

JavaScript

Table of contents

- [HTML vs JSP](#)
- [XML vs HTML](#)
- [CSS](#)
- [HTML links](#)
- [HTML table tags](#)
- [Block-level Elements and inline containers](#)
- [Class vs ID](#)
- [HTML data-* Attribute](#)
- [List](#)
- [Link vs Script](#)
- [Input](#)
- [Form](#)
- [Script attribute](#)
- [All CSS Simple Selectors](#)
- [Position](#)
- [clear property](#)
- [box-shadow](#)
- [z-index](#)
- [Website layout](#)
- [Bootstrap](#)
- [Focus in input](#)
- [Introduction](#)
- [DOM](#)
- [jQuery vs Javascript](#)
- [Javascript variables](#)
- [Javascript arithmetic](#)
- [Javascript function](#)
- [Javascript array](#)
- [Return value from js function](#)
- [File loading](#)
- [Code snippets](#)
- [Regex for forms](#)
- [Bind](#)
- [Toggle vs show](#)
- [jQuery Method Chaining](#)**
- [jQuery Callback Functions](#)**
- [Right Coding practices](#)
- [jQuery Get or Set Contents and Values](#)**
- [Toggle checkbox](#)
- [.click vs on\("click"\)](#)
- [Append & prepend. After & before](#)
- [Wrap](#)
- [empty, remove, unwrap & removeAttr](#)
- [DOM vs jQuery objects](#)
- [this vs \\$\(this\)](#)

this vs [event.target](#) vs event.currentTarget

[Traversing](#)

[each vs filter vs not](#)

[Event bubbling](#)

[toggleclass\(class1\), removeclass\(\), addclass\(class1 class2\)](#)

[css](#)

[Dimension](#)

[Closure in javascript](#)

[Filtering](#)

[Has](#)

[Slice](#)

[What happens if you submit a form?](#)

[Struts](#)

[Ajax](#)

[Async await](#)

[Fetch then](#)

[JQuery callbacks](#)

[Model driven](#)

[Async and deferred in script](#)

[Extra Topics](#)

HTML

▼ HTML vs JSP

The main difference between JSP and HTML is that **JSP is a technology to create dynamic web applications while HTML is a standard markup language to create the structure of web pages**. In brief, JSP file is an HTML file with Java code.

J S P	V E R S U S	H T M L
JSP		HTML
A technology that helps software developers to create dynamically generated web pages based on HTML		A standard markup language for creating web pages and web applications
Stands for Java Server Pages		Stands for Hyper Text Markup Language
Helps to develop dynamic web applications		Helps to create the structure of web pages
		Visit www.PEDIAA.com

▼ XML vs HTML

How XML and HTML compare	
XML	HTML
Used for storing data as structured information	Used for representing content
Strict validation	Loose validation
Defines encoded data	Defines display formatting
No predefined tags or semantics	Predefined tags and semantics
Tags are case-sensitive	Tags are not case-sensitive
New tags/elements can be added by the user	New tags/elements cannot be added by the user

▼ CSS

Using CSS

CSS can be added to HTML documents in 3 ways:

- **Inline** - by using the `style` attribute inside HTML elements
- **Internal** - by using a `<style>` element in the `<head>` section
- **External** - by using a `<link>` element to link to an external CSS file

```
<link rel="stylesheet" href="myStyle.css" type="text/css" >
```

The type attribute **specifies the media type of the linked document/resource**. The most common value of type is "text/css". If you omit the type attribute, the browser will look at the rel attribute to guess the correct type. So, if rel="stylesheet", the browser will assume the type is "text/css".

▼ HTML links

The target Attribute

By default, the linked page will be displayed in the **current browser window**. To change this, you must specify another target for the link.

The target attribute specifies where to open the linked document.

The `target` attribute can have one of the following values:

- `_self` - Default. Opens the document in the same window/tab as it was clicked
- `_blank` - Opens the document in a new window or tab

- `_parent` - Opens the document in the parent frame
- `_top` - Opens the document in the full body of the window

▼ HTML table tags

Tag	Description
<code><table></code>	Defines a table
<code><th></code>	Defines a header cell in a table
<code><tr></code>	Defines a row in a table
<code><td></code>	Defines a cell in a table
<code><caption></code>	Defines a table caption
<code><colgroup></code>	Specifies a group of one or more columns in a table for formatting
<code><col></code>	Specifies column properties for each column within a <code><colgroup></code> element
<code><thead></code>	Groups the header content in a table
<code><tbody></code>	Groups the body content in a table
<code><tfoot></code>	Groups the footer content in a table

▼ Block-level Elements and inline containers

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: `<p>` and `<div>`.

The `<p>` element defines a paragraph in an HTML document.

The `<div>` element defines a division or a section in an HTML document.

The `<div>` Element

The `<div>` element is often used as a container for other HTML elements.

The `<div>` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `<div>` element can be used to style blocks of content:

Example

```
<div style="background-color:black;color:white;padding:20px;"> <h2>London</h2> <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p></div>
```

[Try it Yourself »](#)

The Element

The `` element is an inline container used to mark up a part of a text, or a part of a document.

The `` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `` element can be used to style parts of the text:

Example

```
<p>My mother has <span style="color:blue;font-weight:bold">blue</span> eyes and my father  
has <span style="color:darkolivegreen;font-weight:bold">dark green</span> eyes.</p>
```

[Try it Yourself »](#)

▼ Class vs ID

Difference Between Class and ID

A class name can be used by multiple HTML elements, while an id name must only be used by one HTML element within the page.

Can all HTML elements have ID?

In HTML5, the **id attribute can be used on any HTML element** (it will validate on any HTML element).

The class attribute can be used on any HTML element

The class name can be used by CSS and JavaScript to perform certain tasks for elements with the specified class name.

Using multiple classes:

HTML elements can have more than one class name, where each class name must be separated by a space.

```
< body >  
  
  < h2 class = "country middle" >CHINA</ h2 >  
  
  < h2 class = "country" >INDIA</ h2 >  
  
  < h2 class = "country" >UNITED STATES</ h2 >  
  
</ body >
```

▼ HTML data-* Attribute

Definition and Usage

The `data-*` attribute is used to store custom data **private to the page or application**.

The `data-*` attribute gives us the ability to embed custom data attributes on **all HTML elements**.

The stored (custom) data can then be used in the page's JavaScript to create a more engaging user experience (without any Ajax calls or server-side database queries).

The `data-*` attribute consist of two parts:


1. The attribute name should not contain any uppercase letters, and must be at least one character long after the prefix "data-"
2. The attribute value can be any string

Note: Custom attributes prefixed with "data-" will be completely ignored by the user agent.

▼ List

HTML Lists

HTML lists allow web developers to group a set of related items in lists. Try it Yourself " An unordered list starts with the tag. Each list item starts with the tag. The list items will be marked with bullets (small black circles) by default: An ordered list starts with the tag.

 https://www.w3schools.com/html/html_lists.asp



▼ Link vs Script

<link>: The External Resource Link element

The `<link>` HTML element specifies relationships between the current document and an external resource. This element is most commonly used to link to stylesheets, but is also used to establish site icons (both "favicon" style icons and icons for the home screen and apps on mobile devices) among other things.

Eg. `<link rel="stylesheet" href="main.css">`

```
<link rel="icon" href="favicon.ico">
```

<script>: The Script element

The `<script>` HTML element is used to embed executable code or data; this is typically used to embed or refer to JavaScript code. The `<script>` element can also be used with other languages, such as WebGL's GLSL shader programming language and JSON.

▼ Input

```
<label for="username">Username: </label>
<input type="text" name="Username" id="username" />
```

The `name` attribute specifies the name of an `<input>` element.

The `name` attribute is used to reference elements in a JavaScript, or to reference form data after a form is submitted.

Note: Only form elements with a `name` attribute will have their values passed when submitting a form.

▼ Form

The `method` attribute specifies how to send form-data (the form-data is sent to the page specified in the `action` attribute).

The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).

Notes on GET:

- Appends form-data into the URL in name/value pairs
- The length of a URL is limited (about 3000 characters)
- Never use GET to send sensitive data! (will be visible in the URL)
- Useful for form submissions where a user wants to bookmark the result
- GET is better for non-secure data, like query strings in Google

Notes on POST:

- Appends form-data inside the body of the HTTP request (data is not shown in URL)
- Has no size limitations
- Form submissions with POST cannot be bookmarked

CSS

https://www.youtube.com/watch?v=3_9znKVNe5g

▼ Script attribute

```
<button id="myButton" onclick="displayResult()">Click here</button>

<ol>
  <li onclick="myMethod(this)" data-type="car">Hyundai</li>
  <li onclick="myMethod(this)" data-type="bike">Honda</li>
  <li onclick="myMethod(this)" data-type="machinary">Caterpillar</li>
</ol>

<script>
  function displayResult(){
    //alert(document.getElementById("message")); //results -> [object HTMLHeadingElement]
    document.getElementById("message").innerHTML = "Have a nice day!";
  }

  function myMethod(object){
    //object -> [object HTMLLIElement]
    //object.innerHTML -> Hyundai
    //object.getAttribute("data-type") -> car
    alert(object.innerHTML + " is a " + object.getAttribute("data-type"));
  }
</script>
```

▼ All CSS Simple Selectors

Selector	Example	Example description
<u>element</u>	p	Selects all <p> elements
<u>#id</u>	#firstname	Selects the element with id="firstname"
<u>.class</u>	.intro	Selects all elements with class="intro"
<u>*</u>	*	Selects all elements
element#id	span#author	Selects author id present in the span tag.
<u>element.class</u>	p.intro	Selects only <p> elements with class="intro"
<u>element,element,...</u>	div, p	Selects all <div> elements and all <p> elements

Jquery will not return the HTMLElement, it returns a jQuery object.

A **jQuery object contains a collection of Document Object Model (DOM) elements** that have been created from an HTML string or selected from a document. Since, jQuery methods often use CSS selectors to match elements from a document, the set of elements in a jQuery object is often called a set of "matched elements" or "selected elements".

