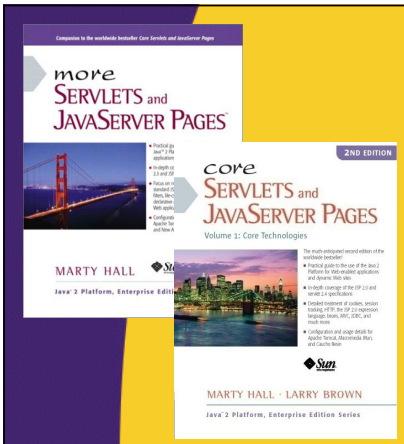
Ajax: Handling Different Server Data Formats

Basics: XML, JSON, and String

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html



Topics in This Section

- **Building HTML tables in JavaScript**
- **XML data**
  - Parsing results
  - Building XML data on server with MVC
- **JSON data**
  - Parsing results
  - Building JSON data on server with MVC
- **String data**
  - Parsing results
  - Building String data on server with MVC
- **Combination data**
  - Deciding what data format to use at run time

Data-Centric Ajax

Three Styles of Ajax

- **Content-Centric Ajax**
  - The server sends the exact HTML to be displayed
  - The client simply inserts the content into the page
    - But can use style sheets to somewhat customize the look
  - This was the approach used in the previous sections
- **Data-Centric Ajax**
  - The server sends raw data to the client
  - The client parses the data and builds the HTML
  - This is the approach used in this section
- **Script-Centric Ajax**
  - The server sends JavaScript functions to the client
  - The client executes the functions
  - This approach is not discussed in this tutorial
    - Inflexible: Requires the server to know too much about client

Data-Centric Ajax: Motivation

- **In many cases, the server data is intimately tied to a specific HTML form on the client**
  - In that case, it makes good sense for the server to send HTML tags and for the client to merely insert them
    - This is what we did previously (content-centric Ajax)
- **In other cases, the same server data may be used in several forms or in different pages**
  - And the data may be used in different ways by different applications
  - In that case, it makes sense for the server to send some standard data format
    - The client must parse (extract info from) this data format
    - The client must build HTML based upon the data

Review: Steps for Content-Centric Ajax

- **JavaScript**
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
    - Initiate a POST or GET request to a servlet
      - Put POST data in the send method
      - Data based on document.getElementById(id).value of some textfield
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract text with responseText
    - Use innerHTML to insert that exact text into designated element
- **HTML**
  - Load JavaScript from centralized directory
  - Designate control that initiates request
  - Give ids to input elements
  - Define a blank placeholder element with a known id

Content-Centric Ajax: Typical Approach

```
function ajaxResultPost(address, data, resultRegion) {
  var request = getRequestObject();
  request.onreadystatechange =
    function() { showResponseText(request,
                                  resultRegion); };
  request.open("POST", address, true);
  request.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
  request.send(data);
}

function showResponseText(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    htmlInsert(resultRegion, request.responseText);
  }
}
```

Steps for Data-Centric Ajax

- **JavaScript**
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
    - Initiate a POST or GET request to a servlet
      - Put POST data in the send method
      - Data based on document.getElementById(id).value of some textfield
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract data with responseText or responseXML
      - Build new text based on this data
    - Use innerHTML to insert that new text into designated element
- **HTML**
  - Load JavaScript from centralized directory
  - Designate control that initiates request
  - Give ids to input elements
  - Define a blank placeholder element with a known id

Data-Centric Ajax: Typical Approach

```
function ajaxResultPost(address, data, resultRegion) {
  var request = getRequestObject();
  request.onreadystatechange =
    function() { showResponseText(request,
                                  resultRegion); };
  request.open("POST", address, true);
  request.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
  request.send(data);
}

function showResponseText(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    var text = someStringBasedOn(request.responseText);
                                  // or request.responseXML
    htmlInsert(resultRegion, text);
  }
}
```

Building HTML Tables in JavaScript

Utility: Building HTML Tables

```
function getTable(headings, rows) {
  var table = "<table border='1' class='ajaxTable'>\n" +
              getTableHeadings(headings) +
              getTableBody(rows) +
              "</table>";
  return(table);
}
```

- **Note**
  - The first argument contains the headings
    - To be inserted into th elements
  - The second argument is an array-of-arrays, where each sub-array is a table row
    - The elements in the sub-arrays will be go in td elements

Utility: Building HTML Tables (Continued)

```
function getTableHeadings(headings) {
  var firstRow = "  <tr>";
  for(var i=0; i<headings.length; i++) {
    firstRow += "<th>" + headings[i] + "</th>";
  }
  firstRow += "</tr>\n";
  return(firstRow);
}

function getTableBody(rows) {
  var body = "";
  for(var i=0; i<rows.length; i++) {
    body += "  <tr>";
    var row = rows[i];
    for(var j=0; j<row.length; j++) {
      body += "<td>" + row[j] + "</td>";
    }
    body += "</tr>\n";
  }
  return(body);
}
```


Other Utilities (From Last Section)

```
// Insert the html data into the element
// that has the specified id.

function htmlInsert(id, htmlData) {
  document.getElementById(id).innerHTML = htmlData;
}

// Return escaped value of textfield that has given id.
// The builtin "escape" function url-encodes characters.

function getValue(id) {
  return(escape(document.getElementById(id).value));
}
```

