# Rich Internet Applications

5. Ajax and services

## Ajax patterns in this lecture

- **Programming Patterns**
  - *Web Services*
    - **RESTful Service**
    - **RPC Service**
  - *Browser Server dialogue*
    - **Cross-Domain Proxy**
- **Foundational Technology Patterns**
  - *Web Remoting*
    - **XmlHttpRequest Call**
    - **On-demand JavaScript/JSONP**
    - **iFrame Call**



## Web services

- The term refers to clients and servers that communicate over the HTTP protocol
- Can be used as a means of communicating between components within a web application, as in the examples you have seen
- Can be a set of public services that together form a reliable, scalable, and inexpensive computing platform “in the cloud”*

*This description is from Amazon Web Services

## RESTful service pattern

- **Forces**
  - The browser side of an Ajax App needs to call server-side services
  - It's often desirable for third-party applications to invoke the server-side service too.
  - With numerous different applications—and a variety of developers—accessing the service, it ought to be easy to use.
  - The basic architecture of the Web does not force any type of service architecture—any given functionality could be exposed in a wide variety of styles.
- **Solution**
  - Expose web services according to RESTful principles...

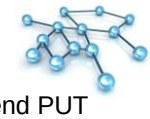


## RESTful principles

- Representational State Transfer (REST) is a term coined by Roy Fielding in his Ph.D. dissertation to describe an **architecture style**
- A style, not a standard
- Principles:
  - Resources as URLs
  - Operations as HTTP methods (GET, POST, PUT, DELETE)
  - GET for queries and only for queries
  - Other methods to change state of resource



## RESTful requests



- Most browsers do not currently send PUT and DELETE requests
- Can configure XHR to send any method, though
- Ideally URLs should simply look like paths, with no parameters
  - /destinations/asia ☑
  - /destinations.svc?destination=Asia ☒

## RESTful URLs

- **GET /users** - return a list of all records
- **GET /users /new** - return a form for creating a new record
- **POST /users** - submit fields for creating a new record
- **GET /users** /1 return the first record
- **DELETE/users /1** – delete the first record
- **GET /users** /1/edit return a form to edit the first record
- **PUT/users /1** - submit fields for updating the first record



## Public REST web service example

- Geonames offers a range of web services providing geographic information

**Find nearby postal codes / reverse geocoding**

This service comes in two flavors. You can either pass the lat/long or a postalcode/placename.

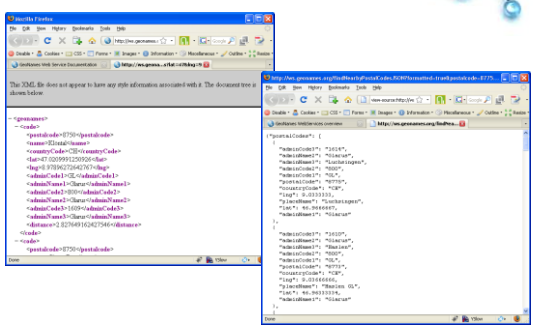
Webservice Type : REST
Url : ws.geonames.org/findNearbyPostalCodes?
Parameters :
lat,lng, radius (in km), maxRows (default = 5),style (verbosity : *SHORT,MEDIUM,LONG,FULL*), country (default = all countries)
or
postalcode,country, radius (in Km), maxRows (default = 5)
Result : returns a list of postalcodes and places for the lat/lng query as xml document
Example:
http://ws.geonames.org/findNearbyPostalCodes?lat=47&lng=9
or
ws.geonames.org/findNearbyPostalCodes?postalcode=8775&country=CH&radius=10

This service is also available in JSON format :
ws.geonames.org/findNearbyPostalCodesJSON?postalcode=8775&country=CH&radius=10

XML and JSON versions available



## Public REST web service example

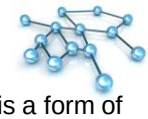


## RPC service pattern

- **Forces**
  - The browser side of an Ajax App needs to call server-side services
  - It's often desirable for third-party applications to invoke the server-side service too.
  - With numerous different applications—and a variety of developers—accessing the service, it ought to be easy to use.
  - The basic architecture of the Web does not force any type of service architecture—any given functionality could be exposed in a wide variety of styles.
- **Solution**
  - Expose web services as Remote Procedure calls