The jQuery object itself behaves much like an array; it has a length property and the elements in the object can be accessed by their numeric indices [0] to [length-1]. Note that a jQuery object is not actually a Javascript Array object, so it does not have all the methods of a true Array object such as `join()`. <http://api.jquery.com/Types/#jQuery>.

```
<script>
$(document).ready(function(){
    console.log($("#ul li").toString());
    console.log($("#ul li"));
    console.log($("#ul li")[0].toString());
    console.log($("#ul li")[0]);
    console.log(typeof $("#ul li")[0].innerHTML);
});
</script>

//where html body is
<body>
    <ul>
        <li>aadi</li>
        <li>b</li>
        <li>
            <ul>
                <li>c</li>
                <li>d</li>
            </ul>
        </li>
        <li>e</li>
    </ul>
</body>

//OUTPUT
[object Object]
//OUTPUT
jQuery.fn.init(5) [li, li, li, li, li, prevObject: jQuery.fn.init(1)]
0: li
1: li
2: li
3: li
```



```

4: li
length: 5
prevObject: jQuery.fn.init [document]
[[Prototype]]: Object(0)
/*Reason last two lines were printed when you use a selector you are creating
an instance of the jquery function; when found an element based on the
selector criteria it returns the matched elements; when the criteria does
not match anything it returns the prototype object of the function.*/

//OUTPUT
[object HTMLLIElement]
//OUTPUT
<li>
  :marker
  "aadi"
</li>
//OUTPUT
aadi

```

In CSS, selectors are patterns used to select the element(s) you want to style. Following is the complete list:

Selector	Example	Example description
<u>.class</u>	.intro	Selects all elements with class="intro"
.class1.class2	.name1.name2	Selects all elements with both <i>name1</i> and <i>name2</i> set within its class attribute
.class1 .class2	.name1 .name2	Selects all elements with <i>name2</i> that is a descendant of an element with <i>name1</i>
<u>#id</u>	#firstname	Selects the element with id="firstname"
<u>*</u>	*	Selects all elements
<u>element</u>	p	Selects all <p> elements
<u>element.class</u>	p.intro	Selects all <p> elements with class="intro"
<u>element.element</u>	div, p	Selects all <div> elements and all <p> elements
<u>element element</u>	div p	Selects all <p> elements inside <div> elements
<u>element>element</u>	div > p	Selects all <p> elements where the parent is a <div> element
<u>element+element</u>	div + p	Selects the first <p> element that is placed immediately after <div> elements
<u>element1~element2</u>	p ~ ul	Selects every element that is preceded by a <p> element
<u>[attribute]</u>	[target]	Selects all elements with a target attribute
<u>[attribute=value]</u>	[target=_blank]	Selects all elements with target="_blank"
<u>[attribute~=value]</u>	[title~=flower]	Selects all elements with a title attribute containing the word "flower"
<u>[attribute]=value]</u>	[lang=en]	Selects all elements with a lang attribute value equal to "en" or starting with "en-"
<u>[attribute^=value]</u>	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"
<u>[attribute\$=value]</u>	a[href\$=".pdf"]	Selects every <a> element whose href attribute value ends with ".pdf"
<u>[attribute*=value]</u>	a[href*="w3schools"]	Selects every <a> element whose href attribute value contains the substring "w3schools"
<u>:active</u>	a:active	Selects the active link
<u>::after</u>	p::after	Insert something after the content of each <p> element

Selector	Example	Example description
<u>::before</u>	p::before	Insert something before the content of each <p> element
<u>:checked</u>	input:checked	Selects every checked <input> element
<u>:default</u>	input:default	Selects the default <input> element
<u>:disabled</u>	input:disabled	Selects every disabled <input> element
<u>:empty</u>	p:empty	Selects every <p> element that has no children (including text nodes)
<u>:enabled</u>	input:enabled	Selects every enabled <input> element
<u>:first-child</u>	p:first-child	Selects every <p> element that is the first child of its parent
<u>::first-letter</u>	p::first-letter	Selects the first letter of every <p> element
<u>::first-line</u>	p::first-line	Selects the first line of every <p> element
<u>:first-of-type</u>	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
<u>:focus</u>	input:focus	Selects the input element which has focus
<u>:fullscreen</u>	:fullscreen	Selects the element that is in full-screen mode
<u>:hover</u>	a:hover	Selects links on mouse over
<u>:in-range</u>	input:in-range	Selects input elements with a value within a specified range
<u>:indeterminate</u>	input:indeterminate	Selects input elements that are in an indeterminate state
<u>:invalid</u>	input:invalid	Selects all input elements with an invalid value
<u>:lang(<i>language</i>)</u>	p:lang(it)	Selects every <p> element with a lang attribute equal to "it" (Italian)
<u>:last-child</u>	p:last-child	Selects every <p> element that is the last child of its parent
<u>:last-of-type</u>	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
<u>:link</u>	a:link	Selects all unvisited links
<u>::marker</u>	::marker	Selects the markers of list items
<u>:not(<i>selector</i>)</u>	:not(p)	Selects every element that is not a <p> element
<u>:nth-child(<i>n</i>)</u>	p:nth-child(2)	Selects every <p> element that is the second child of its parent
<u>:nth-last-child(<i>n</i>)</u>	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
<u>:nth-last-of-type(<i>n</i>)</u>	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
<u>:nth-of-type(<i>n</i>)</u>	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
<u>:only-of-type</u>	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
<u>:only-child</u>	p:only-child	Selects every <p> element that is the only child of its parent
<u>:optional</u>	input:optional	Selects input elements with no "required" attribute
<u>:out-of-range</u>	input:out-of-range	Selects input elements with a value outside a specified range
<u>::placeholder</u>	input::placeholder	Selects input elements with the "placeholder" attribute specified
<u>:read-only</u>	input:read-only	Selects input elements with the "readonly" attribute specified
<u>:read-write</u>	input:read-write	Selects input elements with the "readonly" attribute NOT specified
<u>:required</u>	input:required	Selects input elements with the "required" attribute specified
<u>:root</u>	:root	Selects the document's root element
<u>::selection</u>	::selection	Selects the portion of an element that is selected by a user

Selector	Example	Example description
<u>:target</u>	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
<u>:valid</u>	input:valid	Selects all input elements with a valid value
<u>:visited</u>	a:visited	Selects all visited links

div p vs div > p:

[div > p] selects only the p that are **children of the div**. So if you had a div with lists or whatever inside that had their own p, their properties would not be affected by this selector. [div p] selects all descendant p in the div. So any p that is inside, or descendant, of a div would be affected.20-Apr-2016

▼ Position

Main position types

Value	Description
static	Default value. Elements render in order, as they appear in the document flow
absolute	The element is positioned relative to its first positioned (not static) ancestor element
fixed	The element is positioned relative to the browser window
relative	The element is positioned relative to its normal position, so "left:20px" adds 20 pixels to the element's LEFT position

absolute & relative: https://www.w3schools.com/css/tryit.asp?filename=trycss_position_absolute

▼ clear property

[Example: ResponsiveWebpage.html program]

The `clear` property controls the flow next to floated elements.

The `clear` property specifies what should happen with the element that is next to a floating element.

Value	Description
none	Default. The element is not pushed below left or right floated elements
left	The element is pushed below left floated elements
right	The element is pushed below right floated elements
both	The element is pushed below both left and right floated elements
initial	Sets this property to its default value. Read about initial
inherit	Inherits this property from its parent element. Read about inherit

▼ box-shadow

box-shadow = *h-offset v-offset blur spread color*

box-shadow: 5px 5px 20px grey;

▼ z-index

The `z-index` property specifies the stack order of an element.

An element with greater stack order is always in front of an element with a lower stack order.

Note: `z-index` **only works on positioned elements** (position: absolute, position: relative, position: fixed, or position: sticky) and flex items (elements that are direct children of `display: flex` elements).

Note: If two positioned elements overlap without a `z-index` specified, the element positioned last in the HTML code will be shown on top.

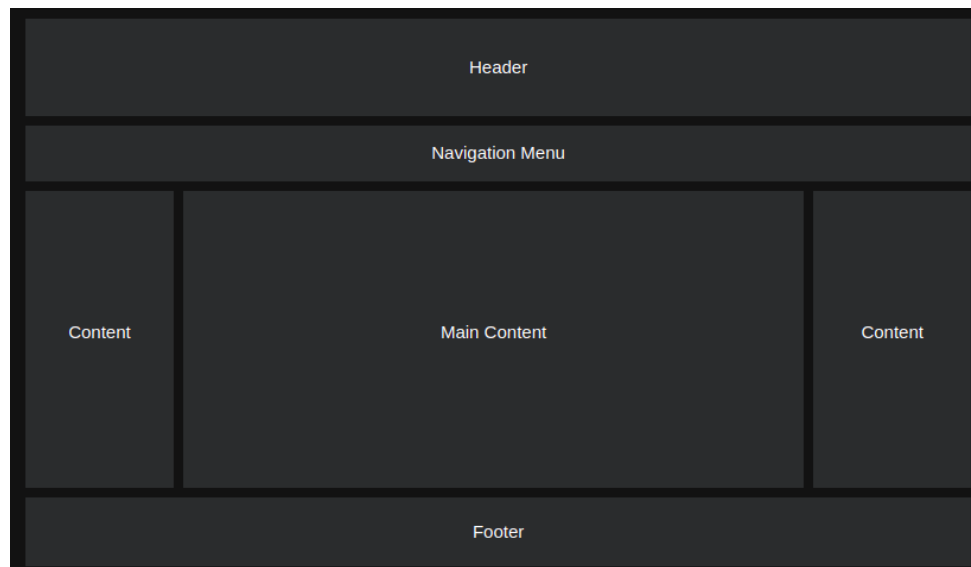
[Show demo >](#)

▼ Website layout

A website is often divided into headers, menus, content and a footer:


- Header
- Navigation Menu
- Content
- Main Content
- Content
- Footer

There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.



CSS Website Layout

A website is often divided into headers, menus, content and a footer: There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.

 https://www.w3schools.com/css/css_website_layout.asp



▼ Bootstrap

What is Bootstrap?

- Bootstrap is a free front-end framework for faster and easier web development
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
- Bootstrap also gives you the ability to easily create responsive designs

What is Responsive Web Design?

Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.

It works on the concept of class name. The class name provided in the div links the js of bootstrap with the HTML div which we create.

▼ Focus in input

The `:focus` selector is used to select the element that has focus.

Tip: The `:focus` selector is allowed on elements that accept keyboard events or other user inputs.

In the `:blur` selector: Attach a function to the blur event. The blur event occurs when the `<input>` field loses focus:

jQuery

▼ Introduction

jQuery is a powerful and widely used JavaScript library to simplify common web scripting task.

jQuery is a fast, lightweight, and feature-rich JavaScript library that is based on the principle "write less, do more". Its easy-to-use APIs makes the things like HTML document traversal and manipulation, event handling, adding animation effects to a web page much simpler that works seamlessly across all the major browsers like Chrome, Firefox, Safari, Internet Explorer, etc.

jQuery also gives you the **ability to create an Ajax based application in a quick and simple way**. Big companies like Google, Microsoft and IBM are using the jQuery for their applications. So you can easily understand how popular and powerful the jQuery is?

jQuery was originally created by John Resig in early 2006. The jQuery project is currently run and maintained by a distributed group of developers as an open-source project.

What You Can Do with jQuery

There are lot more things you can do with jQuery.

- You can easily select elements to perform manipulation.
- You can easily create effect like show or hide elements, sliding transition, and so on.
- You can easily create complex CSS animation with fewer lines of code.
- You can easily manipulate DOM elements and their attributes.
- You can easily implement Ajax to enable asynchronous data exchange between client and server.
- You can easily traverse all around the DOM tree to locate any element.
- You can easily perform multiple actions on an element with a single line of code.
- You can easily get or set dimensions of the HTML elements.