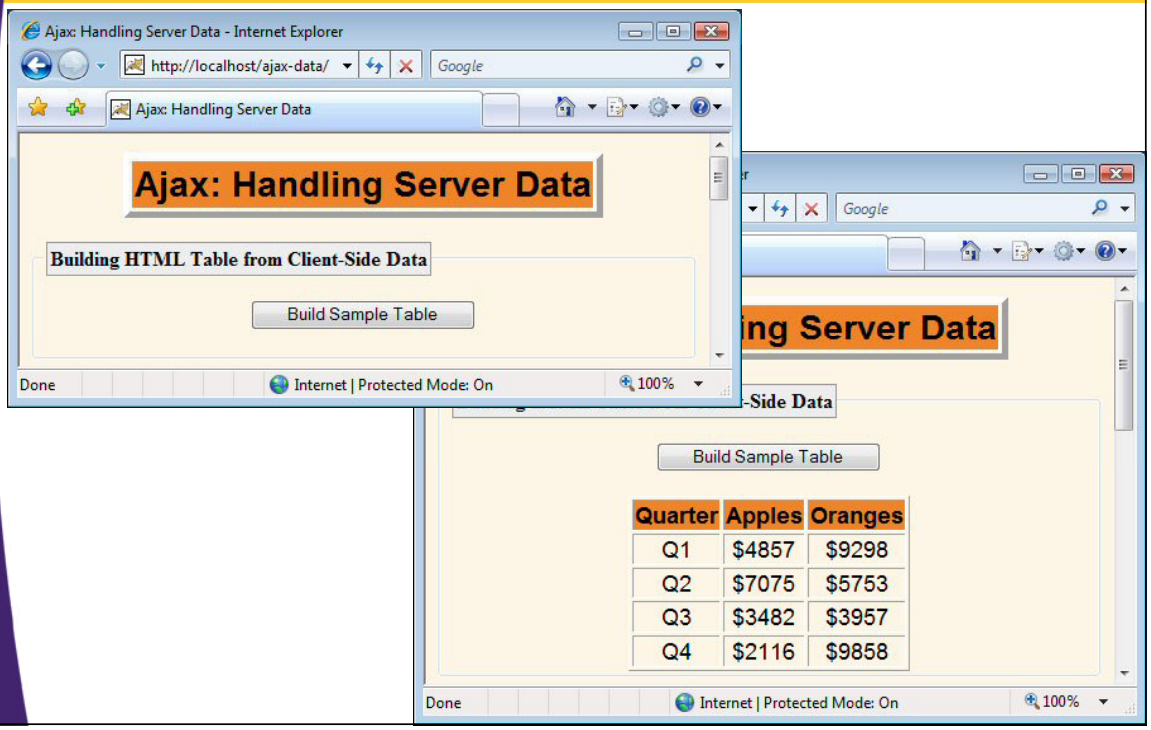
Example Usage (JavaScript)

```
// Build a table from purely client-side information.
// To test the getTable function.

function clientTable(displayRegion) {
  var headings = ["Quarter", "Apples", "Oranges"];
  var rows = [["Q1", randomSales(), randomSales()],
              ["Q2", randomSales(), randomSales()],
              ["Q3", randomSales(), randomSales()],
              ["Q4", randomSales(), randomSales()]];
  var table = getTable(headings, rows);
  htmlInsert(displayRegion, table);
}

function randomSales() {
  var sales = 1000 + (Math.round(Math.random() * 9000));
  return("$" + sales);
}
```


Example Usage (HTML)

```
...
<fieldset>
  <legend>
    Building HTML Table from Client-Side Data
  </legend>
  <form action="#">
   <input type="button" value="Build Sample Table"
          onclick='clientTable("client-table")'/>
  </form>
  <p/>
  <div id="client-table"></div>
</fieldset>
...
```

Example Usage (Result)



Handling XML Data

Basic Tasks in XML

- **How to treat the Ajax response as XML**
  - var xmlDoc = response.responseXML;
- **How to get an array of subelements**
  - xmlDoc.getElementsByTagName(…)
- **How to get an attribute of an element**
  - someElement.getAttribute(…)
- **How to get the body content of an element**
  - someElement.firstChild.nodeValue

Basic Tasks

- **Getting the raw XML data**
  - var xmlDocument = request.responseXML;
- **Finding array of XML elements**
  - xmlDocument.getElementsByTagName (xmlElementName);
- **Finding the text between start and end tags**
  - someElement.childNodes[0].nodeValue
    - Call the following at least once first: xmlDocument.documentElement.normalize();
- **Note**
  - In an earlier section we gave much more detail on XML manipulation in JavaScript

XML Utility Functions

```
// Given an element, returns the body content.

function getBodyContent(element) {
  element.normalize();
  return(element.firstChild.nodeValue);
}
```


XML Utility Functions

```
// Given a doc and the name of an XML element, returns an
// array of the values of all elements with that name.
// E.g., for
//   <foo><a>one</a><q>two</q><a>three</a></foo>
// getXmlValues(doc, "a") would return
//   ["one", "three"].

function getXmlValues(xmlDocument, xmlElementName) {
  var elementArray =
     xmlDocument.getElementsByTagName(xmlElementName);
  var valueArray = new Array();
  for(var i=0; i<elementArray.length; i++) {
     valueArray[i] = getBodyContent(elementArray[i]);
  }
  return(valueArray);
}
```

