The Desktop Advantage

- Large selection of GUI components
- Interactive
  - Rich event model
- Responsive
  - Partial redraw

Events and Event Handler

- Events
  - onfocus, onblur, onkeypress, onkeydown, onkeyup, onclick, ondbclick, onmousedown, onmouseup, onmousemove, onmouseover ...
- Specify event handler
  - `<element event="code">`
  - For example:
    `<button onclick="clickHandler();">click</button>`

HTML Event Models

- HTML 4 Event Model
  - HTML 4.01 Specification - http://www.w3.org/TR/REC-html40/interact/scripts.html#h-18.2.3
  - Limited but widely supported
- Standard Event Model
  - DOM Level 2 HTML Specification - http://www.w3.org/TR/DOM-Level-2-Events/events.html
- Browser specific event models

Example: Event Handling

- `j1.html`
  - Uses X Library from http://cross-browser.com/
  - Event handler
    - Written in JavaScript
    - Modify the HTML document
JavaScript

- Interpreted language
- Originally developed by Netscape
- Syntax is similar to Java

Core JavaScript

- Mainly covers language syntax, which is similar to Java
- Some "un-Java-like" language features
  - Object creation
  - Functions as first-class citizens

Object Creation – Approach 1

```javascript
var car = new Object();
car.make = 'Honda';
car.model = 'Civic';
car.year = 2001;

var owner = new Object();
owner.name = 'Chengyu';

car.owner = owner;
```

- A JavaScript object consists of a set of properties which can be added dynamically

Object Creation – Approach 2

```javascript
var car = {
  make: 'Honda',
  model: 'Civic',
  year: 2001,
  owner: {
    name: 'Chengyu'
  }
};
```

- Object Literal

Object Creation – Approach 3

```javascript
var car = {
  'make': 'Honda',
  'model': 'Civic',
  'year': 2001,
  'owner': {
    'name': 'Chengyu'
  }
};
```

- JSON (JavaScript Object Notation)

Functions as First-class Citizens

- In JavaScript, functions are considered objects like other object types
  - Assigned to variables
  - Assigned as a property of an object
  - Passed as a parameter
  - Returned as a function result
  - Function literals (i.e. functions without names)
Function Examples

```javascript
function foo() {
    alert('foo');
}

bar = function() {
    alert('bar');
}

setTimeout( bar, 5000 );
setTimeout( function() {
    alert('foobar');},
    5000 )
```

Regular function creation

• Function literal
• Function assignment

Function as parameter

• Function literal
• Function literal as parameter

Client-Side JavaScript

◆ Embed JavaScript in HTML
  ■ `<script>`
     • `type="text/javascript"`
     • `language="JavaScript"`
     • `src="path_to_script_file"`
  ◆ Run inside a browser

Document Object Model (DOM)

◆ Representing documents as objects so they can be processed more easily by a programming language

Processing an HTML Document

```html
<html>
    <head><title>JavaScript Example</title></head>
    <body>
        <h1>JavaScript Example</h1>
        <p>Some content.</p>
    </body>
</html>
```

◆ As a text file – very difficult
◆ As an object

DOM Representation

```
  document
  └── html
      ├── head
      │   └── title JavaScript Example
      └── body
          ├── title JavaScript Example
          ├── h1 JavaScript Example
          └── p Some content.
```

Manipulate a Document

◆ Find Elements
◆ Modify Elements
◆ Create Elements
Find Elements
- `document.getElementById()`
- `document.getElementsByTagName()`
- `document.getElementsByName()`

Modify Elements ...
- HTMLElement properties and methods
  - IE
    - `innerHTML`
    - `innerText`
    - `insertAdjacentHTML()`
    - `insertAdjacentText()`
  - Netscape/Mozilla
    - `innerHTML`
    - Element-specific

... Modify Elements
- node
  - `setAttribute()`, `removeAttribute()`
  - `appendChild()`, `removeChild()`
  - `insertBefore()`, `replaceChild()`

Create Elements
- `document`
  - `createElement()`
  - `createTextNode()`

Example: Document Manipulation
- `j2.html`
  - Read and display the text input
  - Display "Hello <name>"??
  - Add text input to table??