The list does not end here, there are many other interesting things that you can do with jQuery. You will learn about all of them in detail in upcoming chapters.

Advantages of Using jQuery

If you're not familiar with jQuery, you might be wondering what makes jQuery so special. There are several advantages why one should opt for jQuery:

- **Save lots of time** — You can save lots of time and efforts by using the jQuery inbuilt effects and selectors and concentrate on other development work.
- **Simplify common JavaScript tasks** — jQuery considerably simplifies the common JavaScript tasks. Now you can easily create feature rich and interactive web pages with fewer lines of codes, a typical example is implementing Ajax to update the content of a page without refreshing it.
- **Easy to use** — jQuery is very easy to use. Anybody with the basic working knowledge of HTML, CSS and JavaScript can start development with jQuery.
- **Compatible with browsers** — jQuery is created with modern browsers in mind and it is compatible with all major modern browsers such as Chrome, Firefox, Safari, Internet Explorer, etc.
- **Absolutely Free** — And the best part is, it is completely free to download and use.



Tip: In JavaScript, you often need to write several lines of code to select an element in an HTML document, but with **jQuery robust selector mechanism you can traverse the DOM tree and select elements in an easy and efficient manner to perform any manipulation.**

What This Tutorial Covers

This jQuery tutorial series covers all the features of the jQuery, including its selector mechanism, event handling system, as well as, effect methods to create interactive user interface features like showing and

hiding elements, animating the elements on a web page, etc.

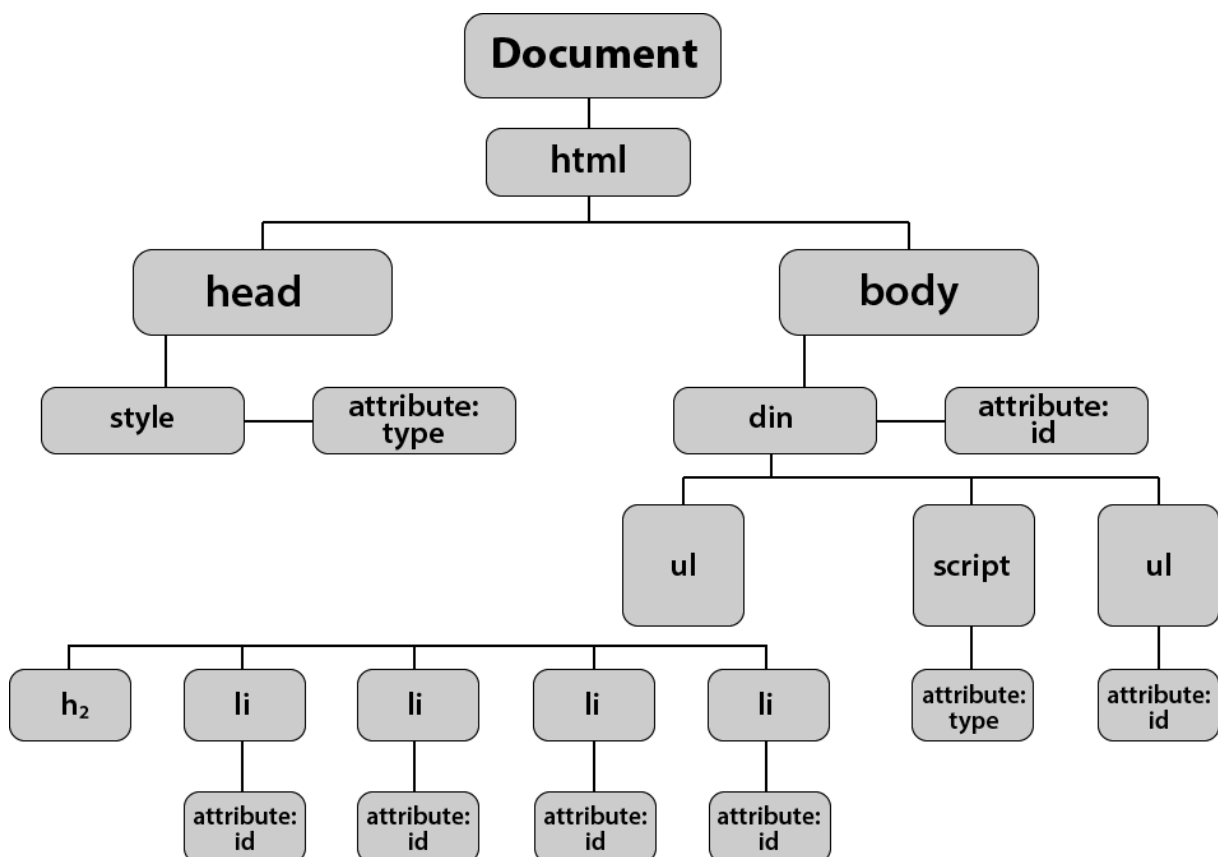
Later you will see some other interesting features of the jQuery such as chaining multiple methods, as well as how to perform common DOM manipulation task such as get or set contents and values of an HTML element on a web page, add or remove elements or their attributes, get and set CSS properties of an element, get or set width and height of the element, and so on.

Finally, you will explore one of the most powerful features of jQuery that is traversing the DOM tree to get the child, parent and sibling elements, as well as features like filtering element's selection, implementing Ajax to retrieve the information from a server and update the page content without refreshing it, and how to avoid conflicts between jQuery and other JavaScript library.

▼ DOM

When writing web pages and apps, one of the most common things you'll want to do is manipulate the document structure in some way. This is usually done by using the Document Object Model (DOM), **a set of APIs for controlling HTML and styling information that makes heavy use of the Document object.**

Even head includes title element.



Correction: din* = div

▼ JQuery vs Javascript

	JavaScript	jQuery
Definition	▸ A unique programming language.	▸ A concise JavaScript library. ▸ jQuery is still dependent on JavaScript.
Language	▸ A high-level interpreted client-side scripting language. ▸ JS combined between ECMAScript and DOM.	▸ A lightweight JavaScript library and - It has only the DOM
Animations	▸ You must use many lines of code to create animations in JavaScript.	▸ jQuery allows you to add animation effects easily with fewer lines of code.
Cross-browser Compatibility	Yes	No
Performance	▸ Directly processed by the browser. JavaScript can be faster for DOM selection/manipulation than jQuery.	▸ jQuery has to be converted into JavaScript to make it run in a browser.
Size	▸ This language is heavier than jQuery.	▸ jQuery is lightweight as a library.

▼ Javascript variables

4 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let`
- Using `const`
- Using nothing

When to Use JavaScript var?

Always declare JavaScript variables with `var`, `let`, or `const`.

The `var` keyword is used in all JavaScript code from 1995 to 2015.

The `let` and `const` keywords were added to JavaScript in 2015.
If you want your code to run in older browser, you must use `var`.

When to Use JavaScript const?

If you want a general rule: always declare variables with `const`.
If you think the value of the variable can change, use `let`.

let vs var

The `let` keyword was introduced in ES6 (2015).
Variables defined with `let` cannot be Redeclared.
Variables defined with `let` must be Declared before use.
Variables defined with `let` have Block Scope.

Cannot be Redeclared

Variables defined with `let` cannot be **redeclared**.
You cannot accidentally redeclare a variable.

With `let` you can not do this:

```
let x = "John Doe";  
let x = 0;  
// SyntaxError: 'x' has already been declared
```

With `var` you can:

```
var x = "John Doe";  
var x = 0;
```

Block Scope

Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.
ES6 introduced two important new JavaScript keywords: `let` and `const`.
These two keywords provide **Block Scope** in JavaScript.
Variables declared inside a `{ }` block cannot be accessed from outside the block:

Example

```
{  
  let x = 2;  
}  
// x can NOT be used here
```

Variables declared with the `var` keyword can NOT have block scope.

Variables declared inside a `{ }` block can be accessed from outside the block.

Example

```
{  
var x = 2;  
}  
  
// x CAN be used here
```

Redeclaring Variables

Redeclaring a variable using the `var` keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example

```
var x = 10;  
// Here x is 10  
  
{  
var x = 2;  
// Here x is 2  
}  
  
// Here x is 2
```

Redeclaring a variable using the `let` keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
let x = 10;  
// Here x is 10  
  
{  
let x = 2;  
// Here x is 2  
}  
  
// Here x is 10
```

▼ Javascript arithmetic

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

```
let x = 16 + 4 + "Volvo";    //20volvo
let x = "Volvo" + 16 + 4;    //volvo164
```

JavaScript types are dynamic

```
let x;           // Now x is undefined
x = 5;           // Now x is a Number
x = "John";      // Now x is a String
```

JavaScript Objects

JavaScript objects are written with curly braces `{ }`.

Object properties are written as **name:value** pairs, separated by commas.

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Type of

```
typeof ""        // Returns "string"
typeof "John"     // Returns "string"
typeof "John Doe" // Returns "string"
typeof 0          // Returns "number"
typeof 314        // Returns "number"
typeof 3.14       // Returns "number"
typeof (3)        // Returns "number"
typeof (3 + 4)    // Returns "number"
```

In JavaScript, a variable without a value, has the value `undefined`. The type is also `undefined`.

```
let car; // Value is undefined, type is undefined
```

▼ Javascript function

```
let x = myFunction(4, 3); // Function is called, return value will end up in x
function myFunction(a, b) {
  return a * b;           // Function returns the product of a and b
}
```

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
return (5/9) * (fahrenheit-32);
}

document.getElementById("demo").innerHTML = toCelsius(77);
```

[Try it Yourself »](#)

The () Operator Invokes the Function

Using the example above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result. Accessing a function without `()` will return the function object instead of the function result.

Example

```
function toCelsius(fahrenheit) {
return (5/9) * (fahrenheit-32);
}

document.getElementById("demo").innerHTML = toCelsius; //you will get whole function
```

▼ Javascript array

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

▼ Return value from js function

```
userExist:function(rollCheck)
{
varreturnObj =null;

$("table tr").find("td:eq(0)").each(function(index, element)
{
// console.log($(element).parent()); //it is jquery object
// console.log(this.parentElement); //it is just text

varelementText = ($(element).text());

if(elementText === rollCheck)
{
returnObj = $(element).parent();
}
});
});
```

```
returnreturnObj;
//If you return this object inside the each function then it would be send to
//the each function and not the userExist function because js works in a
// different manner then java.
},
```

The `concat()` method returns **a new string**.

▼ File loading

Window: DOMContentLoaded event

The `DOMContentLoaded` event fires when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading.

Window: load event

The `load` event is fired when the whole page has loaded, including all dependent resources such as stylesheets and images. This is in contrast to `DOMContentLoaded`, which is fired as soon as the page DOM has been loaded, without waiting for resources to finish loading.

