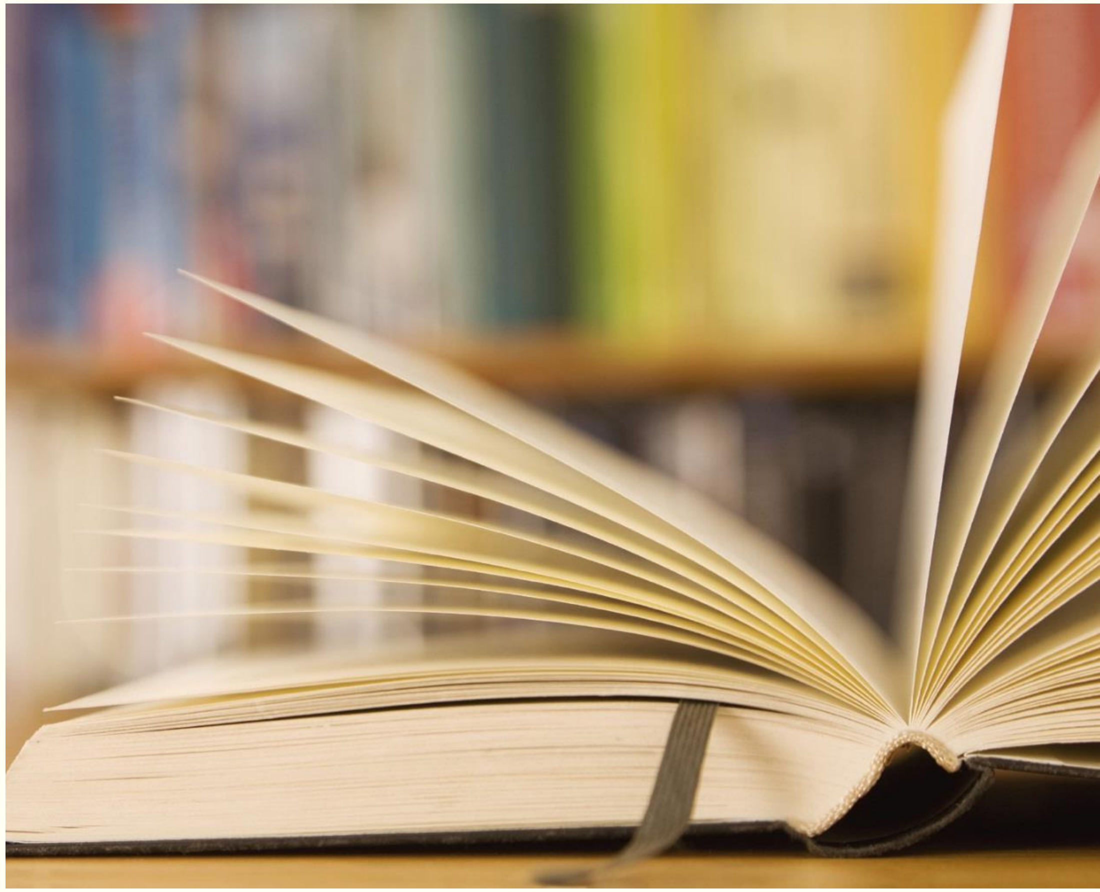
JQUERY

banuprakashc@yahoo.co.in

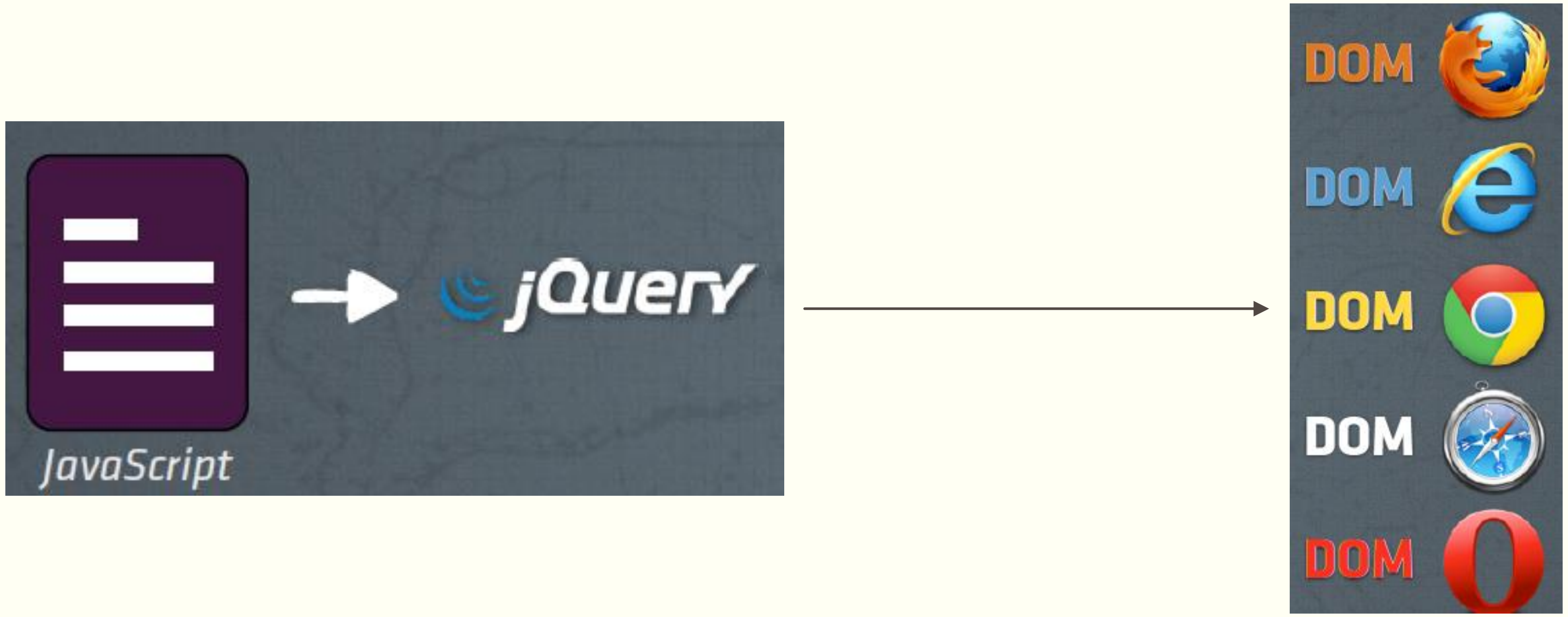


Title and Content Layout with List

- Add your first bullet point here
- Add your second bullet point here
- Add your third bullet point here

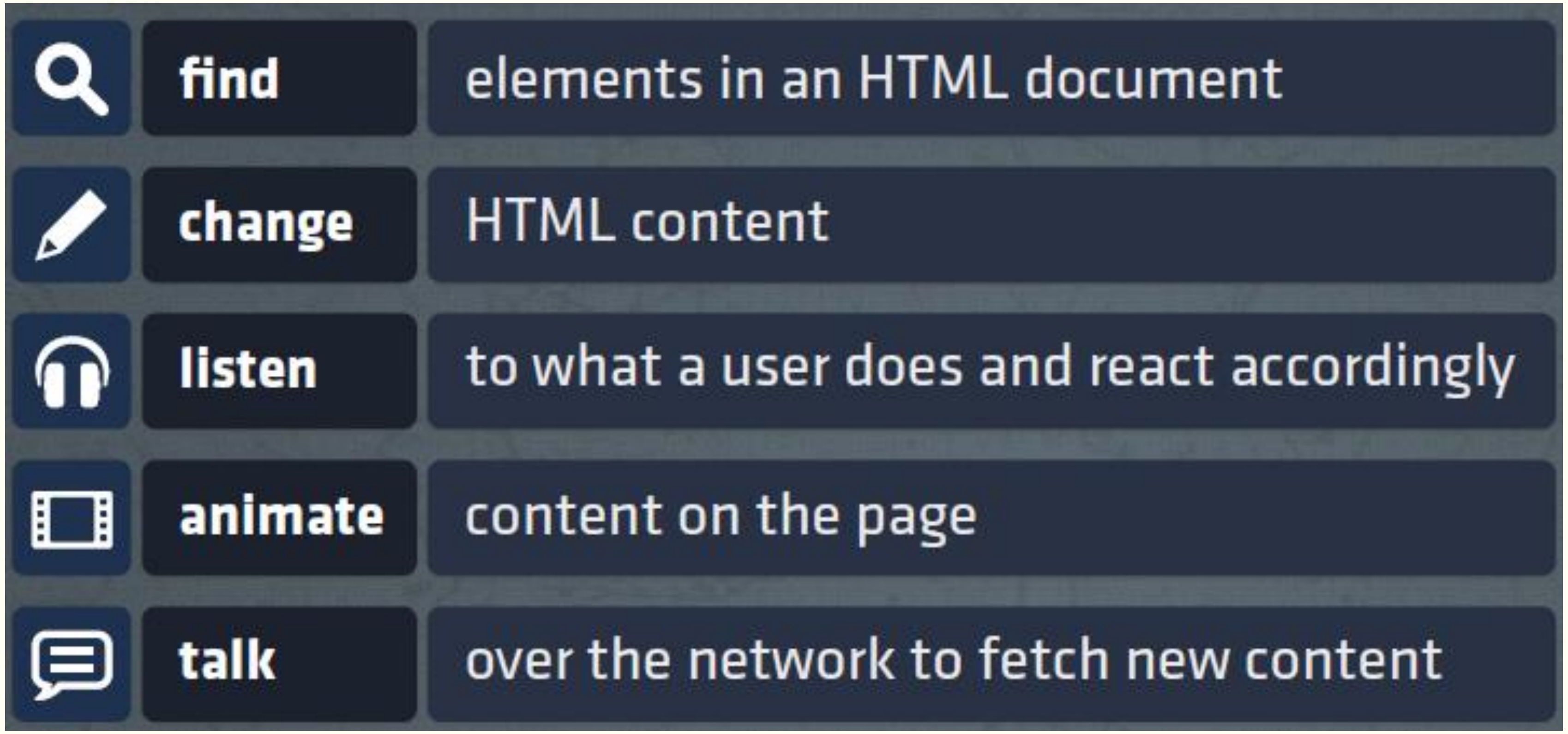
Why jQuery?

- Each Browser has a slightly different DOM interface
- If our JavaScript uses jQuery to interact with the DOM then it will work on most modern browsers.



Why jQuery?

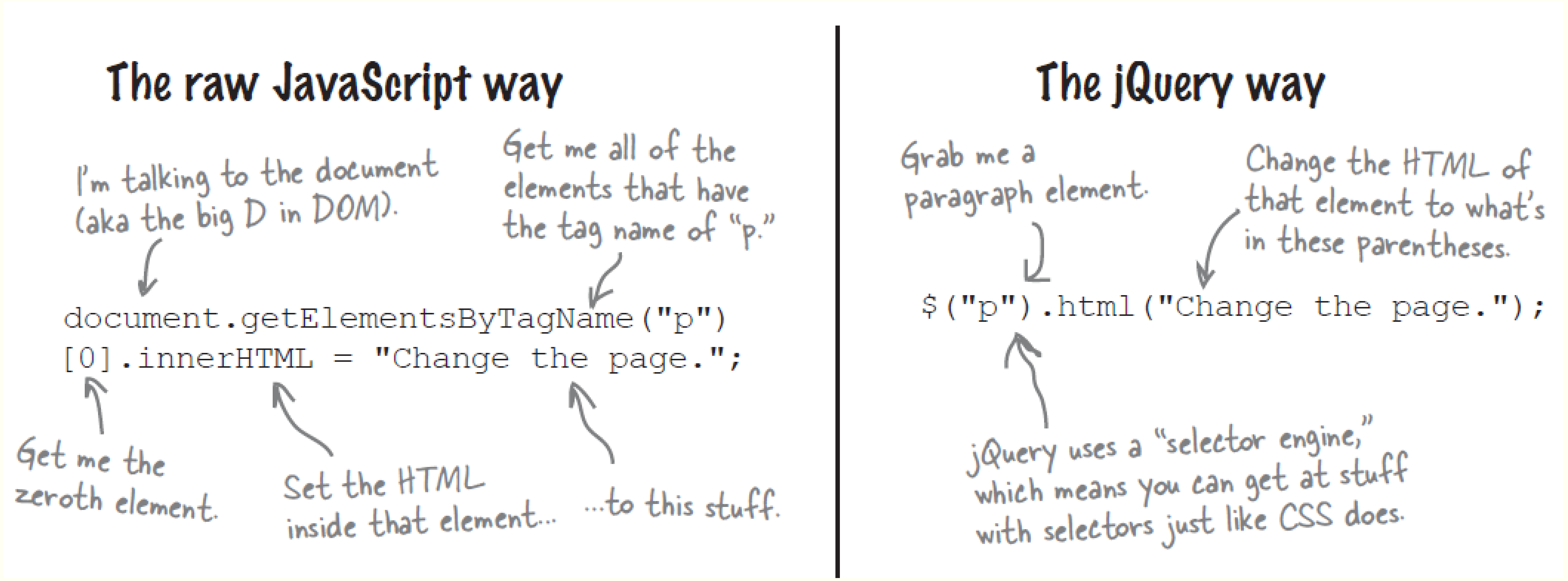
- jQuery makes it easy to



🔍	**find**	elements in an HTML document
✎	**change**	HTML content
🎧	**listen**	to what a user does and react accordingly
🎞	**animate**	content on the page
💬	**talk**	over the network to fetch new content

Why jQuery?

- jQuery makes the DOM less scary

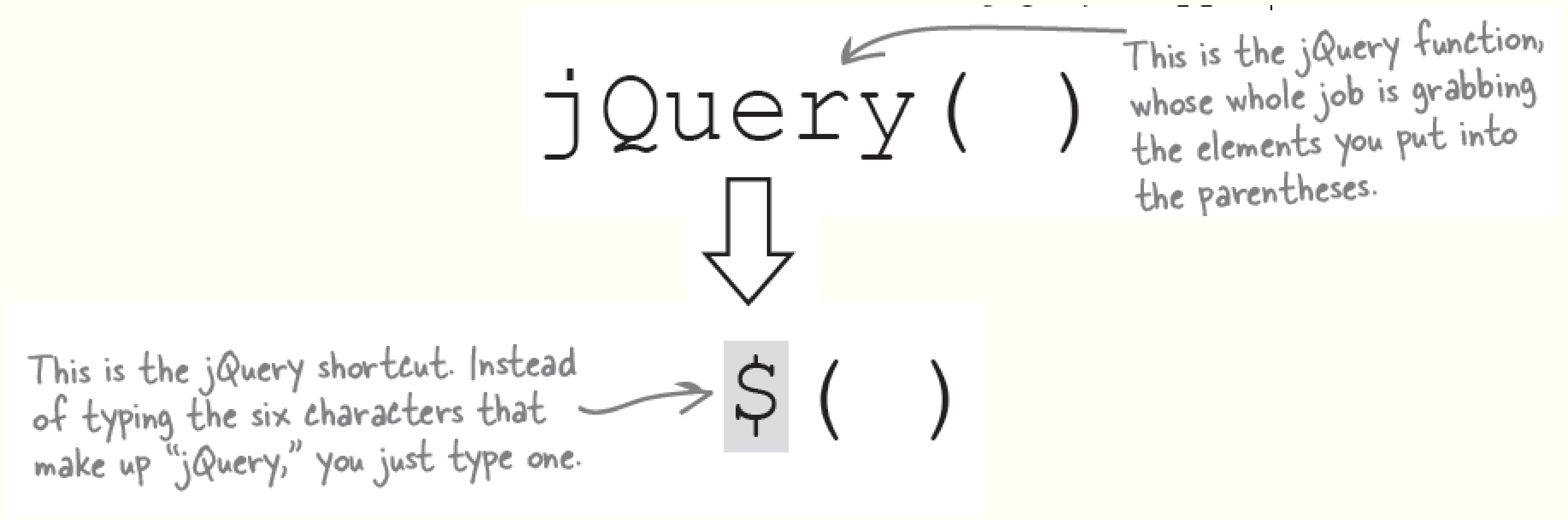

The raw JavaScript way
I'm talking to the document (aka the big D in DOM).
Get me all of the elements that have the tag name of "p."
Get me the zeroth element.
Set the HTML inside that element...
...to this stuff.
The jQuery way
Grab me a paragraph element.
Change the HTML of that element to what's in these parentheses.
jQuery uses a "selector engine," which means you can get at stuff with selectors just like CSS does.

```
document.getElementsByTagName("p")
[0].innerHTML = "Change the page.";
```

```
$("p").html("Change the page.");
```

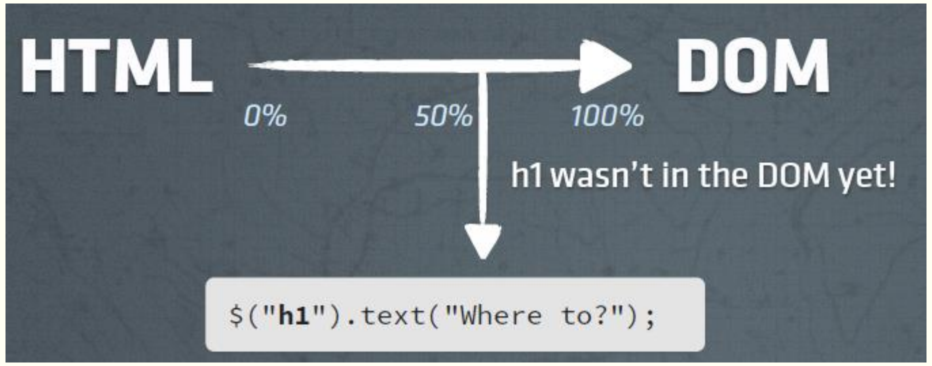
The jQuery function

- The dollar sign with the parentheses is the shorter name of the jQuery **function**.
- This shortcut saves us from writing “jQuery()” every time we want to call the jQuery function. The jQuery function is also often referred to as the jQuery **wrapper**.

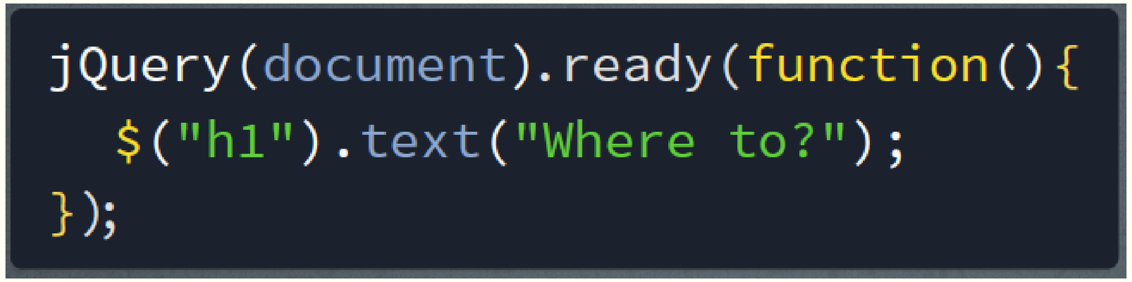

jQuery( )
This is the jQuery function, whose whole job is grabbing the elements you put into the parentheses.
This is the jQuery shortcut. Instead of typing the six characters that make up "jQuery," you just type one.
$( )

jQuery(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 jQuery(document).ready() will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute.



