jQuery Ajax Support: Advanced Capabilities

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.

coreservlets.com – custom onsite training

For customized training related to JavaScript or Java, email hall@coreservlets.com

Marty is also available for consulting and development support

Taught by lead author of *Core Servlets & JSP*, co-author of *Core JSF* (4th Ed), and this tutorial.

Available at public venues, or custom versions can be held on-site at your organization.

- **Courses developed and taught by Marty Hall**
  - JavaScript, jQuery, Ext JS, JSF 2.3, PrimeFaces, Java 8 programming, Spring Framework, Spring MVC, Android, GWT, custom mix of topics
  - Courses available in any state or country.
  - Maryland/DC companies can also choose afternoon/evening courses.
- **Courses developed and taught by coreservlets.com experts (edited by Marty)**
  - Hadoop, Hibernate/JPA, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- Displaying temporary messages during Ajax requests
- Handling JSON data
- Example: static JSON data
- Example: dynamic JSON data
- $.ajax options
- Ajax shortcut functions

Displaying Temporary Messages During Ajax Requests

Showing "Getting Data..." Message While Waiting

- **Idea**
  - You have slow server operation
  - Display animated GIF and message when request sent
  - Hide GIF and message when response completes
- **Approach**
  - Get animated GIF
    - http://ajaxload.info/ lets you build your own with unrestricted use
  - Display image plus message in region that has "display: none"
    - So it is hidden initially
  - Before sending Ajax request, change region to use "display: inline"
    - Use $(selector).show()
  - When request finishes, hide the region again
    - Use $(selector).hide() from complete: handler (not success:)

Temporary Data: HTML

```
…
<fieldset>
<legend>$.ajax: Showing Temporary Messages</legend>
  <input type="button" value="Show Time (Slow)"
         id="slow-time-button"/>
  <h2 id="working" style="display: none">
    <img src="images/ajax-loader.gif"/>
    Waiting for server...
  </h2>
  <h2 id="slow-time-result"></h2>
</fieldset>
```

Temporary Data: JavaScript

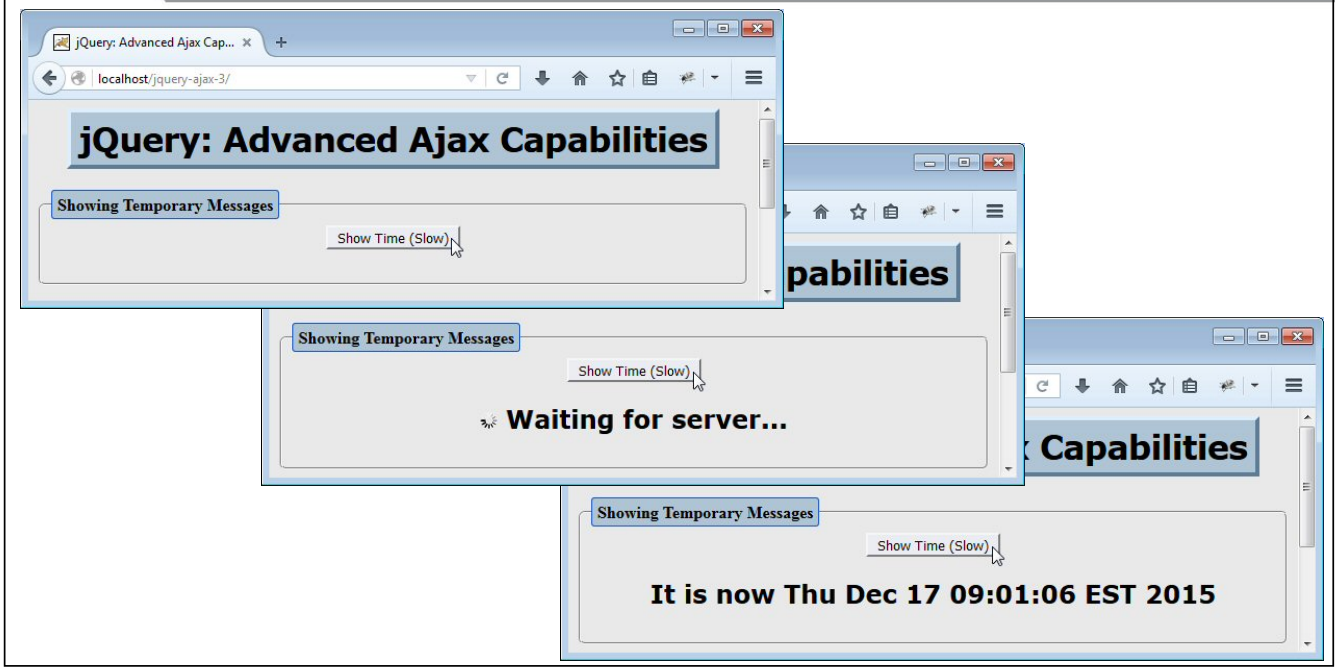
```
$(function() {
    $("#slow-time-button").click(showTime3); …
});

function showTime3() {
  var workingRegion = "#working";
  var resultRegion = "#slow-time-result";
  $(resultRegion).html(""); // Erase any previous results
  $(workingRegion).show();
  $.ajax({
    url: "server-time-slow.jsp",
    success: function(text) { $(resultRegion).html(text); },
    complete: function() { $(workingRegion).hide(); }
  });
}
```