`$(document).ready()`

A page can't be manipulated safely until the document is "ready." jQuery detects this state of readiness for you. Code included inside `$(document).ready()` will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute. Code included inside `$(window).on("load", function() { ... })` will run once the entire page (images or iframes), not just the DOM, is ready.

```
$( document ).ready()
```

A page can't be manipulated safely until the document is "ready." jQuery detects this state of readiness for you. Code included inside `$(document).ready()` will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute. Code

 <https://learn.jquery.com/using-jquery-core/document-ready/>



▼ Code snippets

IMP: Selecting on the basis of element, id, class and attribute
 element: button || class: .className || id: #myText || attribute: 'input[type="text"]'

```
//double click
<script>
$(document).ready(function(){
    $("p").dblclick(function(){
        $(this).slideUp();
    });
});
</script>
```

```
//keypress. Similar: keyup and keydown
<script>
$(document).ready(function(){
    var i = 0;
    $('input[type="text"]').keypress(function(){
        $("span").text(i += 1);
        $("p").show().fadeOut();
    });
});
</script>
```

```
//scroll
<script>
$(document).ready(function(){
    $(window).scroll(function() {
        $("p").show().fadeOut("slow");
    });
});
```

```
//window size
<script>
$(document).ready(function(){
    $(window).resize(function() {
        $(window).bind("resize", function(){
            $("p").text("Window width: " + $(window).width() + ", " + "Window height: " + $(window).height());
        });
    });
});
</script>
```


```
//window scroll
<script>
$(document).ready(function(){
    $(window).scroll(function() {
        $("p").show().fadeOut("slow");
    });
});
</script>
```

```
//fade
//fadeIn fadeOut
//fadeTo("fast", 0)
    speed, opacity(from 0 to 1)
```

Remove attribute removes only the inline css attributes applied to it. The part present in internal and external CSS is not affected.

Example of jQuery Method Chaining

Try and test HTML code online in a simple and easy way using our free HTML editor and see the results in real-time.

 <https://www.tutorialrepublic.com/codelab.php?topic=jquery&file=breaking-method-chain-ing-code-in-multiple-lines>



```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Method Chaining</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
/* Some custom styles to beautify this example */
p {
    width: 200px;
    padding: 40px 0;
    font: bold 24px sans-serif;
    text-align: center;
    background: #aaccaa;
    border: 1px solid #63a063;
    box-sizing: border-box;
}
</style>
<script>
$(document).ready(function(){
    $(".start").click(function(){
        $(".p")
            .animate({width: "100%"})
            .animate({fontSize: "46px"})
            .animate({borderWidth: 30});
    });
    $(".reset").click(function(){
        $(".p").removeAttr("style");
    });
});
</script>
</head>
<body>
<p>Hello World!</p>
<button type="button" class="start">Start Chaining</button>
<button type="button" class="reset">Reset</button>
</body>
</html>

```

▼ Regex for forms

^[a-zA-Z] vs [^a-zA-Z]

^ outside of the character class ("[a-zA-Z]") notes that it is the "begins with" operator. ^ inside of the character negates the specified class.

So, "[a-zA-Z]" translates to "begins with character from a-z or A-Z", and "[^a-zA-Z]" translates to "is not either a-z or A-Z"

Basics

1. A regex (*regular expression*) consists of a sequence of *sub-expressions*. In this example, `[0-9]` and `+`.
2. The `[...]`, known as *character class* (or *bracket list*), encloses a list of characters. It matches any SINGLE character in the list. In this example, `[0-9]` matches any SINGLE character between 0 and 9 (i.e., a digit), where dash (`-`) denotes the *range*.
3. The `+`, known as *occurrence indicator* (or *repetition operator*), indicates one or more occurrences (`1+`) of the previous sub-expression. In this case, `[0-9]+` matches one or more digits.

4. A regex may match a portion of the input (i.e., substring) or the entire input. In fact, it could match zero or more substrings of the input (with global modifier).
5. This regex matches any numeric substring (of digits 0 to 9) of the input. For examples, Take note that this regex matches number with leading zeros, such as `"000"`, `"0123"` and `"0001"`, which may not be desirable.
 - a. If the input is `"abc123xyz"`, it matches substring `"123"`.
 - b. If the input is `"abcxyz"`, it matches nothing.
 - c. If the input is `"abc00123xyz456_0"`, it matches substrings `"00123"`, `"456"` and `"0"` (three matches).
6. You can also write `\d+`, where `\d` is known as a *metacharacter* that matches any digit (same as `[0-9]`). There are more than one ways to write a regex! Take note that many programming languages (C, Java, JavaScript, Python) use backslash `\` as the prefix for escape sequences (e.g., `\n` for newline), and you need to write `"\\d+"` instead.

\$: Matches the end of input. If the multiline flag is set to true, also matches immediately before a line break character. For example, `/t$/` does not match the "t" in "eater", but does match it in "eat".

▼ Bind

The **bind()** is an inbuilt method in jQuery which is used to attach one or more event handlers for selected element and this method specifies a function to run when event occurs. **Syntax:**

```
$(selector).bind(event, data, function);
```

Parameter: It accepts three parameters that are specified below-

1. **event:** This is an event type which is passed to the selected elements.
2. **data:** This is the data which can be shown over the selected elements.
3. **function:** This is the function which is perform by the selected elements.

Return Value: It returns all the modification made on the selected element.

jQuery codes to show the working of bind() method:

```
<script>
  $(document).ready(function() {
    $("p").bind("click", function() {
      alert("Given paragraph was clicked.");
    });
  });
</script>
```

```
<script>
  function handlerName(e) {
    alert(e.data.msg); //see how we retrieve our message passed in bind
  }

  <!-- Here data is passing along with a function in bind method -->
  $(document).ready(function() {
```



```

        $("p").bind("click", {
            msg: "You just clicked the paragraph!"
        }, handlerName)
    });
</script>

```

As of jQuery 3.0, `.bind()` has been deprecated. It was superseded by the `.on()` method for attaching event handlers to a document since jQuery 1.7, so its use was already discouraged. For earlier versions, the `.bind()` method is used for attaching an event handler directly to elements. Handlers are attached to the currently selected elements in the jQuery object, so those elements *must exist* at the point the call to `.bind()` occurs. For more flexible event binding, see the discussion of event delegation in `.on()`.

▼ Toggle vs show

The `toggle()` method toggles between `hide()` and `show()` for the selected elements. This method checks the selected elements for visibility. **show() is run if an element is hidden.** `hide()` is run if an element is visible - This creates a toggle effect.

▼ jQuery Method Chaining

The jQuery provides another robust feature called method chaining that allows us to perform multiple action on the same set of elements, all within a single line of code.

This is possible because most of the jQuery methods return a jQuery object that can be further used to call another method.

Tip: The method chaining not only helps you to keep your jQuery code concise, but it also can improve your script's performance since browser doesn't have to find the same elements multiple times to do something with them.

A typical example of this scenario is the `html()` method. If no parameters are passed to it, the HTML contents of the selected element is returned instead of a jQuery object.

▼ jQuery Callback Functions

JavaScript statements are executed line by line. But, since **jQuery effect takes some time to finish the next line code may execute while the previous effect is still running**. To prevent this from happening jQuery provides a callback function for each **effect** method.

A callback function is a function that is executed once the effect is complete. The callback function is passed as an argument to the effect methods and they typically appear as the last argument of the method. For example, the basic syntax of the jQuery `slideToggle()` effect method with a callback function can be given with:

```

$(selector).slideToggle(duration, callback);

```

Consider the following example in which we've placed the `slideToggle()` and `alert()` statements next to each other. If you try this code the alert will be displayed immediately once you click the trigger button without waiting for slide toggle effect to complete.

And, here's the modified version of the previous example in which we've placed the `alert()` statement inside a callback function for the `slideToggle()` method. If you try this code the alert message will be displayed once the slide toggle effect has completed.

Similarly, you can define the callback functions for the other jQuery effect methods, like `show()`, `hide()`, `fadeIn()`, `fadeOut()`, `animate()`, etc.

Note: If the effect method is applied to multiple elements, then the callback function is executed once for each selected element, not once for all.

```
$( "h1, p" ).slideToggle("slow", function(){
    // Code to be executed once effect is complete
    alert("The slide toggle effect has completed.");
});
```

A callback is a plain JavaScript function passed to some method as an argument or option. Some callbacks are just events, called to give the user a chance to react when a certain state is triggered. jQuery's event system uses such callbacks everywhere:

```
$( "body" ).click(function( event ) {
    console.log( "clicked: " + event.target );
});
```

Most callbacks provide arguments and a context. In the event-handler example, the callback is called with one argument, an Event. The context is set to the handling element, in the above example, document.body.

Some callbacks are required to return something, others make that return value optional. To prevent a form submission, a submit event handler can return false:

```
$( "#myform" ).submit(function() {
    return false;
});
```

Instead of always returning false, the callback could check fields of the form for validity, and return false only when the form is invalid.

▼ Right Coding practices

- Keeping body empty → single page application
- Creating separate javascript file
 - Creating separate variables which would include different functions of each file
- Create different css file which would include classes and would include fix properties that would be applied to the elements
- Create a js file which would include a HTML Loader function which adds HTML data into the div main using `.html` and also the jquery methods created using `on("action", "#elementId", "methodName")`

- Calling the only function HTML loader using `<script>...</script>` block which is placed just before the closing of body tag.

▼ jQuery Get or Set Contents and Values



.text() is used in case of getting content when we receive a html element.
.val() is used in case of input elements.

Some jQuery methods can be used to either assign or read some value on a selection. A few of these methods are `text()`, `html()`, `attr()`, and `val()`.

When these methods are called with no argument, it is referred to as a *getters*, because it gets (or reads) the value of the element. When these methods are called with a value as an argument, it's referred to as a *setter* because it sets (or assigns) that value.

GETTERS: The jQuery `text()` retrieves the values of all the selected elements (i.e. combined text), whereas the other getters such as `html()`, `attr()`, and `val()` returns the value only from the first element in the selection.

SETTERS: When the jQuery `text()`, `html()`, `attr()`, and `val()` methods are called with a value as an argument it sets that value to every matched element.

Note: `attr()` only returns the **inline values**.

Eg. `$("p").text()` gives data in all paragraphs.

`$("p").text("Text to replace")` sets all the paragraphs with the given text.

.value vs val()

There is actually quite a big difference between `.val()` and `.attr("value")`. **The former gets the objects desired value (from the HTML code) whereas the latter gets the objects actual value once the HTML document is created.**

To get the value from the event from the selected dropdown option → `event.currentTarget.value`

To get the value of the tag's attribute → `$("#p1").val()` or `$("#p1").attr("value")`

The jQuery `val()` method is mainly used to get or set the current value of the HTML form elements such as `<input>`, `<select>` and `<textarea>`.

```
<script>
$(document).ready(function(){
    $("#button.get-name").click(function(){
        var name = $("#name").val();
        alert(name);
    });
});
```

```

$("button.get-comment").click(function(){
    var comment = $("#comment").val();
    alert(comment);
});
$("button.get-city").click(function(){
    var city = $("#city").val();
    alert(city);
});
});
</script>

```

▼ Toggle checkbox

```

$('input[type="checkbox"]').each(function(){
    this.checked = !this.checked
});

```

▼ .click vs on("click")

NO DIFFERENCE.

