AJAX and JSON – Lessons Learned

Jim Riecken, Senior Software Engineer, Blackboard® Inc.

About Me

- Jim Riecken
- Senior Software Engineer
- At Blackboard for 4 years.
- Work out of the Vancouver office.
- Working a lot with User Interface components of Blackboard Learn™.
  - Lots of JavaScript, CSS, JSP, Tag Libraries

About this presentation

- Goals
  - Quickly introduce AJAX and JSON.
  - Describe the approaches we've taken for implementing asynchronous behavior in Blackboard Learn.
  - Share some experiences and lessons we've learned in the following areas:
    - Performance
    - Security
    - Accessibility
  - Show "real-life" examples wherever possible from Blackboard Learn.

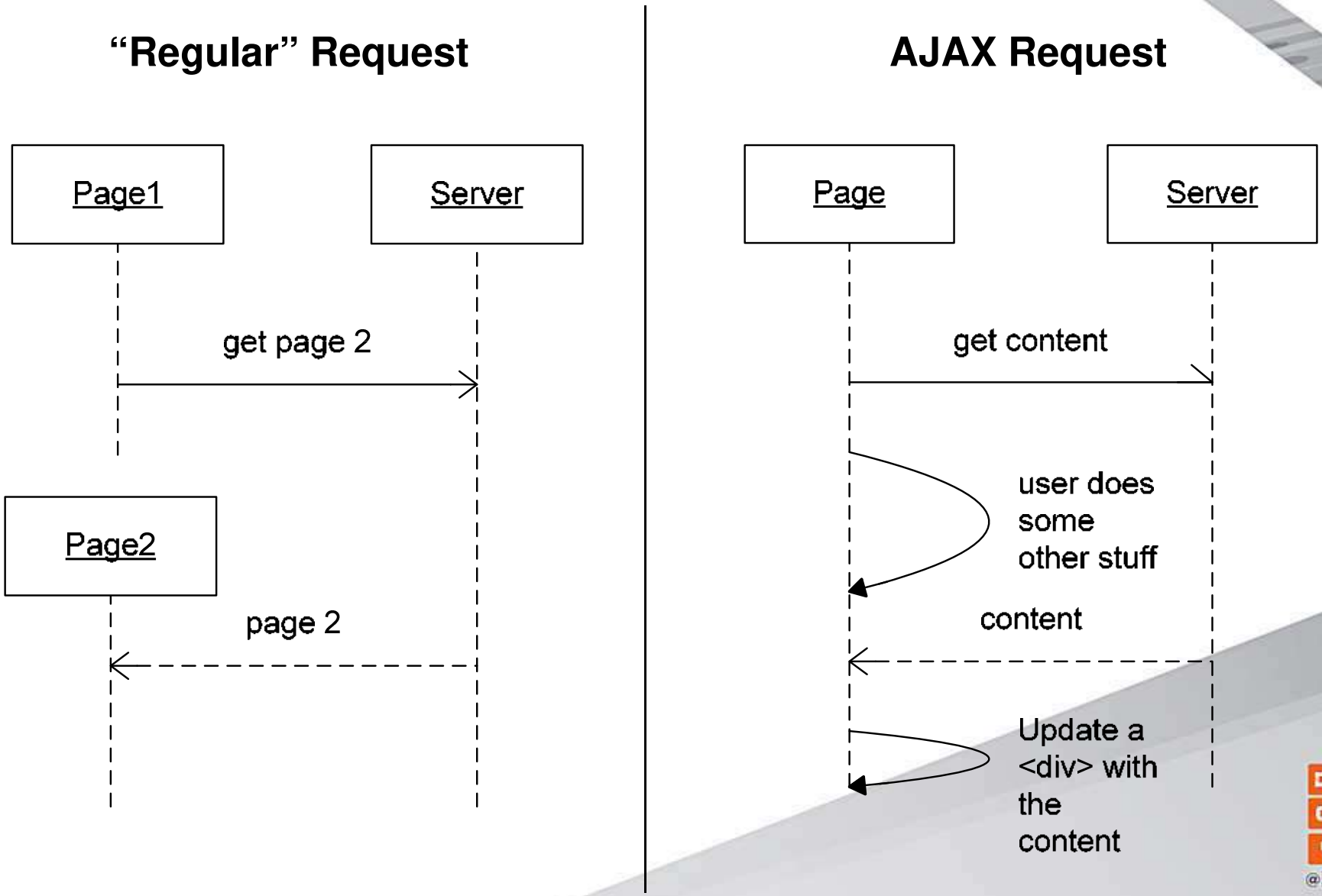
Quick Introduction to AJAX and JSON

What is AJAX?