## RPC services



- A Remote Procedural Call (RPC) is a form of communication where the client invokes a remote procedure on the server
- RPCs are generally characterized as actions
- The URL is usually verb-like; e.g., **/Tourdates/getTourDates?destination=USA**
- Sometimes known as "Big web services"
- Examples: SOAP, XML-RPC



## SOAP services

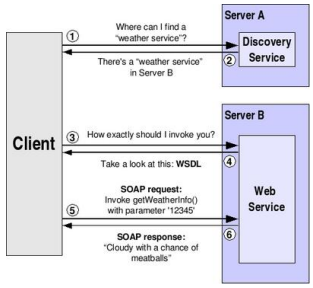
- Simple Object Access Protocol
- Protocol for exchanging XML messages over HTTP
- Procedure call encoded as XML message to server
- Procedure return value encoded as XML message to client
- XML messages are SOAP packets

## SOAP services

- Web Services Description Language (WSDL) gives public description of the methods available on a service
- Discovery services advertise available web services
- Client frameworks use WDSL information to build a client-side stub procedure
- Calling stub procedure on client passes message to server and collects response for use in client-side code



## SOAP information flow






## SOAP request

```
POST /gcutours/Tourdates.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/getTourDates"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getTourDates xmlns="http://tempuri.org/">
      <destination>USA</destination>
    </getTourDates>
  </soap:Body>
</soap:Envelope>
```

operation

parameter

## SOAP request

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <getTourDatesResponse xmlns="http://tempuri.org/">
      ...data returned by operation...
      </getTourDatesResponse>
   </soap:Body>
</soap:Envelope>
```

Data can be text, XML or JSON written to response by service or can be a representation of a server object



## Public SOAP web services

- Some large web services providers prefer to concentrate on REST services
  - e.g. Yahoo
- Others offer SOAP and REST services
- e.g. Amazon
- Sites which list SOAP services from a range of providers:
  - www.xmethods.net
  - www.seekda.com



## Comparing REST and RPC services

- RPC approach, e.g. SOAP, adds complexity and network overhead
- Requires creation of client-side stub, although available frameworks make this fairly straightforward
- REST approach allows services to be called with simple HTTP requests
- Easy to consume service from client-side JavaScript

## Cross-origin requests

- Same-origin policy
- Most browsers disallow cross-domain, or cross-origin XmlHttpRequest calls by default
- This means that a page cannot directly call a public or third-party web service with this technique
- Need to find ways of implementing origin calls



## CORS

- Cross-origin Resource Sharing
- Mechanism by which browsers allow cross-origin XHRs, with safeguards
- Transparent to client code
- W3C Working Draft July 2010
- Supported in Chrome, Firefox 3.5+, Safari 4+, IE 10
- IE8/9 support XDomainRequest object which has similar purpose



## Simple CORS request

- GET or POST request
- Browser sends **Origin** header identifying current domain
- If server decides request should be allowed it sends **Access-Control-Allow-Origin header**
  - Can be a specific domain or *
- If this header is missing or the origins don't match, then browser disallows the request
- If all is well, then the browser processes the response

## Simple CORS request

Params | Headers | Response | JSON

Response Headers

Access-Control-Allow-Orig... *
Cache-Control no-cache
Connection Keep-Alive
Content-Type application/json;charset=UTF-8
Date Sun, 12 Feb 2012 20:44:03 GMT
Keep-Alive timeout=15, max=100
Server Apache/2.2.21 (Linux/SUSE)
Transfer-Encoding chunked

Request Headers

Accept application/json, text/javascript, */*; q=0.01
Accept-Encoding gzip, deflate
Accept-Language en-gb,en;q=0.5
Connection Keep-alive
Host ws.geonames.org
Origin http://localhost:1593
Referer http://localhost:1593/Home/PlacenameXhr
User-Agent Mozilla/5.0 (Windows NT 6.1; rv:10.0) Gecko/20100101 Firefox/10.0



## CORS with pre-flight

- CORS allows the use of custom headers, methods other than GET or POST through a transparent mechanism of server verification called preflighted requests
- When you try to make a request with one of the advanced options, a “preflight” request is made to the server
- This request uses the OPTIONS method and sends additional headers
- Still transparent to client code



## Pre-flight request headers

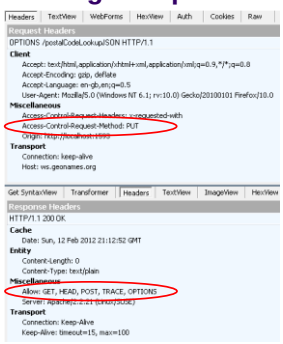
- **Origin** – same as in simple requests.
- **Access-Control-Request-Method** – the method that the request wants to use.
- **Access-Control-Request-Headers** – (Optional) a comma separated list of the custom headers being used.

## Pre-flight response

- The server communicates whether it will accept this request by sending the following headers in the response:
  - **Access-Control-Allow-Origin**
  - **Access-Control-Allow-Methods** – a comma separated list of allowed methods.
  - **Access-Control-Allow-Headers** – a comma separated list of headers that the server will allow.
  - **Access-Control-Max-Age** – the amount of time in seconds that this preflight request should be cached for



## Pre-flight request




- Use Fiddler to examine headers
- PUT method was not allowed in this example



## JSONP

- JSON with Padding
- Dynamic script loading
- Requires a web service which:
  - Returns data as JSON
  - Supports dynamic scripting by allowing a callback function to be specified
- Look at the example of a **Geonames** service which returns geographic data for a specified postcode or place name

## JSON web service example

**URL**
http://ws.geonames.org/postalCodeLookupJSON?formatted=true&postalcode=G4&country=UK&style=full

**Response**

