Fundamentals of Javascript with jQuery

Chris Langtiw, Training Connection

The Three Layers of the Web

- Content – Hyper Text Markup Language (HTML)
- Presentation – Cascading Style Sheets (CSS)
- Behavior – Javascript

How The Layers Work Together

- HTML – Describes structure and semantic context of content and is parsed into Document Object Model (DOM)
- CSS – Determines how DOM nodes are rendered on media
- Javascript – Manipulates the live DOM directly

Javascript Libraries

- NOT a replacement for knowing Javascript
- Lets you write less code (usually)
- Common libraries:
  jQuery (http://jquery.com)
  Prototype (http://prototypejs.org)
  Yahoo User Interface (http://yuilibrary.com)

Using Javascript On Your Page

- Code placed inside SCRIPT element

```
<script type="text/javascript>
    // actual code
</script>
```

- Use src attribute to link external scripts

```
<script src="code.js" type="text/javascript">
```

- Script placed at end of BODY element (preferred) or in HEAD element

Basic Syntax

- Statements end with semicolon (;)

```
document.write('<p>This space for rent.</p>');
```

- Blocks of statements are contained within curly brackets { }

```
if (1) {
    document.write('<p>Line 1</p>');
    document.write('<p>Line 2</p>');
    document.write('<p>Line 3</p>');
}
```

- Blocks are not terminated with ; unless part of an assignment statement

- Statements flow from top to bottom unless disrupted
- Comments
  - Block

```
/*
Block comment - everything in between is ignored
*/
```

  - Line

```
// Line comment - everything afterwards is ignored
```

- Names
  - Can consist of letters, numbers, $ and _
  - Case sensitive

- (Almost) Everything in Javascript is an object
- Objects are containers of properties
- Properties usually accessed using dot (.) notation

```
document.write('Some output');
```

- Data primitives are wrapped in a object when instantiated

```
'This string is placed in an object'.toUpperCase();
```

- Javascript supports chaining

Chaining

- An object's sub-objects or returned objects is immediately accessible when the parent is referenced.

```
// instead of this:
var a = 'some text.';
var b = a.toUpperCase();
var c = b.toLowerCase();
var d = c.length;

// we can write this instead:
var a = 'some text'
            .toUpperCase()
            .toLowerCase()
            .length;
```

Variables

- Container to store values
- Data types are strings, numbers, booleans (true and false), `null`, `undefined`, other objects (arrays, functions, user-defined, etc.)
- Loosely typed
- Values assigned via = operator
- Value is referenced in place of variable name

```
message = '<p>This space for rent</p>';
document.write(message); // <p>This space for rent</p>
```

- Do NOT put quotes around variable name

Variables

- Container to store values
- Data types
  - Strings
  - Numbers
  - Booleans (true and false)
  - `null`
  - `undefined`
  - Other objects (arrays, functions, user-defined, etc.)

- Loosely typed
- Values assigned via = operator
- Value is referenced in place of variable name

```
message = '<p>This space for rent</p>';
document.write(message); // <p>This space for rent</p>
```

- Do NOT put quotes around variable name

- Common operators:

=	++	( )
+	--	
-	+=	
*	-=	
/	*=	
%	/=	

- Be careful with + as concatenation takes precedence

```
var sum = 10 + '3'; // '103'
```

- parseInt() and parseFloat() can correct this

```
var sum = 10 + parseInt('3');
```

`null`, `NaN` and `undefined`

- `undefined` – a variable or member that does not exist or does not have a value

```
typeof abc; // undefined
var abc;
typeof abc; // undefined
```

- `null` – a placeholder meaning ‘no value’

```
var xyz = null; // null
```

- `NaN` – a number type meaning ‘not a number’

```
var product = 13 * ‘orange’; // NaN
isNaN(product); // true
```

Arrays

- Collection of values (think egg carton)
- Defined as a list of comma-separated values enclosed within brackets [ ]
- Elements (individual values) referenced using numeric key starting at 0

```
var names = ['John', 'Peter', 'Nancy', 'Betty'];
document.write(names[0]); // John
document.write(names[2]); // Nancy
```

- Elements can be reassigned using =
- Can contain mixed data types
- Internal methods to manipulate collection