- **A**synchronous **J**avaScript **A**nd **X**ML
  - Buzzword
  - Don't have to use XML.
  - Really just an asynchronous HTTP request.
    - Browser doesn't refresh the page.
  - Server-side, it's just a normal HTTP request.
  - Done via the `XMLHttpRequest` object built into modern browsers.
    - Various libraries are available that hide minor browser differences in how AJAX requests are constructed.
    - E.g. Prototype's `Ajax.Request` object, Direct Web Remoting (DWR)

```
new Ajax.Request( "/some/url", {
  method: 'get',
  parameters: { param1: 'value1', param2: 'value2' },
  onSuccess: function( req )
  {
    alert( req.responseText );
  }
});
```

AJAX vs. Non-AJAX



Why use AJAX?

- Helps your application become:
  - More “Desktop”-like
    - Drag and drop.
    - In-place editing.
    - Etc.
  - More responsive
    - Instead of whole page refreshes, only parts need to update.
    - Immediately show feedback that an operation is in progress and let the user do other things while it’s processing.
  - More usable
    - Immediate feedback.
    - No long waits.
    - Less clicks.
    - Better overall user experience.

What is JSON?

- **J**ava**S**cript **O**bject **N**otation (text/x-json)
  - Stems from how Object literals are specified in JavaScript.

```
var myObject = {
 one: 'value1',
 two: [ 1, 2, 3, 4, { one: 'two' } ],
 three: { another: 'object' },
 four: function( a ) { alert( a ); },
 five: someOtherVariable
};
```

  - JSON is a subset of this.
    - Can have arrays and nested objects.
    - No functions or variable references though.
    - Property names must be strings.

```
{ 'property1': 'value1', 'property2': 42 }
{ 'items' : [ 'item1', 'item2', 'item3' ] }
```

What is JSON?

- A pretty good alternative to using XML for data transfer.
  - More succinct and easy to read.
  - Easy to generate.
  - Don't have to do any XML DOM processing.
  - Browsers can parse it very fast into native JavaScript objects
    - Using **`eval`**, or a parsing function from a JavaScript library.
    - E.g. Prototype's **`String.evalJSON`**

```
var jsonString = "{'name':'value','items':[1,2,3,4]}";
var parsed = jsonString.evalJSON( true );

alert( parsed.name ) // alerts "value"
alert( parsed.items[2] ) // alerts "3"
```

JSON vs. XML - Format

XML

```
<result>
 <people>
  <person firstName="Alice" lastName="Anderson"/>
  <person firstName="Bill" lastName="Brown"/>
  ...
 </people>
</result>
```

JSON

```
{
  people:
  [
   { 'firstName': 'Alice', 'lastName', 'Anderson' },
   { 'firstName': 'Bill', 'lastName', 'Brown' },
   ...
  ]
}
```

JSON vs. XML - Processing

XML – DOM traversal

```
// req is a XMLHttpRequest with text/xml content
var result = req.responseXML;
var people = result.getElementsByTagName('person');
for ( var i = 0, l = people.length; i < l; i++ )
{
  alert( people[i].getAttribute('firstName') );
}
```

JSON – Native JS objects

```
// req is a XMLHttpRequest with text/x-json content
var result = req.responseText.evalJSON( true );
var people = result.people;
for ( var i = 0, l = people.length; i < l; i++ )
{
  alert( people[i].firstName );
}
```

What are we using?

What are we using?

- In Blackboard Learn we use several different approaches to asynchronous requests.
  1. Direct Web Remoting (DWR)
     - DWR allows Java classes to be exposed directly to JavaScript callers.
     - Extremely simple.
     - http://directwebremoting.org
  2. Normal AJAX requests.
     - Using Prototype's `Ajax.Request` on the client-side.
       - http://prototypejs.org
     - Using Struts actions that return JSON content on the server-side.
       - Using Json-lib to generate the JSON.
       - http://json-lib.sourceforge.net/

What are we using? DWR

Service class

```
public class MyDwrService
{
  public List<String> getItems( String parameter1, String parameter2 )
  {
    List<String> result = // construct the list somehow
    ...
    return result;
  }
}
```

In dwr.xml

```
<create creator="new" javascript="MyDwrService">
  <param name="class" value="my.package.MyDwrService"/>
</create>
```