Create Desktop-Like Web Applications
- Interactivity
  - HTML events
  - JavaScript for event handling
  - DOM for document manipulation
- Responsiveness??
Communicate with Server

- The synchronous request-response model is still a limiting factor in responsiveness
- Solution: XMLHttpRequest
  - A JavaScript object
  - Send request and receive response
  - Response can be handled asynchronously
  - Do not need to wait for the response

Understand Asynchronous ...

- Synchronous
  - send( request );
  - // wait for response
  - process( response );
  - // do other things
  - ...

- Asynchronous
  - send( request );
  - // don't wait for response
  - process( response );
  - // do other things
  - ...

What's the problem??
What's the solution??

... Understand Asynchronous

- Asynchronous
  
  ```javascript
  // callback function
  function foo( response ) {
    process( response );
  }

  Same as handling events like click.
  ```

An XMLHttpRequest Example

- a1.html
  
  ```html
  a1.html
  ```

About the Example

- clickHandler()
- newXMLHttpRequest()
- updateDocument()
- getReadyStateHandler()

XMLHttpRequest - Properties

- onreadystatechange
- readyState
  - 0 - uninitialized
  - 1 - loading
  - 2 - loaded
  - 3 - interactive
  - 4 - complete
- status
- statusText
- responseBody
- responseStream
- responseText
- responseXML
**XMLHttpRequest - Methods**

- `abort()`
- `getAllResponseHeaders()`
- `getResponseHeader( header )`
- `open( method, url, asyncFlag, username, password )`
  - asyncFlag, username, password are optional
- `send( messageBody )`
- `setRequestHeader( name, value )`

**So What is Ajax?**

- Asynchronous JavaScript and XML
- JavaScript + XMLHttpRequest
- Characteristics of Ajax
  - Non-blocking – the server response is handled asynchronously with a callback function
  - Partial page update using JavaScript

**More About AJAX**

- XMLHttpRequest used to be an IE specific feature that received little attention
- It’s all started by Google Maps
- The beginning of “Web 2.0”

**Key Elements of an Ajax Operation**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Process the request</td>
</tr>
<tr>
<td>Event handler</td>
<td>Send back a response</td>
</tr>
<tr>
<td></td>
<td>Create a XMLHttpRequest</td>
</tr>
<tr>
<td></td>
<td>Attach a callback function</td>
</tr>
<tr>
<td></td>
<td>Send the request</td>
</tr>
<tr>
<td>Callback function</td>
<td>Process the response</td>
</tr>
<tr>
<td></td>
<td>Update the HTML Page</td>
</tr>
</tbody>
</table>

**Problems of Plain JavaScript + XMLHttpRequest**

- Each browser has their own JavaScript implementation
  - Code that works on some browsers may not work on others
- Lack of pre-made GUI components
- Implementing Ajax operations is quite tedious

**JavaScript/Ajax Frameworks and Libraries**

  - Cross-browser compatibility
    - New JavaScript API, e.g. X Lib, JQuery
    - New language, e.g. ZK, Taconite
  - Pre-made, Ajax-enabled GUI component
  - Simplify the implementation of Ajax operations
One Library to Rule Them All - jQuery

- jQuery - http://jquery.com/
- jQuery UI - http://jqueryui.com/
- The market share of jQuery

A jQuery Example

- j3.html
  - Usage
  - jQuery wrapper
  - Selectors
  - Elements
  - Events and event handling
  - DOM manipulation

Use jQuery Library

- http://jquery.com/download/
  - Local copy vs. CDN hosted
  - 1.x vs 2.x

jQuery Wrapper

- $() or jQuery()
  - Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.
  - $( "input[name='firstName']" )
  - $( "#who" ) | $( "#tl" )

Basic Selectors

- By id
  - $("#foo")
- By tag name
  - $("div")
- By CSS class
  - $(".foo")

By attribute

- $("[name]")
- $("[name='joe']")

Combine Selectors

- Select all the <div> elements with CSS class foo and an attribute bar
  - $(“div.foo[bar]”)
- Select all the <div> elements, and all the elements with CSS class foo, and all the elements with an attribute bar
  - $(“div, foo, [bar]”)
Other Selectors and Filters

- Form selectors
- Hierarchy selectors
- Filters

What Can We Do With An Element

- Get and set
  - html()
  - attr()
  - prop()
  - val()
- Manipulate CSS
  - addClass()
  - removeClass()
  - toggleClass()
  - hasClass()

```
Property tag Name
  <input name="username" value="cyrus" />
  Attribute name    val()
```