XML Utility Functions

```
// Given an element object and an array of sub-element names,
// returns an array of the values of the sub-elements.
// E.g., for <foo><a>one</a><c>two</c><b>three</b></foo>,
// if the element points at foo,
// getElementValues(element, ["a", "b", "c"]) would return
// ["one", "three", "two"]

function getElementValues(element, subElementNames) {
  var values = new Array(subElementNames.length);
  for(var i=0; i<subElementNames.length; i++) {
    var name = subElementNames[i];
    var subElement = element.getElementsByTagName(name)[0];
    values[i] = getBodyContent(subElement);
  }
  return(values);
}
```


General Utility Function (Update from Previous Section)

```
// Generalized version of ajaxResultPost. In this
// version, you pass in a response handler function
// instead of just a result region.

function ajaxPost(address, data, responseHandler) {
  var request = getRequestObject();
  request.onreadystatechange =
    function() { responseHandler(request); };
  request.open("POST", address, true);
  request.setRequestHeader
                 ("Content-Type",
                  "application/x-www-form-urlencoded");
  request.send(data);
}
```

General Utility Function (Same as in Previous Sections)

```
function getRequestObject() {
  if (window.XMLHttpRequest) {
    return(new XMLHttpRequest());
  } else if (window.ActiveXObject) {
    return(new ActiveXObject("Microsoft.XMLHTTP"));
  } else {
    return(null);
  }
}
```


Handling XML Data: Example

Steps

- **JavaScript**
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
    - Initiate a POST request to a servlet
      - Put POST data in the send method
      - Data based on document.getElementById(id).value of some textfield
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
      - Get text from XML with getElementsByTagName and firstChild.nodeValue
      - Build HTML table or other HTML data out of the text
    - Use innerHTML to insert result into designated element
- **HTML**
  - Load JavaScript from centralized directory
  - Designate control that initiates request
  - Give ids to input elements
  - Define a blank placeholder element with a known id

Initiate Request

```
function xmlCityTable(inputField, resultRegion) {
  var address = "show-cities";
  var data = "cityType=" + getValue(inputField) +
             "&format=xml";
  ajaxPost(address, data,
           function(request) {
             showXmlCityInfo(request, resultRegion);
           });
}
```

Handle Response

```
function showXmlCityInfo(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    var xmlDocument = request.responseXML;
    var headings = getXmlValues(xmlDocument, "heading");
    var cities = xmlDocument.getElementsByTagName("city");
    var rows = new Array(cities.length);
    var subElementNames = ["name", "time", "population"];
    for(var i=0; i<cities.length; i++) {
      rows[i] =
        getElementValues(cities[i], subElementNames);
    }
    var table = getTable(headings, rows);
    htmlInsert(resultRegion, table);
  }
}
```


HTML Code

```
...
<fieldset>
  <legend>Getting XML Data from Server, Building HTML Table</legend>
  <form action="#">
   <label for="city-type-1">City Type:</label>
   <select id="city-type-1">
     <option value="top-5-cities">Largest Five US Cities</option>
     <option value="second-5-cities">Second Five US Cities</option>
     <option value="cities-starting-with-s">
       US Cities Starting with 'S'</option>
     <option value="superbowl-hosts">
       Most Recent Superbowl Hosts</option>
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='xmlCityTable("city-type-1", "xml-city-table")'/>
  </form>
  <p/>
  <div id="xml-city-table"></div>
</fieldset>
...
```

Server Design: MVC

- **Logic**
  - Set the headers, read the request parameters, compute the results
  - Do this in Java (called by a servlet)
- **Presentation**
  - Build an XML file
  - Do this in JSP
    - Use the JSP expression language to access the results
- **Minor variation from usual MVC**
  - So that you can set Content-Type in servlet, use RequestDispatcher.include instead of RequestDispatcher.forward
- **Reminder**
  - Details on MVC and on the JSP expression language are given in other sections.
    - From the servlet and JSP tutorials

Servlet Code

```
public class ShowCities extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setHeader("Cache-Control", "no-cache");
    response.setHeader("Pragma", "no-cache");
    String cityType = request.getParameter("cityType");
    List<City> cities = CityUtils.findCities(cityType);
    request.setAttribute("cities", cities);
    String format = request.getParameter("format");
    String outputPage;
    if ("xml".equals(format)) {
      response.setContentType("text/xml");
      outputPage = "/WEB-INF/results/cities-xml.jsp";
    } ...
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
  }
```

Servlet Code (Continued)

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
  doGet(request, response);
}
```

- **I will use POST from the JavaScript**
  - But having GET support makes it easier to test interactively
  - So support both

Supporting Code (City.java)

- **Constructor**

```
public City(String name, int timeZone, int pop) {
  setName(name);
  setTimeZone(timeZone);
  setPop(pop);
}
```

- **Getter methods**
  - getName
  - getTime, getTimeZone
    - Assumes server is in US east coasts, subtracts 0-3 hours based on time zone
  - getPop
    - Raw population as an int
  - getPopulation
    - Formatted population as a String with commas

Supporting Code (CityUtils.java)

- **Map that associates city name with City**

```
private static Map<String,City> biggestAmericanCities =
  new HashMap<String,City>();
```

  - Populate it with 40 largest US cities

- **Lookup function**

```
public static City getCity(String name) {
  name = name.toUpperCase();
  return(biggestAmericanCities.get(name));
}
```


Supporting Code Continued (CityUtils.java)

- **Map that associates category of cities with List of City**