Temporary Data: JSP (server-time-slow.jsp)

```
<% try { Thread.sleep(5000);
   } catch(Exception e) {} %>
It is now <%= new java.util.Date() %>
```

This waits five seconds, then outputs a string showing the current date and time.

Temporary Messages: Results



Handling JSON Data: Overview

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.

Content-Centric vs. Data-Centric Ajax

- **Content-centric Ajax**
  - The server-side program builds the exact content we need
    - Simple string or chunk of HTML
  - The client-side JavaScript merely inserts the server content into the page verbatim
    - Without examining it or modifying it
- **Data-centric Ajax**
  - The server-side program sends raw data
    - JSON is the most common format, but you could also send XML or a string containing a custom data format
  - The client-side JavaScript reads the data, then builds a simple string or a chunk of HTML based on that data
  - This string (built by the JavaScript code, not returned directly by server) is what is inserted into the page

Recap: Strict JSON

- **Strict JSON according to json.org**
  - Subset of JavaScript where
    - Object property names must be in double quotes
    - Strings use double quotes only (not single quotes)
  - This is what recent jQuery versions and JSON.parse support
    - jQuery automatically uses JSON.parse when you specify dataType: "json"
    - If the JSON data has any errors, the Ajax request fails and the success handler function is never invoked
  - Since this is what is clearly described at json.org, you should follow this format when sending JSON from the server.
- **MIME type for JSON from server**
  - RFC 4627 says JSON responses should have "application/json" MIME type
  - No known libraries enforce this

Data-Centric Ajax with JSON: Approach

- **Server**
  - Returns strict JSON object containing data. E.g.:

```
{ "cto": "Resig ", "ceo": "Gates ", "coo": "Ellison" }
```

- **Code that calls $.ajax**
  - Specifies dataType of json. E.g.:

```
$.ajax({ url: address, success: handler, dataType: "json" });
```

- **Response handler**
  - Receives JavaScript data as first argument. No need for JSON.parse or eval. Must build HTML or plain strings from the result. E.g.:

```
function handler(companyExecutives) {
  $("#some-id").html("<b>Chief Technology Officer is " +
                     companyExecutives.cto + "</b>");
}
```

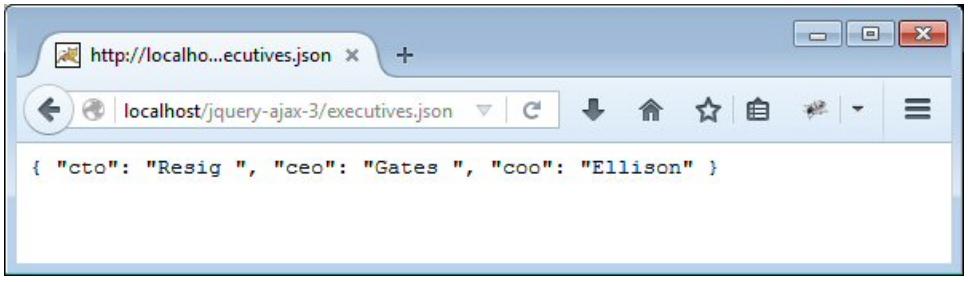

Handling JSON Data: Example with Static Data

Overview

- **Server-side file (executives.json)**
  - Returns static JSON object with properties that show company executives
    - File extension of ".json" is just for documentation; executives.txt would also work
- **Client-side JavaScript code**
  - Main function calls to server-side program and specifies dataType: "json"
  - Handler function takes the JSON object and builds a <h2> heading based on the cto property of the JSON data
- **HTML**
  - Has button to start the process
  - Has named region where result will be shown

Server File (executives.json)

```
{ "cto": "Resig ", "ceo": "Gates ", "coo": "Ellison" }
```



The file extension (.json) is just to remind the developers that it contains JSON. It would work equally well with .txt.