▼ answer

- '.click' requires the event handler to be attached to all elements that match the same selector. This results in needless overhead when the defined selector occurs in the DOM multiple times, because each of these elements needs to be bound and monitored to and by the event handler.
- '.on' on the other hand, *allows* to create an event handler that gets attached to a single parent element to ultimately give you the same effect without the previously mentioned overhead. Dynamically generated descendent elements that match the selector within this parent container will automatically be handled for you, while additional binding of the handler to the added element would otherwise be required when opting for the first (.click) approach. If not clear, below the example I'll explain a bit more extensively.

Consider the following example (taken from jQuery's official documentation):

Imagine a table with 1000 rows and the following event handler:

```

| $('#dataTable tbody tr').click

```

The previous definition is equivalent to:

```

| $( "#dataTable tbody tr" ).on( "click", function() {alert( $( this ).text() );});

```

resulting into an event handler that gets attached to and monitors 1000 elements.

But this is not equal to:

```

| $( "#dataTable tbody" ).on( "click", "tr", function() {alert( $( this ).text() );});

```

This strategy uses a native Javascript occurrence that is called '**event bubbling**': it practically means that the event from the child element is *propagated* through its ancestors. In the example, the dataTable tbody element handles the events that bubbles up from all child elements that match the same selector (tr). This strategy is called **event delegation**.

It is important to know how event delegation works in the scope of more processing intensive events than 'click' (such as mousemove or scroll) for performance. Quote from jQuery's documentation and important to know when using 'on':

Attaching many delegated event handlers near the top of the document tree can degrade performance. Each time the event occurs, jQuery must compare all selectors of all attached events of that type to every element in the path from the event target up to the top of the document. For best performance, attach delegated events at a document location as close as possible to the target elements. Avoid excessive use of document or document.body for delegated events on large documents.

Similarity: Direct events are only attached to elements at the time the method is called. In cases, where our anchor might not exist when it was called, it does not get the event handler.

Difference: I think, the difference is in usage patterns.

I would prefer `.on` over `.click` because the former *can* use **less memory** and work for dynamically added elements.

Consider the following html:

```
<html>
  <button id="add">Add new</button><div id="container">
    <button class="alert">alert!</button>
  </div>
</html>
```

where we add new buttons via

```
$("#button#add").click(function() {
  var html = "<button class='alert'>Alert!</button>";
  $("#button.alert:last").parent().append(html);
});
```

and want "Alert!" to show an alert. We can use either "click" or "on" for that.

When we use `click`

```
$("#button.alert").click(function() {
    alert(1);
});
```

with the above, a *separate* handler gets created for *every single element* that matches the selector. That means

1. **many matching elements would create many identical handlers and thus increase memory footprint**
2. dynamically added items won't have the handler - ie, in the above html the **newly added "Alert!" buttons won't work** unless you rebind the handler.

When we use `.on`

```
$("#div#container").on('click', 'button.alert', function() {
    alert(1);
});
```

with the above, a *single* handler for *all elements* that match your selector, including the ones created dynamically.

...another reason to use `.on`

If you add a handler with `.on("click", handler)` you normally remove it with `.off("click", handler)` which will remove that very handler. Obviously this works only if you have a reference to the function, so what if you don't ? You use namespaces:

```
$("#element").on("click.someNamespace", function() { console.log("anonymous!"); });
```

with unbinding via

```
$("#element").off("click.someNamespace");
//or
process.exit()
```

▼ Append & prepend. After & before

The `append()` method inserts specified content at the end of the selected elements.

Tip: To insert content at the beginning of the selected elements, use the `prepend()` method.

.append(): This function Insert the data or content inside an element at last index. Means it puts the content inside an element (making the content its child) specified by the parameter at the end of element in the set of matched elements.

Whereas ,

.after(): This function puts the element after the specified element. Or you can say that it inserts data outside an element (making the content its sibling) in the set of matched elements.

Let's see an example to get the difference in practice:

Suppose you have HTML code like this.

```
<div class='a'> //<---you want div c to append in this div at last.
<div class='b'>b</div>
</div>
```

Then you should use **.append()** function like this.

```
$("#div.a").append("<div class='c'>C</div>");
```

Then, the HTML code looks like this.

```
<div class='a'>
<div class='b'>b</div>
<div class='c'>C</div> /<----this will be placed here.
</div>On other hand,
```

And, if you are using **.after()** method in the above same HTML code, then it will add at the last div having class a as its sibling.

```
$("#div.a").after("<div class='c'>C</div>");
<div class='a'>
<div class='b'>b</div>
</div>
<div class='c'>C</div> //<----this will be placed here.
```

So, this will clearly understand from the above that:

The **.append()** inserts the parameter element inside the selector element's tag at the last index position, whereas the **.after()** puts the parameter element after the matched element's tag.

▼ Wrap

```
<script>
$(document).ready(function(){
    // Wrap div container with another div on document ready
    $(".container").wrap('<div class="wrapper"></div>');

    // Wrap paragraph's content on button click
    $("button").click(function(){
        $("p").contents().wrap("<em><b></b></em>");
    });
});
</script>
```

▼ empty, remove, unwrap & removeAttr

empty: clears the content inside the element

remove: it completely removes the whole element

unwrap: it removes the parent

removeAttr: it removes the attribute of all the elements received from the selector

▼ DOM vs jQuery objects

In one sentence, DOM objects are the objects that the web browser is using to render elements on the web page whereas jQuery objects are basically wrapper objects around a set of DOM elements.

What are JavaScript DOM objects?

As mentioned earlier, DOM objects are used by browser directly to render the webpage in browser window. The browser receives an HTML document from a web server, which is just text. The browser proceeds to parse this text into an internal structure that it can actually use to render the page visually. **The DOM represents that internal structure a browser has of an HTML document. A DOM object represents a visual or functional object on the page which was created from the original HTML document.**

Even when browser has fully rendered the webpage, you can use JavaScript to change the DOM objects, it's **attributes** and values. Any change done in such way automatically refreshes the visual representation shown in browser window.

The **advantage with working with DOM objects** is that you have direct access to everything you need to manipulate the HTML element. The **disadvantage of DOM objects** is that most of the attached functions and attributes are things that the browser needs and are not necessarily useful when you're working with JavaScript. It makes working with them a little slower, at least for less-experienced developers.

What are jQuery objects?

jQuery objects are wrapper objects around single or multiple DOM elements. The jQuery objects (though technically still JavaScript objects) provide access to the wrapped DOM elements — however, in a much different, much easier, and often much more effective way.

▼ this vs \$(this)

this keyword: In JavaScript, **this** keyword is used to refer to the object it belongs to. The value that **this** stores is the **current execution context** of the JavaScript program. Thus, when used inside a function **this**'s value will change depending on how that function is defined, how it is invoked and the default execution context.

\$(this): It also refers to the object it belongs to. Basically, both are the same. But when **this** keyword is used inside **\$()**, then it becomes a jQuery object, and now we can use all **properties of jQuery** on this method.
[Properties: <https://api.jquery.com/>]

▼ this vs event.target vs event.currentTarget

Similarity: All 3 of them are DOM object

To use them as jQuery object we need to use them as `$(this)`, `$(event.target)` or `$(event.currentTarget)`. Then the methods can be used like `$(event.target).text()` or `$(event.target).val()` or `$(event.target).html("<div id='newDiv'>Hello World</div>")`.

Difference: There is a difference between `this` and `event.target`, and quite a significant one.

While `this` (or `event.currentTarget`, see below) always refers to the DOM element the listener was attached to, `event.target` is the actual DOM element that was clicked. Remember that due to event bubbling, if you have

```
<div class="outer">
  <div class="inner"></div>
</div>
```

and attach click listener to the outer div

```
$('.outer').click( handler );
```

then the `handler` will be invoked when you click inside the outer div as well as the inner one (unless you have other code that handles the event on the inner div and stops propagation).

In this example, when you click inside the inner div, then in the `handler`:

- `this` refers to the `.outer` DOM element (because that's the object to which the handler was attached)
- `event.currentTarget` also refers to the `.outer` element (because that's the *current target* element handling the event)
- `event.target` refers to the `.inner` element (this gives you the element where the event originated)

The jQuery wrapper `$(this)` only wraps the DOM element in a jQuery object so you can call jQuery functions on it. You can do the same with `$(event.target)`.

Also note that if you rebind the context of `this` (e.g. if you use Backbone it's done automatically), it will point to something else. You can always get the actual DOM element from `event.currentTarget`.

▼ Traversing

The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

`parent()`, `parents()`, `parents("div")`, `parentsUntil("html")`

parent → child

html → body → p → em

```
$("#p").parent().addClass("frame") -> will add frame to body only
$("#p").parents().addClass("frame") -> will add frame to body, html
$("#em").parentsUntil("html").addClass("frame") -> will add frame to p, body
$("#p").children().addClass("frame") -> will add frame to em only
```

```
$("#div").find("li").addClass("frame") -> find all descendent which are li
$("#div").find("*").addClass("frame") -> find all descendants
```

```
//html is:
<html>
  <body>
    <p>Hello world. <em>How are you</em></p>
  </body>
</html>
//where frame is defined as follows
//.frame{
//  border: 2px solid green;
//}
```

```
$("#p").next().addClass("highlight") -> Selecting the Next Sibling Element in jQuery
$("#p").nextAll().addClass("highlight") -> Selecting All the Following Sibling Elements in
$("#h1").nextUntil("ul").addClass("highlight") -> Selecting All Following Siblings between Two Elements in jQuery
$("#ul").prev().addClass("highlight") -> Selecting the Previous Sibling Element in jQuery
$("#ul").prevAll().addClass("highlight") -> Selecting All the Preceding Sibling Elements
$("#ul").prevUntil("h1").addClass("highlight") -> Selecting All Preceding Siblings between Two Elements
```

Sibling elements

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Selecting All the Sibling Elements in jQuery</title>
<style>
  .highlight{
    background: yellow;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("#p").siblings().addClass("highlight").wrap("<b></b>");
  //$("#p").siblings("ul").addClass("highlight").wrap("<b></b>");
  //get only the sibling which is ul
});
</script>
</head>
<body>
  <div class="container">
    <span>Hello World</span>    <!--this will be wrapped -->
    <p>This is a <em>simple paragraph</em></p>
    <ul>    <!--this whole will be wrapped -->
      <li>Item One</li>
      <li>Item Two</li>
    </ul>
    hello<br>
    hi
  </div>
</body>
</html>
```

▼ each vs filter vs not

The main difference between each and filter is that each just iterate over a jQuery object, executing a function for each matched element but filter executes the callback and check its return value. If the value is true

element remains in the resulting array but if the return value is false the element will be removed for the resulting array.

each() is very useful for multi-element DOM manipulation, as well as iterating over arbitrary arrays and object properties.

not() is opposite of filter().

