScript1.2">var _js12_ = 1.2</script>
<!-- Now avoid the problems with JavaScript 1.2 on Netscape by running -->
<!-- the following code on any browser that supports JavaScript 1.1. If -->
<!-- the browser does not support JavaScript 1.2, however, we'll display -->
<!-- an error message and suppress any syntax errors that occur. -->
<script language="JavaScript1.1">
 // If JavaScript 1.2 is not supported, fail gracefully
 function supressErrors() { return true; }
 if (!_js12_) {
 window.onerror = supressErrors;
 alert("This program requires a browser with JavaScript 1.2 support");
 }
 // Now proceed with the JavaScript 1.2 code
</script>
```

- Solutions:
  - Loading a New Page for Compatibility

```
<head>
<script language="JavaScript1.2">
 // If JavaScript 1.2 is supported, extract a new URL from the portion of
 // our URL following the question mark, and load in that new URL
 location.replace(location.search.substring(1));

 // Enter a really long, empty loop, so that the body of this document
 // doesn't get displayed while the new document is loading
 for(var i = 0; i < 10000000; i++);
</script>
</head>
<body>
<hr size="4">
<h1>This Page Requires JavaScript 1.2</h1>
Your browser cannot run this page.
Please upgrade to a browser that supports JavaScript 1.2,
such as Netscape 4 or Internet Explorer 4.
<hr size="4">
</body>
```

- The Closure Compiler is a tool for making JavaScript download and run **faster**.
  - Instead of compiling from a source language to machine code, it compiles from JavaScript to better JavaScript.
  - It parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left.
  - It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.



The screenshot shows the Closure Compiler web interface. On the left, there's a form with fields for 'Add a URL' (containing 'http://'), 'Optimization' (set to 'Simple'), and 'Formatting' (set to 'Pretty print'). Below these are 'Compile' and 'Reset' buttons. The main text area contains a JavaScript snippet: 

```
// ==ClosureCompiler==
// @compilation_level SIMPLE_OPTIMIZATIONS
// @output_file_name default.js
// ==/ClosureCompiler==

// ADD YOUR CODE HERE
function hello(name) {
 alert('Hello, ' + name);
}
hello('New user');
```

 On the right, the results section shows a green message 'Compilation was a success!'. Below this, it lists 'Original Size: 92 bytes (102 bytes gzipped)' and 'Compiled Size: 55 bytes (68 bytes gzipped)', noting a 40.22% reduction in size. There are tabs for 'Compiled Code', 'Warnings', 'Errors', and 'POST data'. The 'Compiled Code' tab is active, showing the minimized code: 

```
function hello(a){alert("Hello, "+a)}hello("New user");
```

 This line of code is highlighted with a red border.

REST API | Help

Add a URL:  Add

Example: <http://www.example.com/bigfile.js>

Optimization: ☐ Whitespace only ☒ Simple ☐ Advanced  
[Which optimization is right for my code?](#)

Formatting: ☐ Pretty print ☐ Print input delimiter

[Reset](#)

```
// ==ClosureCompiler==
// @compilation_level SIMPLE_OPTIMIZATIONS
// @output_file_name default.js
// ==/ClosureCompiler==

// ADD YOUR CODE HERE
function hello(name) {
 alert('Hello, ' + name);
}
hello('New user');
```

**Compilation was a success!**

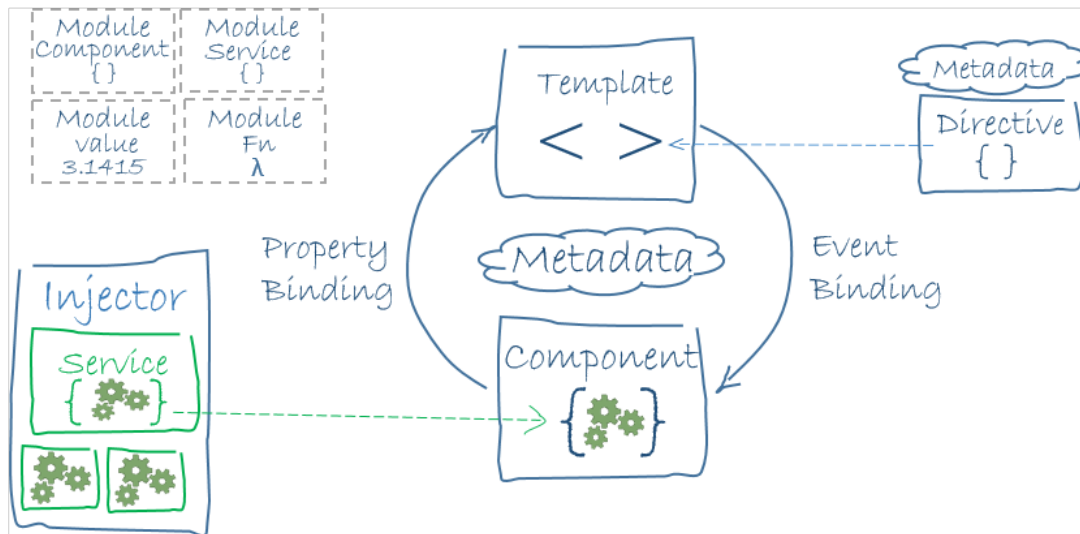
Original Size: 92 bytes (102 bytes gzipped)  
Compiled Size: 55 bytes (68 bytes gzipped)  
Saved 40.22% off the original size (33.33% off the gzipped size)  
The code may also be accessed at [default.js](#).

**Compiled Code** Warnings Errors POST data