```
{"postalcodes": [{
  "postalcode": "G4",
  "countryCode": "GB",
  "lng": -4.25,
  "placeName": "Glasgow",
  "lat": 55.8333333
}]}
```



## JSON web service example

**URL**
http://ws.geonames.org/postalCodeLookupJSON?formatted=true&postalcode=G4&country=UK&style=full**&callback=getGeo**

add name of callback function to request

**Response**

```
getGeo({"postalcodes": [{
  "postalcode": "G4",
  "countryCode": "GB",
  "lng": -4.25,
  "placeName": "Glasgow",
  "lat": 55.8333333
}]});
```

web service is programmed to wrap JSON as a **parameter** of a call to a function with the name specified in the request



## Calling the JSON service dynamically

- To make use of this our page needs JavaScript to do the following:
  - Build a URL for the web service
  - URL may include data input by user
  - Dynamically add a new <script> element to the DOM with the URL as the SRC
  - Provide a callback function with the name specified in the URL

## How does this work?

- Any part of the page DOM can be manipulated dynamically, so it is possible to add a new <script> element

```
<script type="text/javascript" src="http://ws.geonames.org/postalCodeLookupJSON
?formatted=true&postalcode=G4&country=UK&style=full&callback=getGeo" />
```

- This is a bit like being able to add a new method to a Java program while it is running

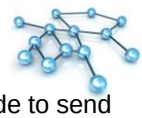


## Immediate execution

- Normally, **src** attribute of a script tag retrieves a library of JavaScript **function declarations**
- *This* web service returns a **function call** (to the callback) rather than function declarations
- A function call is an **expression** and is executed immediately
- Function can make use of the JSON data which the web service has returned because the data is a parameter of the function call



## What's the trick?



- The trick is that the browser is made to send a **request to a web service** by disguising its URL as a **source of JavaScript code**
- Not an XmlHttpRequest, so no domain restrictions
- The web service helps by looking at the **callback** parameter in the URL and wrapping its data in a **function call** with that name
- The web service therefore returns an **expression** which is executed immediately

## What's the catch?

- A possible issue is that we are allowing the browser to execute whatever code it gets back from the service
- This is a bit like using the eval function on JSON data returned from a service
- Potential security vulnerability through executing arbitrary code, although risk is low as you are consuming code from a known third party



## JSONP with jQuery example