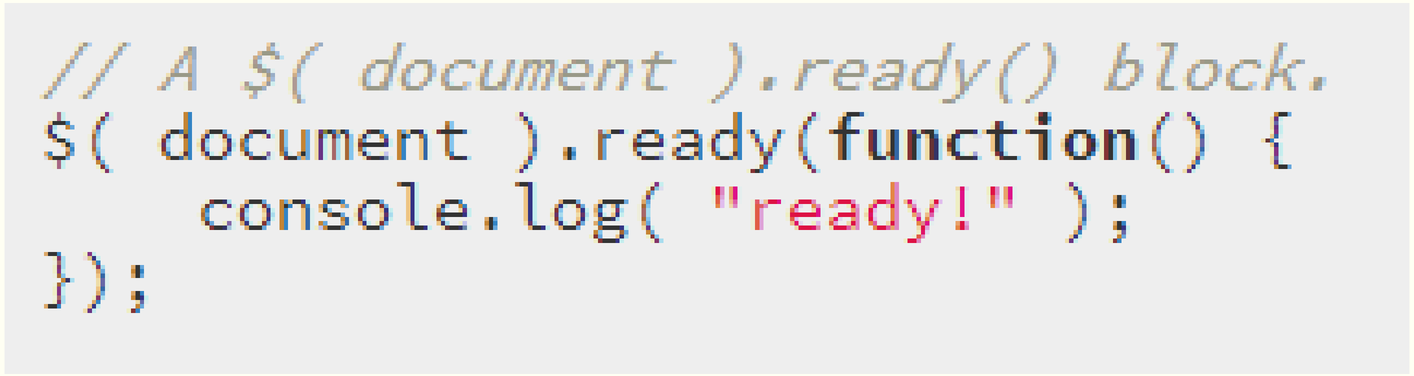
```
$("h1").text("Where to?");
```



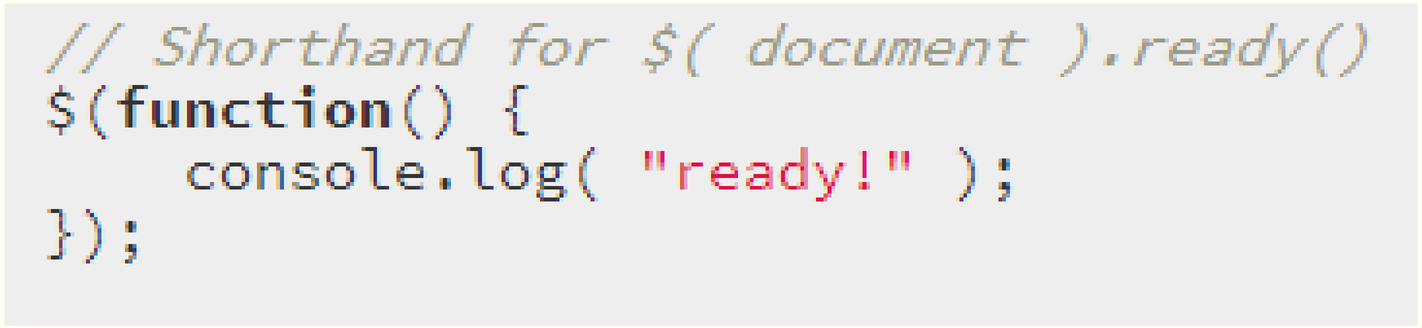
```
jQuery(document).ready(function(){
  $("h1").text("Where to?");
});
```

jQuery(document).ready()

- Other syntax



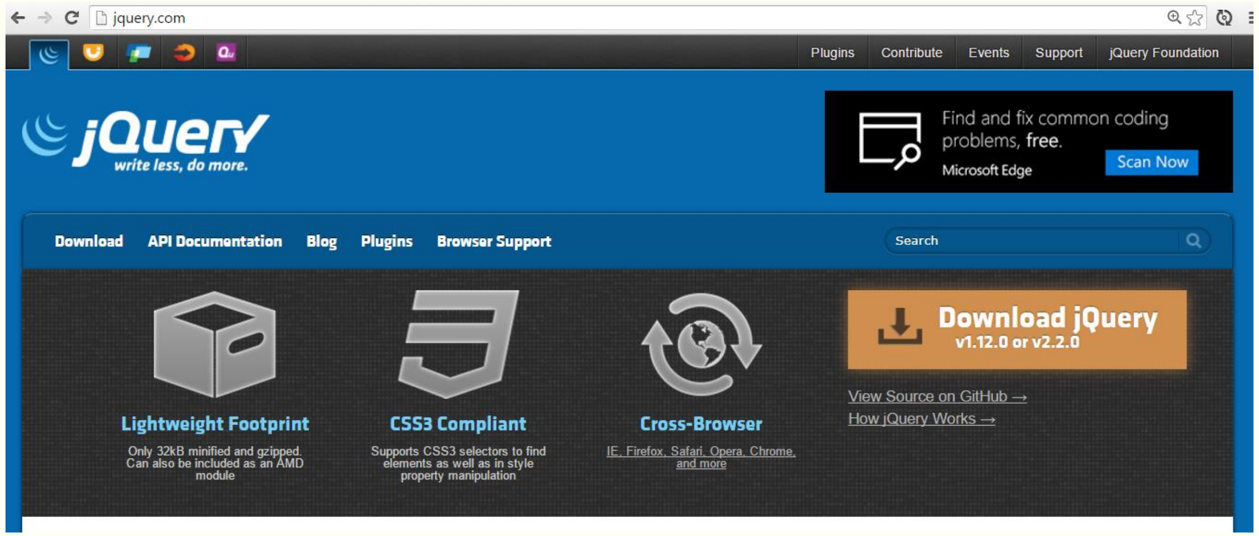
```
// A $( document ).ready() block.
$( document ).ready(function() {
    console.log( "ready!" );
});
```



```
// Shorthand for $( document ).ready()
$(function() {
    console.log( "ready!" );
});
```

Getting Started

- Download jQuery



Find and fix common coding problems, free.

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

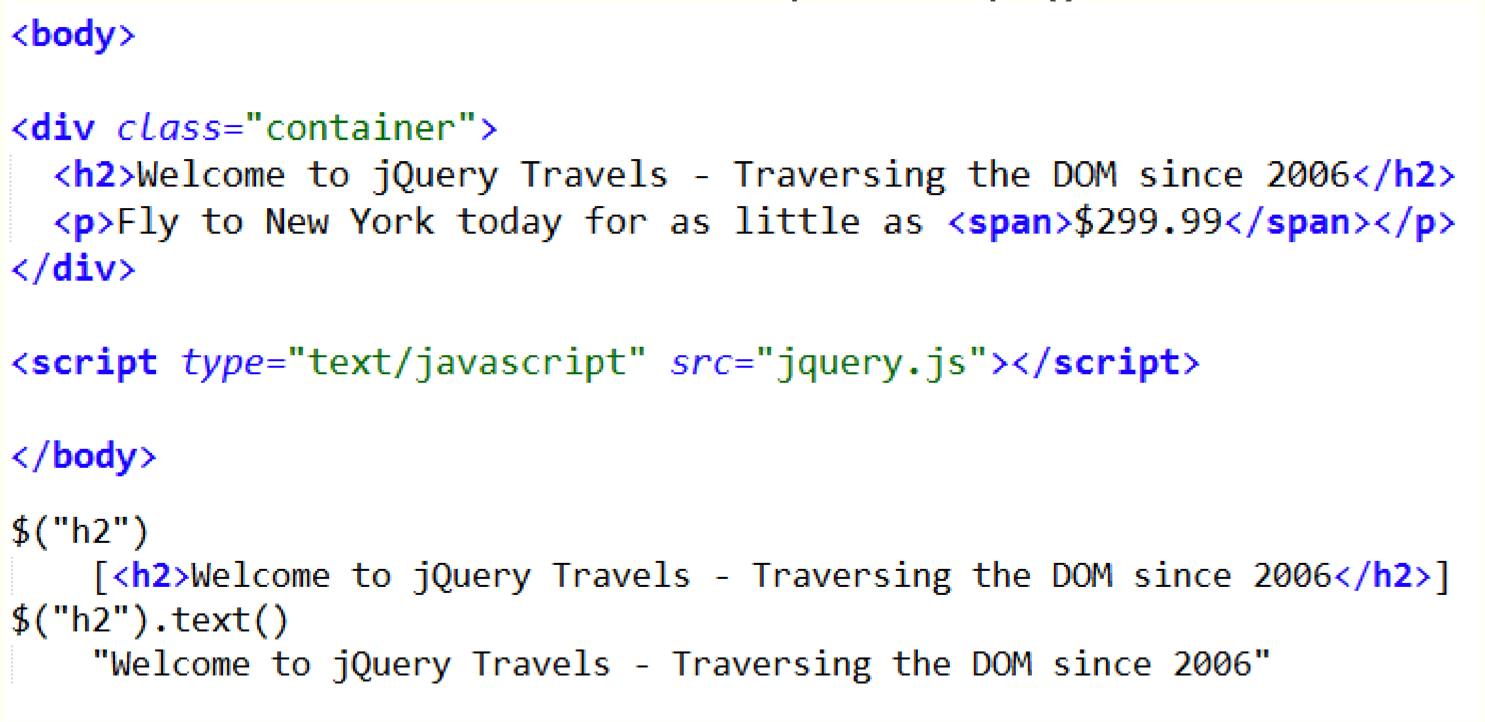
Supports CSS3 selectors to find elements as well as in style property manipulation

IE, Firefox, Safari, Opera, Chrome, and more

- Load it in your HTML document
  - <script src=“jquery.min.js”></script>
- Start using it
  - <script src=“app.js”></script>

Element Selector

- Select the h2 element of this simple web page.



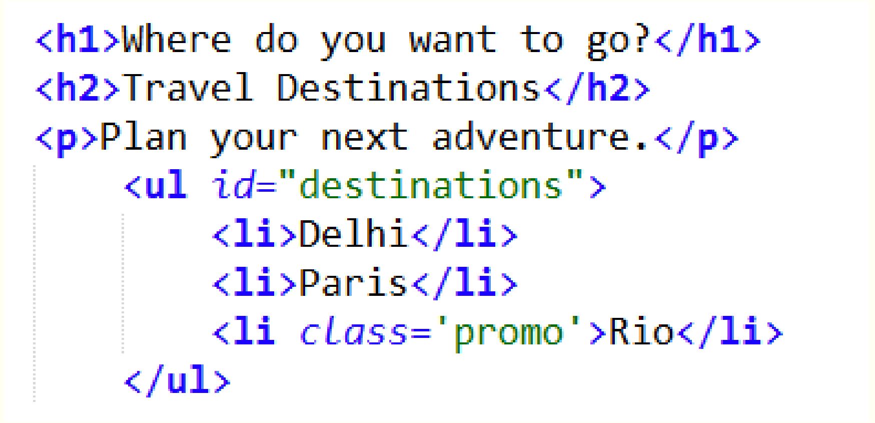
```
<body>

<div class="container">
  <h2>Welcome to jQuery Travels - Traversing the DOM since 2006</h2>
  <p>Fly to New York today for as little as <span>$299.99</span></p>
</div>

<script type="text/javascript" src="jquery.js"></script>

</body>
```

```
$("h2")
    [<h2>Welcome to jQuery Travels - Traversing the DOM since 2006</h2>]
$("h2").text()
    "Welcome to jQuery Travels - Traversing the DOM since 2006"
```

- change the price to *$100*.
  - $("span").text("$100");

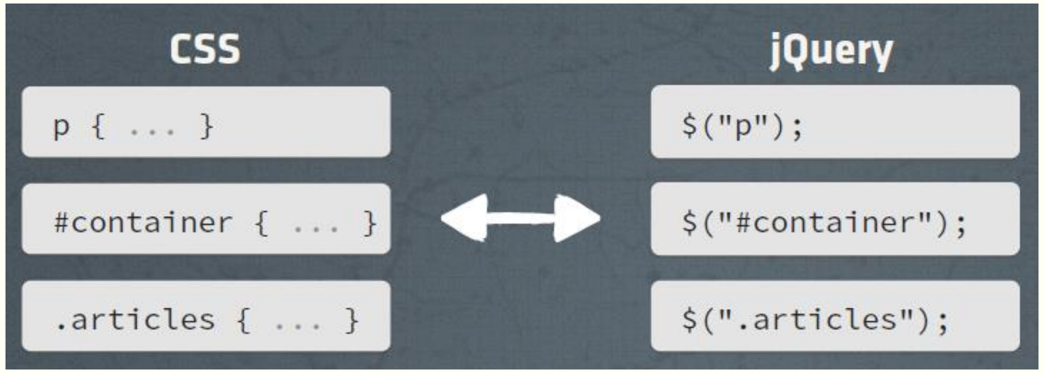
Selecting multiple elements

- $("li")



```
<h1>Where do you want to go?</h1>
<h2>Travel Destinations</h2>
<p>Plan your next adventure.</p>
    <ul id="destinations">
        <li>Delhi</li>
        <li>Paris</li>
        <li class='promo'>Rio</li>
    </ul>
```

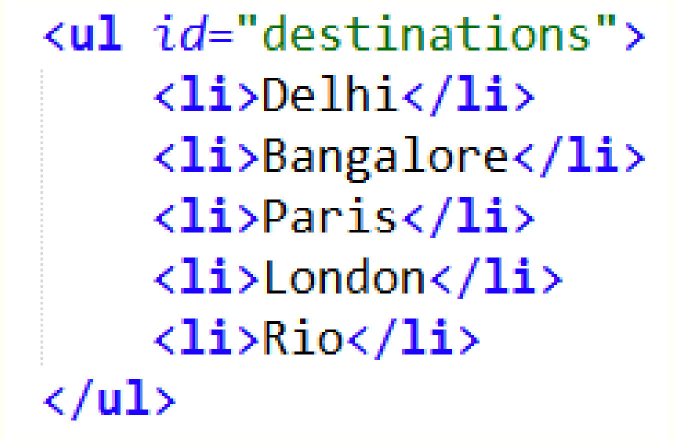
- We can find elements by ID or Class



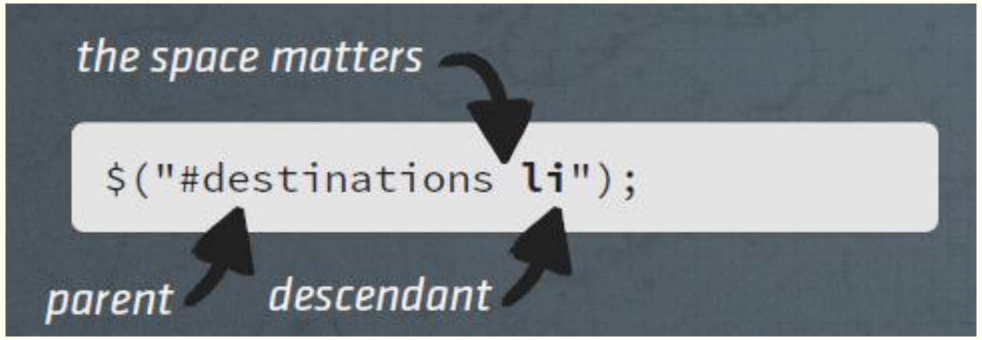
CSS	jQuery
`p { ... }`	`$("p");`
`#container { ... }`	`$("#container");`
`.articles { ... }`	`$(".articles");`

Selecting descendants

- How do we find only the <li> elements that are children of the “destinations” <ul>?



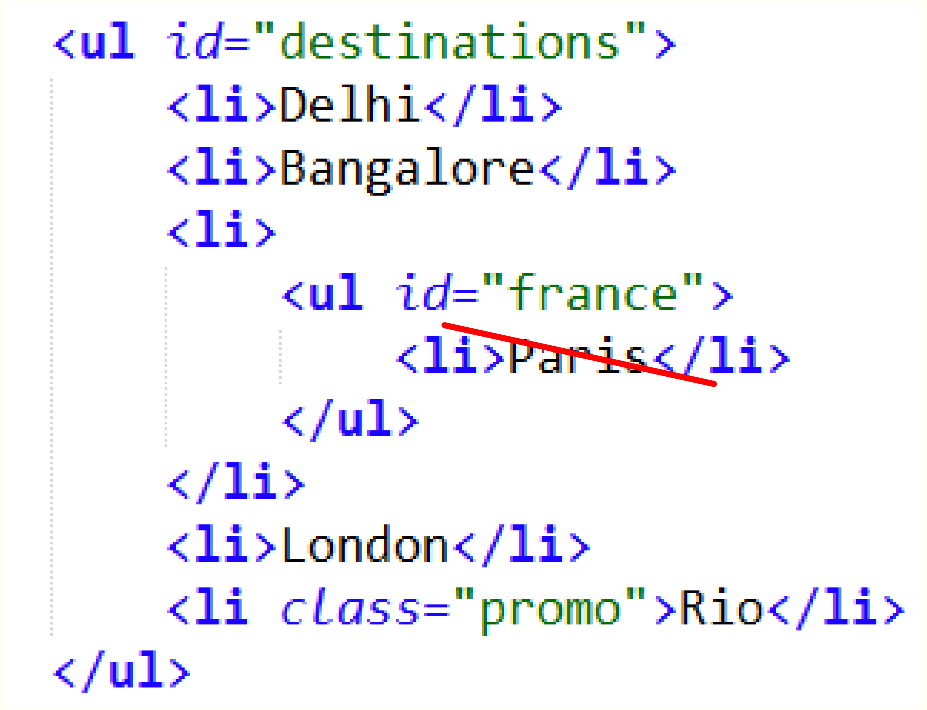
```
<ul id="destinations">
    <li>Delhi</li>
    <li>Bangalore</li>
    <li>Paris</li>
    <li>London</li>
    <li>Rio</li>
</ul>
```



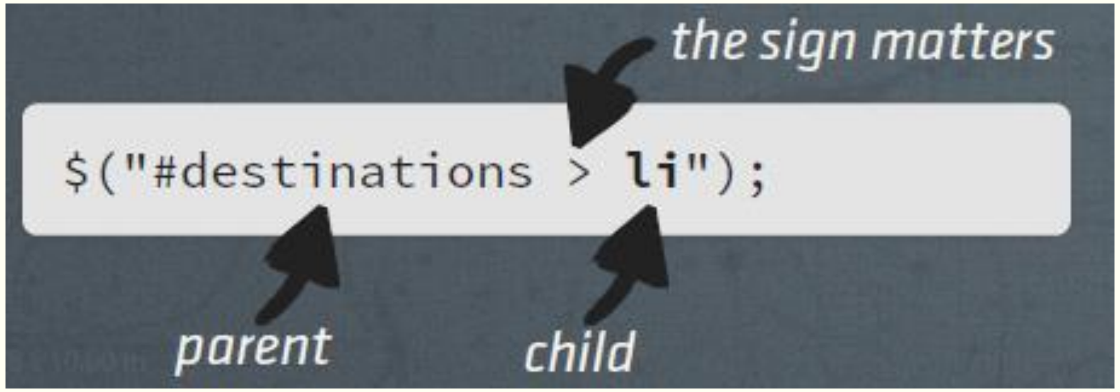
```
$("#destinations li");
```

Selecting direct children

- How do we find only the <li> elements that are children of the “destinations” <ul>?