Objects

- Containers of properties
- Self-contained entity
- Properties can be any data type
- Attributes (values) and methods (functions)
- Declared as `name:value` comma-separated pairs within braces `{}`
- Properties can be accessed via dot notation or array notation
- Use `new` to create new objects (instances) based on existing objects

```
var pillbox =
{
   Sun: ‘white’,
   Mon: ‘white’,
   Tue: ‘none’,
   Wed: ‘blue’,
   Thu: ‘orange’,
   Fri: ‘red’,
   Sat: ‘green’
};
document.write(pillbox.Mon);      // dot notation
document.write(pillbox[‘Fri’]);   // array notation
var pillbox2 = new pillbox(); // create new pillbox object
pillbox2[‘Bob’] = ‘rainbow’;      // original pillbox intact
```

Object Literals vs JSON

- Javascript Object Notation (JSON) is a subset of object literals used for serializing data
- Only strings, numbers, arrays, simple objects, boolean and null values may be serialized using JSON
- JSON requires strings to be enclosed in double quotes
- Property names must be treated as strings

```
{"name":"John Doe"}
```

- http://www.json.org

Exercise: Displaying the Date

Output the current date in the following format:

Today is Saturday, November 5th, 1955

Hints

- Arrays are your friend
- Create an instance of a Date object using the following:

```
var today = new Date();
```

```
// set up arrays for days and months
var weekdays = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
   'Thursday', 'Friday', 'Saturday'];

var months = ['January', 'February', 'March', 'April', 'May', 'June',
   'July', 'August', 'September', 'October', 'November', 'December'];

// instantiate the Date object and get a date to work with
var today = new Date();

// deal with the ordinal...
var ordinal =
   ['X','st','nd','rd','th','th','th','th','th','th','th','th','th','t
   h','th','th','th','th','th','th','th','st','nd','rd','th','th','th'
   ,'th','th','th','th','st'];

// now output the date
document.write('<p>Today is ' +
            weekdays[today.getDay()] + ', ' +
            months[today.getMonth()] + ' ' +
            today.getDate() + ordinal[today.getDate()] + ', ' +
            today.getFullYear() + '</p>'
          );
```

Program Flow

- Script executes top-down unless flow is disrupted
- 2 types of disruption
  - Branching
  - Looping

Branching

- Statements executed conditionally
- 3 types of branching
  - Optional path
  - Either/or
  - Multiple choice
- Use `if`, `if…else` and `switch`

Optional Path

```
if (condition) {
  // statements to execute
}
```

- Comparison operators:

  `==` `===` `!=` `!==`

  `<` `>` `>=` `<=`

  `&&` `||` `!`

- Don’t confuse `=` and `==`

Either/Or

```
if (condition) {
  // execute if condition is true
} else {
  // execute if condition is false
}
```

Multiple Choice

```
// link multiple if statements
if (door == 1) {
  // door 1 code
} else if (door == 2) {
   // door 2 code
} else {
  // door 3 code
}
```

```
// multiple choice using switch statement
switch(door) {
  case 1:
   // do stuff
  break;
  case 2:
   // do stuff
  case 3:
   // do other stuff
  break;
  default:
   // if no matching case label do this stuff
  break;
}
```

```
// alternate version of switch
switch (true) {
  case door == 1:
   // do stuff
  break;
  case door > 1 && door < 7:
   // make sure ranges do not overlap
   // do stuff
  break;
  case door == 3:
   // do stuff
  break;
}
```

Date example revisited…

```
// deal with the ordinal...
var m = today.getDate() % 10; // get modulo

switch(true) {
   case m == 1 && today.getDate() != 11:
    var ordinal = 'st';
   break;
   case m == 2 && today.getDate() != 12:
    var ordinal = 'nd';
   break;
   case m == 3 && today.getDate() != 13:
    var ordinal = 'rd';
   break;
   default:
    var ordinal = 'th';
   break;
}
```

Ternary Operator

- Used to do inline conditional assignment or output
- Generally faster than if…else
- Format: condition ? trueValue : falseValue;

```
var isDoor1 = door == 1 ? true : false;

document.write(
   'This ' +
   (door == 1 ? 'is ' : 'is not ') +
   'door 1'
); // ternary inside () makes it an expression
```

Looping (`for` and `while`)

```
for (var c = 0; c < 10; c++) {
  document.write(c);
}
```

```
var c = 0;
while (c < 10) {
  document.write(c);
  c++;
}
```

```
var c = 0;
do {
  document.write(c);
  c++;
} while (c < 10);
```