```
function getPlacename() {
        var postalcode = $('#postalcode').val();
        var country = $('#country').val();
        var url = 'http://ws.geonames.org/postalCodeLookupJSON';
        $.ajax({
            url: url,
            data: { formatted: "true", postalcode: postalcode,
                    country: country, style: "full" },
            dataType: "jsonp",
            jsonpCallback: "jsonpcallback",
        });
}
```

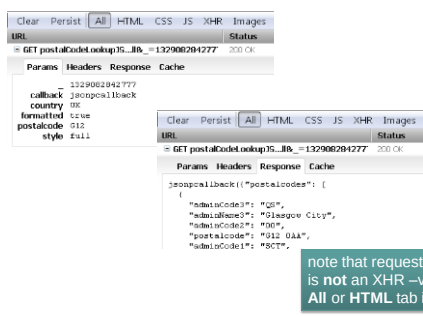
get user input

set data type

name of callback to wrap data

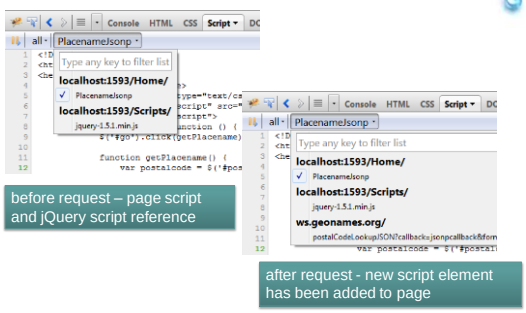


## JSONP with jQuery example




note that request to web service is **not** an XHR –view it in **All** or **HTML** tab in Firebug

## JSONP with jQuery example




before request – page script and jQuery script reference

after request - new script element has been added to page



## On-demand JavaScript

- Dynamic scripting is a special case of the On-demand JavaScript pattern
- Script tags can also be created dynamically to load JavaScript libraries when needed
- Avoids overhead of loading a large amount of JavaScript over the network when page loads
- Avoids loading code which may never be required depending on what the user does with the page



## Comparing CORS and JSONP

- XHR with CORS is safer and more flexible than JSONP
- Allows control of request type and headers
- Not executing code from an external source
- Main motivation for using JSONP is current level of browser support – will work in a wider range of browsers

## JSONP support in WCF 4

```
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true"/>
      <serviceDebug includeExceptionDetailInFaults="true"/>
    </behavior>
  </serviceBehaviors>
  <endpointBehaviors>
    <behavior name="webHttpBehavior">
      <webHttp />
    </behavior>
  </endpointBehaviors>
</behaviors>
<bindings>
    <webHttpBinding>
        <binding name="webHttpBindingWithJsonP" crossDomainScriptAccessEnabled="true" />
    </webHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior" name="AjaxDestinations.PackagesService">
    <endpoint address="" binding="webHttpBinding"
              bindingConfiguration="webHttpBindingWithJsonP" contract="AjaxDestinations.IPackagesService"
              behaviorConfiguration="webHttpBehavior"/>
  </service>
</services>
```



## WCF JSONP example

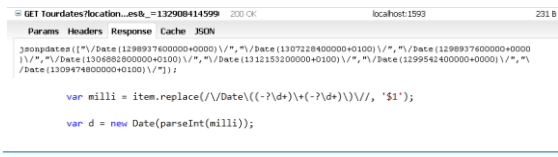
- jQuery.ajax will send XHR if URL is not remote unless **crossDomain** property = true

```
function getTourDates() {
        var selectedDestination = $('#destinationlist').val();
        var url = "http://localhost:1593/PackagesService.svc/Tourdates?
          location=" + escape(selectedDestination);