In HTML

```
<script type="text/javascript" src="/path/to/dwr/engine.js"></script>
<script type="text/javascript" src="/path/to/dwr/interface/MyDwrService.js"></script>
<script type="text/javascript">
 MyDwrService.getItems( 1, 2, function( result )
 {
   alert( result[0] );
 });
</script>
```

What are we using? `Ajax.Request`

In JavaScript

```
new Ajax.Request( '/path/execute/adder', {
  method: 'get',
  parameters: 'one=1&two=2'
  onSuccess: function( req )
  {
    var result = req.responseText.evalJson( true );
    alert( result.answer );
  }
});
```

Struts Action

```
public class AdderAction extends Action
{
  public ActionForward execute( ActionMapping m, ActionForm f,
                                HttpServletRequest req, HttpServletResponse res ) throws Exception
  {
    Map<String, String> result = new HashMap<String,String>();
    int i = Integer.parseInt( req.getParameter('one');
    int j = Integer.parseInt( req.getParameter('two');
    result.put( 'answer', String.valueOf( i + j ) );
    res.setContentType( "text/x-json" );
    res.setCharacterEncoding( "UTF-8" );
    JSON json = JSONSerializer.toJSON( result )
    json.write( res.getWriter() );
    return null;
  }
}
```

Lessons Learned

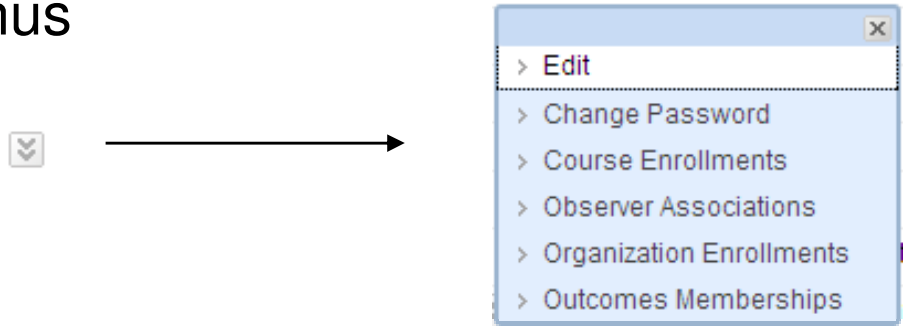
Performance

Be Lazy!

- Don’t do upfront processing if you don’t need to and defer expensive processing to as late as possible.
  - If a large component is hidden on the page,
    - Can dynamically load the content with AJAX only when it’s accessed.
  - If a JavaScript component doesn’t need to be initialized until the user interacts with it,
    - Can initialize the component when the user hovers over, or puts focus on the component.
  - If JavaScript code isn’t needed right away by the page when it loads,
    - Can dynamically load scripts using AJAX.
- Example of lazy “on-demand” loading in Blackboard Learn:
  - Context Menus
    - Make use of the first two items in the list above.

Be Lazy! – Context Menus

- A large change in the UI of 9.0 was the introduction of Context Menus



  - Large lists of data can potentially have hundreds of them on one page.
  - Menu items can differ depending on the object the menu is acting upon.
  - Users are not likely to click on all (or even many) of the menus on a page.
  - Prime candidate for lazy loading and initialization.

Be Lazy! – Context Menus

- How does it work?
  - The user hovers or focuses on the context menu link.
    - Event handler runs that initializes the context menu JavaScript and then removes itself (so it only runs once.)
  - The user clicks on the context menu link.
  - An AJAX request is sent off to the server (e.g. to a Struts action.)
  - Server constructs the contents of the context menu.
    - Using parameters sent in the request to determine the context.
  - The server returns the contents in a JSON response.
    - The response is in a specific format the JavaScript knows how to process.
  - The HTML elements for the menu are constructed in JavaScript
    - The contents are cached.
    - No need for another AJAX request if the user opens the menu again.
  - The menu is shown to the user!

Be Lazy! – Context Menus

- Example loading time savings if:
  - The page, without any of the context menus, takes **100ms** to generate.
  - Each context menu takes **5ms** to generate in-page.
  - The context menu takes **50ms** to request dynamically.
  - There is a list with **100** context menus on it in the page.