Looping

- Used to repeat one or more statements
- 2 basic types of loops
  - for used when number of iterations is known
  - while used when number of iterations is unknown or unimportant
  - while performs zero or more iterations
  - do…while performs one or more iterations

Functions

```
function greeting() {
  document.write('<p>Hello!</p>');
}
greeting();

// using return value rather than direct output
function greeting2() {
  return 'Hello!';
}
document.write('<h1>' + greeting2() + '</h2>');
```

```
function foo() {
  // functions have their own scope
  var c = 100; // DON'T forget the var
  return c;
}
var c = 1;
document.write(c); // 1
document.write(foo()); // 100
document.write(c); // 1
```

```
function foo() {
  c = 100; // note lack of var keyword
  return c;
}
var c = 1;
document.write(c); // 1
document.write(foo()); // 100
document.write(c); // 100
```

```
// parameters can be passed into a function
function greeting(name) {
  return 'Hello ' + name + '!';
}
document.write(
'<h1>' + greeting('Hans') + '</h1>'
);

function foo(a, b) {
  b = typeof b == 'undefined'? 10: b;
  return a * b;
}
document.write(foo(10)); // 100
```

```
// assign an anonymous function
var foo = function() {
  return 100;
};
document.write(foo());

var Car = {
  running: false,
  startEngine: function() {
   // 'this' refers to current object
   this.running = true;
  }
}
var myCar = new Car();
myCar.startEngine(); // call startEngine method
```

```
function foo() {
  return 'Hello';
}

document.write(foo()); // Hello

var bar = foo(); // Hello

var bar2 = foo; // reference to function
document.write(bar2()); // Hello
```

- Extremely useful for making multiple references to the same function

Functions

- Makes code reusable and modular
- Can be named or anonymous
- Has own variable scope
- Can return a value to be manipulated
- Have zero or more parameters
- Called a method when inside an object
- Referenced directly by omitting ()

Using Objects As Namespaces

- Avoid cluttering up the global scope
- Use objects to organize code

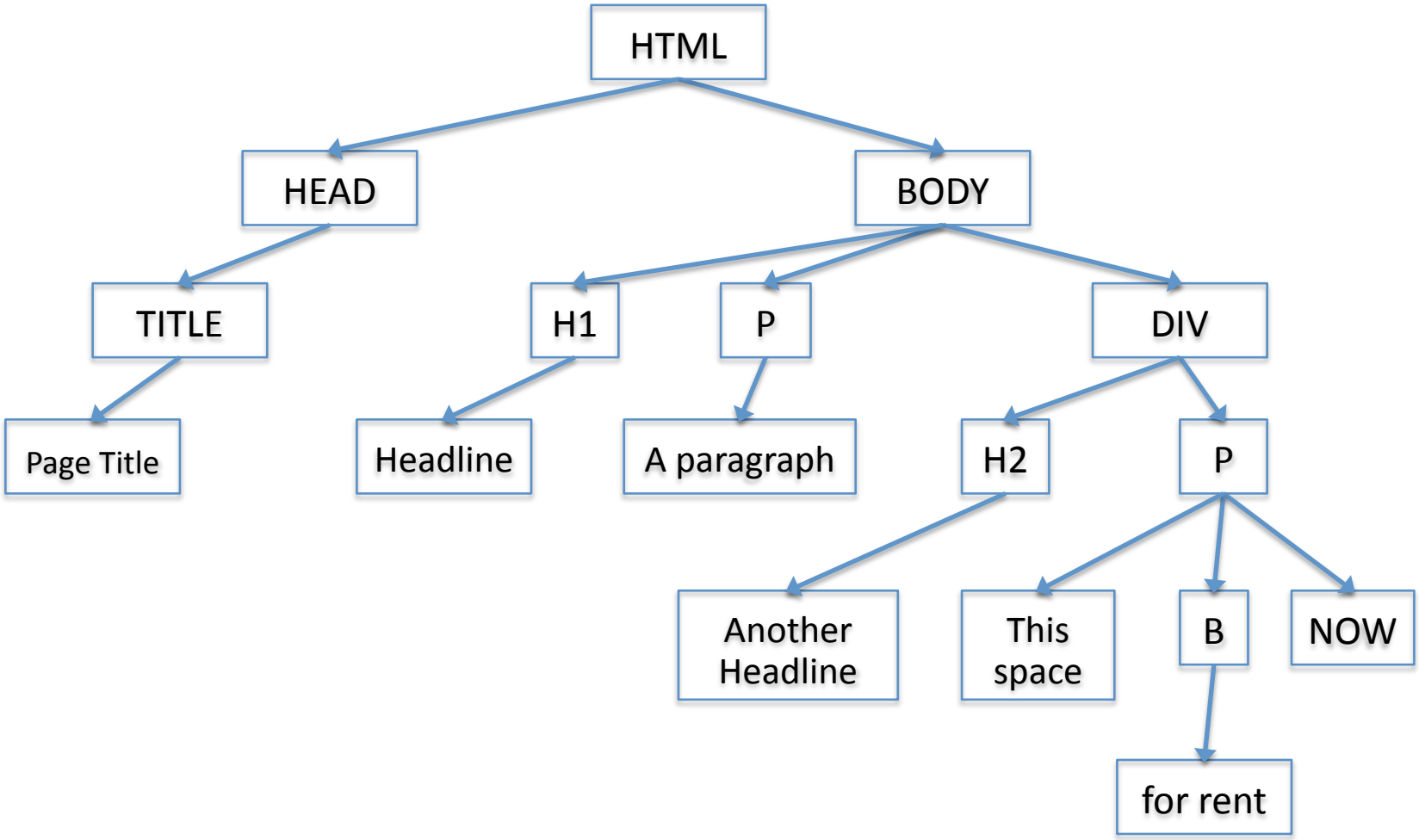
```
// singleton example
var App = {
   counter: 0,
   increment: function() { App.counter++; },
   getCount: function() { return App.counter; },
   main: function() {
    App.increment();
    App.getCount();
   }
}
App.main();
```