However, it is necessary that the file contains strict JSON: quotes around property names and double, not single, quotes.

Static JSON Example: JavaScript

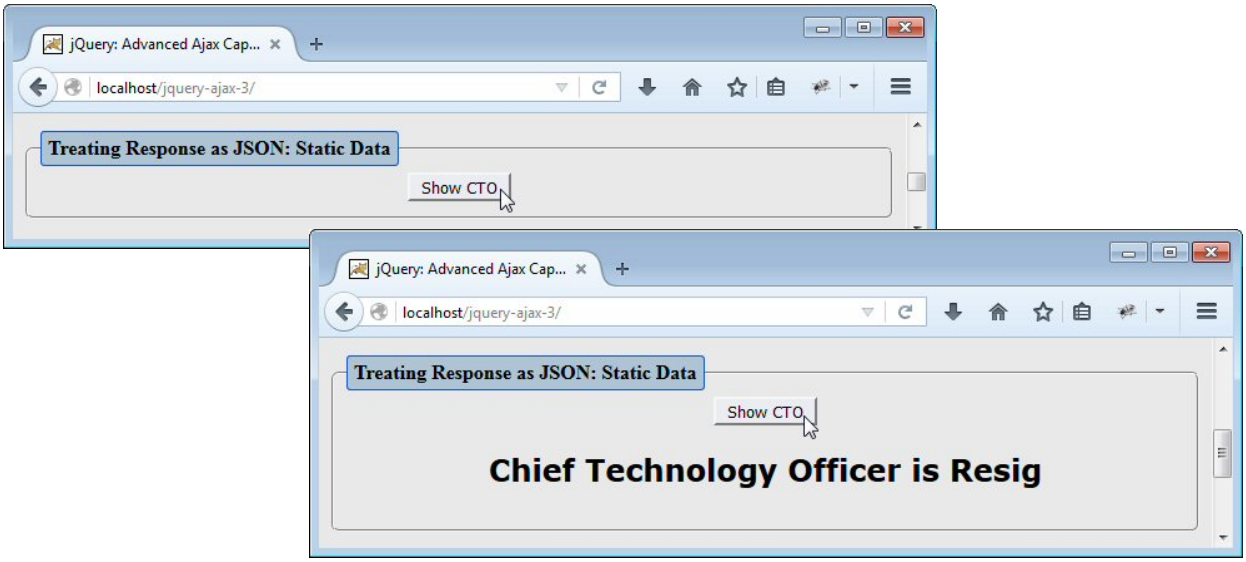
```
$(function() {
    $("#cto-button").click(showCto); ...
});

function showCto() {
  $.ajax({ url: "executives.json",
           dataType: "json",
           success: showCtoHeading });
}

function showCtoHeading(companyExecutives) {
  $("#cto-result").html("<h2>Chief Technology Officer is " +
                        companyExecutives.cto + "</h2>");
}
```

The response handler is passed a JavaScript object, not a string.
The object was built behind-the-scenes in jQuery by using JSON.parse.

Static JSON Example: HTML

```
<fieldset>
<legend>Treating Response as JSON: Static Data</legend>
  <input type="button" value="Show CTO"
         id="cto-button"/>
  <div id="cto-result"></div>
</fieldset>
```