```
private static Map<String,String[]> cityTypeMap;
```

- **Lookup function**

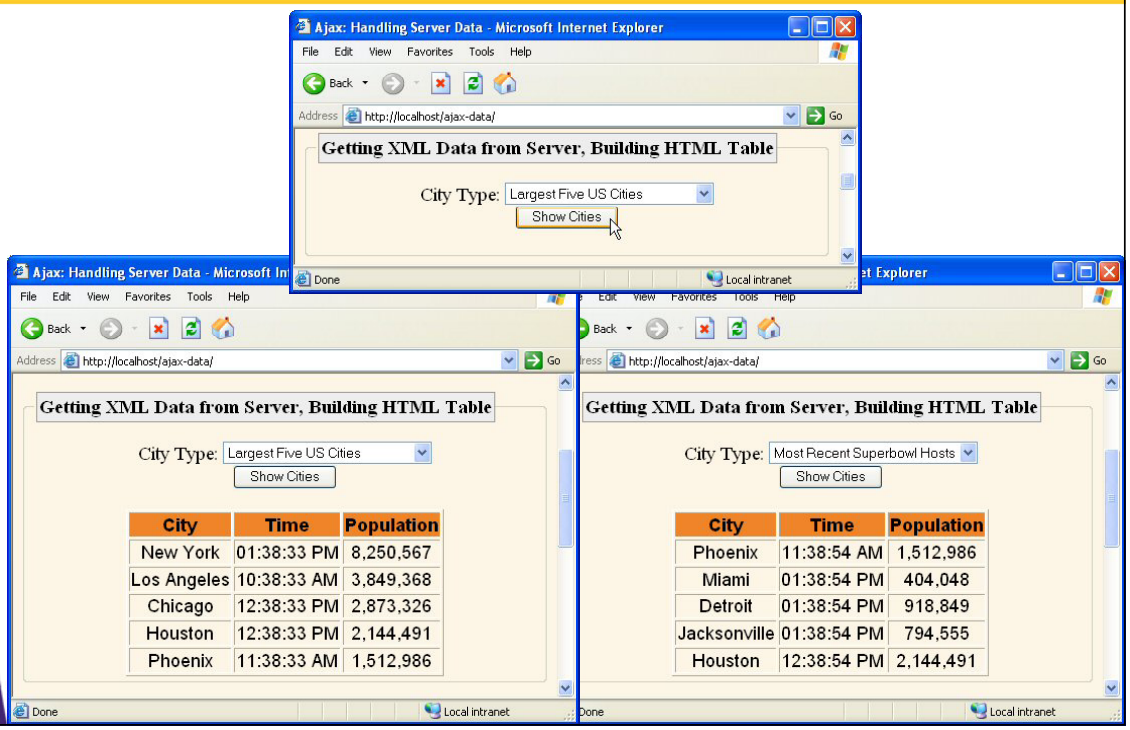
```
public static List<City> findCities(String cityType) {
  String[] cityNames = cityTypeMap.get(cityType);
  if (cityNames == null) {
    String[] twoCities = { "New York", "Los Angeles" };
    cityNames = twoCities;
  }
  List<City> cities = new ArrayList<City>();
  for(String cityName: cityNames) {
    cities.add(getCity(cityName));
  }
  return(cities);
}
```

JSP Code (/WEB-INF/results/cities-xml.jsp)

```
<?xml version="1.0" encoding="UTF-8"?>
<cities>
 <headings>
    <heading>City</heading>
    <heading>Time</heading>
    <heading>Population</heading>
  </headings>
  <city>
    <name>${cities[0].name}</name>
    <time>${cities[0].time}</time>
    <population>${cities[0].population}</population>
  </city>
  ...
  <city>
    <name>${cities[4].name}</name>
    <time>${cities[4].time}</time>
    <population>${cities[4].population}</population>
  </city>
</cities>
```

Three more cities (omitted to make space on slide)

XML Data: Results



Major Flaw in Design

- **Client-side code (good)**
  - Can handle any number of city entries
    - I.e., any number of entries in array that represents the table rows
- **Servlet code (good)**
  - Can handle any number of City objects
    - Just stores List<City> in request scope
- **JSP code (bad)**
  - Problems
    - Must know how many cities there are
    - Repeats description for each city
  - Solution
    - JSTL (covered in upcoming section)

Handling JSON Data

Basic Tasks

- **JSON**
  - JavaScript Object Notation. A simple textual representation of JavaScript objects that is already directly supported in JavaScript.
  - More details will be provided in later section
- **Directly in JavaScript**
  - var someObject =
    { property1: value1,
    property2: value2, ... };
- **In a string (e.g., when coming in on network)**
  - Surround object representation in parens
  - Pass to the builtin "eval" function

Basic Tasks: Details

- **Main object**
  - Surround entire value in curly braces
  - Put field names in single or double quotes
    - Quotes can be omitted if field name is legal variable name
  - Use colons between field names and values
  - Put commas after each fieldname: fieldvalue pair.
- **Field values**
  - Strings: use single or double quotes
  - Numbers: no quotes needed
  - Arrays: use comma-separated values inside *square* braces
- **Putting JSON in strings**
  - Enclose in parens and quotes
    - Use single quotes on the outside if you have double quotes inside
  - Pass result to "eval" to get an object back

Basic Tasks: Example