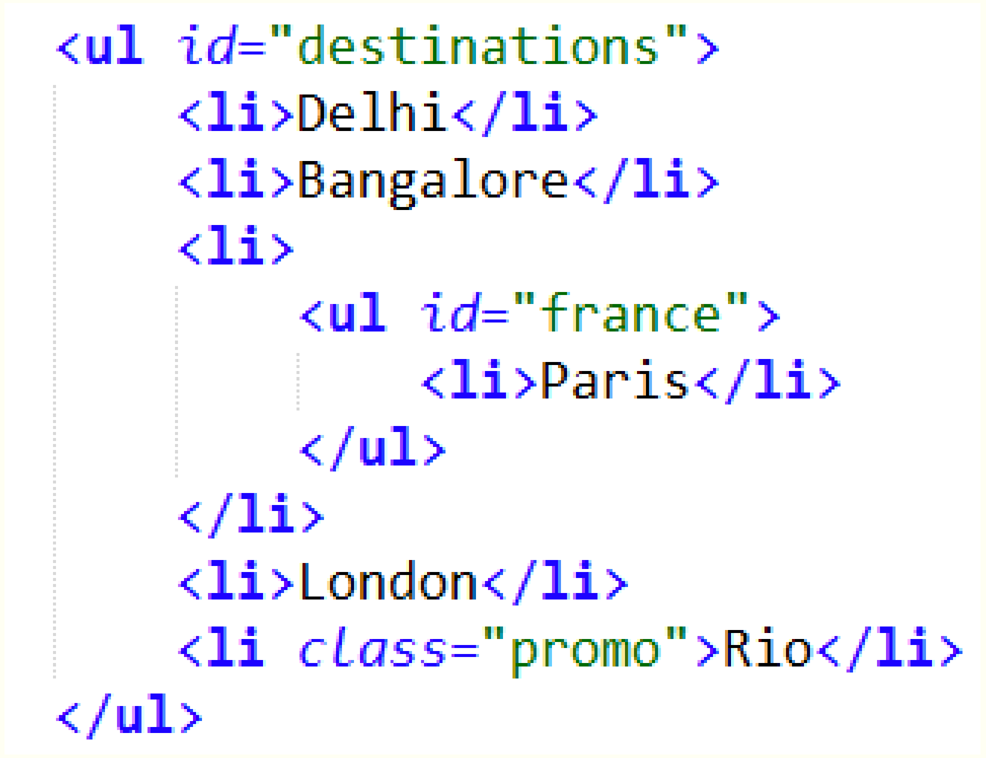


```
<ul id="destinations">
    <li>Delhi</li>
    <li>Bangalore</li>
    <li>
        <ul id="france">
            ~~<li>Paris</li>~~
        </ul>
    </li>
    <li>London</li>
    <li class="promo">Rio</li>
</ul>
```

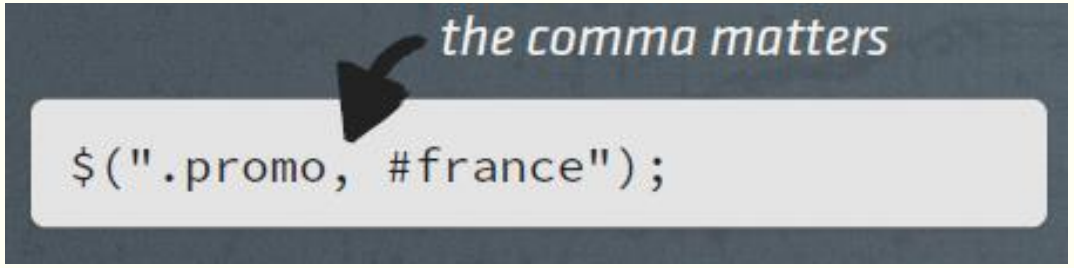


```
$("#destinations > li");
```

Selecting multiple elements

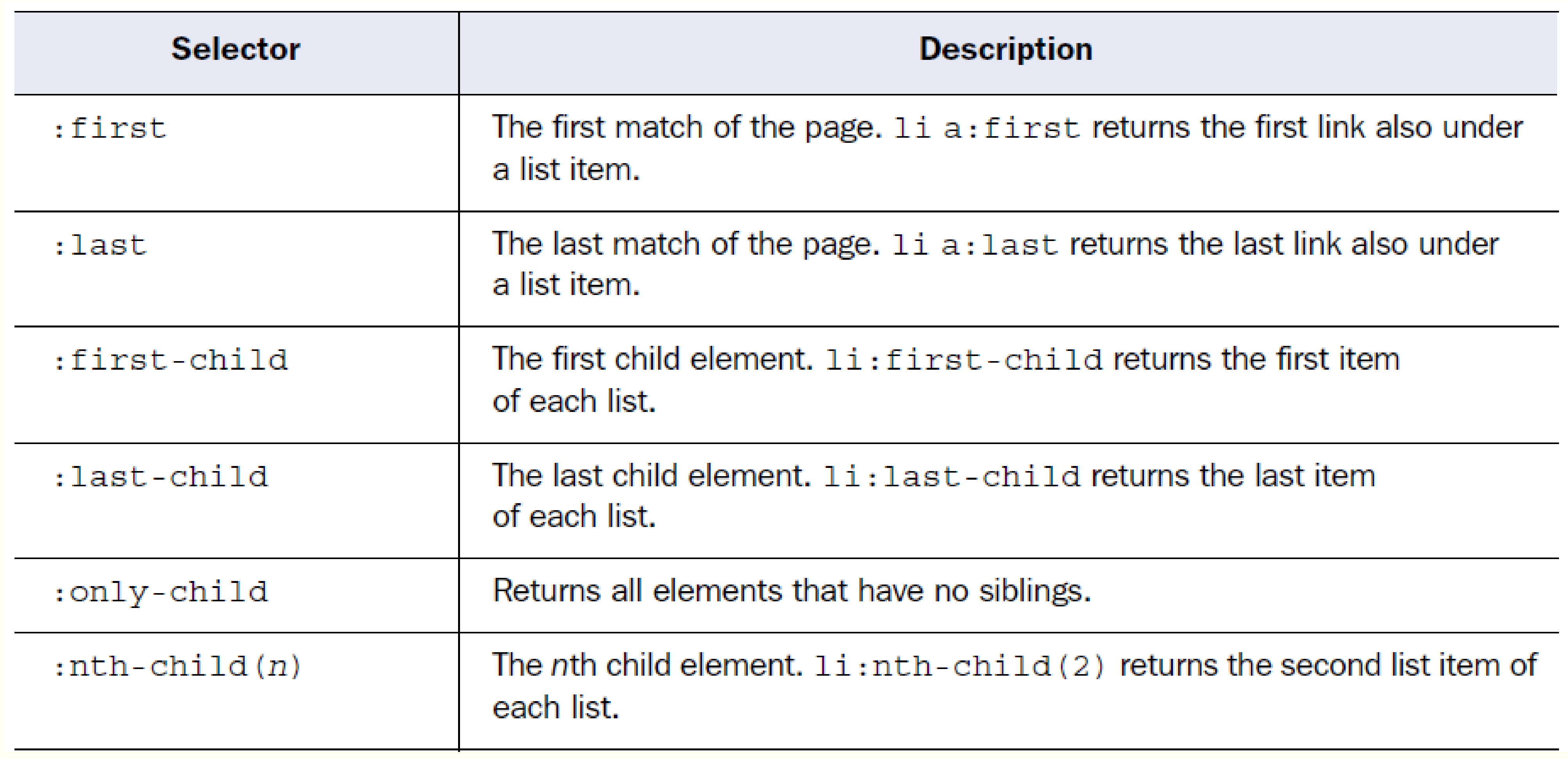


```
<ul id="destinations">
    <li>Delhi</li>
    <li>Bangalore</li>
    <li>
        <ul id="france">
            <li>Paris</li>
        </ul>
    </li>
    <li>London</li>
    <li class="promo">Rio</li>
</ul>
```



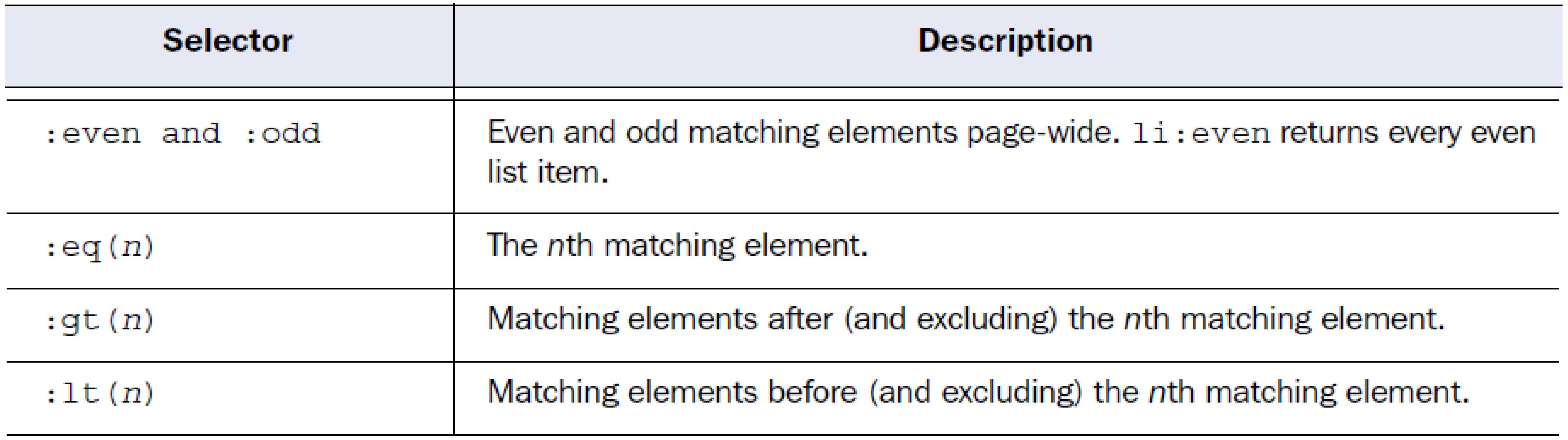
```
$(".promo, #france");
```

jQuery Filters



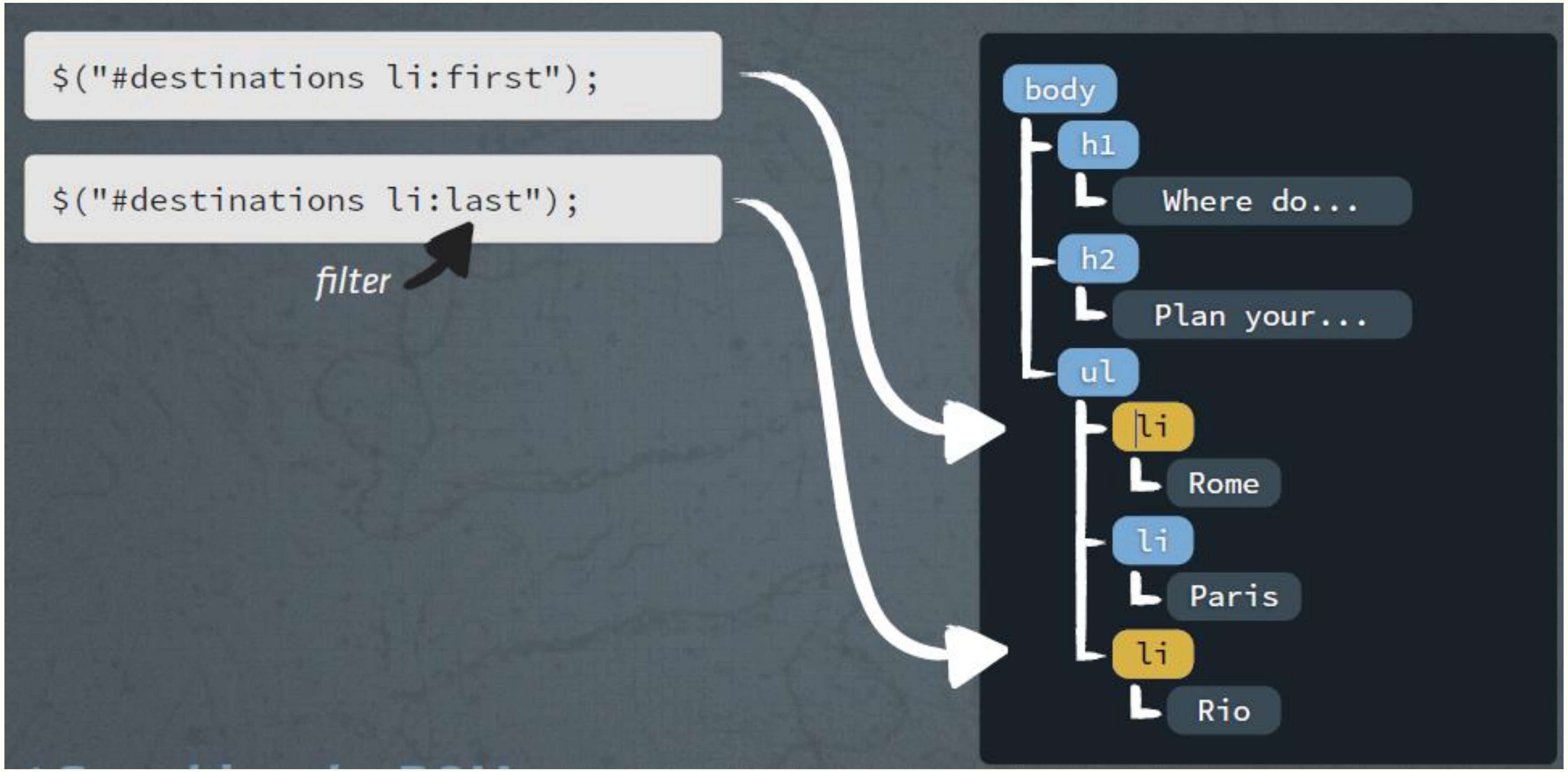
Selector	Description
`:first`	The first match of the page. `li a:first` returns the first link also under a list item.
`:last`	The last match of the page. `li a:last` returns the last link also under a list item.
`:first-child`	The first child element. `li:first-child` returns the first item of each list.
`:last-child`	The last child element. `li:last-child` returns the last item of each list.
`:only-child`	Returns all elements that have no siblings.
`:nth-child(`*n*`)`	The *n*th child element. `li:nth-child(2)` returns the second list item of each list.

jQuery Filters



Selector	Description
`:even and :odd`	Even and odd matching elements page-wide. `li:even` returns every even list item.
`:eq(`*n*`)`	The *n*th matching element.
`:gt(`*n*`)`	Matching elements after (and excluding) the *n*th matching element.
`:lt(`*n*`)`	Matching elements before (and excluding) the *n*th matching element.

jQuery Filters

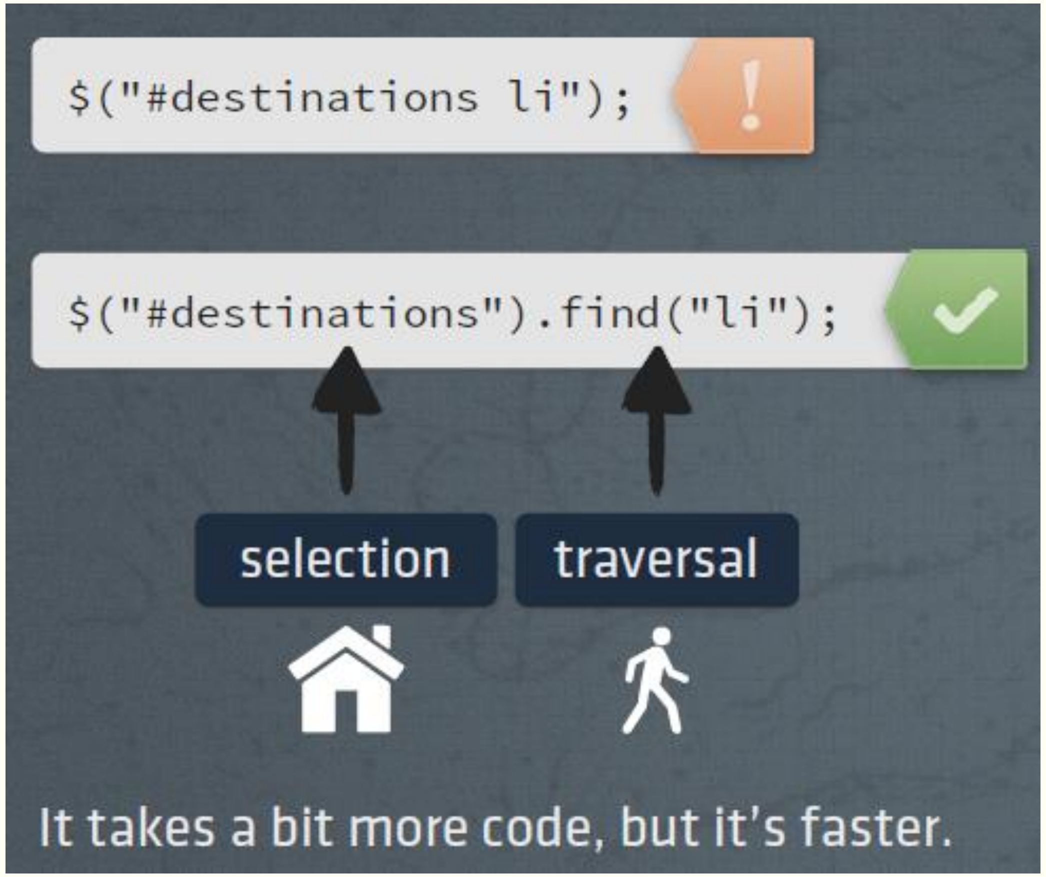


```
$("#destinations li:first");
```

```
$("#destinations li:last");
```

DOM Traversing

- Filtering by traversing



```
$("#destinations li");
```

```
$("#destinations").find("li");
```

It takes a bit more code, but it's faster.