Static JSON Example: Results



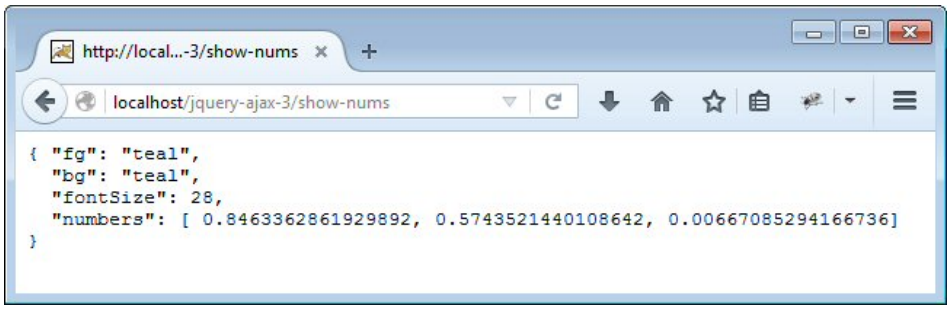
Handling JSON Data: Example with Dynamic Data

Overview

- **Server-side program**
  - Returns complex JSON object with values that change randomly
- **Client-side JavaScript code**
  - Main function calls to server-side program and specifies dataType: "json"
  - Handler function reads the JSON object and builds a <ul> list based on the information
    - Object contains a font size (number) and colors (strings) that will be used to control the look of the <ul> list
    - Object contains array of numbers that become the <li> elements
- **HTML**
  - Has button to start the process
  - Has named region where result will be shown

Typical Server Result (show-nums)

```
{ "fg": "lime",
  "bg": "blue",
  "fontSize": 32,
  "numbers": [ 0.9830393089180433,
               0.204857758495537,
               0.2637957765810426]
}
```



Dynamic JSON Example: Auxiliary JavaScript

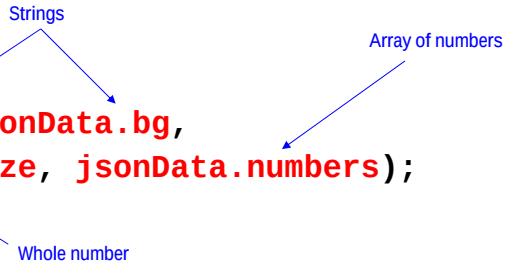
```
function makeList(fg, bg, fontSize, nums) {
  return(listStartTags(fg, bg, fontSize) +
         listItems(nums) +
         listEndTags());
}

function listStartTags(fg, bg, fontSize) {
  return(
    "<div style='color:" + fg + "; " +
                "background-color:" + bg + "; " +
                "font-size:" + fontSize + "px'>\n" +
    "<ul>\n");
}
```


Dynamic JSON Example : Auxiliary JavaScript (Continued)

```
function listItems(items) {
  var result = "";
  for(var i=0; i<items.length; i++) {
    result = result + "<li>" + items[i] + "</li>\n";
  }
  return(result);
}

function listEndTags() {
  return("</ul></div>");
}
```

Dynamic JSON Example: Core JavaScript

```
$(function() { …
    $("#nums-button").click(showNums); ...
});

function showNums() {
  $.ajax({ url: "show-nums",
           dataType: "json",
           success: showNumberList });
}

function showNumberList(jsonData) {
  var list = makeList(jsonData.fg, jsonData.bg,
                      jsonData.fontSize, jsonData.numbers);
  $("#nums-result").html(list);
}
```



Dynamic JSON Example : HTML

```
<fieldset>
<legend>Treating Response as JSON: Dynamic Data</legend>
  <input type="button" value="Show Nums"
         id="nums-button"/>
  <div id="nums-result"></div>
</fieldset>
```

Dynamic JSON Example: Servlet

```
public class ShowNumbers extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setHeader("Cache-Control", "no-cache");
    response.setHeader("Pragma", "no-cache");
    String fg = ColorUtils.randomColor();
    request.setAttribute("fg", fg);
    String bg = ColorUtils.randomColor();
    request.setAttribute("bg", bg);
    String fontSize = "" + (10 + ColorUtils.randomInt(30));
    request.setAttribute("fontSize", fontSize);
    double[] nums = { Math.random(), Math.random(), Math.random() };
    request.setAttribute("nums", nums);
    response.setContentType("application/json");
    String outputPage = "/WEB-INF/results/show-nums.jsp";
    RequestDispatcher dispatcher = request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
  }
}
```

It is not expected that non-Java programmers will understand this code!

Just understand the earlier slide titled "Typical Server Result" – this code produces JSON with a random fg property (string), a random bg property (string), a random fontSize (whole number), and a random numbers property (array containing 3 random numbers).