```
var firstObject =
  { field1: "string-value1",
    field2: 3,
    field3: ["a", "b", "c"]
   };
var someString =
  '({ f1: "val1", f2: "val2" })';
var secondObject = eval(someString);
```

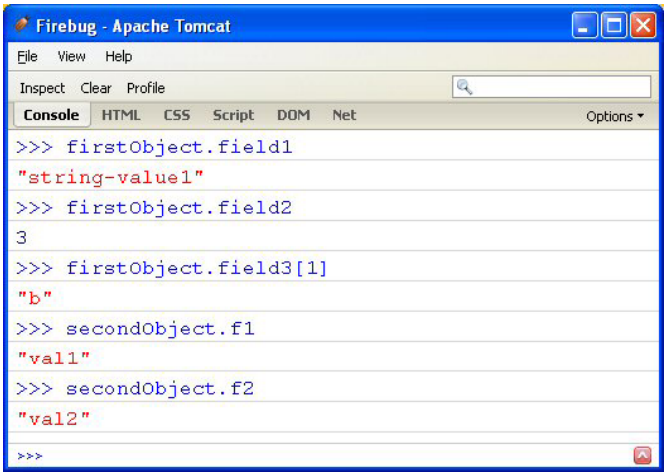
- **Results**
  - firstObject.field1 → "string-value1"
  - firstObject.field2 → 3
  - firstObject.field3[1] → "b"
  - secondObject.f1 → "val1"
  - secondObject.f2 → "val2"

Testing

- **Don't use HTML: use Firebug**
  - Open Firebug
    - F12 or Tools → Firebug → Open Firebug
  - Go to the Console
  - Cut/paste the expressions into the command line
    - Either at the bottom or the right, depending on Options
- **Reminder**
  - Firebug is indispensible for Ajax development and testing
  - Download from http://getfirebug.com/
  - For details, see "Ajax Development Tools" section

Testing in Firebug: Example

- **Steps**
  - Opened Firebug with F12
  - Cut/pasted code from earlier slide
  - Interactively entered the expressions shown in blue



More on JSON

- **This section**
  - Constructs JSON explicitly using MVC
  - Uses normal servlets
  - Reads request parameters as strings
- **Upcoming sections**
  - Constructs JSON automatically from Java objects
  - Uses RPC approach to
    - Hide the use of normal servlets
    - Pass ordinary arguments instead of request parameter strings
- **Earlier section (JavaScript Core)**
  - Gives more examples of basic JSON usage in ordinary JavaScript programs

Handling JSON Data: Example

Steps

- **JavaScript**
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
    - Initiate a POST request to a servlet
      - Put POST data in the send method
      - Data based on document.getElementById(id).value of some textfield
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
      - Add parens then pass string to "eval" to get a real JavaScript object
      - Access fields, array elements, etc., with normal JavaScript syntax
    - Use innerHTML to insert result into designated element
- **HTML**
  - Load JavaScript from centralized directory
  - Designate control that initiates request
  - Give ids to input elements
  - Define a blank placeholder element with a known id

Initiate Request

```
function jsonCityTable(inputField, resultRegion) {
  var address = "show-cities";
  var data = "cityType=" + getValue(inputField) +
             "&format=json";
  ajaxPost(address, data,
           function(request) {
             showJsonCityInfo(request, resultRegion);
           });
}
```


Handle Response

```
function showJsonCityInfo(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    var rawData = request.responseText;
    var data = eval("(" + rawData + ")");
    var table = getTable(data.headings, data.cities);
    htmlInsert(resultRegion, table);
  }
}
```

HTML Code

```
...
<fieldset>
  <legend>Getting JSON Data from Server, Building HTML Table
  </legend>
  <form action="#">
   <label for="city-type-2">City Type:</label>
   <select id="city-type-2">
     <option value="top-5-cities">Largest Five US Cities</option>
     <option value="second-5-cities">Second Five US Cities</option>
     <option value="cities-starting-with-s">
       US Cities Starting with 'S'</option>
     <option value="superbowl-hosts">
       Most Recent Superbowl Hosts</option>
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='jsonCityTable("city-type-2",
                                 "json-city-table")'/>
  </form>
  <p/>
  <div id="json-city-table"></div>
</fieldset>...
```


Server Design: MVC

- **Logic**
  - No changes to basic logic
  - Only addition is logic to decide which results page applies
- **Presentation**
  - Build a plain-text page instead of an XML page
  - Embed data in JSON format

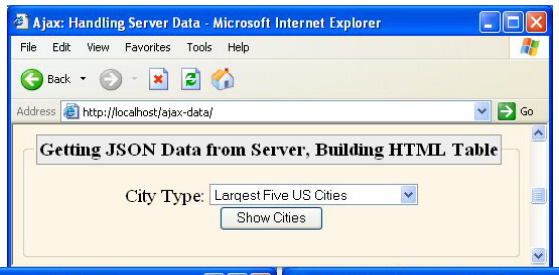
Servlet Code

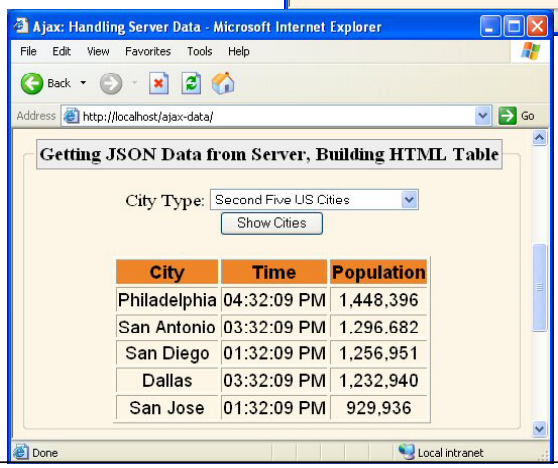
```
public class ShowCities extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    ...
    request.setAttribute("cities", cities);
    String format = request.getParameter("format");
    String outputPage;
    if ("xml".equals(format)) {
      response.setContentType("text/xml");
      outputPage = "/WEB-INF/results/cities-xml.jsp";
    } else if ("json".equals(format)) {
      response.setContentType("application/json");
      outputPage = "/WEB-INF/results/cities-json.jsp";
    } ...
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
  }
```

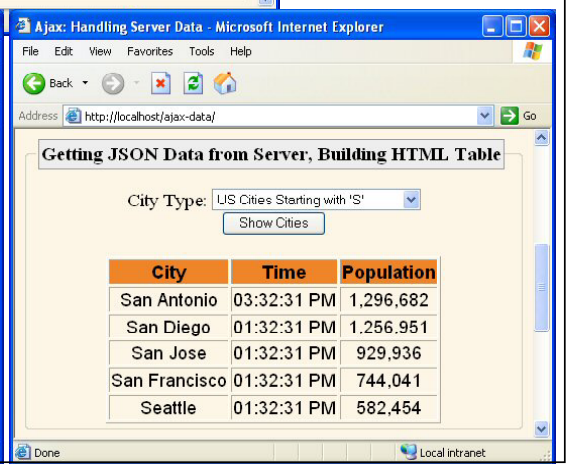

JSP Code (/WEB-INF/results/cities-json.jsp)