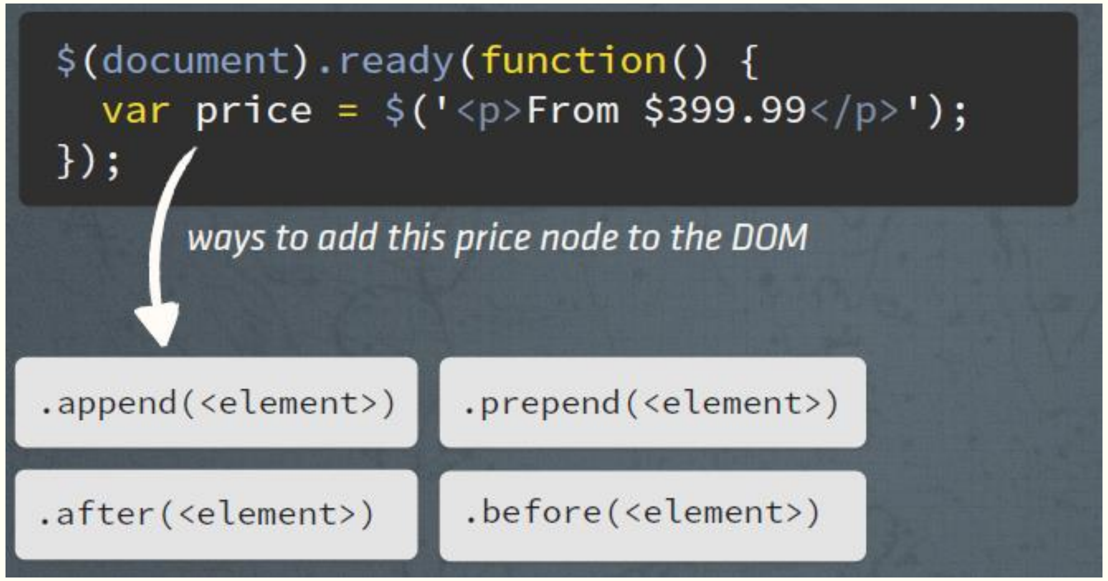
DOM Traversing

- $("li:first"); can be done as $("li").first();
- $("li:last"); can be done as $("li").last();

- Walking up the DOM
  - $("li").first();
  - $("li").first().next();
  - $("li").first().next().prev();
  - $("li").first().parent();
  - $("#destinations").children("li");
    - children(), unlike find(), only selects direct children

Working with the DOM

- Appending to the DOM



```
$(document).ready(function() {
  var price = $('<p>From $399.99</p>');
});
```

- $('.vacation').before(price);
  - *Puts the price node before* ***.vacation***
- $('.vacation').after(price);
  - *Puts the price node after* ***.vacation***
- $('.vacation').prepend(price);
  - *Puts the price node at the top of* ***.vacation***
- $('.vacation').append(price);
  - *Puts the price node at the bottom of* ***.vacation***

Working with the DOM

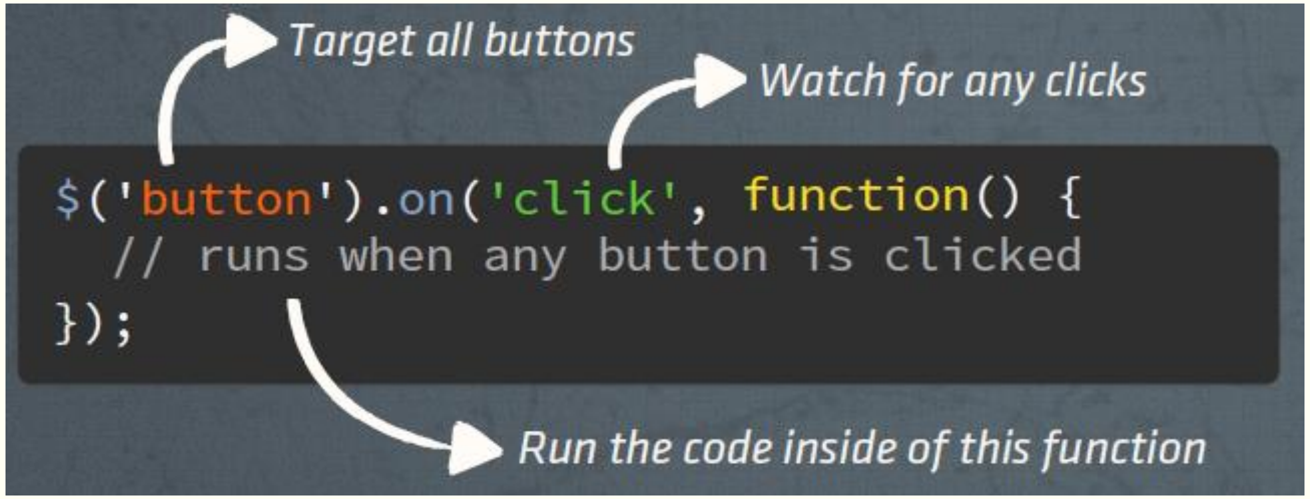
- Removing from the DOM
  - $('button').remove();
    - *Removes the* ***<button>*** *from the DOM*

Appending to the DOM

- var price = $('<p>Starting from $399.99</p>');
  - price.appendTo($('.vacation'));
  - Methods:
    - .appendTo(<element>)
    - .prependTo(<element>)
    - .insertBefore(<element>)
    - .insertAfter(<element>)

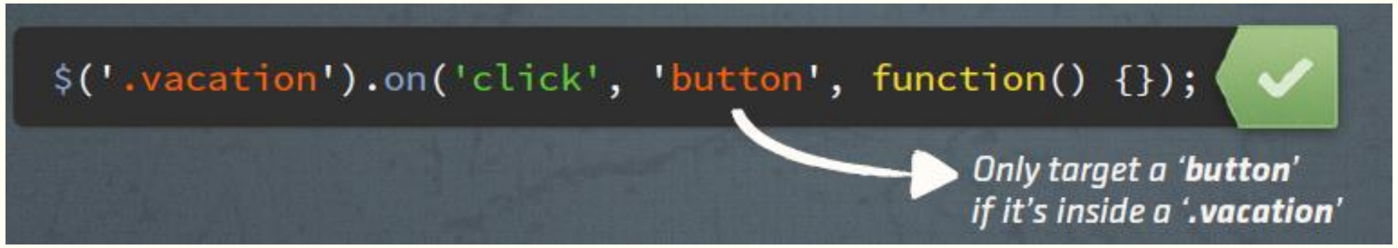
jQuery Interactions [Events]

- jQuery Object methods
  - .on(**<event>**, **<event handler>**)



```
$('button').on('click', function() {
  // runs when any button is clicked
});
```

  - On With a Selector



```
$('.vacation').on('click', 'button', function() {});
```

jQuery Interactions [Events]

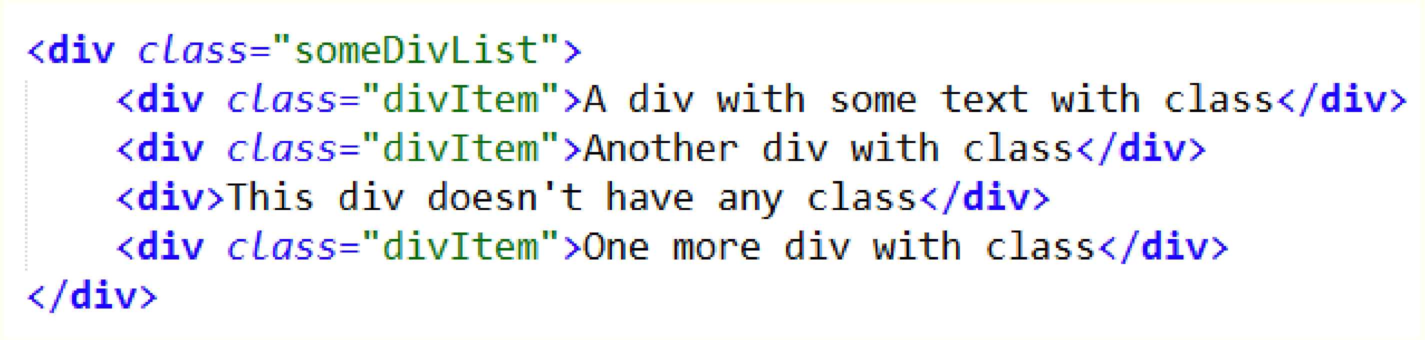
- MouseEvents
  - click
  - dblclick
  - focusin
  - focusout
  - mousedown
  - mouseup
  - mousemove
  - mouseout
  - mouseover
  - mouseleave
  - Mouseenter
- KeyboardEvents
  - keypress
  - keydown
  - keyup

- Form events
  - blur
  - focus
  - select
  - change
  - submit

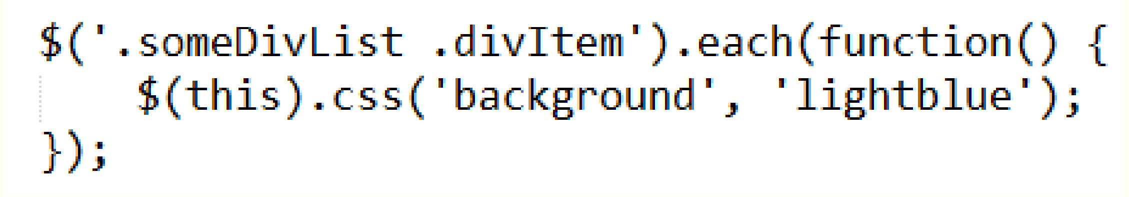
jQuery's this

What is "this"?

- In many object-oriented programming languages, this (or self) is a keyword which can be used in instance methods to refer to the object on which the currently executing method has been invoked.



```html
<div class="someDivList">
    <div class="divItem">A div with some text with class</div>
    <div class="divItem">Another div with class</div>
    <div>This div doesn't have any class</div>
    <div class="divItem">One more div with class</div>
</div>
```



```javascript
$('.someDivList .divItem').each(function() {
    $(this).css('background', 'lightblue');
});
```

jQuery's this

```css
.someLinksList a {
    display: block;
    background: #d5d5d5;
    color: #000;
    text-decoration: none;
    padding: 5px;
    margin-bottom: 5px;
}

.someLinksList a.hover {
    color: #FFF;
    background: green;
}
```

```html
<div class="someLinksList">
            <a href="#">A Simple Link</a>
            <a href="#">Another Simple Link</a>
            <a href="#">One More Simple Link</a>
</div>
```

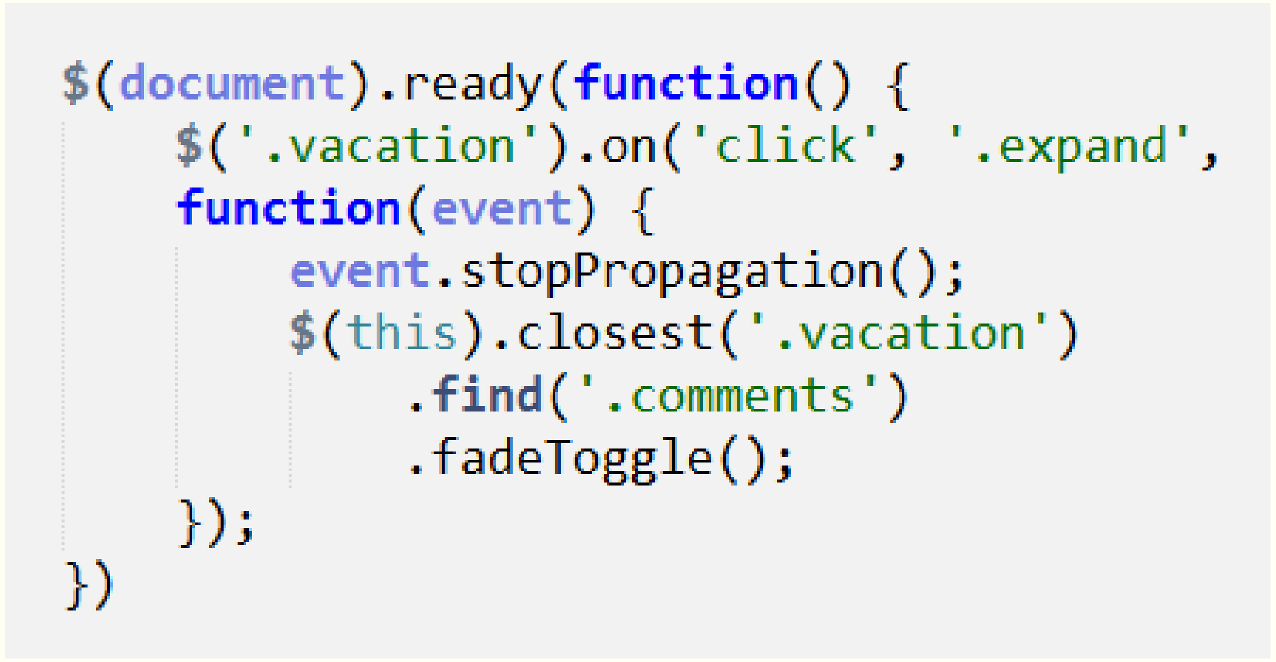
```javascript
$('.someLinksList > a').hover(function() {
    $(this).toggleClass('hover');
});
```

Using .closest(<selector>)

- **.closest()**
  - For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.
  - **$(this).closest('.vacation').append(price);**

Event object stopPropagation()

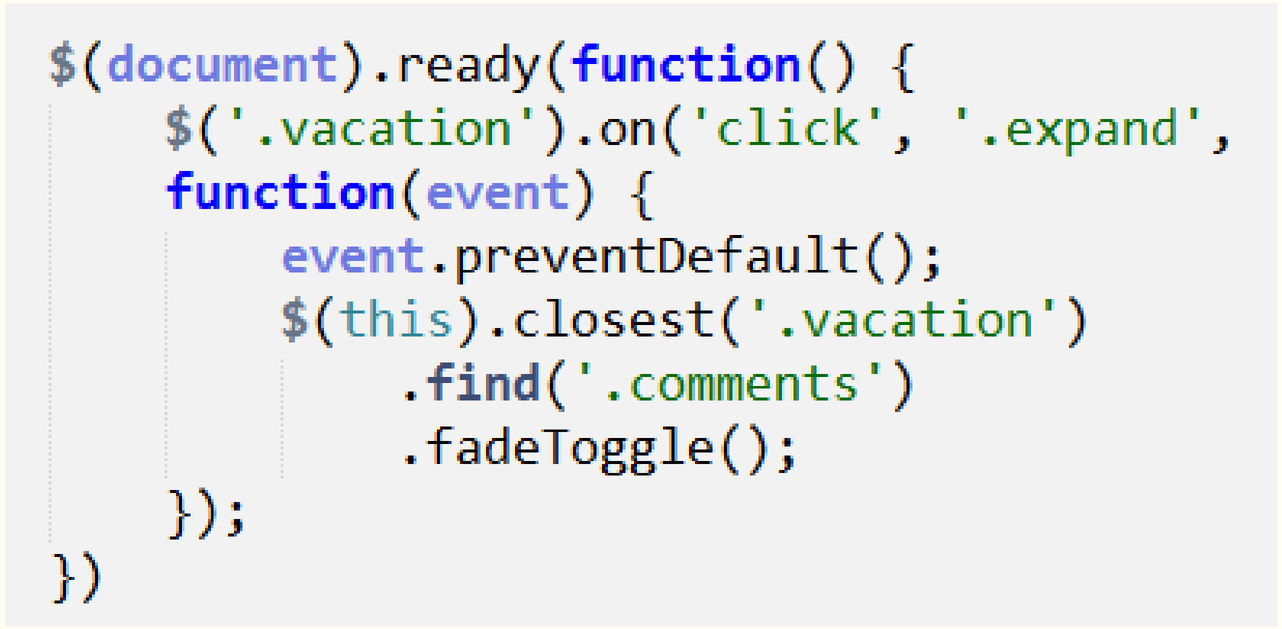
- *The browser will still handle the click event but will prevent it from “bubbling up” to each parent node.*



```
$(document).ready(function() {
    $('.vacation').on('click', '.expand',
    function(event) {
        event.stopPropagation();
        $(this).closest('.vacation')
            .find('.comments')
            .fadeToggle();
    });
})
```

Event object preventDefault()

- *The click event will “bubble up” but the browser won’t handle it*



```
$(document).ready(function() {
    $('.vacation').on('click', '.expand',
    function(event) {
        event.preventDefault();
        $(this).closest('.vacation')
            .find('.comments')
            .fadeToggle();
    });
})
```

jQuery CSS

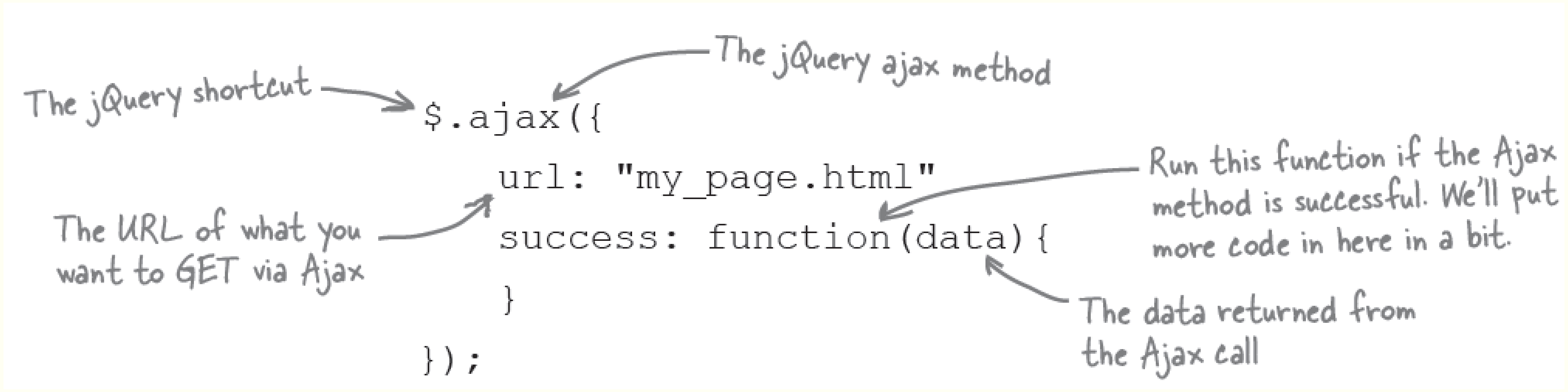
- jQuery Object Methods for CSS
  - .css(<attr>, <value>)
  - .css(<attr>)
  - .css(<object>)
  - .addClass(<class>)
  - .toggleClass(<class>)
  - .removeClass(<class>)
  - .hasClass(<class>)
  - **Examples:**
    - $(this).css({'background-color': '#252b30', 'border-color': '1px solid #967'});
    - $(this).addClass('highlighted');
    - $(this).toggleClass('highlighted');
    - $(this).hasClass('highlighted') //Returns true or false

jQuery Effects

- **.animate()**
  - Perform a custom animation of a set of CSS properties.
  - Example:
    - $(this).animate({'top': '-10px'});
- **.fadeIn()**
  - Display the matched elements by fading them to opaque.
- **.fadeOut()**
  - Hide the matched elements by fading them to transparent.
- **.hide()**
  - Hide the matched elements.
- **.show()**
  - Display the matched elements.

.slideDown()
Display the matched elements with a sliding motion.
.slideUp()
Hide the matched elements with a sliding motion.
.slideToggle()
Display or hide the matched elements with a sliding motion.