Dynamic JSON Example : JSP

```
{ "fg": "${fg}",
  "bg": "${bg}",
  "fontSize:" ${fontSize},
  "numbers": [ ${nums[0]}, ${nums[1]}, ${nums[2]}]
}
```

- **Notes**
  - Quotes around property names. Double, not single, quotes
  - Client-side code does *not* need to call JSON.parse. JSON evaluation handled automatically by jQuery
  - Types
    - fg and bg: Strings
    - fontSize: int
    - numbers: Array of doubles

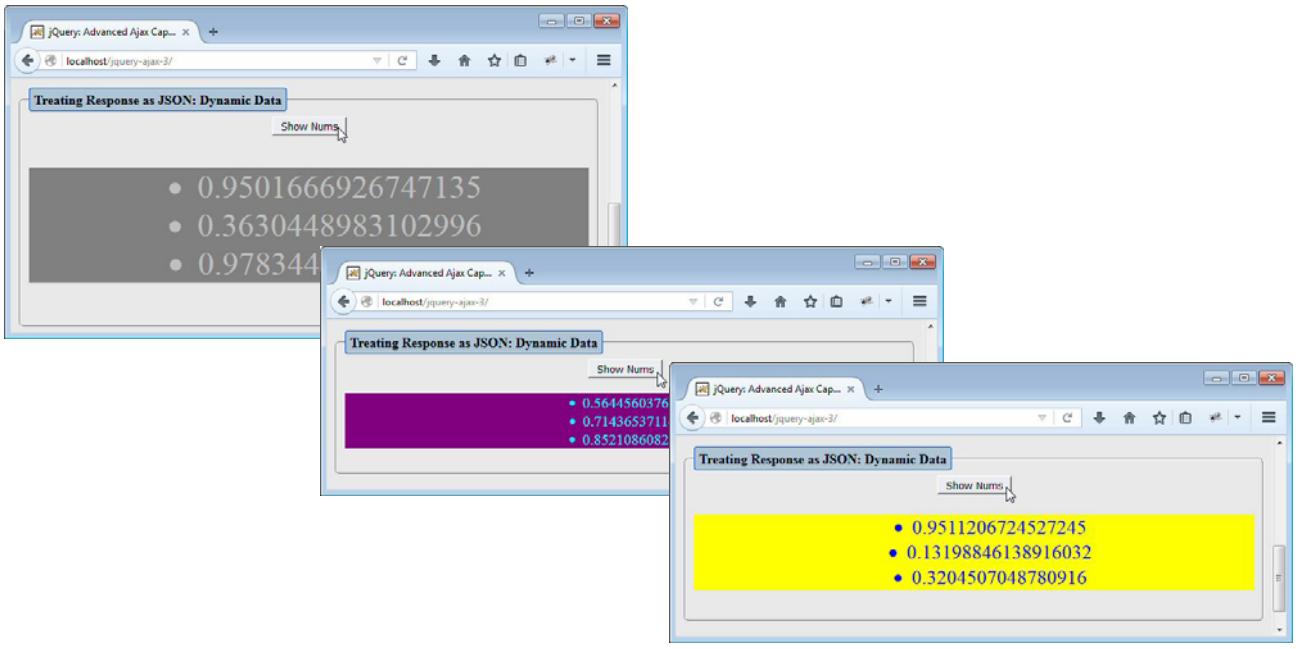
Dynamic JSON Example: Auxiliary Java Code

```
public class ColorUtils {
  private static String[] colors = {
    "aqua", "black", "blue", "fuchsia", "gray",
    "green", "lime", "maroon", "navy", "olive",
    "purple", "red", "silver", "teal", "white", "yellow" };

  /** One of the official HTML color names, at random. */

  public static String randomColor() {
    return(RandomUtils.randomElement(colors));
  }

  private ColorUtils() {}  // Uninstantiatable class}
}
```


JSON Example Code: Auxiliary Java Code

```
public class RandomUtils {
  private static Random r = new Random();

  public static int randomInt(int range) {
    return(r.nextInt(range));
  }

  public static int randomIndex(Object[] array) {
    return(randomInt(array.length));
  }

  public static <T> T randomElement(T[] array) {
    return(array[randomIndex(array)]);
  }
}
```

Dynamic JSON Example: Results



$.ajax Options

Overview

- **Options (almost) always used: url, success**

```
$.ajax({url: "some-address", success: someFunction});
```

  - success is not strictly required; you might want to just fire off some data to the server and not display anything