```
{ headings: ["City", "Time", "Population"],
  cities: [["${cities[0].name}", "${cities[0].time}",
            "${cities[0].population}"],
           ["${cities[1].name}", "${cities[1].time}",
            "${cities[1].population}"],
           ["${cities[2].name}", "${cities[2].time}",
            "${cities[2].population}"],
           ["${cities[3].name}", "${cities[3].time}",
            "${cities[3].population}"],
           ["${cities[4].name}", "${cities[4].time}",
            "${cities[4].population}"]]
}
```

JSON Data: Results





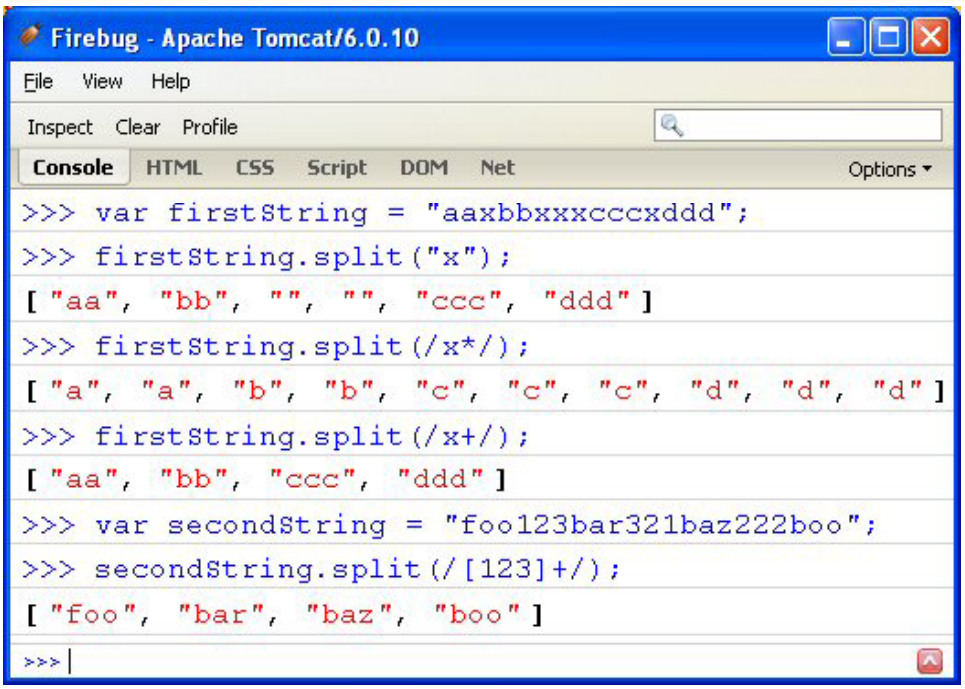


Handling String Data

Basic Tasks

- **General Approach**
  - Server-side code invents a custom data format
  - Client-side code parses it
- **Specific Common Approach**
  - Server-side code sends delimited strings
  - Client-side code uses String.split to break strings into arrays
- **String.split in JavaScript**
  - Quite similar to String.split in Java
  - With a one-char delimiter, use single or double quotes
  - With a regular expression, use slashes
    - JavaScript regex's similar to Perl (and Java) regular expressions
    - More details will be given in a later section
- **Online references**
  - http://www.evolt.org/article/Regular_Expressions_in_JavaScript/17/36435/
  - http://www.javascriptkit.com/javatutors/re.shtml

String.split: Example



Handling String Data: Example

Steps

- **JavaScript**
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
    - Initiate a POST request to a servlet
      - Put POST data in the send method
      - Data based on document.getElementById(id).value of some textfield
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
      - Break it into array with String.split and regular expression delimiters
      - Access array elements (perhaps using String.split again)
    - Use innerHTML to insert result into designated element
- **HTML**
  - Load JavaScript from centralized directory
  - Designate control that initiates request
  - Give ids to input elements
  - Define a blank placeholder element with a known id

Initiate Request