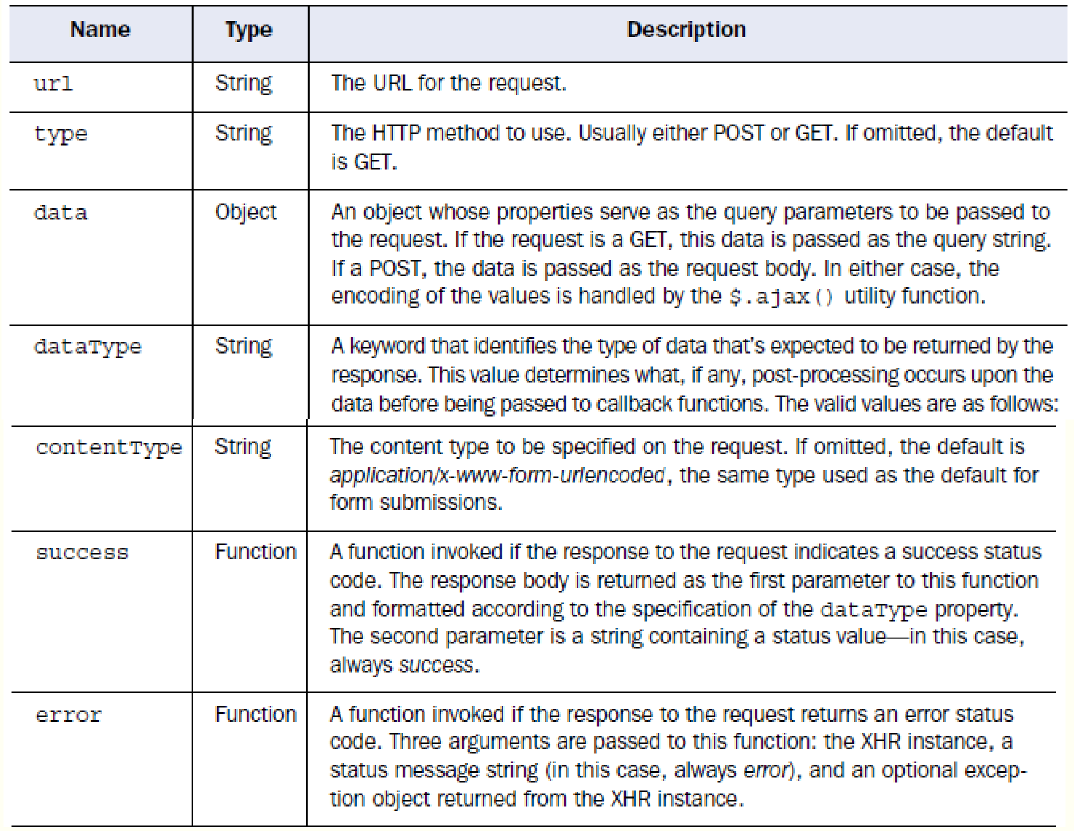
AJAX and jQuery



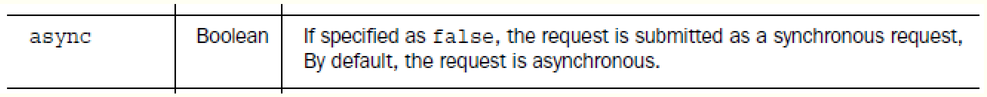
```
$.ajax({
    url: "my_page.html"
    success: function(data){
    }
});
```

The jQuery shortcut
The jQuery ajax method
Run this function if the Ajax method is successful. We'll put more code in here in a bit.
The URL of what you want to GET via Ajax
The data returned from the Ajax call

Options for the **$.ajax()** utility function

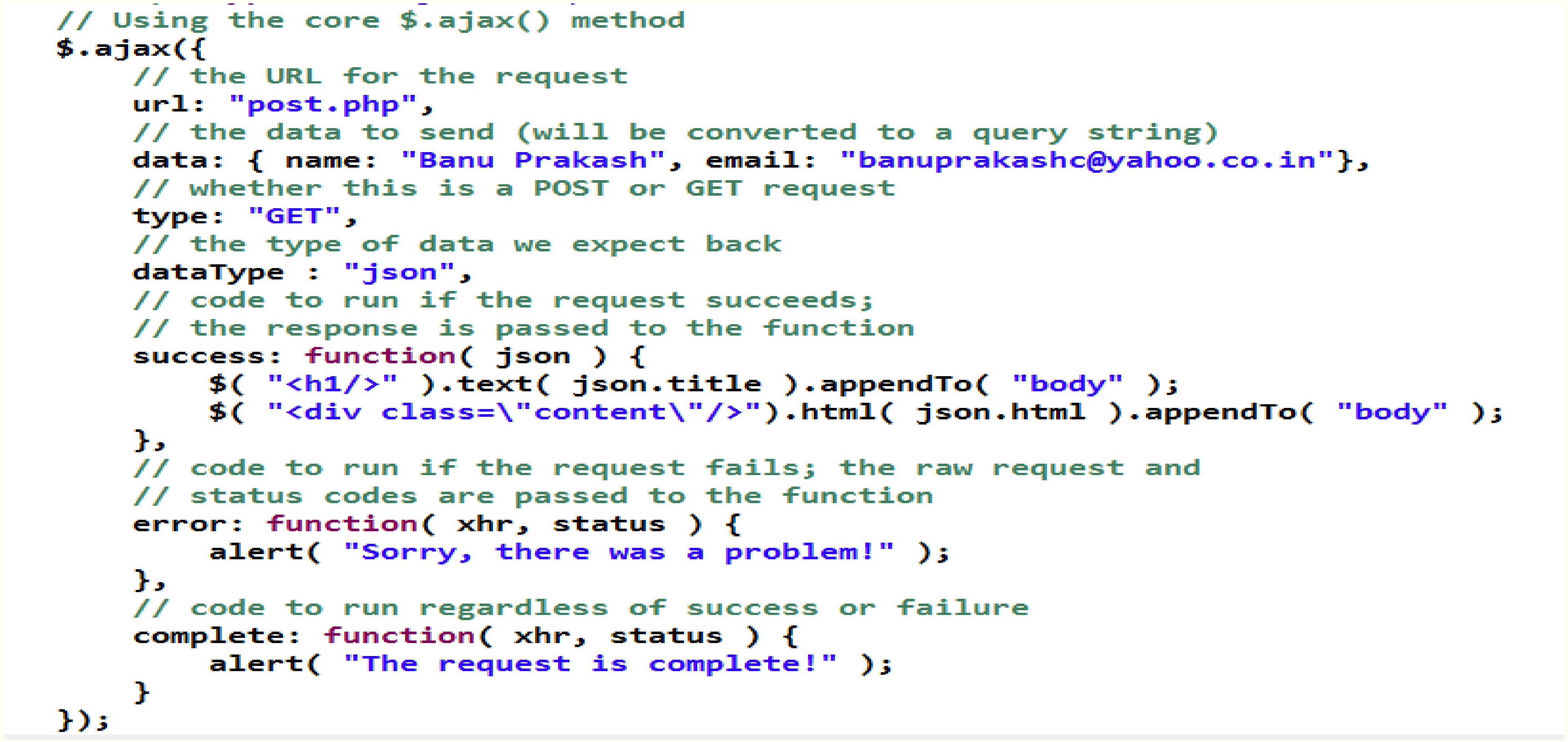


Name	Type	Description
`url`	String	The URL for the request.
`type`	String	The HTTP method to use. Usually either POST or GET. If omitted, the default is GET.
`data`	Object	An object whose properties serve as the query parameters to be passed to the request. If the request is a GET, this data is passed as the query string. If a POST, the data is passed as the request body. In either case, the encoding of the values is handled by the `$.ajax()` utility function.
`dataType`	String	A keyword that identifies the type of data that's expected to be returned by the response. This value determines what, if any, post-processing occurs upon the data before being passed to callback functions. The valid values are as follows:
`contentType`	String	The content type to be specified on the request. If omitted, the default is *application/x-www-form-urlencoded*, the same type used as the default for form submissions.
`success`	Function	A function invoked if the response to the request indicates a success status code. The response body is returned as the first parameter to this function and formatted according to the specification of the `dataType` property. The second parameter is a string containing a status value—in this case, always *success*.
`error`	Function	A function invoked if the response to the request returns an error status code. Three arguments are passed to this function: the XHR instance, a status message string (in this case, always *error*), and an optional exception object returned from the XHR instance.



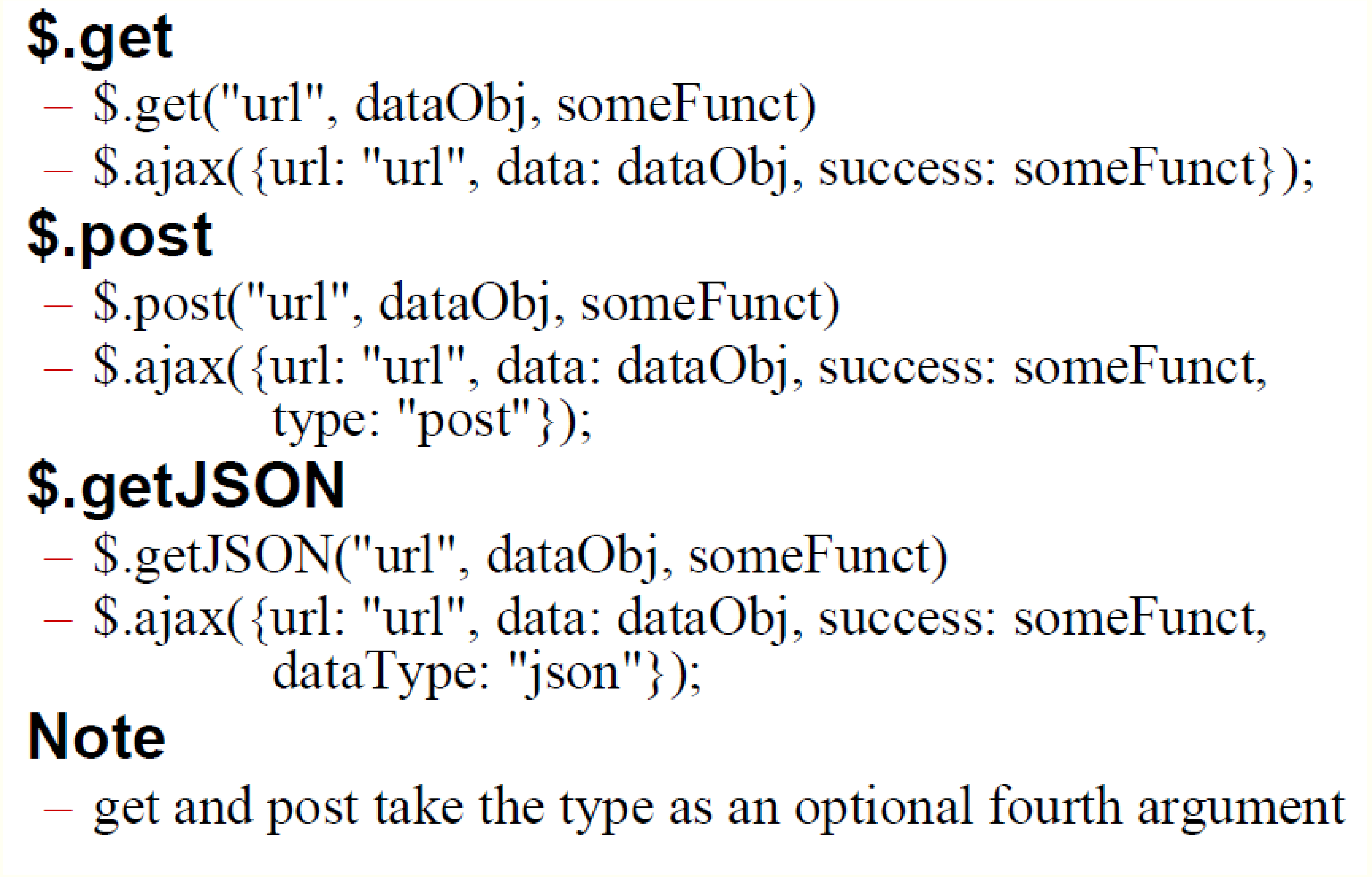
`async`	Boolean	If specified as `false`, the request is submitted as a synchronous request, By default, the request is asynchronous.

The ajax() options



```
// Using the core $.ajax() method
$.ajax({
    // the URL for the request
    url: "post.php",
    // the data to send (will be converted to a query string)
    data: { name: "Banu Prakash", email: "banuprakashc@yahoo.co.in"},
    // whether this is a POST or GET request
    type: "GET",
    // the type of data we expect back
    dataType : "json",
    // code to run if the request succeeds;
    // the response is passed to the function
    success: function( json ) {
        $( "<h1/>" ).text( json.title ).appendTo( "body" );
        $( "<div class=\"content\"/>").html( json.html ).appendTo( "body" );
    },
    // code to run if the request fails; the raw request and
    // status codes are passed to the function
    error: function( xhr, status ) {
        alert( "Sorry, there was a problem!" );
    },
    // code to run regardless of success or failure
    complete: function( xhr, status ) {
        alert( "The request is complete!" );
    }
});
```

Shortcuts for AJAX



$.get

- $.get("url", dataObj, someFunct)
- $.ajax({url: "url", data: dataObj, success: someFunct});

$.post

- $.post("url", dataObj, someFunct)
- $.ajax({url: "url", data: dataObj, success: someFunct,
  type: "post"});

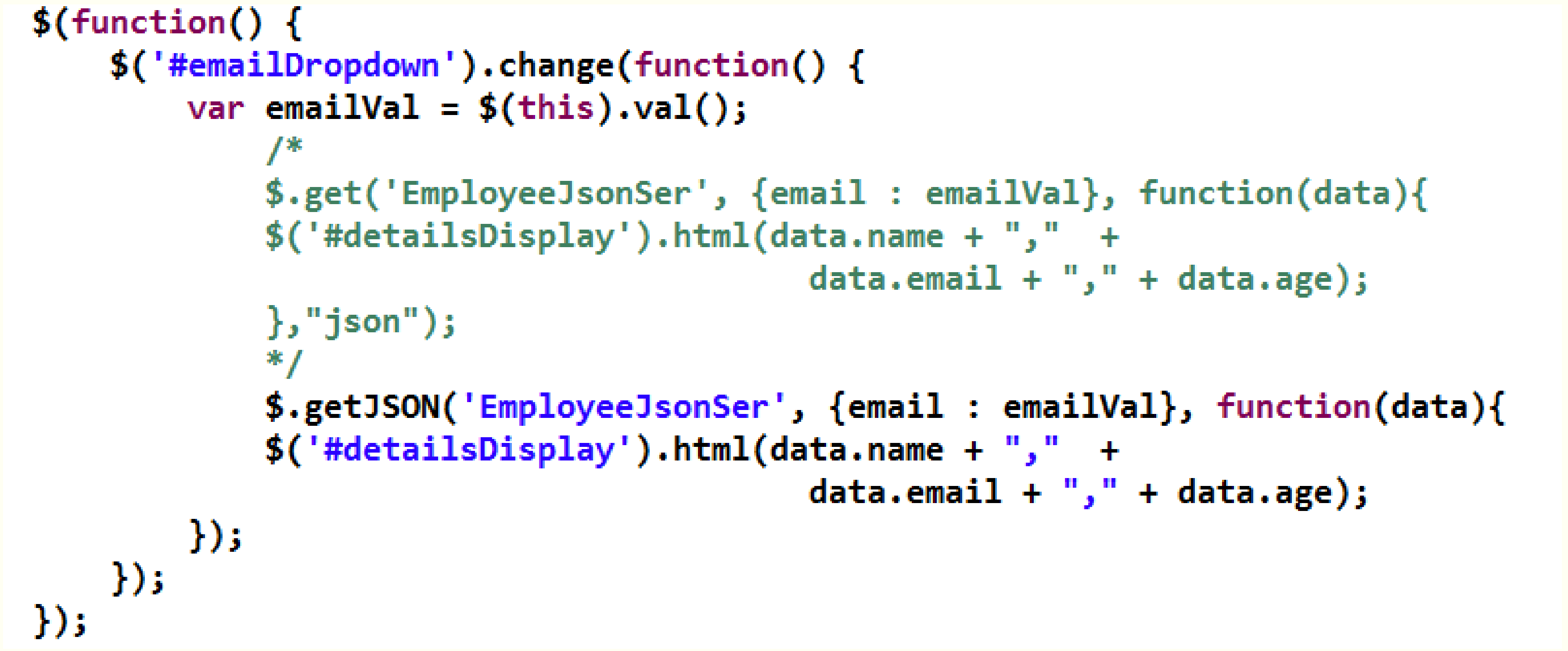
$.getJSON

- $.getJSON("url", dataObj, someFunct)
- $.ajax({url: "url", data: dataObj, success: someFunct,
  dataType: "json"});

Note

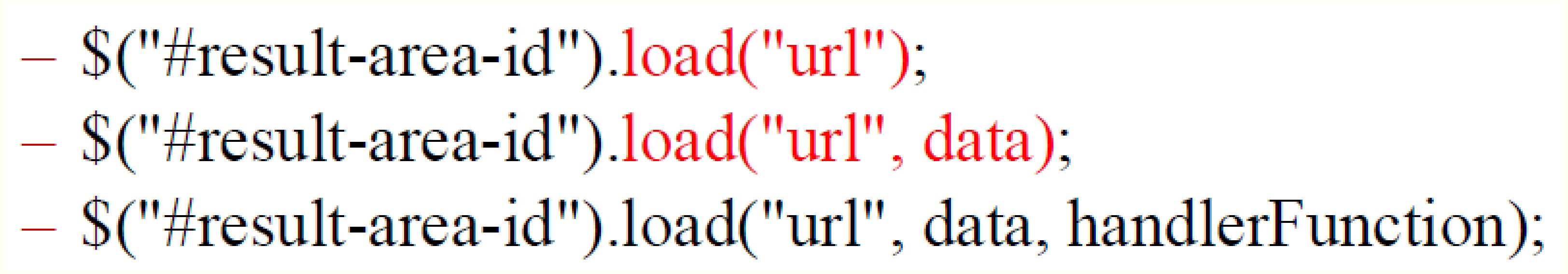
- get and post take the type as an optional fourth argument

Example $.get Vs. $.getJSON



```
$(function() {
    $('#emailDropdown').change(function() {
        var emailVal = $(this).val();
            /*
            $.get('EmployeeJsonSer', {email : emailVal}, function(data){
            $('#detailsDisplay').html(data.name + ","  +
                                        data.email + "," + data.age);
            },"json");
            */
            $.getJSON('EmployeeJsonSer', {email : emailVal}, function(data){
            $('#detailsDisplay').html(data.name + ","  +
                                        data.email + "," + data.age);
        });
    });
});
```

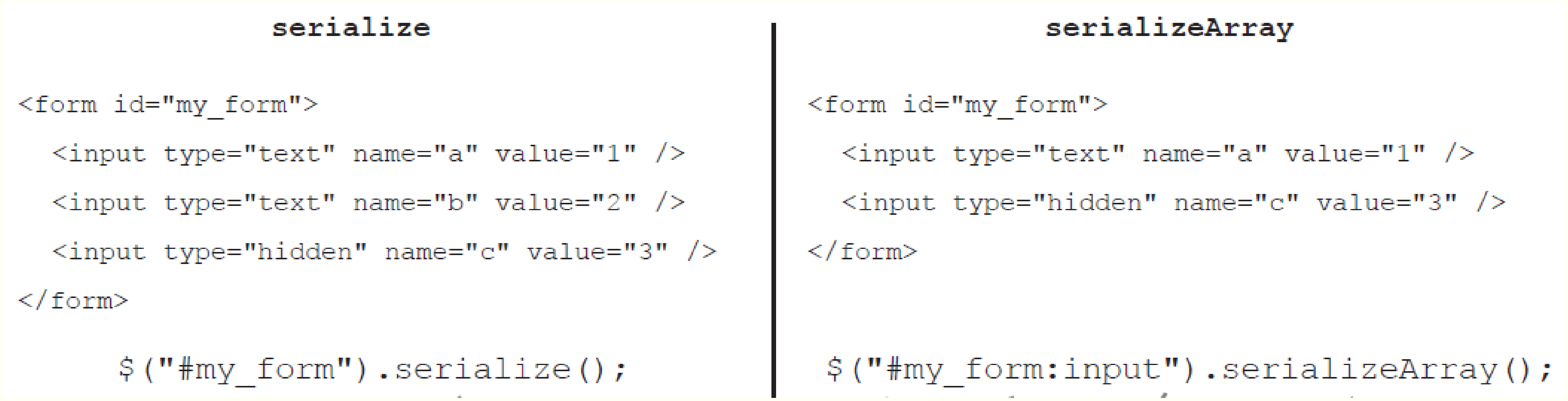
Simplifying Inserting Results into HTML: the “load” Function



- `$("#result-area-id").load("url");`
- `$("#result-area-id").load("url", data);`
- `$("#result-area-id").load("url", data, handlerFunction);`

Format the data before you send it.