```
// each - function()
<script>
$(document).ready(function(){
    $("ul li li").each(
        function(){
            $(this).addClass("highlight");
        });
});
</script>

// each - function(index)
<script>
$(document).ready(function(){
    $("ul li li").each(
        function(index){
            alert(index);
        });
});
</script>

// each - function(index, element)
<script>
$(document).ready(function(){
    $("ul li li").each(
        function(index, element){
            $(element).addClass("highlight");
        });
});
</script>
```

▼ Event bubbling

Event bubbling is a method of event propagation in the HTML DOM API when an event is in an element inside another element, and both elements have registered a handle to that event. It is a process that starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy. In event bubbling, the event is first captured and handled by the **innermost element and then propagated to outer elements**.

Bubbling and capturing

Let's start with an example. This handler is assigned to , but also runs if you click any nested tag like or : Isn't it a bit strange? Why does the handler on run if the actual click was on ? The bubbling principle is simple.

✂ <https://javascript.info/bubbling-and-capturing>



JAVASCRIPT.INFO
The Modern JavaScript Tutorial

CurrentTarget vs Target in Js

| What is the difference between event.currentTarget and event.target in JS ?

To understand that we need to understand what are events and how they occur in JS.

Interaction of User with the dom is known as events. Clicking of buttons, hovering of mouse over an element, pressing of any keys are all events.

Whenever these events occur we would like to run some JS function which is done using event handlers.

Now every event goes through three phases of event propagation:

1. From window to the target element phase.
2. The event target phase and
3. From the event target back to the window phase.

event.currentTarget tells us on which element the event was attached or the element whose eventListener triggered the event.

event.target tells where the event started.

Suppose there's an event which shows an alert on click of the element. This event has been attached to the body. Now when the user clicks on the strong tag, `currentTarget().nodeName` will show the body whereas `target` will show strong as the alert output.

```
<body onclick="myFunction(event)">
<p>Click on a paragraph. An alert box will alert the element whose
eventlistener triggered the event.</p>
<p><strong>Note:</strong> The currentTarget property does not necessarily
return the element that was clicked on, but the element whose eventlistener
triggered the event.</p>
<script>
  function myFunction(event) {
    alert(event.currentTarget.nodeName); // body
    alert(event.target.nodeName);      // strong
  }
</script>
```

Links:

[Event Bubbling](#)

▼ `toggleClass(class1)`, `removeClass()`, `addClass(class1 class2)`

```
<style>
  p{
    padding: 10px;
    cursor: pointer;
    font: bold 16px sans-serif;
  }
  .highlight{
    background: yellow;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).toggleClass("highlight");
  });
});
```

```
});
});
</script>
```

▼ CSS

```
<script>
$(document).ready(function(){
    $("div").click(function(){
        var color = $(this).css("background-color");
        $("#result").html(color);//prints rgb(238, 130, 238)
    });
});
</script>

//in body we have
<b id="result"></b>
```

```
<script>
$(document).ready(function(){
    $(".box").click(function(){
        $(this).css("background-color", "blue");
    });
});
</script>
```

```
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").css({"background-color": "yellow", "padding": "20px"});
    });
});
</script>
```

▼ Dimension

`.width()`: It provides from the css as well

`attr("width")`: It provides only the inline attribute value

Note: jQuery `innerWidth()` includes the CSS properties (**width + padding-left + padding-right**), whereas the `innerHeight()` includes (**height + padding-top + padding-bottom**).

`outerWidth()` does not include the **margin**.

`outerWidth(true)` it includes the margin.

▼ Closure in javascript

See program [AccessingValue.js](#)

`this.done` is not defined. But then when `undefined` is used with `not` then `undefined` implicitly converts to `false`, and then `!` negates it and results in **true**.

```
this.xyz:undefined || !this.xyz:true
```

Closures is an ability of a function to remember the variables and functions that are declared in its outer scope.

```
var Person = function(pName){
  var name = pName;

  this.getName = function(){
    return name;
  }
}

var person = new Person("Neelesh");
console.log(person.getName());
```

Let's understand closures by example:

```
function randomFunc(){
  var obj1 = {name:"Vivian", age:45};

  return function(){
    console.log(obj1.name + " is " + "awesome"); // Has access to obj1 even when the randomFunc function is executed
  }
}

var initialiseClosure = randomFunc(); // Returns a function

initialiseClosure();
```

Let's understand the code above,

The function `randomFunc()` gets executed and returns a function when we assign it to a variable:

```
var initialiseClosure = randomFunc();
```

The returned function is then executed when we invoke `initialiseClosure`:

```
initialiseClosure();
```

The line of code above outputs "Vivian is awesome" and this is possible because of closure.

When the function `randomFunc()` runs, it sees that the returning function is using the variable `obj1` inside it:

```
console.log(obj1.name + " is " + "awesome");
```

Therefore `randomFunc()`, instead of destroying the value of `obj1` after execution, saves the value in the memory for further reference.

This is the reason why the returning function is able to use the variable declared in the outer scope even after the function is already executed.

This ability of a function to store a variable for further reference even after it is executed, is called Closure.

▼ Filtering

first → first element

last → last element

eq(1) → second element

`$("ul li:even").addClass("highlight")` → if there are 2 ul list having 4 and 3 elements respectively then it fetches all of them and then apply highlight to the returned list. so 1 and 3 in first list and 1 and 3 in second list would be highlighted.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filtering the Selection of Elements in jQuery via Function</title>
<style>
    .highlight{
        background: yellow;
    }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
    $("ul li").filter(function(index){
        return index % 2 !== 0;
    }).addClass("highlight");
});
</script>
</head>
<body>
    <h2>Unordered List</h2>
    <ul>
        <li>First list item</li>
        <li>Second list item</li>
        <li>Third list item</li>
        <li>Fourth list item</li>
    </ul>
    <hr>
    <h2>Another Unordered List</h2>
    <ul>
        <li>First list item</li>
        <li>Second list item</li>
        <li>Third list item</li>
        <li>Fourth list item</li>
    </ul>
</body>
</html>
```

▼ Has

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

has expects only the type of element "ul", "p", ...

▼ Slice

```

<script>
$(document).ready(function(){
    //includes 0 and excludes 2
    $("ul li").slice(0, 2).addClass("highlight");
});
</script>

<script>
$(document).ready(function(){
    //includes third last and second last only. i.e. includes -3 and excludes -1
    $("ul li").slice(-3, -1).addClass("highlight");
});
</script>

```

▼ What happens if you submit a form?

The form will be submitted to the server and the browser will redirect away to the current address of the browser and append as query string parameters the values of the input fields.

In terms of the HTTP protocol the following GET request HTTP request will be sent:

```

GET http://example.com/?namefield1=value1&namefield2=value2 HTTP/1.1
Host: example.com

```

Since your `<form>` is missing an `action` attribute, the browser will simply redirect to the current url by appending the values as query string parameters. So if this form was loaded from `http://example.com/foo.php` after submitting it, the browser will redirect to `http://example.com/foo.php?namefield1=value1&namefield2=value2` where `value1` and `value2` will be the values entered by the user in the corresponding input fields.

▼ Struts

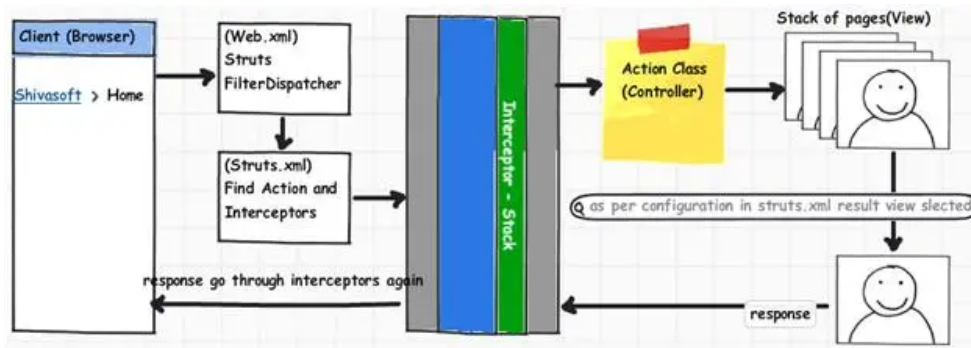
Struts 2

Share your videos with friends, family, and the world

 <https://www.youtube.com/playlist?list=PLB7BB551126EDD5E0>



▼ Complete flow



To use **ModelDriven** actions, make sure that the **Model Driven Interceptor** is applied to your action. This interceptor is part of the default interceptor stack **defaultStack** so it is applied to all actions by default.

Working summary: <https://www.jitendrazaa.com/blog/java/struts/what-is-struts-2-and-how-it-works/>

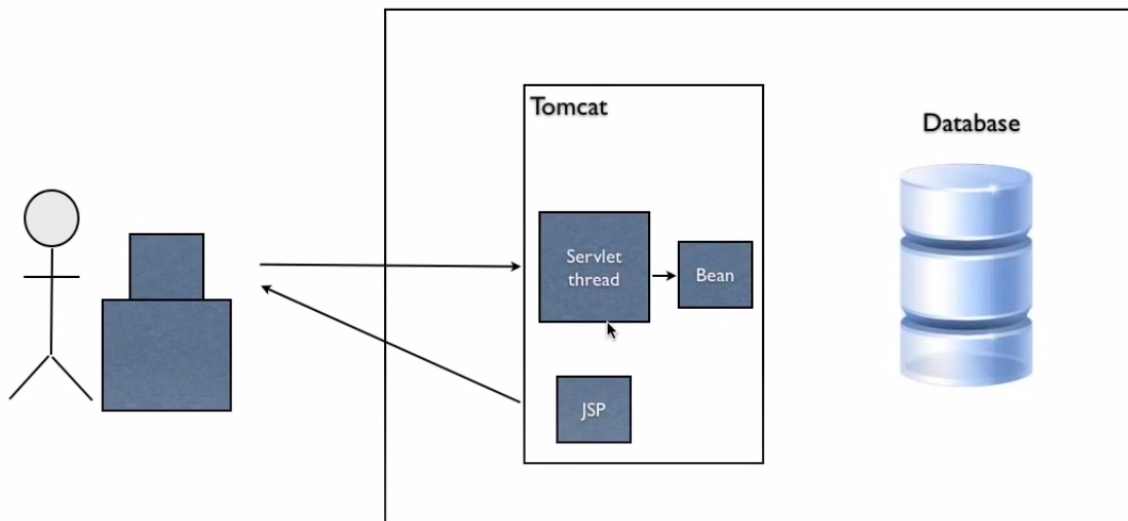
▼ MVC

Model: A business service or a class calling a business service. Data as well as the behaviour that pulls the data.

View: Renders the model which is presentable.