```
function stringCityTable(inputField, resultRegion) {
  var address = "show-cities";
  var data = "cityType=" + getValue(inputField) +
             "&format=string";
  ajaxPost(address, data,
           function(request) {
             showStringCityInfo(request, resultRegion);
           });
}
```


Handle Response

```
function showStringCityInfo(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    var rawData = request.responseText;
    var rowStrings = rawData.split(/[\n\r]+/);
    var headings = rowStrings[0].split("#");
    var rows = new Array(rowStrings.length-1);
    for(var i=1; i<rowStrings.length; i++) {
      rows[i-1] = rowStrings[i].split("#");
    }
    var table = getTable(headings, rows);
    htmlInsert(resultRegion, table);
  }
}
```

HTML Code

```
...
<fieldset>
  <legend>Getting String Data from Server, Building HTML Table
  </legend>
  <form action="#">
   <label for="city-type-3">City Type:</label>
   <select id="city-type-3">
     <option value="top-5-cities">Largest Five US Cities</option>
     <option value="second-5-cities">Second Five US Cities</option>
     <option value="cities-starting-with-s">
       US Cities Starting with 'S'</option>
     <option value="superbowl-hosts">
       Most Recent Superbowl Hosts</option>
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='stringCityTable("city-type-3",
                                    "string-city-table")'/>
  </form>
  <p/>
  <div id="string-city-table"></div>
</fieldset>...
```


Server Design: MVC

- **Logic**
  - No changes to basic logic
  - Only addition is logic to decide which results page applies
- **Presentation**
  - Build a plain-text page
  - Embed data between delimiters

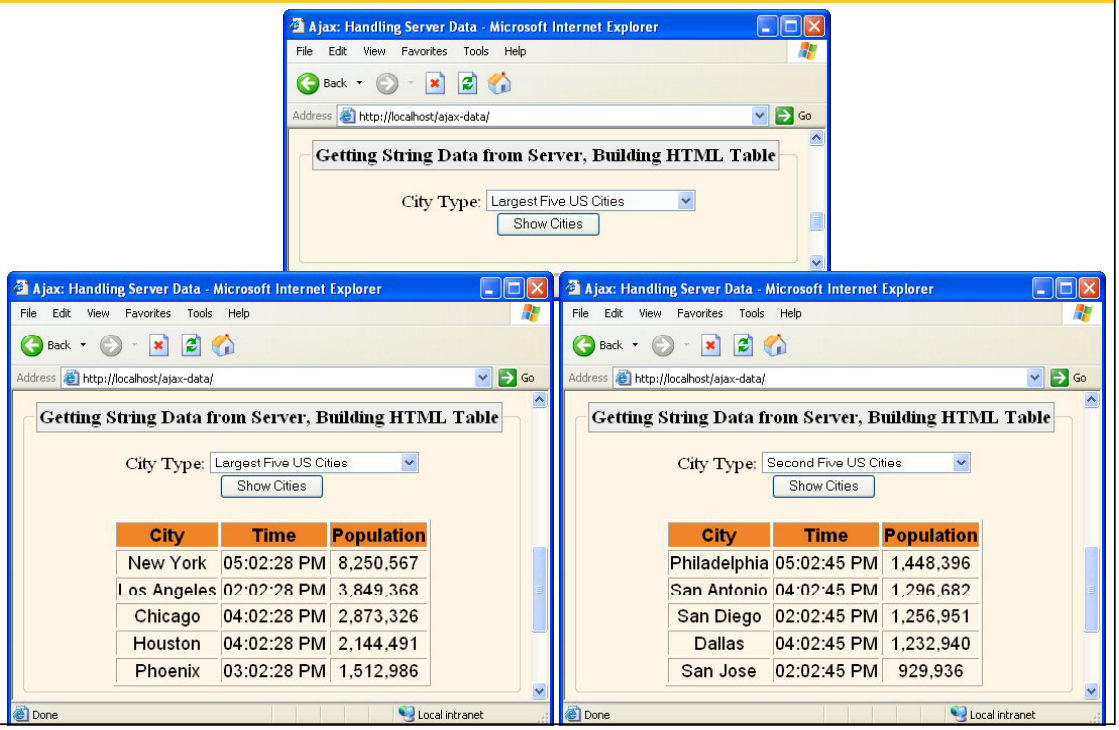
Servlet Code

```
public class ShowCities extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    ...
    if ("xml".equals(format)) {
      response.setContentType("text/xml");
      outputPage = "/WEB-INF/results/cities-xml.jsp";
    } else if ("json".equals(format)) {
      response.setContentType("application/json");
      outputPage = "/WEB-INF/results/cities-json.jsp";
    } else {
      response.setContentType("text/plain");
      outputPage = "/WEB-INF/results/cities-string.jsp";
    }
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
  }
```


JSP Code (/WEB-INF/results/cities-string.jsp)

```
City#Time#Population
${cities[0].name}#${cities[0].time}#${cities[0].population}
${cities[1].name}#${cities[1].time}#${cities[1].population}
${cities[2].name}#${cities[2].time}#${cities[2].population}
${cities[3].name}#${cities[3].time}#${cities[3].population}
${cities[4].name}#${cities[4].time}#${cities[4].population}
```

String Data: Results



Combination Data

Idea

- **Earlier**
  - Server
    - Decided what datatype to return based on “format” parameter
  - Client
    - Hardcoded “format” value
    - Hardcoded response handler function
- **Now**
  - Server
    - No change. Still uses “format” param in same way.
  - Client
    - Gets “format” value from combobox
    - Decides on response handler function based on combobox value

JavaScript