Typical Event and Event Handling in jQuery

```
 Event  Event Handler
  $("#click").click( function(){
    ...
  });
```

Unobtrusive JavaScript: separate style, behavior, and structure.

```
<button id="click" onclick="display();">
  Click Me</button>
```

Document Ready Event

```
Triggered when the DOM hierarchy of the HTML document is fully constructed

$( document ).ready( handler )
  $( document ).ready( handler ) (not recommended)
  $( handler )
```

Other Events

- Mouse events
  - .click()
  - .mousedown()
  - ...
- Keyboard events
  - .keyup()
  - .keydown()
  - .keypress()
  - ...
- Form events
  - .change()
  - .submit()
  - ...
- Browser events
  - .resize()
  - ...
- Document events

DOM Manipulation

- Insertion
  - Around (i.e. parent)
  - Inside (i.e. children)
  - Outside (i.e. sibling)
- Removal
- Replacement

Example:
```
$("#t1").append("<tr><td>John</td><td>Doe</td></tr>");
```
Example: jQuery Tic Tac Toe

```
j4.html

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

```

AJAX with jQuery

```
http://api.jquery.com/category/ajax/

$.ajax( url [, settings])
- url: request URL
- data: data to be sent to the server
- success: a function to be called if the request succeeds

Example: a2.html

```

Example: Who’s Online (I)

```

Who’s Online
- cysun
- admin

```

Who’s Online (II)

```

Use an AJAX request to get the list of online users as a JSON object, then use the JSON object to populate the list
- JSON
- Jackson
- JSON view in Spring
- More jQuery

```

Java JSON Library

```

- Serialize and de-serialize Java objects
- Jackson
  - Maven dependency: jackson-databind
- Gson
  - https://code.google.com/p/google-gson/

```

Jackson and Spring

```

- Add a BeanNameViewResolver
- Add a MappingJackson2JsonView
- Model objects passed to the view will be automatically serialized to JSON

```

```

```
Who’s Online (III)

- Automatically update the Who’s Online list
  - How??

Repeated Requests

- Refresh response header, or
- setInterval(function, interval) in JavaScript

Asynchronous Request Processing

- Introduced in Servlet 3.0 specification
- Supported by Spring 3.2+

Enable Asynchronous Request Processing

- web.xml
  - Add `<async-supported>` to servlets and filters
  - Add `<dispatcher>` to filter mapping
- `<servlet>-context.xml`
  - Add `<mvc:async-support>` to `<mvc:annotation-driven>`

DeferredResult ...

- [http://docs.spring.io/spring/docs/current/spring-api/org/springframework/web/context/request/async/DeferredResult.html](http://docs.spring.io/spring/docs/current/spring-api/org/springframework/web/context/request/async/DeferredResult.html)

Controller code:

```java
@RequestMapping("/whosonline.deferred.json")
@ResponseBody
public DeferredResult<String> wosDeferred()
{
    DeferredResult<String> deferredResult = new DeferredResult<String>();
    return deferredResult;
}
```

Some other code:

```java
deferredResult.setResult(data);
```
DeferredResult

- Controller can return immediately so the associated servlets and filters can finish and their resources released
- The response remains open until `setResult(data)` is called, at which point data is sent back to the client as the response body (per `@ResponseBody`)

How Will Who’s Online Work

```
public class WhoOnineService {
    private Users deferredResults = new Users();

    public Users getUpdatedUserList() {
        DeferredResult<User> deferredResult = new DeferredResult<>();
        deferredResult.setResult(user);
        deferredResult.setHasData(true);
        deferredResult.setException(null);
        return deferredResults;
    }
```

- deferredResult() and removeUser()
  - update Users list
  - setResult() to all DeferredResults in the Results list
- addUser() and removeUser()
  - add a DeferredResult to the Results list

Potential Issues

- Concurrency issues
  - E.g. multiple users login/logout at the same time
- Speed issues
  - E.g. requests coming in faster than we can `setResult()`
- Various exceptions
  - E.g. connection timeout

Some Possible Solutions

- Use thread-safe data structures in `java.util.concurrent`
- Keep track of which client has already been served

Use Jackson’s ObjectMapper

- `http://fasterxml.github.io/jackson-databind/javadoc/2.3.0/com/fasterxml/jackson/databind/ObjectMapper.html`
- JSON to Java
  - `readValue(input, type)`
- Java to JSON
  - `writeValue(output, object)`