Controller: Passes necessary data to the model do its work and calls the right view.

Servlet with business service bean and JSP



Servlet thread: controller

JSP: View

Bean: Model

Why MVC?

Controller, model and view are separate from each other and independent.



Published business services are **Java classes that manage and run business services**.

Business services are Java classes that have one or more methods. Business service methods call business functions, database operations, and other business services to provide a specific, described unit of work.

Framework vs Pattern

- Pattern is the way you can architect your application.
- Framework provides foundation classes and libraries.
- Gets us started quickly
- Leverages industry best practices

▼ Servlets and JSP pages

Java Servlet technology and JavaServer Pages (JSP pages) are server-side technologies that have dominated the server-side Java technology market; they've become the standard way to develop commercial web applications. Java developers love these technologies for myriad reasons, including: the technologies are fairly easy to learn, and they bring the *Write Once, Run Anywhere* paradigm to web applications. More importantly, if used effectively by following best practices, servlets and JSP pages help separate presentation from content. *Best practices* are proven approaches for developing quality, reusable, and easily maintainable servlet- and JSP-based web applications. For instance, embedded Java code (scriptlets) in sections of HTML documents can result in complex applications that are not efficient, and difficult to reuse, enhance, and maintain. Best practices can change all that.

Overview of Servlets and JSP Pages

Servlets support a request and response programming model. **When a client sends a request to the server, the server sends the request to the servlet.** The servlet then constructs a response that the server sends back to the client. Servlets run within the same process as the HTTP server.

When a client request is made, the `service` method is called and passed a request and response object. The servlet first determines whether the request is a `GET` or `POST` operation. It then calls one of the following methods: `doGet` or `doPost`. The `doGet` method is called if the request is `GET`, and `doPost` is called

if the request is `POST`. Both `doGet` and `doPost` take request (`HttpServletRequest`) and response (`HttpServletResponse`).

In the simplest terms, then, servlets are Java classes that can generate dynamic HTML content using `print` statements. What is important to note about servlets, however, is that they run in a container, and the APIs provide session and object life-cycle management. Consequently, when you use servlets, you gain all the benefits from the Java platform, which include the sandbox (security), database access API via JDBC, and cross-platform portability of servlets.

JavaServer Pages (JSP)

The JSP technology--which abstracts servlets to a higher level--is an open, freely available specification developed by Sun Microsystems as an alternative to Microsoft's Active Server Pages (ASP) technology, and a key component of the Java 2 Enterprise Edition (J2EE) specification. Many of the commercially available application servers (such as BEA WebLogic, IBM WebSphere, Live JRun, Orion, and so on) support JSP technology.

How Do JSP Pages Work?

A JSP page is basically a web page with traditional HTML and bits of Java code. The file extension of a JSP page is `.jsp` rather than `.html` or `.htm`, which tells the server that this page requires special handling that will be accomplished by a server extension or a plug-in.

When a JSP page is called, it will be compiled (by the JSP engine) into a Java servlet. At this point the servlet is handled by the servlet engine, just like any other servlet. The servlet engine then loads the servlet class (using a class loader) and executes it to create dynamic HTML to be sent to the browser, as shown in Figure 1. **The servlet creates any necessary object, and writes any object as a string to an output stream to the browser.**

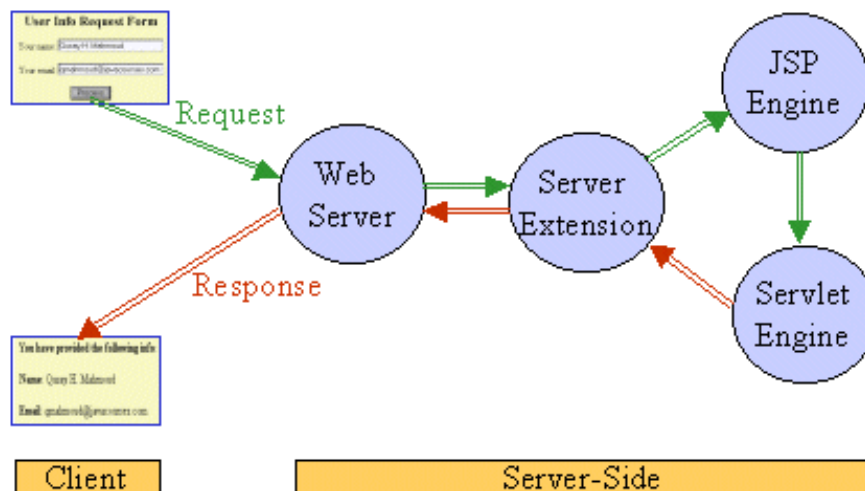


Figure 1: Request/Response flow calling a JSP page

The next time the page is requested, the JSP engine executes the already-loaded servlet unless the JSP page has changed, in which case it is automatically recompiled into a servlet and executed.

Integrating Servlets and JSP Pages

The JSP specification presents two approaches for building web applications using JSP pages: JSP Model 1 and Model 2 architectures. These two models differ in the location where the processing takes place. In Model 1 architecture, as shown in Figure 2, the JSP page is responsible for processing requests and sending back replies to clients.

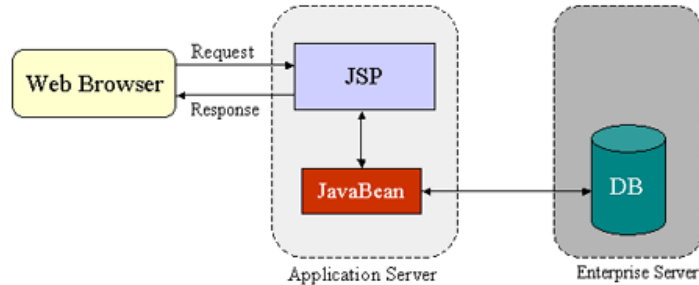


Figure 2: JSP Model 1 Architecture

The Model 2 architecture, as shown in Figure 3, integrates the use of both servlets and JSP pages. In this mode, JSP pages are used for the presentation layer, and servlets for processing tasks. The servlet acts as a *controller* responsible for processing requests and creating any beans needed by the JSP page. The controller is also responsible for deciding to which JSP page to forward the request. The JSP page retrieves objects created by the servlet and extracts dynamic content for insertion within a template.

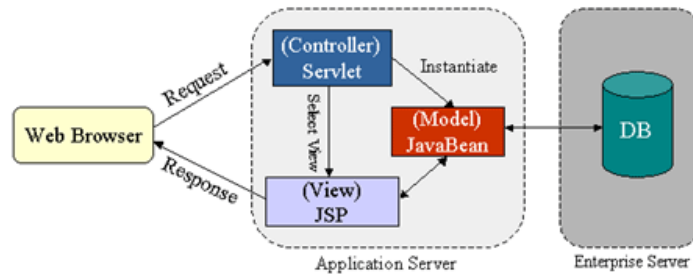
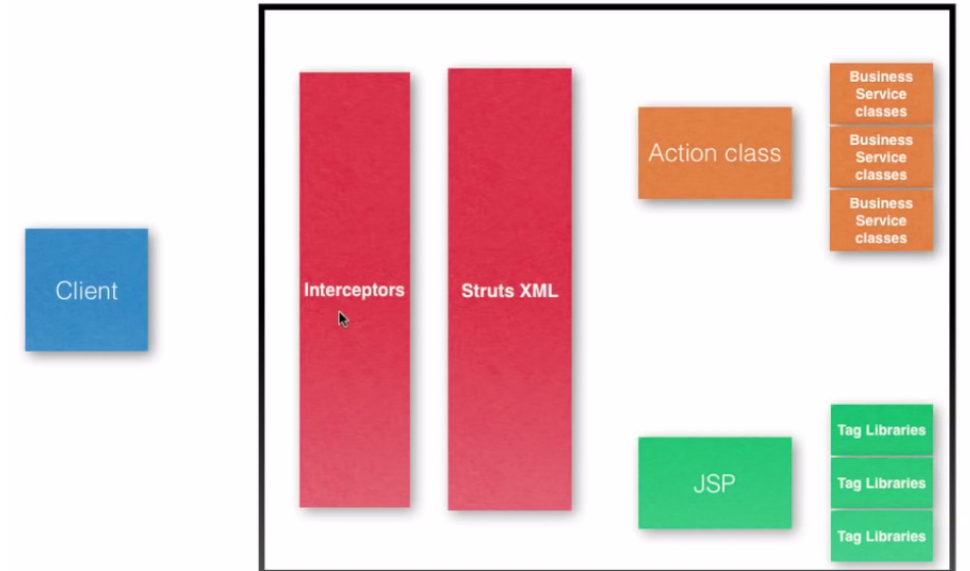


Figure 3: JSP Model 2 Architecture

This model promotes the use of the Model View Controller (MVC) architectural style design pattern. Note that several frameworks already exist that implement this useful design pattern, and that truly separate presentation from content. The Apache Struts is a formalized framework for MVC. This framework is best used for complex applications where a single request or form submission can result in substantially different-looking results.

▼ Understanding Struts MVC



Coding sequence

Write struts XML → Action class → JSP

Struts 2 XML

- Action class result 'code' to JSP

URL	Action Class	Code	JSP
http://mywebapp/getTutorial	TutorialAction	Success	success.jsp
		Failure	error.jsp
		NoSession	login.jsp
http://mywebapp/getBooks	BookAction		
http://mywebapp/getSeminars	SeminarAction		

▼ Configuring Struts

- Create new project maven-webapp
- Create folder java in main and mark directory as sources root
- Create folder resources in main and mark it as resources
- adding dependencies in pom.xml (code 1), filter in web.xml (code 2) and create actions in struts.xml (code 3).

```

<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-core</artifactId>
  <version>${struts2.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-json-plugin</artifactId>
  <version>${struts2.version}</version>
</dependency>

```

```

<filter>
  <filter-name>struts2</filter-name>
  <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
  "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>

  <package name="default" extends="struts-default, json-default">

    <result-types>
      <result-type name="json" class="org.apache.struts2.json.JSONResult">
        <param name="noCache">true</param>
        <param name="excludeNullProperties">true</param>
        <param name="enableGZIP">true</param>
      </result-type>
    </result-types>

    <interceptors>
      <interceptor name="userSession" class="litenms.interceptor.LoginSessionInterceptor"/>
      <interceptor-stack name="loginSession">
        <interceptor-ref name="userSession"/>
        <interceptor-ref name="defaultStack"/>
      </interceptor-stack>
    </interceptors>

    <global-results>
      <result name="loginError">login.jsp</result>
    </global-results>

    <action name="">
      <result>/login.jsp</result>
    </action>
  </package>

</struts>


```

▼ Ajax

Introduction to jQuery Ajax async. The jQuery Ajax async is handling Asynchronous HTTP requests in the element. **It is a procedure to send a request to the server without interruption.** It is an Asynchronous method to send HTTP requests without waiting response.

jQuery.ajax()

Description: Perform an asynchronous HTTP (Ajax) request. A string containing the URL to which the request is sent. A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with \$.ajaxSetup(). See

 <https://api.jquery.com/jquery.ajax/>



timeout

Type: Number

Set a timeout (in milliseconds) for the request. A value of 0 means there will be no timeout. This will override any global timeout set with \$.ajaxSetup(). The timeout period starts at the point the \$.ajax call is made; if several other requests are in progress and the browser has no connections available, it is possible for a request to time out before it can be sent.