        $.ajax({
            url: url,
            dataType: "jsonp",
            jsonpCallback: "jsonpdates",
            crossDomain: "true"
        });
}
```



## WCF JSONP example

- Note that WCF serializes .NET DateTime to JSON differently to MVC
- Includes local offset (UTC+offset), whereas MVC is UTC time
- Need to take account of this in client code



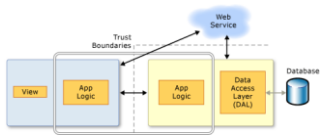
## Coming soon in .NET(?)

- WCF RIA Services jQuery support
- ASP.NET Web API



## WCF RIA Services

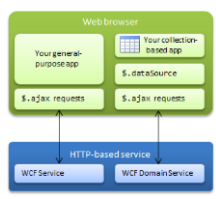
- Intended to coordinate client and service code to provide access to data
- Exposes data as domain services
- Essentially a wrapper round EDM






## WCF RIA Services

- Client support initially for Silverlight
  - Code generation for client proxy which can be used as data source for data bound controls
- Support under development for other clients, for example jQuery
  - RIA/JS library, apparently to be rebranded as upshot.js



## ASP.NET Web API

- Introduced in ASP.NET MVC 4 Beta (Feb 12)
- Provides new base controller class ApiController
- Actions return data rather than views
- Automatically serializes your model to JSON, XML, etc
- Integrates with ASP.NET routing
- …like ASP.NET Page Methods for MVC?



## iFrame Call pattern

- **Goal Story**
  - Bill's ordering a car online and the price is always synchronized with his choice. This happens because of a hidden IFrame. Each time he changes something, the browser populates a form in the IFrame and submits it. The IFrame soon contains the new price, which the browser copies into the visible display. XMLHttpRequest could have accomplished the feat too, but it's not supported in Bill's outdated browser.
- **Problem/Forces**
  - Same as XmlHttpRequest Call
- **Solution**
  - Use IFrames for browser-server communication...



## iFrames

- The iFrame technique is a bit of a “hack” but is still useful
- Works with (very) old browsers which don’t support XmlHttpRequest
- No cross-domain restriction

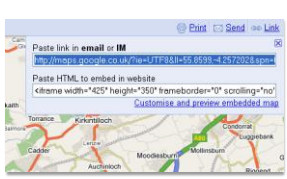
## What is an iFrame?

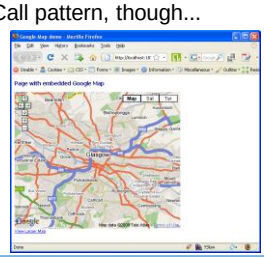
- A web page can be built as a **frameset**
  - Page split into several frames, each of which has a separate HTML document loaded into it
  - Rarely used nowadays
- An **iFrame** is an **inline frame** which contains a separate HTML document
  - iFrame can be positioned anywhere within a page, just like any other inline element, e.g. an image



## A Google Map in an iFrame

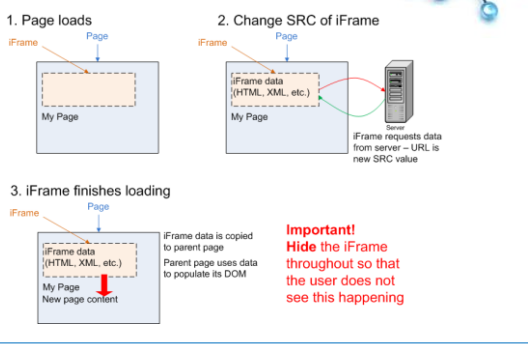
- Google Maps allows you to embed a live map in your own web page as an iFrame
  - This is **not** the iFrame Call pattern, though...






## iFrame call




**Important!**
**Hide** the iFrame throughout so that the user does not see this happening

## iFrame example

- Loads an HTML page containing a list of destinations into an iFrame
- iFrame Page rendered by an MVC view
- Copies the list from the iFrame into the main page (parent)
- Same technique could be used to load XML or JSON data and use it to update the DOM of the main page



## iFrame example – main page