- jQuery offers two form helper methods for serializing data: serialize and serializeArray.

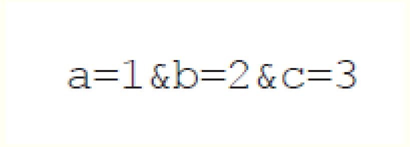


serialize

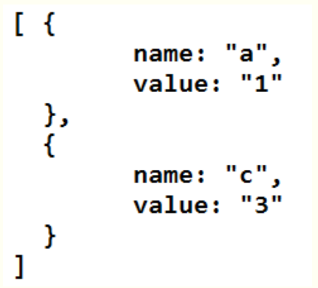
```
<form id="my_form">
  <input type="text" name="a" value="1" />
  <input type="text" name="b" value="2" />
  <input type="hidden" name="c" value="3" />
</form>

$("#my_form").serialize();
```

serializeArray

```
<form id="my_form">
  <input type="text" name="a" value="1" />
  <input type="hidden" name="c" value="3" />
</form>

$("#my_form:input").serializeArray();
```

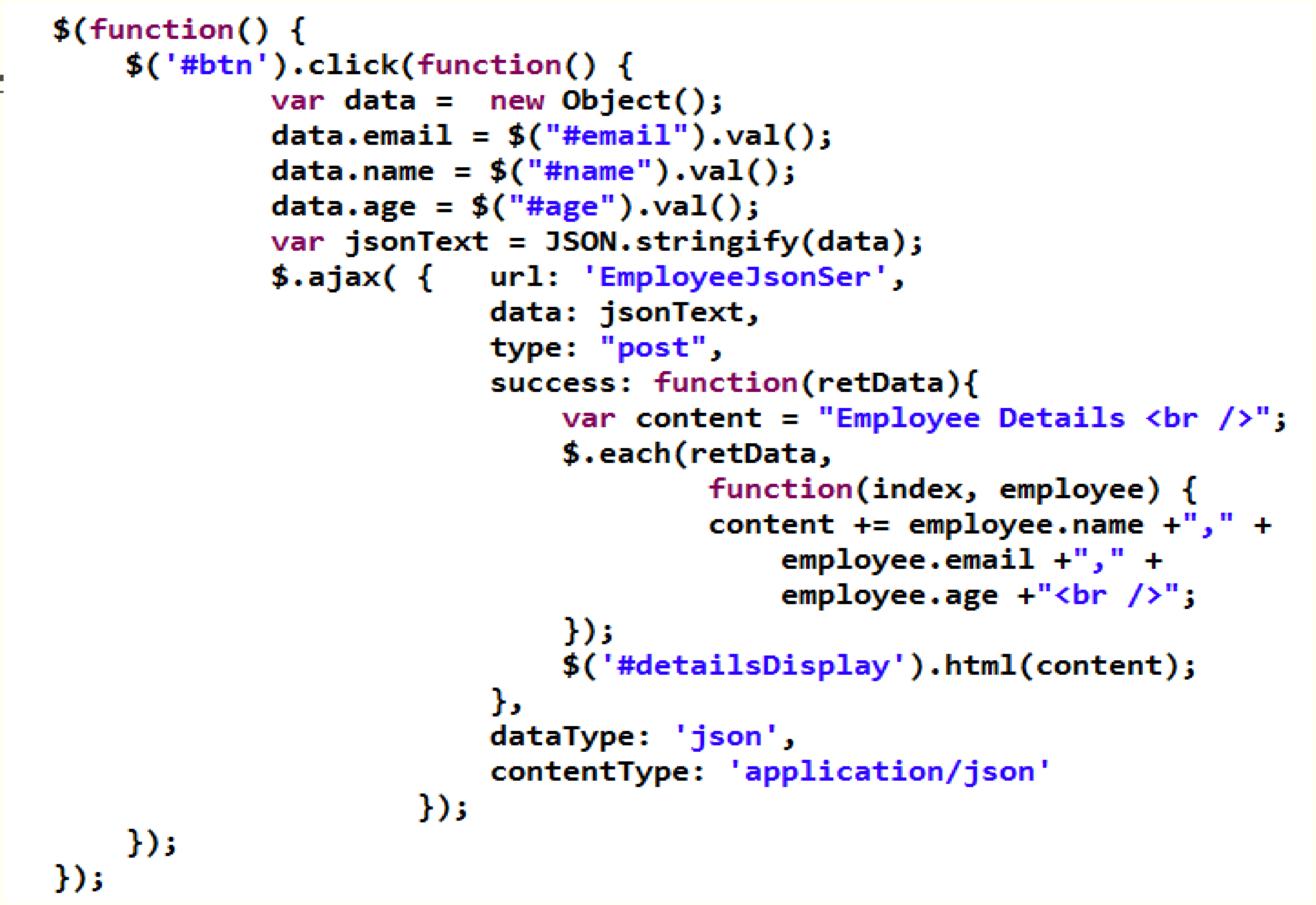


```
a=1&b=2&c=3
```



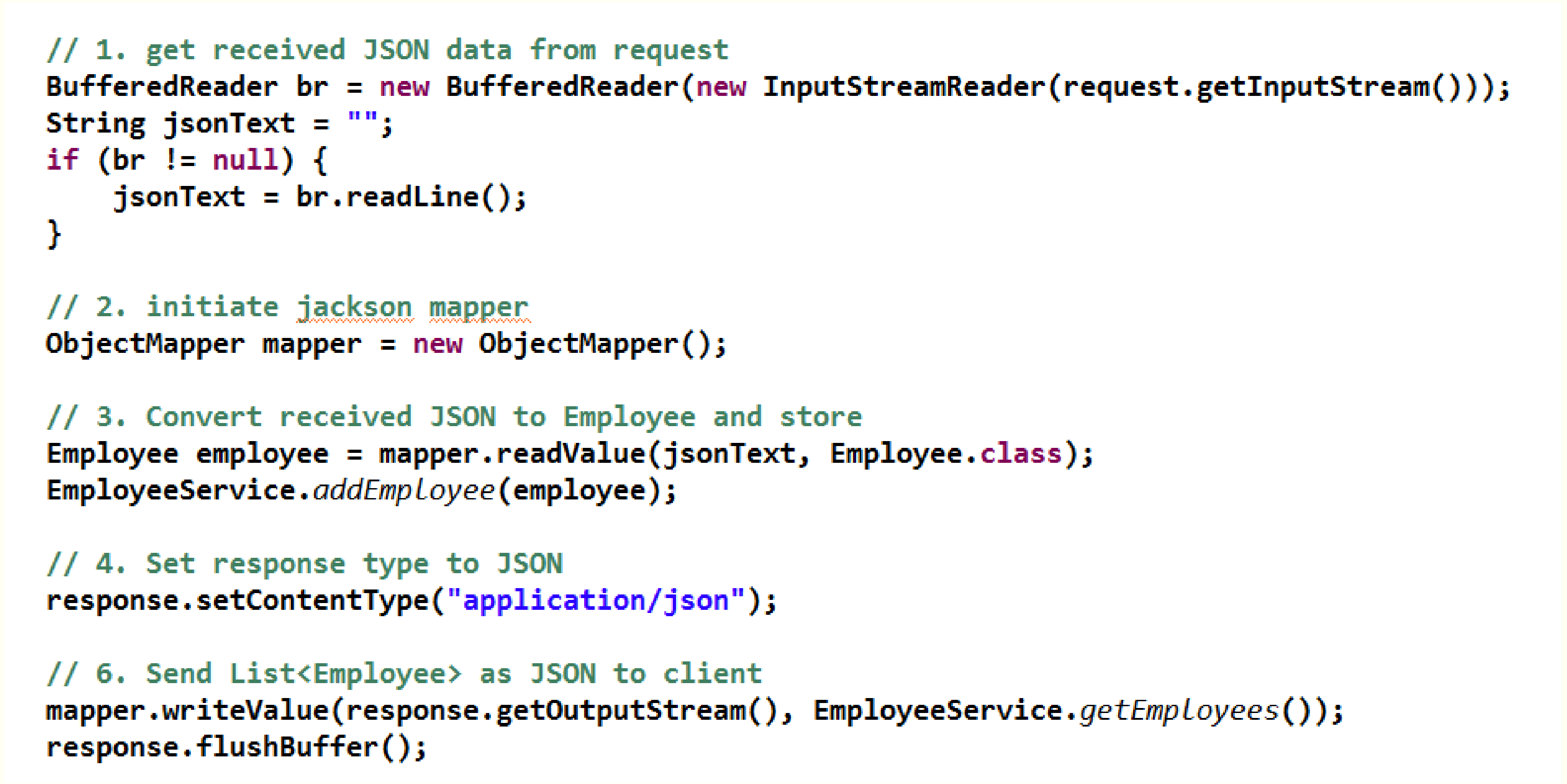
```
[ {
        name: "a",
        value: "1"
  },
  {
        name: "c",
        value: "3"
  }
]
```

Example



```javascript
$(function() {
    $('#btn').click(function() {
            var data =  new Object();
            data.email = $("#email").val();
            data.name = $("#name").val();
            data.age = $("#age").val();
            var jsonText = JSON.stringify(data);
            $.ajax( {   url: 'EmployeeJsonSer',
                        data: jsonText,
                        type: "post",
                        success: function(retData){
                            var content = "Employee Details <br />";
                            $.each(retData,
                                    function(index, employee) {
                                    content += employee.name +"," +
                                        employee.email +"," +
                                        employee.age +"<br />";
                            });
                            $('#detailsDisplay').html(content);
                        },
                        dataType: 'json',
                        contentType: 'application/json'
                    });
    });
});
```

Server Side code

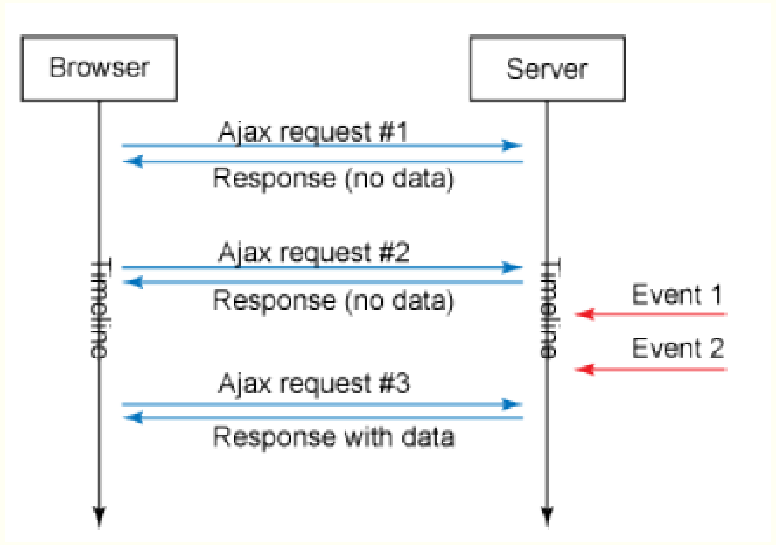


```java
// 1. get received JSON data from request
BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
String jsonText = "";
if (br != null) {
    jsonText = br.readLine();
}

// 2. initiate jackson mapper
ObjectMapper mapper = new ObjectMapper();

// 3. Convert received JSON to Employee and store
Employee employee = mapper.readValue(jsonText, Employee.class);
EmployeeService.addEmployee(employee);

// 4. Set response type to JSON
response.setContentType("application/json");

// 6. Send List<Employee> as JSON to client
mapper.writeValue(response.getOutputStream(), EmployeeService.getEmployees());
response.flushBuffer();
```

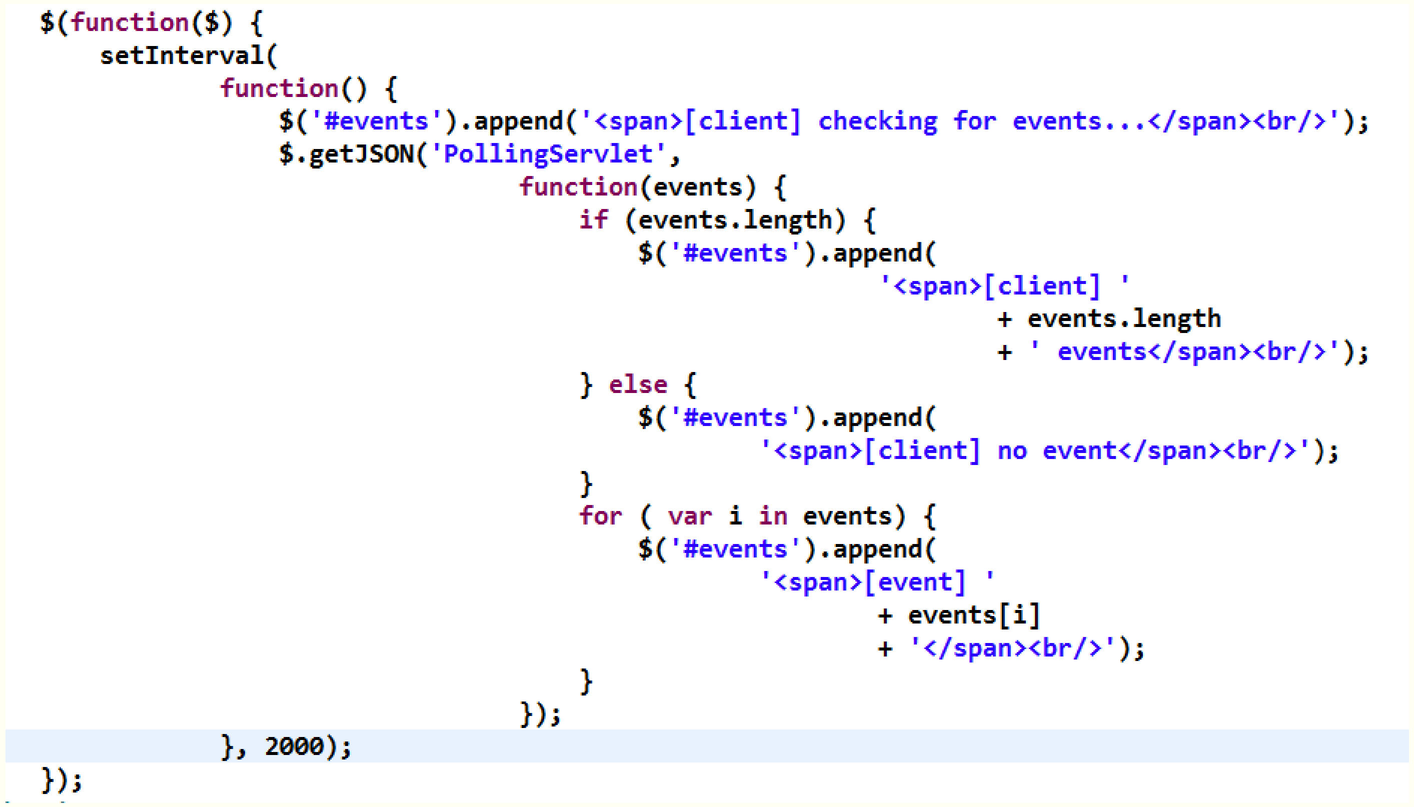
Reverse Ajax techniques

- Ajax requests are stateless by default, and can only be opened from the client to the server.
- The goal of Reverse Ajax is to let the server push information to the client.
- HTTP Polling:
  - *Polling* involves issuing a request from the client to the server to ask for some data. This is obviously a mere Ajax HTTP request.
  - To get the server events as soon as possible, the polling interval (time between requests) must be as low as possible. There's a drawback: if this interval is reduced, the client browser is going to issue many more requests, many of which won't return any useful data, and will consume bandwidth and processing resources for nothing

Reverse Ajax with HTTP polling



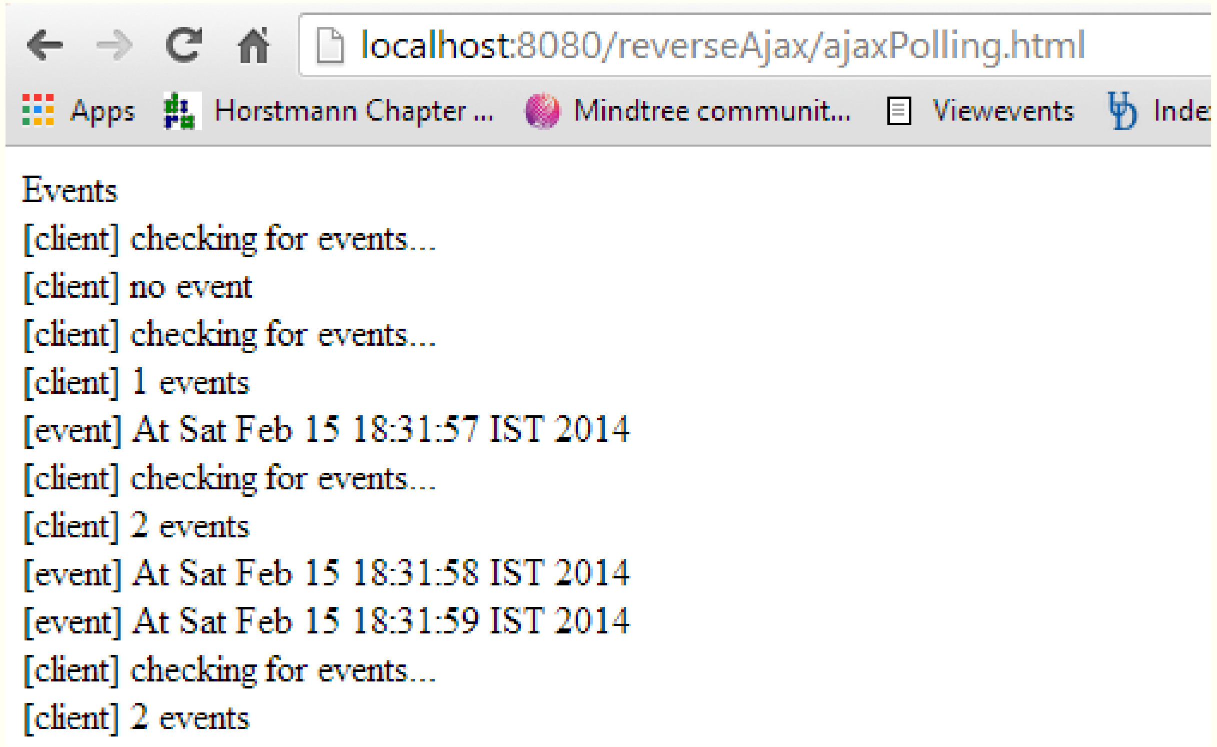
Reverse Ajax with HTTP polling



```
$(function($) {
    setInterval(
            function() {
                $('#events').append('<span>[client] checking for events...</span><br/>');
                $.getJSON('PollingServlet',
                                function(events) {
                                    if (events.length) {
                                        $('#events').append(
                                                        '<span>[client] '
                                                                + events.length
                                                                + ' events</span><br/>');
                                    } else {
                                        $('#events').append(
                                                '<span>[client] no event</span><br/>');
                                    }
                                    for ( var i in events) {
                                        $('#events').append(
                                                '<span>[event] '
                                                        + events[i]
                                                        + '</span><br/>');
                                    }
                                });
            }, 2000);
});
```