	With Lazy Loading	Without Lazy Loading
Page response time	100ms	600ms
Time to open a context menu	50ms	0ms

- Latency threshold for “interactivity” is somewhere between 50-200ms
  - Anything more will feel “sluggish”

Be Lazy! – Context Menus

JSP

```
...
<bbNG:contextMenu dynamic="true" menuGeneratorUrl="/path/to/my/action"
      contextParameters="param1=one&param2=two"/>
...
```

Menu generator action

```
import blackboard.platform.ui.struts.dynamiccontextmenu.BaseContextMenuGenerator;

public class MyGenerator extends BaseContextMenuGenerator
{
  protected List<List<ContextMenuItem>> generateContextMenu( HttpServletRequest request )
  {
    List<List<ContextMenuItem>> result = new ArrayList<List<ContextMenuItem>>();
    List<ContextMenuItem> group = new ArrayList<ContextMenuItem>();
    result.add( group );

    String param1 = request.getParameter("param1");
    //...
    ContextMenuItem item = new ContextMenuItem();
    item.setTitle("Title");
    item.setUrl("<url>");
    group.add( item )
    //...
    return result;
  }
}
```

Be Efficient!

- Try to minimize the number of HTTP requests that you need to do while loading the page.
  - Many small requests are slower than one big one.
    - Combine related JavaScript functions into one file.
    - Don't add an external script file more than once.
  - Can even load JavaScript code on demand.
    - Get code with an AJAX request, then dynamically construct a `<script>` block and add it to the `<head>`.
- Ensure that your scripts and content (if desired) can be cached.
  - Add appropriate `Cache-Control` or `Expires` headers.
  - With AJAX requests
    - Use `GET` requests for content that may be cached – browsers will cache AJAX `GET`s unless told otherwise.
    - Use `POST` requests for saving.

Be Efficient!

- Minimize time spent in “onload” initialization JavaScript code.
  - Some browsers execute JavaScript *much* slower than others.
  - Long-running scripts can cause the page to appear unresponsive.
  - Ways to help:
    - Split up processing into chunks, waiting a short amount of time between each chunk to let the browser do other things.
      - Use `setTimeout`, or Prototype’s `Function.delay` or `Function.defer`

```
document.observe("dom:loaded", function()
{
  (function() {
    //Do first part of long-running item.

    (function() {
      //Do second part of long-running item.
    }).delay( 0.1 );

  }).delay( 0.1 );
}
```

    - Perform lazy initialization of components
      - Like the context menus do.

Be Efficient!

- Batch calls together if you're using DWR.
  - Instead of N small AJAX requests, it will do one large one that performs all the operations in the batch.

```
DWREngine.beginBatch();

MyService.method1( ....., callback1 );
MyService.method2( ....., callback2 );
for ( int i = 0; i < 10; i++ )
{
  MyService.method3( i, callback3 );
}

DWREngine.endBatch();
```

  - Instead of **12** separate AJAX requests, this only makes **1**
    - All the callbacks will be called with the method call's return value when the overall request completes.

Cache!

- If data is not likely to change often, cache it. E.g.
  - Server-side
    - Results of database queries.
    - Generated JSON.
  - Client-side
    - Results of AJAX calls
      - Can do this either with browser caching (by setting appropriate Cache-Control headers)
      - Or saving data in JavaScript objects.
- Disk space (and increasingly, memory) is cheap.
- Time is expensive.
- Example of JSON caching in Blackboard Learn:
  - Blackboard Grade Center

Cache! – Grade Center

- The Blackboard Learn Grade Center makes heavy use of AJAX and JSON
  - Grade data is pushed to the browser and then saved to the server asynchronously.
  - Updates to grade data are retrieved incrementally from the server.
- Generating the JSON data needed to render the Grade Center is very expensive, especially for large courses.
  - E.g. 100 students, 100 columns = 10000 pieces of grade data.
  - Database queries, calculations, etc.
- The grade data is cached at two levels:
  - **On the browser** – Until the instructor leaves the current course.
    - Helps with secondary loads of the Grade Center.
  - **On the server** – JSON in files in the file system.
    - Helps with initial load.

Cache! – Grade Center

- Client-side caching
  - Data is stored in JavaScript variables inside the frameset.
  - Not 100% reliable.
    - Doesn't work with selective SSL due to browser security handling.
    - Doesn't work if the page is opened outside the frameset.
  - But it helps a lot in most cases.