The Document Object Model (DOM)

- The web browser parses the HTML and stores the content in memory as a tree
- EVERYTHING is represented in the tree as nodes (12 kinds of nodes total)
- The order of the nodes is important
- In Javascript, our concern is with element and text nodes

The Document Object Model (DOM)

```
<html>
<head>
    <title>Page Title</title>
</head>
<body>
    <h1>Headline</h1>
    <p>A paragraph</p>
    <div>
      <h2>Another Headline</h2>
      <p>This space <b>for rent</b> NOW</p>
    </div>
</body>
</html>
```


HTML
HEAD
BODY
TITLE
H1
P
DIV
Page Title
Headline
A paragraph
H2
P
Another Headline
This space
B
NOW
for rent

Javascript and the DOM

- Element nodes expose all HTML attributes and inline CSS styles as properties in the element object
- Javascript manipulates the DOM, NOT the markup or stylesheets
- Be mindful of browser-specific properties
- General work pattern:
  - Select part of DOM to manipulate
  - Create new nodes if necessary
  - Set node properties if necessary
  - Attach/remove/move nodes to or in DOM as needed

Demonstration: Build a Table

- Select the node where the table structure will be added
- Create new nodes as needed
- Set any properties
- Attach the nodes together and to the DOM

jQuery

- jQuery is
  - A library that lets you write LESS code
  - Fairly small (94k minified and compressed)
  - Designed to be easily extended
- jQuery is NOT
  - A replacement for Javascript
  - A framework or complete solution
  - Ubiquitous or omnipotent

What jQuery Does Well

- Element selector engine that fully supports CSS selectors
- Traverse and manipulate DOM nodes
- Normalizes event handling
- Basic animation
- Basic utility functions
- Highly leverages chaining