Reverse Ajax with HTTP polling

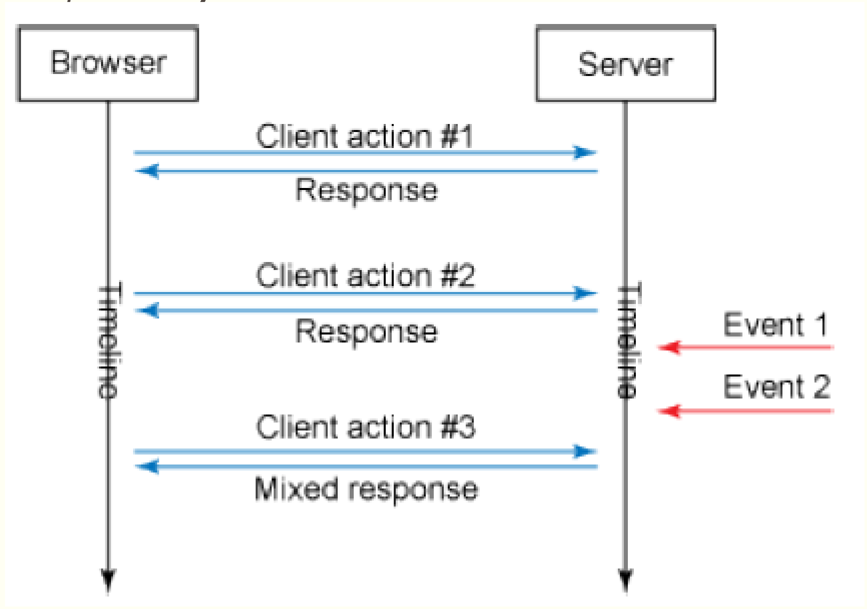
- Sample output



```
Events
[client] checking for events...
[client] no event
[client] checking for events...
[client] 1 events
[event] At Sat Feb 15 18:31:57 IST 2014
[client] checking for events...
[client] 2 events
[event] At Sat Feb 15 18:31:58 IST 2014
[event] At Sat Feb 15 18:31:59 IST 2014
[client] checking for events...
[client] 2 events
```

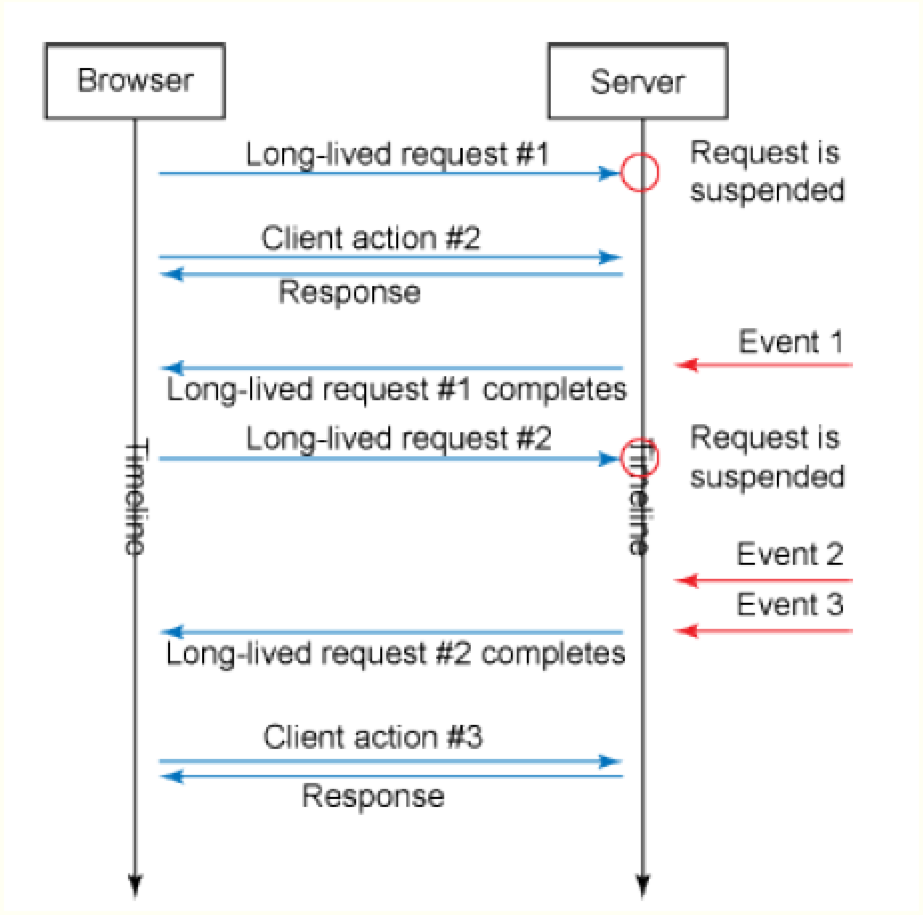
Piggyback

- Piggyback polling is a much more clever method than polling since it tends to remove all non-needed requests (those returning no data).
- There is no interval; requests are sent when the client needs to send a request to the server. The difference lies in the response, which is split into two parts: the response for the requested data and the server events, if any occurred



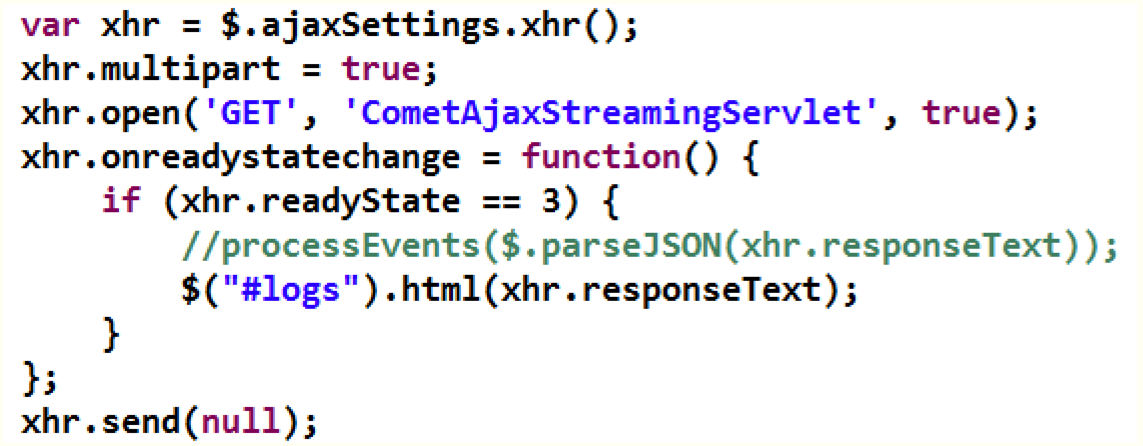
Comet using HTTP streaming

- In streaming mode, one persistent connection is opened.
- There will only be a long-lived request since each event arriving on the server side is sent through the same connection.
- Thus, it requires on the client side a way to separate the different responses coming through the same connection.
- Multi-part feature of the XMLHttpRequest object is used to create Ajax requests in JavaScript



Comet using HTTP streaming

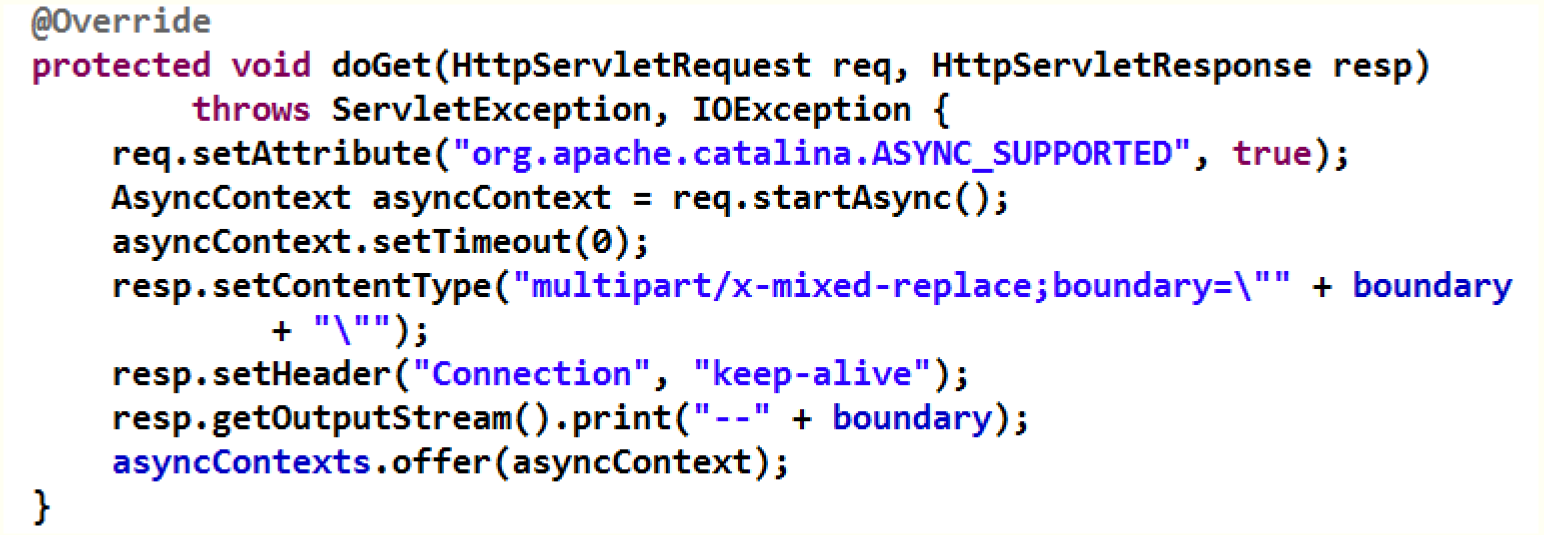
- Ajax Code



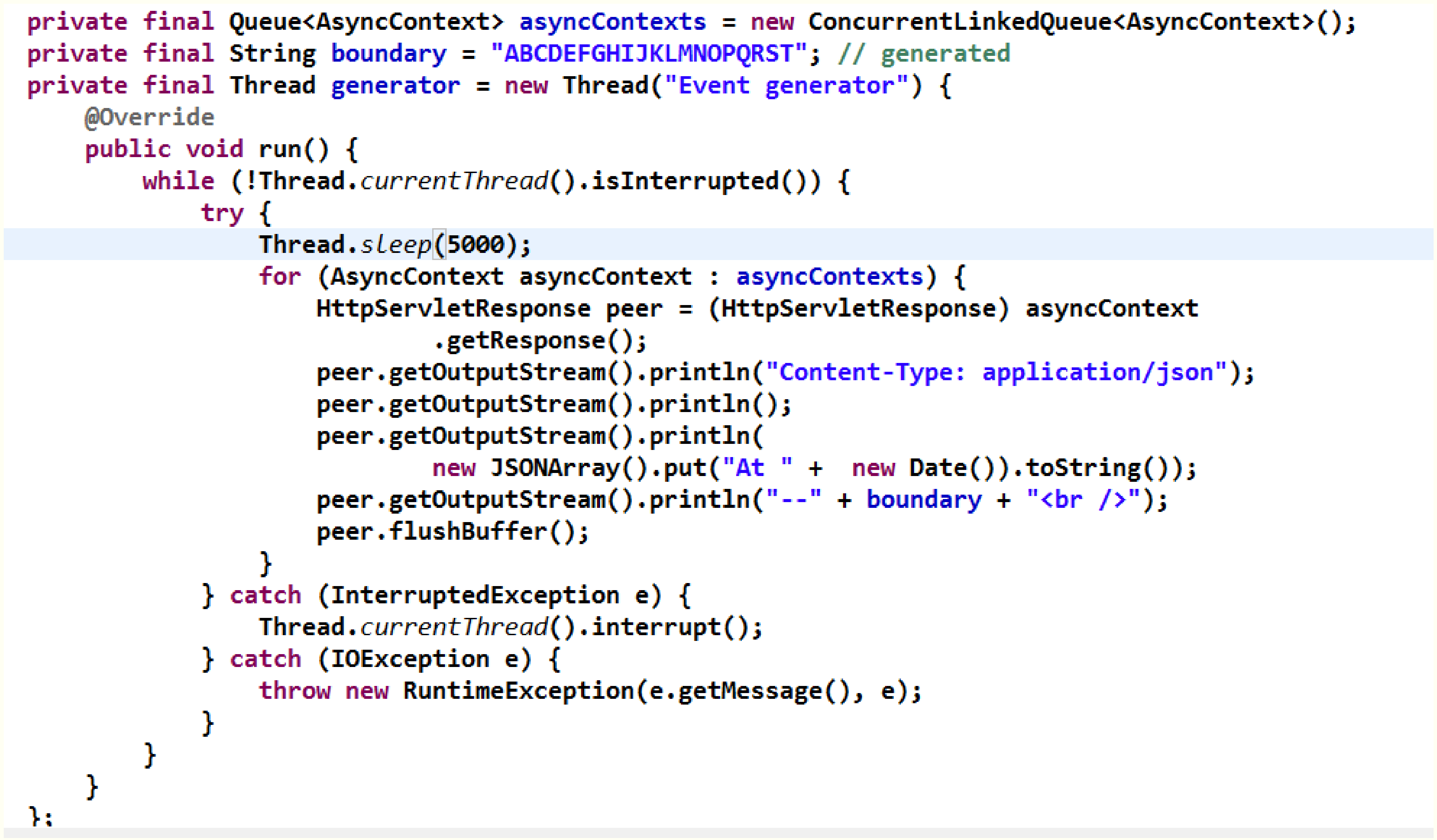
```
var xhr = $.ajaxSettings.xhr();
xhr.multipart = true;
xhr.open('GET', 'CometAjaxStreamingServlet', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 3) {
        //processEvents($.parseJSON(xhr.responseText));
        $("#logs").html(xhr.responseText);
    }
};
xhr.send(null);
```

Comet using HTTP streaming

- Servlet Code
- AsyncContext: the execution context for an asynchronous operation that was initiated on a ServletRequest.



```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    req.setAttribute("org.apache.catalina.ASYNC_SUPPORTED", true);
    AsyncContext asyncContext = req.startAsync();
    asyncContext.setTimeout(0);
    resp.setContentType("multipart/x-mixed-replace;boundary=\"" + boundary
            + "\"");
    resp.setHeader("Connection", "keep-alive");
    resp.getOutputStream().print("--" + boundary);
    asyncContexts.offer(asyncContext);
}
```



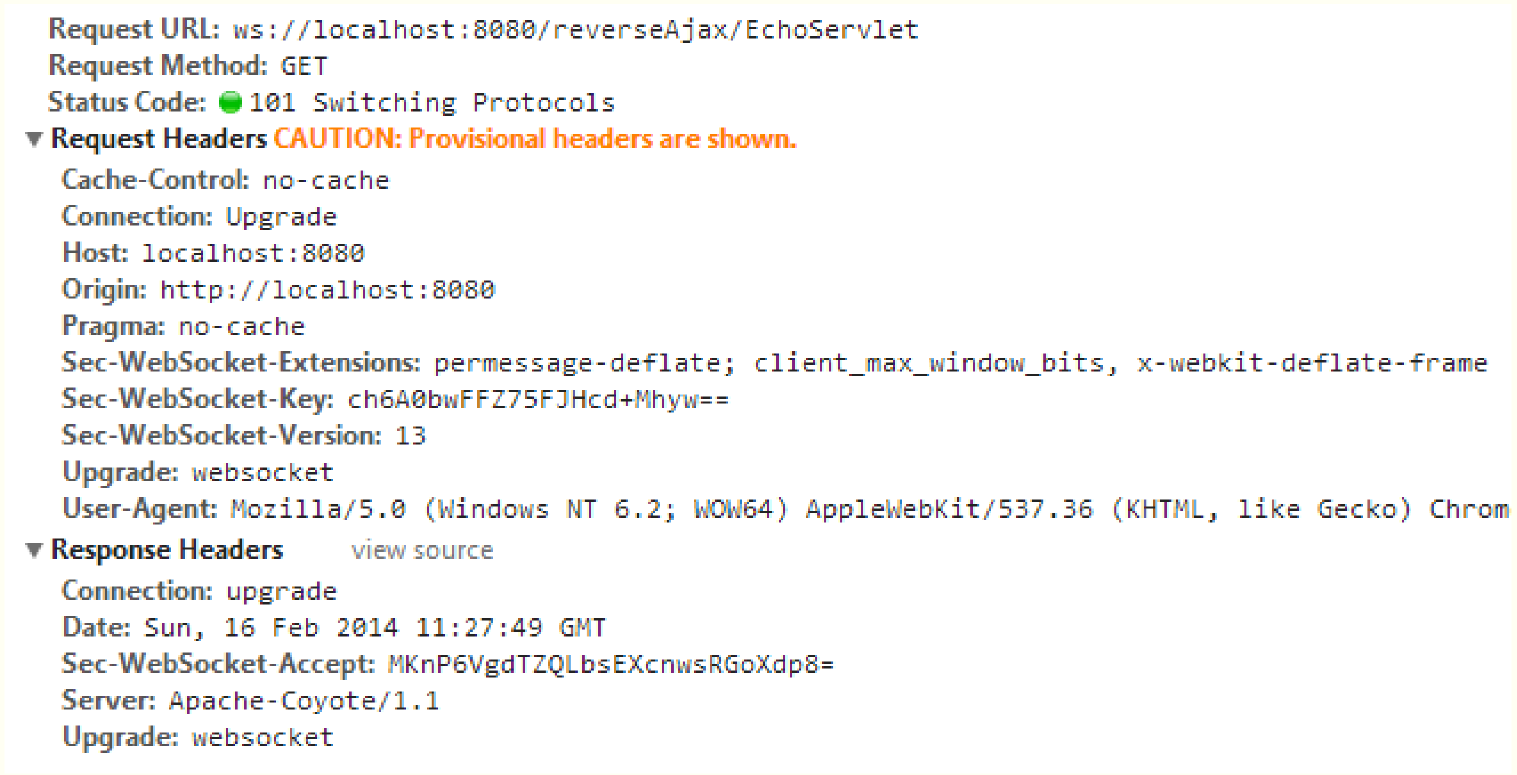
```
private final Queue<AsyncContext> asyncContexts = new ConcurrentLinkedQueue<AsyncContext>();
private final String boundary = "ABCDEFGHIJKLMNOPQRST"; // generated
private final Thread generator = new Thread("Event generator") {
    @Override
    public void run() {
        while (!Thread.currentThread().isInterrupted()) {
            try {
                Thread.sleep(5000);
                for (AsyncContext asyncContext : asyncContexts) {
                    HttpServletResponse peer = (HttpServletResponse) asyncContext
                            .getResponse();
                    peer.getOutputStream().println("Content-Type: application/json");
                    peer.getOutputStream().println();
                    peer.getOutputStream().println(
                            new JSONArray().put("At " +  new Date()).toString());
                    peer.getOutputStream().println("--" + boundary + "<br />");
                    peer.flushBuffer();
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            } catch (IOException e) {
                throw new RuntimeException(e.getMessage(), e);
            }
        }
    }
};
```

WebSockets

- WebSockets are a new way for clients to communicate to servers and vice versa, without the overhead of an HTTP protocol.
- This implementation allowed for asynchronous event driven request processing as well as bi-directional communication.
- The protocol specification makes it clear that one of the design decisions when making this protocol was to ensure that both HTTP based clients and WebSocket based ones can operate on the same port.
- This is why the handshake is such that the client and server ‘upgrade’ from an HTTP based protocol to a WebSocket based protocol.