```
<script language="JavaScript" type="text/JavaScript">
   function getDestinations()
   {
      var url = "/Datasources/LocationsIframe";
      document.getElementById("iFrame").src = url;
   }
   function loadResult(result) {
      document.getElementById("response").innerHTML = result;
   }
   window.onload = getDestinations;
</script>
...
<body>
   <iframe id='iFrame' ></iframe>
   <h3>Dynamically Populating a DIV with HTML using an iFrame</h3>
   <div id='response'></div>
</body>
```

**getDestinations** sets iFrame source, causing an HTTP request to be sent

call **getDestinations** when page loads

iFrame is hidden by CSS



## iFrame example – iFrame content

```
...
<script type="text/javascript">
   function loadInParent() {
      var result = document.getElementById("results").innerHTML;
      if (window.parent) {
          window.parent.loadResult(result);
      }
   }
   window.onload = loadInParent;
</script>
..
<div id="results">
   <ul>
   @foreach (String loc in Model) {
      <li>@loc</li>
   }
   </ul>
</div>
```

A complete HTML document is loaded into the iFrame

call **loadResult** function of **parent** to pass the content of the **results** div to the parent (the main page)

Content rendered by an MVC view in this example – Model is a .NET List<String>

## iFrame example – main page (again)

```
<script language="JavaScript" type="text/JavaScript">
   function getDestinations()
   {
      var url = "/Datasources/LocationsIframe ";
      document.getElementById("iFrame").src = url;
   }
   function loadResult(result) {
      document.getElementById("response").innerHTML = result;
   }
   window.onload = get
</script>
...
<body>
   <iframe id='iFrame' ></iframe>
   <h3>Dynamically Populating a DIV with HTML using an iFrame</h3>
   <div id='response'></div>
</body>
```

**loadResult** is called from iFrame document- processes the iFrame data to update the main page



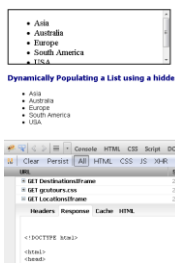
## Hiding the iFrame

- Use CSS to set visibility of iFrame
- Set size to 1px too, otherwise frame may take up space on the page

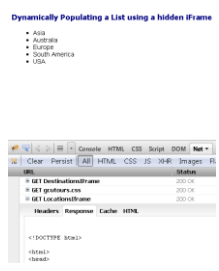
```
#iFrame {
  display    : none;
  visibility : hidden;
  height     : 1px;
}
```



## iFrame example in action




iFrame visible




iFrame hidden

## Google Maps and iFrames

- This (old) example shows a response containing a complete HTML document with calls to parent function
- Passes JSON data
- This response will be loaded into a hidden iFrame
- Google Maps also uses other techniques, including XmlHttpRequest

call to parent function

JSON data



## JavaScript APIs

- Some services provide a package of complex data retrieval and DOM manipulation, rather than simply returning data
- Consumed through client JavaScript library which provides scriptable objects
- API calls are typically translated into REST service calls
- Examples
  - Maps, e.g. Google, Bing
  - LinkedIn API



## JavaScript API example

- Bing Maps API

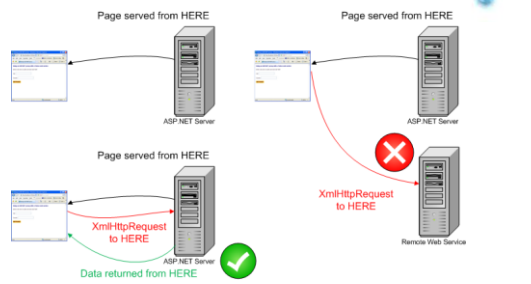
```
<script language="JavaScript" type="text/JavaScript">
<script type="text/javascript" src=
"http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.2">
</script>
   <script type="text/javascript">
      var map = null;