Selecting Elements

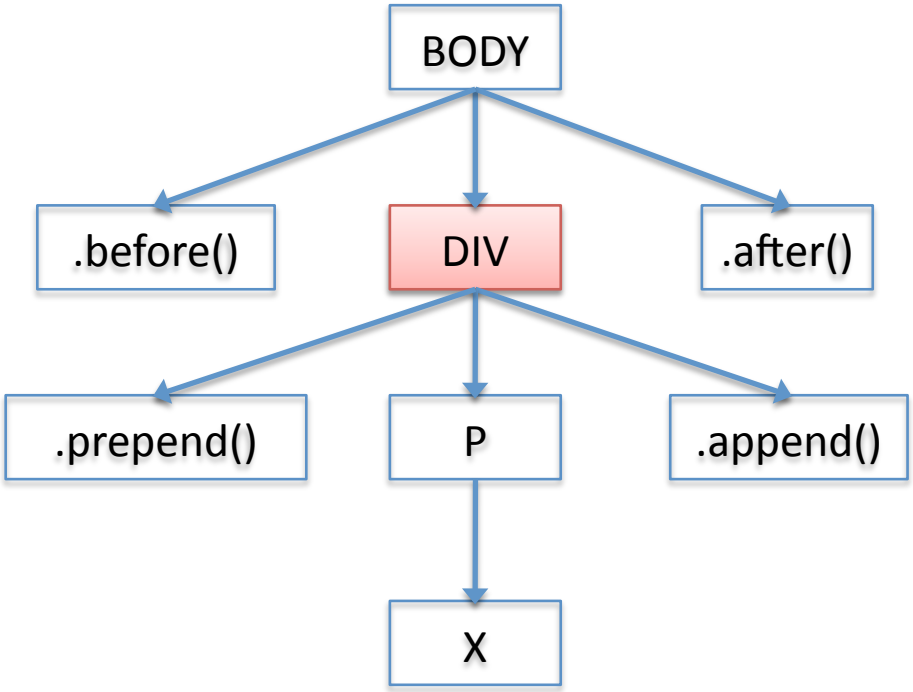
- Main interface is the $() function
- Accepts the following:
  - Selector as text (‘#main h2’)
  - DOM node
  - jQuery collection
  - HTML as string (‘ Text ’)
- Matching elements returned as a jQuery collection object

Adding Nodes

A	B
.append()	.appendTo()
.prepend()	.prependTo()
.before()	.insertBefore()
.after()	.insertAfter()

```
$(target).A(content);

$(content).B(target);
```



$(‘div’)…

Other DOM Manipulation

- Removing nodes
  - detach()
  - remove()
- Copying nodes
  - clone()
- Manipulating attributes
  - attr()
  - prop()
  - css()
  - addClass()
  - removeClass()
  - toggleClass()

Exercise: Building a Table

jQuery Collections

- The $() function returns a jQuery collection object that contains an array of matched elements
- Any changes get applied to EVERY element in the collection (implicit iteration)
- Most of the collection object methods return the modified collection object, allowing chaining
- Retrieving values via getters returns the value from the FIRST element in the collection

Filtering and Changing the Collection

- The collection can be pruned, added to, or changed completely
- Calling a traversal method will apply the traversal to EVERY element in the collection
- When the collection is modified the previous collection will be cached

Iterating Through The Collection

- Implicit iteration lessens the need to manually code our loops
- jQuery provides two methods for explicit iteration
  - `$.each()`
  - `.each()`
- Manages what kind of loop to perform automatically

Storing Data in Elements

- Most Javascript objects are mutable – be careful with this!
- Don’t modify objects you don’t own
- Use `.data()` to associate data with elements
- Explicit iteration is required if each element’s data is unique

Exercise: Dressing Up The Table

Events

- Events are generated based on actions taken by the user or user agent
  - Mouse activity
  - Keyboard activity
  - Window/browser state changes
- Code does not execute in real time, so callback functions are required
- Events are bound using `.on()`

```
<button id="thebutton">Click Me</button>

<script type="text/javascript">
$('#thebutton')
   .on(
    'click',
    function() {
       alert('The button was clicked.');
    }
   );

</script>
```