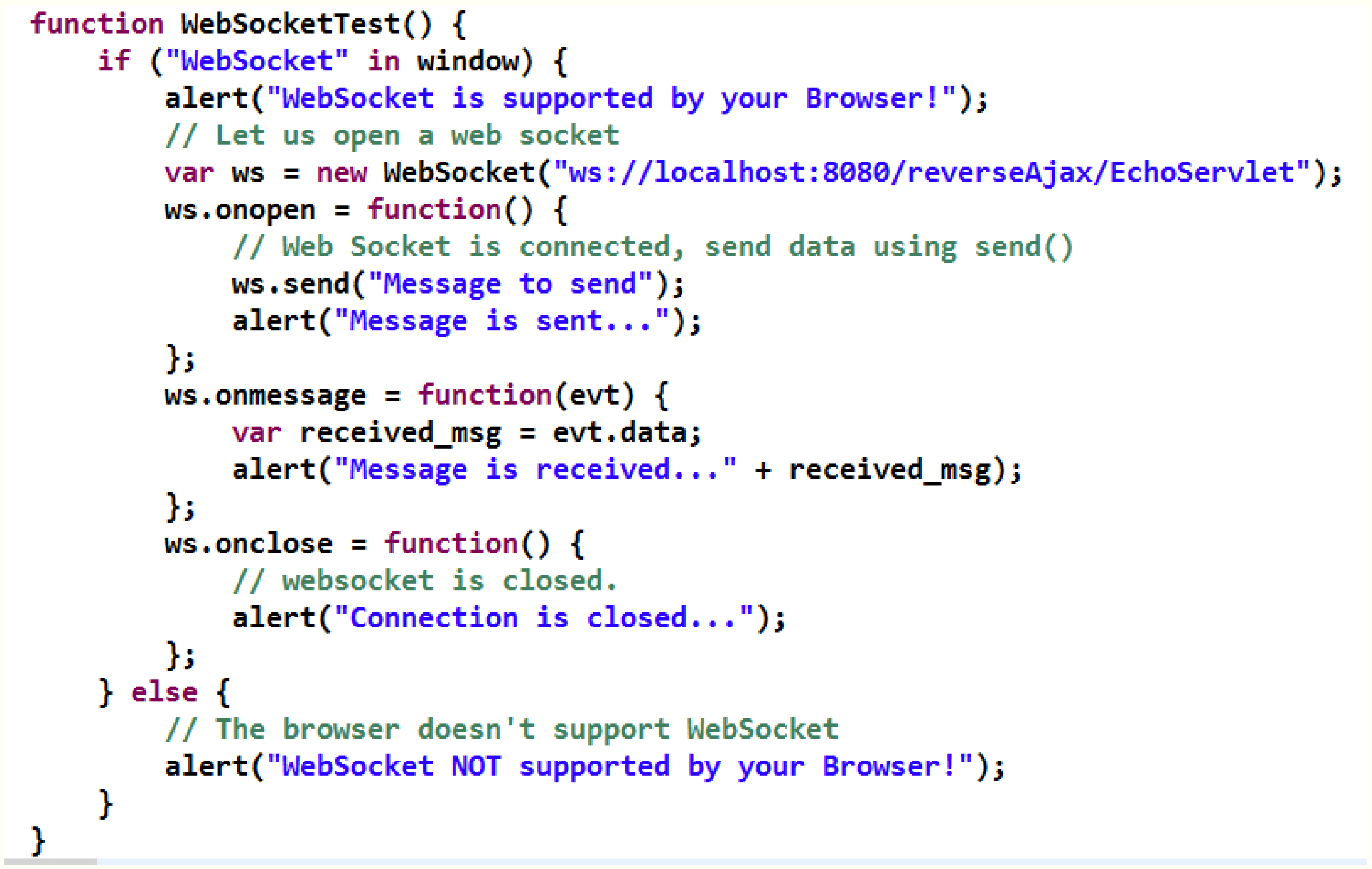
WebSockets

- The response contains the critical status code of 101



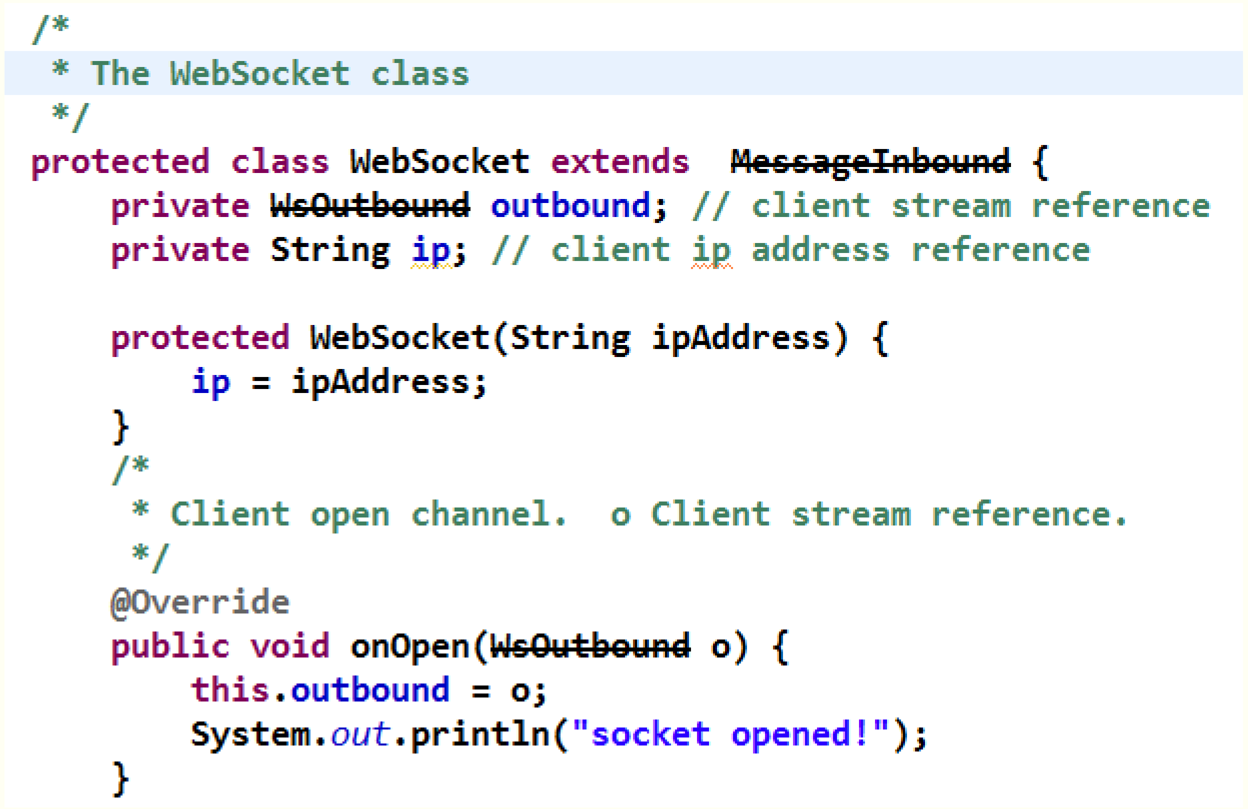
```
Request URL: ws://localhost:8080/reverseAjax/EchoServlet
Request Method: GET
Status Code: ● 101 Switching Protocols
▼ Request Headers CAUTION: Provisional headers are shown.
   Cache-Control: no-cache
   Connection: Upgrade
   Host: localhost:8080
   Origin: http://localhost:8080
   Pragma: no-cache
   Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits, x-webkit-deflate-frame
   Sec-WebSocket-Key: ch6A0bwFFZ75FJHcd+Mhyw==
   Sec-WebSocket-Version: 13
   Upgrade: websocket
   User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
▼ Response Headers     view source
   Connection: upgrade
   Date: Sun, 16 Feb 2014 11:27:49 GMT
   Sec-WebSocket-Accept: MKnP6VgdTZQLbsEXcnwsRGoXdp8=
   Server: Apache-Coyote/1.1
   Upgrade: websocket
```

WebSocket AJAX client



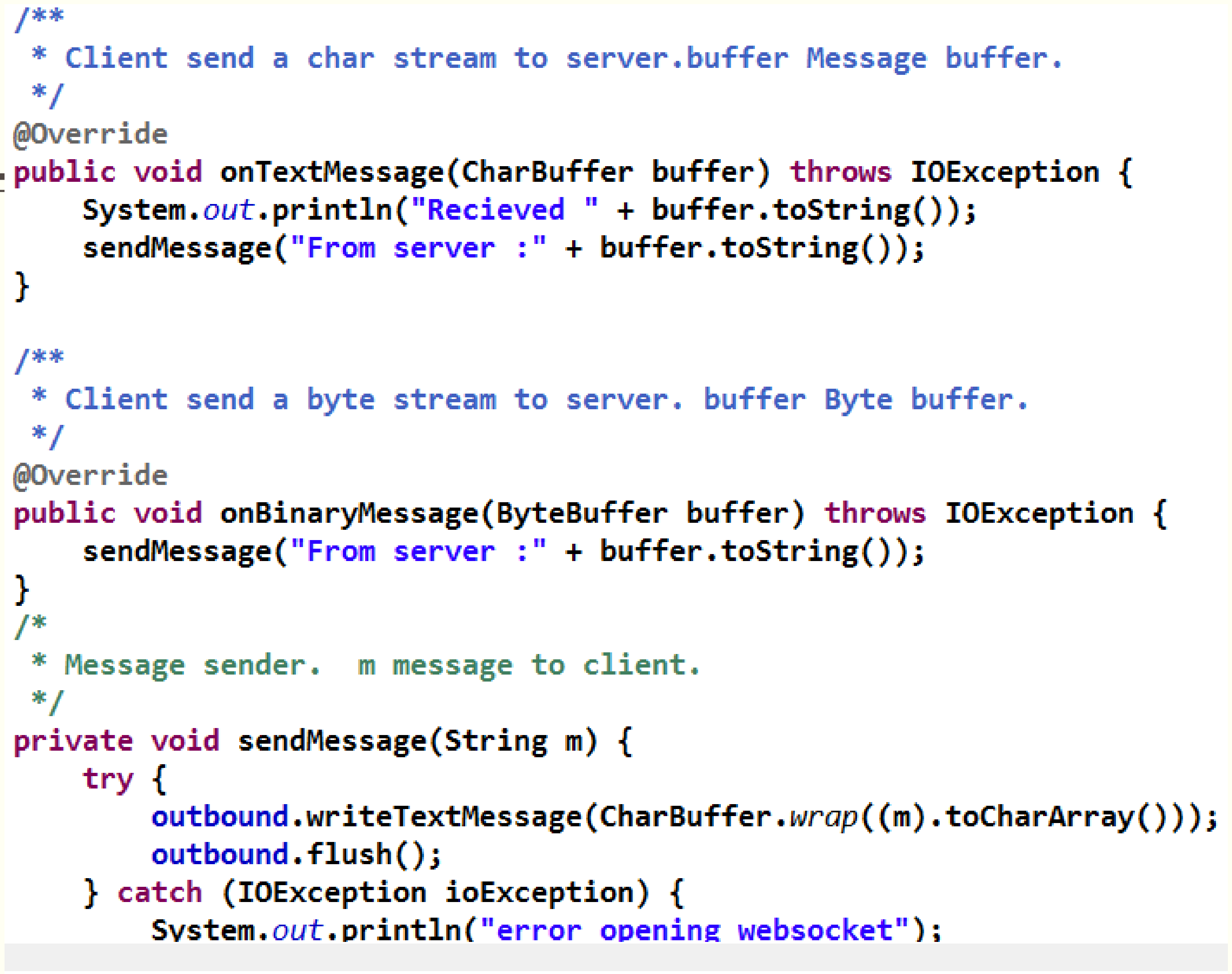
```
function WebSocketTest() {
    if ("WebSocket" in window) {
        alert("WebSocket is supported by your Browser!");
        // Let us open a web socket
        var ws = new WebSocket("ws://localhost:8080/reverseAjax/EchoServlet");
        ws.onopen = function() {
            // Web Socket is connected, send data using send()
            ws.send("Message to send");
            alert("Message is sent...");
        };
        ws.onmessage = function(evt) {
            var received_msg = evt.data;
            alert("Message is received..." + received_msg);
        };
        ws.onclose = function() {
            // websocket is closed.
            alert("Connection is closed...");
        };
    } else {
        // The browser doesn't support WebSocket
        alert("WebSocket NOT supported by your Browser!");
    }
}
```

WebSocket helper code for Servlet



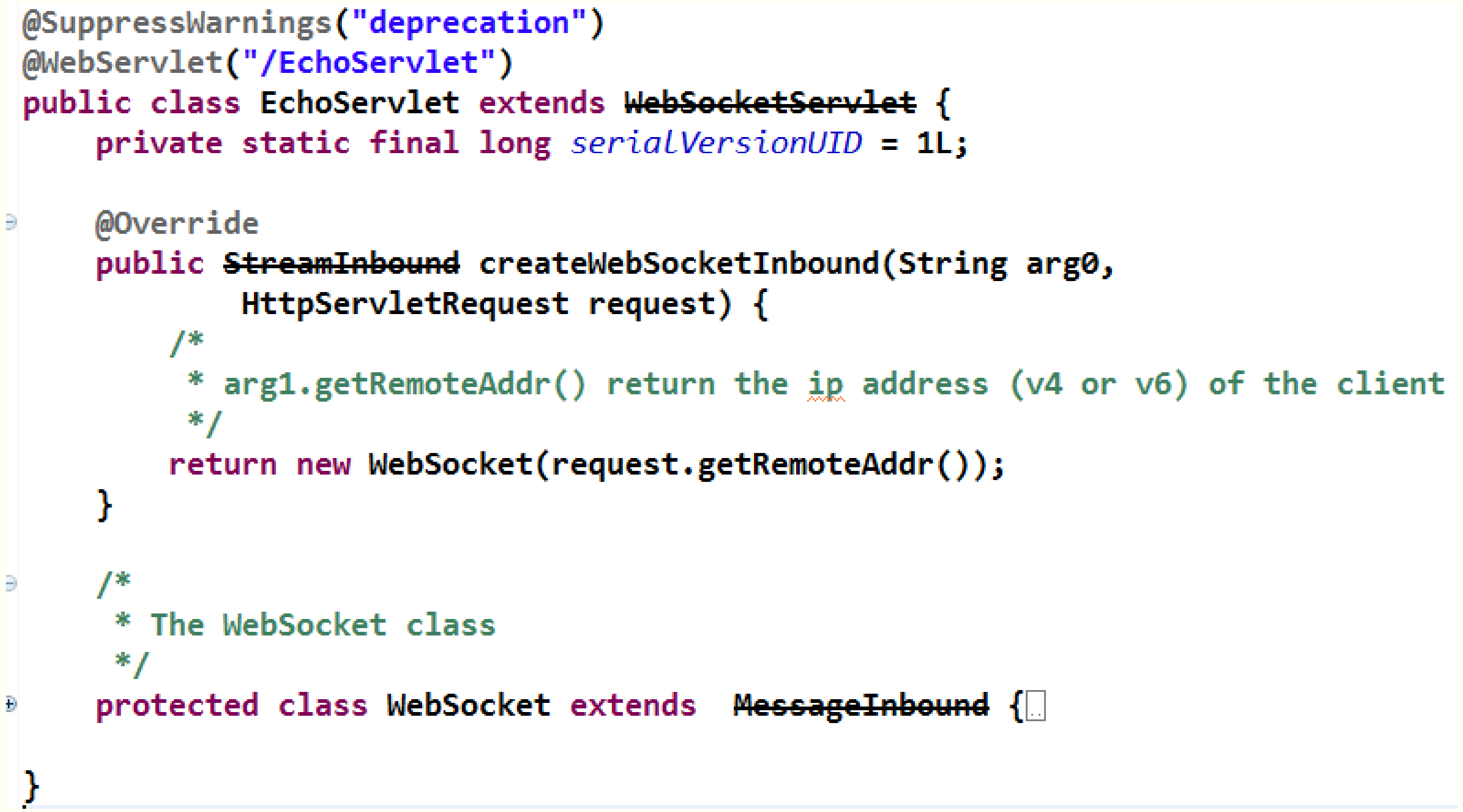
```
/*
 * The WebSocket class
 */
protected class WebSocket extends  ~~MessageInbound~~ {
    private ~~WsOutbound~~ outbound; // client stream reference
    private String ip; // client ip address reference

    protected WebSocket(String ipAddress) {
        ip = ipAddress;
    }
    /*
     * Client open channel.  o Client stream reference.
     */
    @Override
    public void onOpen(~~WsOutbound~~ o) {
        this.outbound = o;
        System.out.println("socket opened!");
    }
```

Contd..



```
/**
 * Client send a char stream to server.buffer Message buffer.
 */
@Override
public void onTextMessage(CharBuffer buffer) throws IOException {
    System.out.println("Recieved " + buffer.toString());
    sendMessage("From server :" + buffer.toString());
}

/**
 * Client send a byte stream to server. buffer Byte buffer.
 */
@Override
public void onBinaryMessage(ByteBuffer buffer) throws IOException {
    sendMessage("From server :" + buffer.toString());
}
/*
 * Message sender.  m message to client.
 */
private void sendMessage(String m) {
    try {
        outbound.writeTextMessage(CharBuffer.wrap((m).toCharArray()));
        outbound.flush();
    } catch (IOException ioException) {
        System.out.println("error opening websocket");
```

WebSocket Servlet



```
@SuppressWarnings("deprecation")
@WebServlet("/EchoServlet")
public class EchoServlet extends ~~WebSocketServlet~~ {
    private static final long serialVersionUID = 1L;

    @Override
    public ~~StreamInbound~~ createWebSocketInbound(String arg0,
            HttpServletRequest request) {
        /*
         * arg1.getRemoteAddr() return the ip address (v4 or v6) of the client
         */
        return new WebSocket(request.getRemoteAddr());
    }

    /*
     * The WebSocket class
     */
    protected class WebSocket extends  ~~MessageInbound~~ {

}
```