In jQuery 1.4.x and below, the XMLHttpRequest object will be in an invalid state if the request times out; accessing any object members may throw an exception. **In Firefox 3.0+ only**, script and JSONP requests cannot be cancelled by a timeout; the script will run even if it arrives after the timeout period.

▼ Async await

Async/await

There's a special syntax to work with promises in a more comfortable fashion, called "async/await". It's surprisingly easy to understand and use. Let's start with the async keyword. It can be placed before a function, like this: The word "async" before a function

 <https://javascript.info/async-await>



SetTimeout

JavaScript Async / Await: Writing Asynchronous Code in a Clearer Syntax


Summary: in this tutorial, you will learn how to write asynchronous code using JavaScript async/ await keywords. Note that to understand how the async / await works, you need to know how promises work. In the past, to deal with asynchronous operations, you often used the callback functions.

 <https://www.javascripttutorial.net/es-next/javascript-async-await/>

▼ Fetch then

JavaScript Fetch API Explained By Examples

Summary: in this tutorial, you'll learn about the JavaScript Fetch API and how to use it to make asynchronous HTTP requests. The Fetch API is a modern interface that allows you to make HTTP requests to servers from web browsers.

 <https://www.javascripttutorial.net/javascript-fetch-api/>

▼ JQuery callbacks

▼ Need of callbacks

Sequence Control

Sometimes you would like to have better control over when to execute a function.

Suppose you want to do a calculation, and then display the result.

You could call a calculator function (`myCalculator`), save the result, and then call another function (`myDisplayer`) to display the result:

Example

```
function myDisplayer(some) { document.getElementById("demo").innerHTML =  
some;}function myCalculator(num1, num2) { let sum = num1 + num2; return sum;}let result =  
myCalculator(5, 5);myDisplayer(result);
```

[Try it Yourself »](#)

Or, you could call a calculator function (`myCalculator`), and let the calculator function call the display function (`myDisplayer`):

Example

```
function myDisplayer(some) { document.getElementById("demo").innerHTML =  
some;}function myCalculator(num1, num2) { let sum = num1 +  
num2; myDisplayer(sum);}myCalculator(5, 5);
```

[Try it Yourself »](#)

The problem with the first example above, is that you have to call two functions to display the result.

The problem with the second example, is that you cannot prevent the calculator function from displaying the result.

Now it is time to bring in a callback.

ADVERTISEMENT

JavaScript Callbacks

A callback is a function passed as an argument to another function.

Using a callback, you could call the calculator function (`myCalculator`) with a callback, and let the calculator function run the callback after the calculation is finished:

Example


```
function myDisplayer(some) { document.getElementById("demo").innerHTML =
some;}function myCalculator(num1, num2, myCallback) { let sum = num1 +
num2; myCallback(sum);}myCalculator(5, 5, myDisplayer);
```

[Try it Yourself »](#)

In the example above, `myDisplayer` is the name of a function.

It is passed to `myCalculator()` as an argument.

When you pass a function as an argument, remember not to use parenthesis.

Right: `myCalculator(5, 5, myDisplayer);`

Wrong: `myCalculator(5, 5, myDisplayer());`

When to Use a Callback?

The examples above are not very exciting.

They are simplified to teach you the callback syntax.

Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

Asynchronous functions are covered in the next chapter.

▼ Details

Description: A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

- **`jQuery.Callbacks(flags)`**

- **flags**

Type: `String`

An optional list of space-separated flags that change how the callback list behaves.

The `$.Callbacks()` function is internally used to provide the base functionality behind the jQuery `$.ajax()` and `$.Deferred()` components. It can be used as a similar base to define functionality for new components.

`$.Callbacks()` supports a number of methods including `callbacks.add()`, `callbacks.remove()`, `callbacks.fire()` and `callbacks.disable()`.

Getting started

The following are two sample methods named `fn1` and `fn2`:

```
function fn1( value ) {
  console.log( value );
}

function fn2( value ) {
  console.log( "fn2 says: " + value );
  return false;
}
```

These can be added as callbacks to a `$.Callbacks` list and invoked as follows:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );

// Outputs: foo!
callbacks.fire( "foo!" );

callbacks.add( fn2 );

// Outputs: bar!, fn2 says: bar!
callbacks.fire( "bar!" );
```

The result of this is that it becomes simple to construct complex lists of callbacks where input values can be passed through to as many functions as needed with ease.

Two specific methods were being used above: `.add()` and `.fire()`. The `.add()` method supports adding new callbacks to the callback list, while the `.fire()` method executes the added functions and provides a way to pass arguments to be processed by the callbacks in the same list.

Another method supported by `$.Callbacks` is `.remove()`, which has the ability to remove a particular callback from the callback list. Here's a practical example of `.remove()` being used:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );

// Outputs: foo!
callbacks.fire( "foo!" );

callbacks.add( fn2 );

// Outputs: bar!, fn2 says: bar!
callbacks.fire( "bar!" );

callbacks.remove( fn2 );

// Only outputs foo!bar, as fn2 has been removed.
callbacks.fire( "foo!bar" );
```

Supported Flags

The `flags` argument is an optional argument to `$.Callbacks()`, structured as a list of space-separated strings that change how the callback list behaves (eg. `$.Callbacks("unique stopOnFalse")`).

Possible flags:


- `once`: Ensures the callback list can only be fired once (like a Deferred).
- `memory`: Keeps track of previous values and will call any callback added after the list has been fired right away with the latest "memorized" values (like a Deferred).
- `unique`: Ensures a callback can only be added once (so there are no duplicates in the list).
- `stopOnFalse`: Interrupts callings when a callback returns false.

By default a callback list will act like an event callback list and can be "fired" multiple times.

▼ Model driven

Model Driven

If an action class implements the interface `com.opensymphony.xwork2.ModelDriven` then it needs to return an object from the `getModel()` method. Struts will then populate the fields of this object with the request parameters, and this object will be placed on top of the stack

 <https://struts.apache.org/core-developers/model-driven.html>

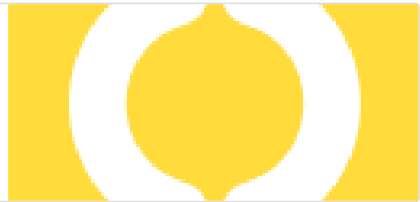


▼ Async and deferred in script

Asynchronous vs Deferred JavaScript

In my article on Understanding the Critical Rendering Path, I wrote about the effect JavaScript files have on the Critical Rendering Path. JavaScript is considered a "parser blocking resource". This means that the parsing of the HTML document itself is blocked by

 <https://bitsofco.de/async-vs-defer/>



Asynchronous and deferred execution of scripts are more important when the `<script>` element is not located at the very end of the document. HTML documents are parsed in order, from the first opening `<html>` element to its close. If an externally sourced JavaScript file is placed right before the closing `</body>` element, it becomes much less pertinent to use an `async` or `defer` attribute. Since the parser will have finished the vast majority of the document by that point, JavaScript files don't have much parsing left to block.

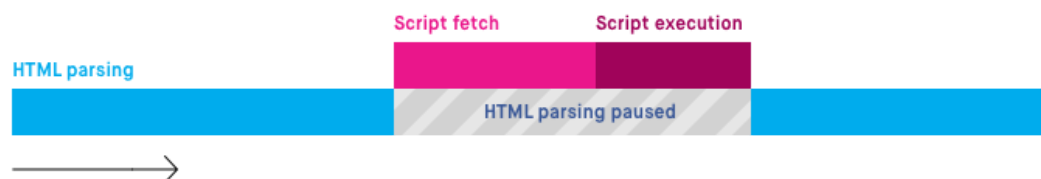
Normal Execution

Before looking into the effect of the two attributes, we must first look at what occurs in their absence. By default, as mentioned above, JavaScript files will interrupt the parsing of the HTML document in order for them to be fetched (if not inline) and executed.

Take, for example, this script element located somewhere in the middle of the page -

```
<html>
<head> ... </head>
<body>
  ...
  <script src="script.js">
  ....
</body>
</html>
```

As the document parser goes through the page, this is what occurs -



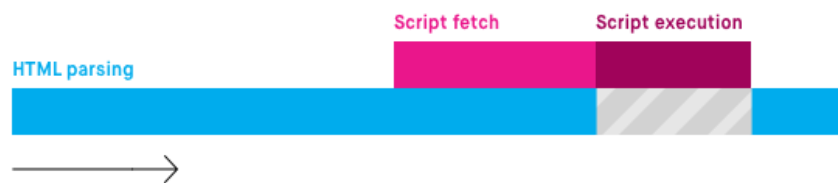
The HTML parsing is paused for the script to be fetched and executed, thereby extending the amount of time it takes to get to first paint.

The `async` Attribute

The `async` attribute is used to indicate to the browser that the script file *can* be executed asynchronously. The HTML parser does not need to pause at the point it reaches the script tag to fetch and execute, the execution can happen whenever the script becomes ready after being fetched in parallel with the document parsing.

```
<script async src="script.js">
```

This attribute is only available for externally located script files. When an external script has this attribute, the file can be downloaded while the HTML document is still parsing. Once it has been downloaded, the parsing is paused for the script to be executed.

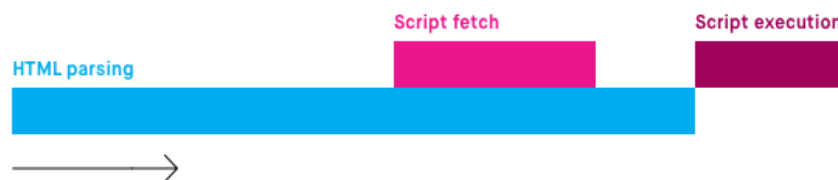


The `defer` Attribute

The `defer` attribute tells the browser to only execute the script file once the HTML document has been fully parsed.

```
<script defer src="script.js">
```

Like an asynchronously loaded script, the file can be downloaded while the HTML document is still parsing. However, even if the file is fully downloaded long before the document is finished parsing, the script is not executed until the parsing is complete.



▼ Extra Topics

- set timeout: USAGE
https://www.w3schools.com/jsref/met_win_settimeout.asp

- **Ajax** is a set of web development techniques that uses various web technologies on the client-side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behaviour of the existing page.
- **async** (default true): Setting `async` to `false` means that the statement you are calling has to complete before the next statement in your function can be called. If you set `async: true` then that statement will begin its execution and the next statement will be called regardless of whether the `async` statement has completed yet.
- The **Promise** object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.