```
function hello(a){alert("Hello, "+a)}hello("New user");
```

- Angular

- a framework for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript.



- from: <https://angular.io/docs/ts/latest/guide/architecture.html>

- jQuery

- is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.



### Lightweight Footprint

Only 32kB minified and gzipped. Can also be included as an AMD module



### CSS3 Compliant

Supports CSS3 selectors to find elements as well as in style property manipulation



### Cross-Browser

Chrome, Edge, Firefox, IE, Safari, Android, iOS, and more

- Bootstrap

- is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.



## Preprocessors

Bootstrap ships with vanilla CSS, but its source code utilizes the two most popular CSS preprocessors, [Less](#) and [Sass](#). Quickly get started with precompiled CSS or build on the source.



## One framework, every device.

Bootstrap easily and efficiently scales your websites and applications with a single code base, from phones to tablets to desktops with CSS media queries.



## Full of features

With Bootstrap, you get extensive and beautiful documentation for common HTML elements, dozens of custom HTML and CSS components, and awesome jQuery plugins.

- React

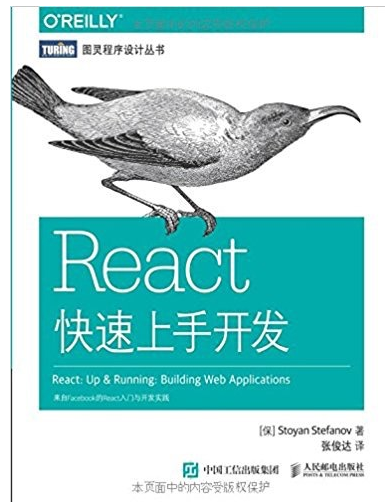
- is an open-source JavaScript library providing a views that is rendered using components specified as custom HTML tags.
- React components implement a `render()` method that takes input data and returns what to display.
- **JSX**, an XML-like syntax, is optional and not required to use React.

JavaScript:

```
class HelloMessage extends React.Component {
 render() {
 return React.createElement("div", null,
 "Hello ", this.props.name);
 }
}

ReactDOM.render(React.createElement(HelloMessage,
 { name: "Jane" }), mountNode);
```

- Download
  - <https://github.com/facebook/react/>
- Reference
  - React快速上手开发
  - <https://github.com/stoyan/reactbook>
  - React Quick Start
  - <https://reactjs.org/docs/hello-world.html>





- HTML 5.1
  - <https://www.w3.org/TR/2016/REC-html51-20161101/dom.html>
- JavaScript
  - <https://www.w3.org/standards/webdesign/script.html>
  - <http://www.w3school.com.cn/jsref>
  - <https://en.wikipedia.org/wiki/JavaScript>
  - <https://www.javascript.com>
  - <https://www.codeschool.com/learn/javascript>
- JavaScript Compatibility Techniques
  - [http://docstore.mik.ua/oreilly/webprog/jsript/ch20\\_01.htm](http://docstore.mik.ua/oreilly/webprog/jsript/ch20_01.htm)
- Javascript 多浏览器兼容性问题及解决方案
  - <http://www.jb51.net/article/21483.htm>
- ES2015 [ES6] cheatsheet
  - <https://github.com/DrkSephy/es6-cheatsheet>
- JavaScript Closure Compiler
  - <https://github.com/google/closure-compiler-js>

- Angular
  - <https://angularjs.org/>
- React
  - <https://facebook.github.io/react/>
- jQuery
  - <https://jquery.com/>
- BootStrap
  - <http://getbootstrap.com/>



- *Web*开发技术
- *Web Application Development*

Thank You!