      function GetMap() {
         map = new VEMap('myMap');
         map.LoadMap(new VELatLong(55.867, -4.249), 15, 'h', false);
      }
</script>
```

## Proxy services

- A **proxy** provides a way to use an XmlHttpRequest with a remote web service
- Gets round the restriction on cross-domain calls by using intermediary, or proxy, service
- Proxy is on the same server that serves the web page to the client, so a call to it remains within the same domain
- Proxy forwards the request to the remote web service on a different domain

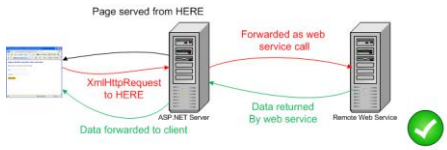


## The need for a proxy






## Using a proxy

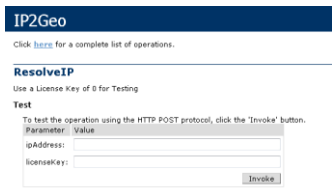



ASP.NET server must run a **proxy web service** which:

- receives XmlHttpRequest from client
- forwards request as a call to the remote web service
- gets data from web server and forwards it back to client
- performs any translation needed between service calls – for example, the XmlHttpRequest could use JSON and the remote service XML or SOAP

## Proxy example

- IP2Geo service (SOAP)
- http://ws.cdyne.com/ip2geo/ip2geo.asmx
- Gives gelocation data for a specified IP






## Proxy example

- Create service client in VS2010
- Call service on server side and make data available to local clients as JSON

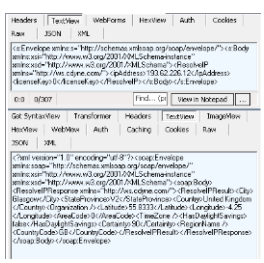
```
public ActionResult IP2Geo(string address)
{
    var client = new IP2GeoService.P2GeoSoapClient();
    var result = client.ResolveIP(address, "0");
    return Json(result, JsonRequestBehavior.AllowGet
}
```

```
function lookupIP() {
    var address = $('#address').val();
    var url = '/DataSources/IP2Geo';

    $.ajax({
        url: url,
        type: "GET",
        data: { address: address},
        dataType: "json",
        success: jsoncallback
    });
}
```

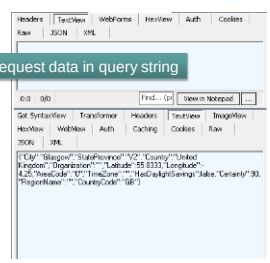
- Consume local data source in client



## Proxy example data exchange



Request data in query string



Browser - Server

Server - Service

## Reasons for using proxy

- Service may only be available as SOAP
  - Possible to create JavaScript SOAP client, but tool support for client creation is generally better in popular server-side languages
- Reduce network traffic between browser and server
- Security
  - Generally more secure than JSONP
  - Service access code is not sent to browser, which may be important if authentication is involved

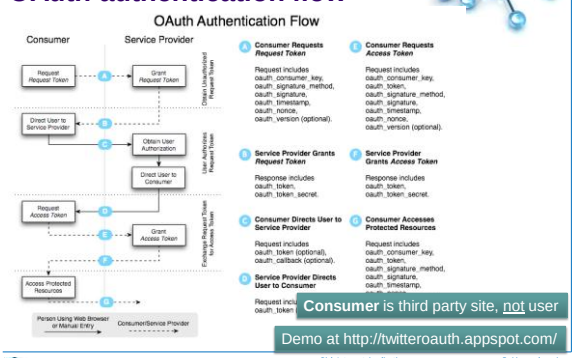


## OAuth

- Commonly used by sites for web services which require authentication
- Open standard
- Allows users to share private resources on one site with another site without providing credentials to the third party site
- OAuth allows users to hand out tokens instead of credentials
- Each token grants access to a specific site



## OAuth authentication flow




**Consumer** is third party site, not user

Demo at http://twitteroauth.appspot.com/

## Using OAuth with proxy

- OAuth is not intended for use in client JavaScript
- Consumer gets a key and secret when app is registered with service provider
- Should not expose these in client code or send tokens to/from browser



## Mashups

- Applications that use and combine data, presentation or functionality from two or more sources to create new services.
- The term implies easy, fast integration, frequently using open APIs and data sources
- Produce enriched results that were not necessarily the original reason for producing the raw source data
- Apply the techniques covered here



## What's next?

- Ajax functionality
- Bookmarking and the Back button in single-page applications
- Long-running connections