Cache! – Grade Center

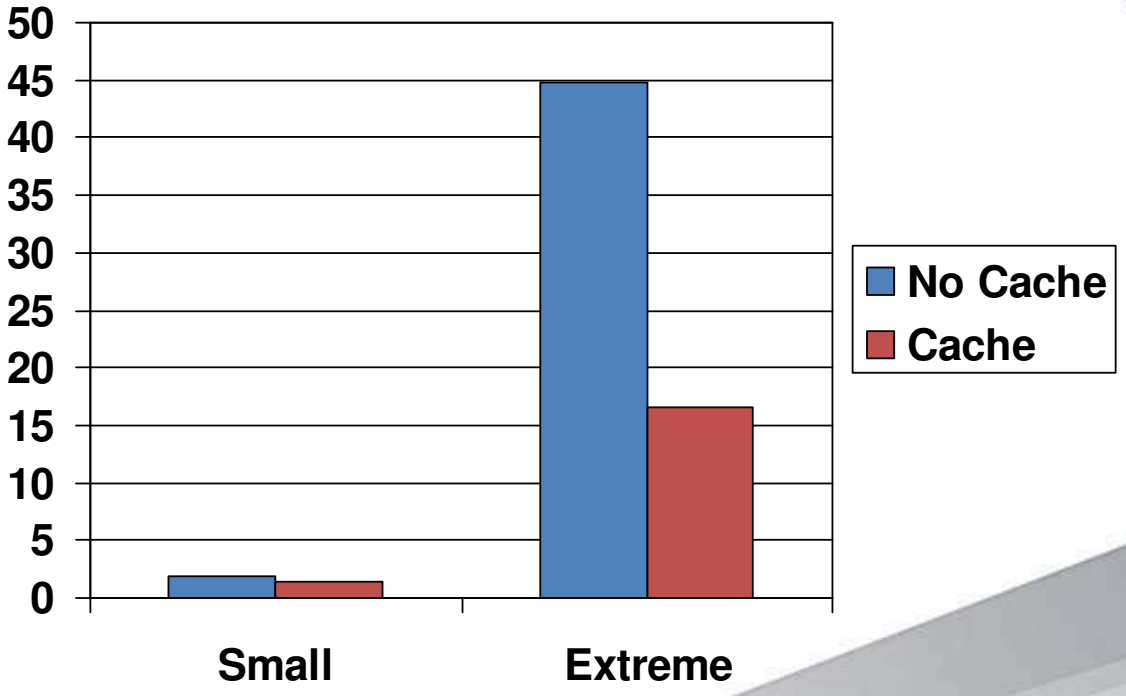
- Server-side JSON caching.
  - Although the client side caching helps, it doesn't help with the loading of the grade data when entering the grade center initially after login.
    - Database queries.
    - Generation of JSON.
  - Gradebook data doesn't change that often
    - So, the JSON can be cached on the file system.
    - Then sent back to the browser on subsequent entries.
    - Along with a "delta" of what's changed between the cached version and the actual live data in the database.
  - The caching gives a significant performance improvement

Cache! – Grade Center

- How the caching works
  - When the cached JSON data **doesn't** exist:
    - It is generated (e.g. all the database queries are run)
    - It is sent back to the browser while *simultaneously* being streamed to the file system.
    - The browser shows the grade center grid.
  - When the cached JSON data **does** exist:
    - It is sent back directly to the browser and a delta computation is immediately started (via a background thread.)
    - The browser retrieves the data and then requests the delta from the server.
      - In most cases, the delta has finished processing by the time the request is made.
      - If not, the request will wait until the delta is complete.
    - The delta is sent back to the browser.
    - The browser shows the grade center grid.

Cache! – Grade Center

- Performance improvement when data is cached:
  - Small Course
    - 40 students
    - 26 columns
    - No caching: 1828ms
    - Caching: 1421ms
    - Improvement: **23%**
  - Extreme Course
    - 500 students
    - 98 columns
    - No caching: 44827ms
    - Caching: 16594
    - Improvement: **63%**



Security

Trust No One

- AJAX requests and responses are just like any other HTTP request
  - Don’t assume input from an AJAX request is clean.
    - Assume the opposite.
    - Filter all input that may be displayed somewhere later.
  - Validate request parameters.
    - Client-side validation can be trivially bypassed.
  - Check that a user is allowed to do the action that they are trying to do
    - If they figure out the URL, students can try to call an instructor-only action.
    - Ensure that the user has the right entitlements.

Stop Cross Site Scripting (XSS)

- Cross-site scripting happens when arbitrary JavaScript code is allowed to be injected into parts of a page.
  - Attacks can perform actions as other users
    - If the instructor looks at an “infected” page, for example, the script could try to make all the student users instructors without the instructor knowing.
- Ways to avoid:
  - Strip out HTML from **all** user input
  - If you need allow HTML, run `XssUtil.filter( String input )` on **all** HTML inputs.
    - It strips out all known XSS attacks from the HTML while still allowing most tags through.