- **Common options: example**

```
$.ajax({
    url: "address",
    success: successHandlerFunction,
    data: { param1: "foo bar", param2: "baz“ },
    error: errorHandlerFunction,
    cache: false,
    dataType: "json",
    username: "resig",
    password: "scriptaculous-fan" });
```


Options

Name	Description	Default
async	Should the request be asynchronous? Use synchronous requests with caution since they lock up the browser.	true
beforeSend	Function to modify XMLHttpRequest object before it is sent (e.g., to set custom headers). The XHR is automatically passed to function.	*None*
cache	Is browser permitted to cache the page? Set to false if you use GET and you could get different responses back from the same data. Equivalent to having the server send Cache-Control: no-cache and Pragma: no-cache.	true (except false if dataType is script or json)
complete	Function to be called after error or success function is finished.	*None*
contentType	Content-Type of data sent to server. Rarely needed.	application/ x-www-form-urlencoded
data	Data to send to server (possibly after conversion). Sent in the appropriate place depending on whether it is GET or POST. Can be a String or an object. If a String, sent unchanged. If an object, property names become param names and property values get URL-encoded and become param values. & and = inserted automatically. If a value is an array, it is serialized with repeated param names.	*Empty*

Options (Continued)

Name	Description	Default
dataFilter	Response-data sanitizing function. Rarely used.	*None*
dataType	The format in which to pass the response to the handler function. Legal values are text, html (same as text except embedded scripts are run), xml, json, jsonp (JSON with Padding), and script (evaluates the response as JavaScript and returns it as plain text).	html, xml, or json *(makes intelligent guess)*
error	Function to be called if request fails. Function is passed 3 args: the XHR object, a string describing the error type, and an optional exception object. Possible values for the second argument are null, "timeout", "error", "notmodified" and "parsererror".	*None*
global	jQuery lets you set global defaults for various handlers: should they be used if set?	true
ifModified	Should the request be considered successful only if the response has changed since the last request (based on the Last-Modified header)?	false
jsonp	Override the callback function name in a jsonp request. JSONP is a JSON extension in which the name of a callback function is specified as an input argument of the call itself.	callback
password username	Username and password to be used in response to HTTP authentication request.	*None*

Options (Continued)

Name	Description	Default
processData	Should the value of the “data” property, if an object, be turned into a URL-encoded query string?	true
scriptCharset	Forces the request to be interpreted as a certain charset. Only for requests with dataType of “jsonp” or “script” and type of “GET”.	*None*
success	Function to be called if request succeeds. Function passed 3 args: the data returned from the server (formatted according to the dataType property), a string describing the status, and the XHR.	*None*
timeout	Timeout in milliseconds. If request takes longer, the error handler will be called instead of the success handler.	Global timeout, if set via $.ajaxSetup
traditional	Should data with arrays be serialized in traditional manner (shallow), or recursively (deep).	false
type	The HTTP method to use for the request. “get” and “post” are main options, but some browsers support other methods.	get
url	The address to request. Should be a relative URL.	*None*
xhr	Callback for creating your own custom XMLHttpRequest object.	ActiveXObject if available (IE), XMLHttpRequest otherwise

Ajax Shortcut Functions

Shortcuts for $.ajax: Equivalent Forms

- **$.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

Pros and Cons of Shortcuts

- **Advantages of shorthand functions**
  - Shorter and (arguably) clearer
- **Disadvantages of shorthand functions**
  - If you want additional options later, or you want to switch from GET to POST or vice versa, you have to change existing code more drastically
  - There is no direct argument for supplying error handler. Instead, you must call ajaxError to set up error handler
  - If you don't have data, you have to pass in null. This is less convenient than just omitting the "data" property entirely

```
$.get("url", null, someHandler);
```

vs.

```
$.ajax({url: "url", success: someHandler});
```


Wrap-Up

Summary

- **Showing temporary messages**

```
$(resultRegion).html(""); // Erase any previous results
$(workingRegion).show();
$.ajax({
   url: "server-time-slow.jsp",
   success: function(text) { $(resultRegion).html(text); },
   complete: function(text) { $(workingRegion).hide(); }
});
```

- **Dealing with JSON from the server**
  - Call to server with dataType: "json"
  - Have handler function take a JavaScript object, not a string
  - Build HTML or string based on the data
  - Insert that resultant string into the page