```
function cityTable(cityTypeField, formatField,
                   resultRegion) {
  var address = "show-cities";
  var cityType = getValue(cityTypeField);
  var format = getValue(formatField);
  var data = "cityType=" + cityType +
             "&format=" + format;
  var responseHandler = findHandler(format);
  ajaxPost(address, data,
           function(request) {
             responseHandler(request, resultRegion);
           });
}

function findHandler(format) {
  if (format == "xml") {            // == is ok for strings!
    return(showXmlCityInfo);
  } else if (format == "json") {
    return(showJsonCityInfo);
  } else {
    return(showStringCityInfo);
  }
}
```

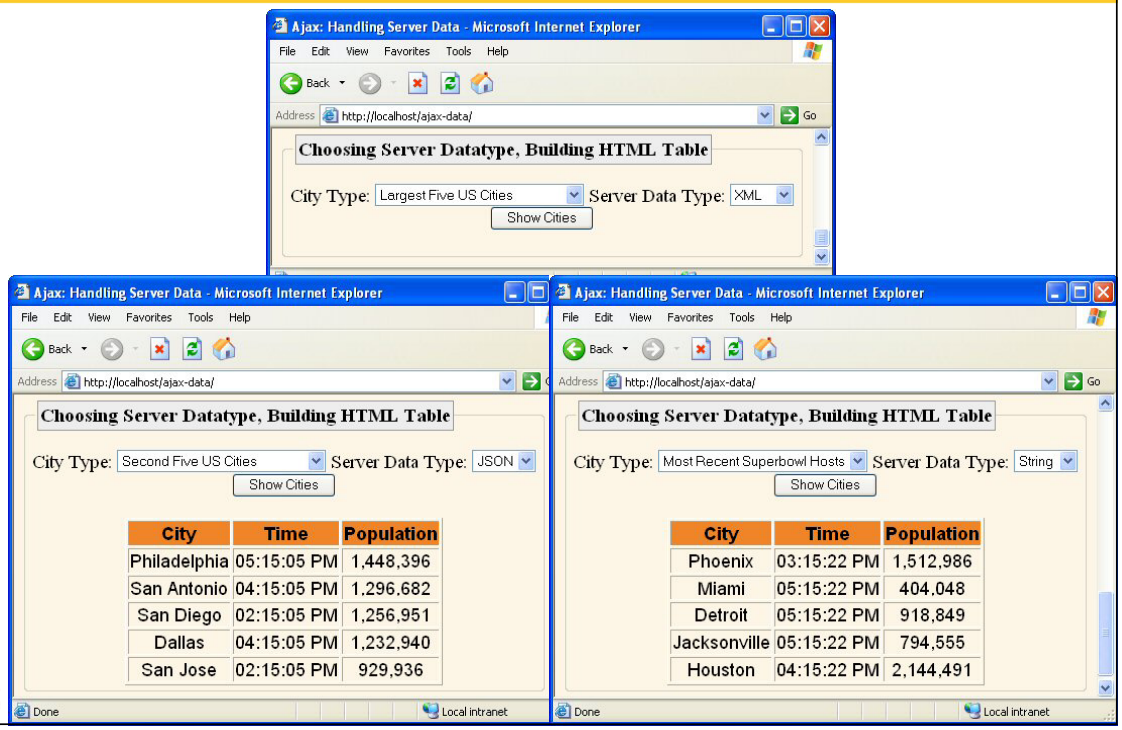
HTML

```
<fieldset>
  <legend>Choosing Server Datatype...</legend>
  <form action="#">
   <label for="city-type-4">City Type:</label>
   <select id="city-type-4">
     <option value="top-5-cities">Largest Five ...</option>
     ...
    </select>
   <label for="data-type">Server Data Type:</label>
   <select id="data-type">
     <option value="xml" selected="selected">XML</option>
     <option value="json">JSON</option>
     <option value="string">String</option>
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='cityTable("city-type-4", "data-type",
                             "city-table")'/>
  </form>
  <p/>
  <div id="city-table"></div>
</fieldset>
```


Server-Side Code

- **No changes whatsoever**

Combination Data: Results



Wrap-up

Preview of Next Sections

- **Problems with JSP pages in this section**
  - Repeated identical information for each of the five entries in the list of cities
  - Cannot handle city lists that are not five entries long
    - Servlet code was independent of the number of cities
    - Client-side (JavaScript) code was independent of the number of cities
    - But JSP code was hard-coded to the number of cities
- **Handling variable-length data in JSP**
  - Bean
  - Custom tag
  - JSTL loop
  - Scripting loop

Preview of Next Sections (Continued)

- **Design**
  - Returning real City objects instead of strings is more in keeping with data-centric approach
- **Automatic JSON generation**
  - Building JSON objects automatically from various Java data types
- **RPC mechanism**
  - Let client side code act as if it is calling a server-side function (not a URL)
  - Let client side code pass and receive regular arguments
  - Let server-side code pass and receive regular arguments
    - Return results converted with JSONObject & JSONArray

Summary

- **Parsing XML data**
  - Call request.responseXML
  - Call getElementsByTagName
  - Get body text via someElement.firstChild.nodeValue
- **Parsing JSON data**
  - Add parens
  - Pass to eval
  - Treat as normal JavaScript object
- **Parsing string data**
  - Use String.split and (possibly) regular expressions
- **Server**
  - Use MVC

Questions?