Stop Cross-Site Request Forgery

- Cross-Site Request Forgeries (XSRF) are a subset of XSS attacks where the attack originates on another website altogether.
  - Instructor visits “Site X” which has an XSS attack on it that makes a request to the Blackboard Learn server to make all the students instructors.
  - Since the instructor is logged in to Blackboard Learn, the request succeeds.
- Ways to fix
  - Use `POST`s for actions that modify data
    - Marginally more difficult to attack.
  - Include a unique token (or “nonce”) generated each time a request is made (that the server knows)
    - The server checks for the existence of this parameter, and rejects requests that don’t have it.

XSRF – Cont.

- For AJAX requests, you can “double submit” cookies.
  - Cookies are automatically sent in request headers.
  - JavaScript can access cookies, but **only for the domain that the current page is on**.
  - In the AJAX request, send the `document.cookie` as a request parameter and on the server side check if it matches what’s in the request header.
  - DWR > 2.0 does this by default on all requests.

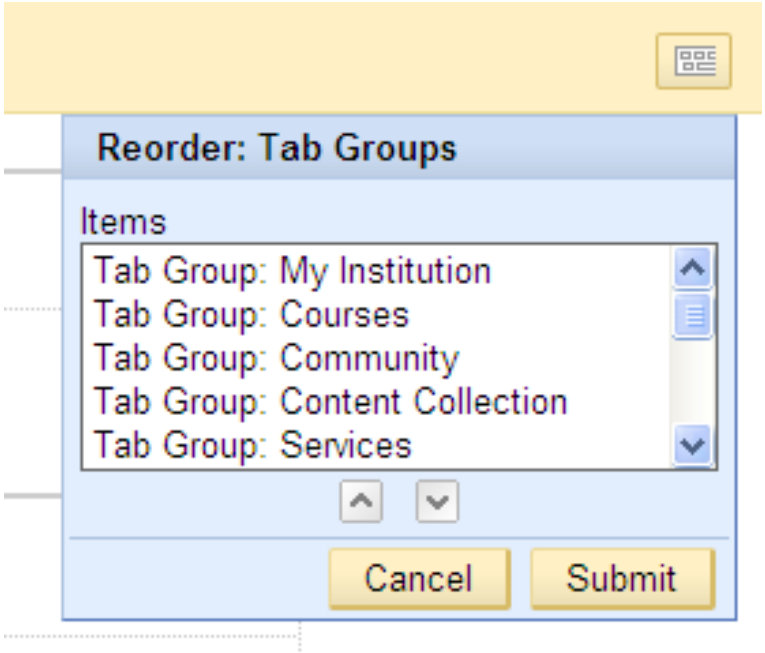
```
new Ajax.Request('/url/to/action',
{
  method: 'post',
  parameters: { cookie: document.cookie, .... },
  onSuccess: function() { ... }
});
```

```
//in action code
String cookie = request.getHeader("Cookie");
String jsCookie = request.getParameter("cookie");
if ( StringUtil.isEmpty( jsCookie ) || !jsCookie.equals( cookie ) )
{
  //INVALID REQUEST
}
```

Accessibility

Design for Accessibility

- Not all users have sight, or can use a mouse.
- Ensure your rich UI components are keyboard accessible
  - E.g. Drag and Drop in Blackboard Learn
    - Completely separate set of controls that allow reordering of items to be performed using the keyboard.



Design For Accessibility

- Ensure components are described appropriately.
  - Give form elements `<label>`s
  - Can give additional context to a screen reader by placing content off-screen with CSS

```
<span style="position: absolute; top: -10000px; left: -10000px;">Info for screen
   reader</span>
```

- Give feedback that operations have completed.
  - Show an alert, or set focus on a message.
  - E.g. Inline Receipts in Blackboard Learn
    - Are focused on when they appear.

Use ARIA

- ARIA (Accessible Rich Internet Applications) is an emerging W3C standard for accessible “Web 2.0” sites.
  - Specialized HTML markup lets assistive technologies (e.g. screen readers) know what role a given component on the page has.
    - E.g. that a specific `<ul>` is actually a popup menu
- Live regions in specific for AJAX.
  - They indicate areas of the page that will be updated with new content (e.g. from an AJAX call)
  - They will notify the user when the content has updated based on the mode the live region is set
    - “polite” – announce updates when the user is idle.
    - “assertive” – announce updates as soon as possible.

```
<div aria-live="assertive">
 ...
</div>
```

Q & A