Exercise: Adding Interaction

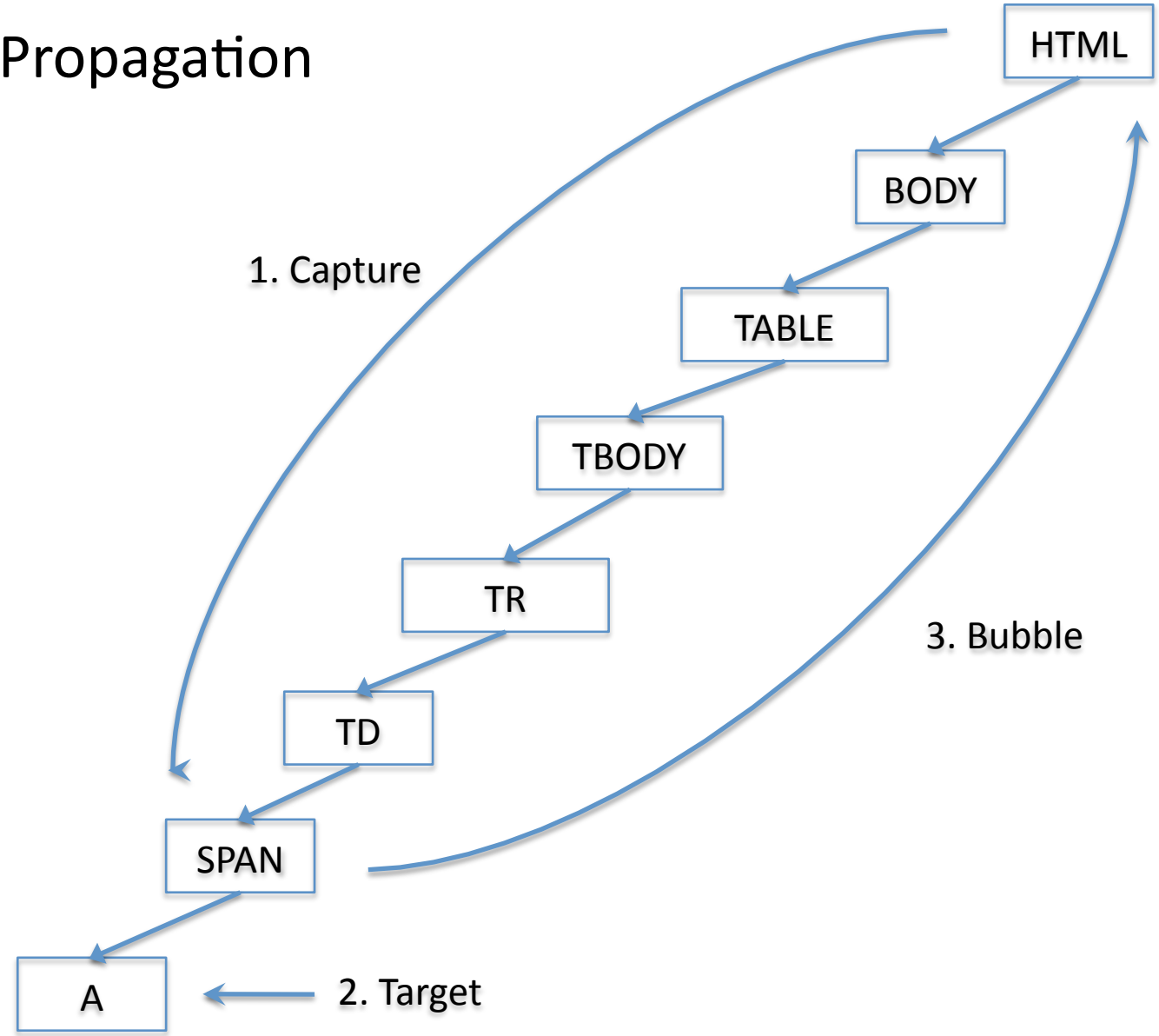
The Event Object

- Information about the event (whodunit, button/key state, mouse position, etc) are stored in the event object
- Most browsers pass the event object to the event handler (IE doesn't)
- jQuery normalizes browser-specific implementation details into a custom object passed to the handler

Event Propagation

- Events are triggered in three phases
  - Capture phase
  - Target phase
  - Bubble phase
- Few browsers support capture
- Not all events bubble
- Use propagation to optimize event code

Event Propagation



Deferring Script Execution

- Scripts usually must wait for the DOM to load before being executed
- window.onload is too slow
- Use jQuery's $(document).ready() handler

Additional Event Handling

- Default actions can be stopped using Event.preventDefault()
- Event propagation can be interrupted using Event.stopPropagation()
- Event listening can be filtered using selectors
- Data can be passed to the event handler and is accessible via the event object

Effects

- Effects are accomplished by changing CSS properties in real time or over time via Javascript
- CSS properties may be changed by modifying classes, applying inline styles, or via .animate()

Exercise

Forms

- jQuery has custom selectors to make selection of form elements simpler
- Form element values are retrieved using .val()
- Forms are made dynamic by using CSS to show/hide/change form content triggered by events

Exercise: Creating a Dynamic Form

Form Validation

- Basic approach is “innocent until proven guilty”
- Assume form data is valid
- Test data against a validation rule
- If the data fails the test, mark the data invalid
- Provide some sort of user feedback

Exercise: Form Validation

Additional References

- jQuery Documentation
  http://api.jquery.com/
- W3Schools CSS selector reference
  http://www.w3schools.com/cssref/css_selectors.asp
- Javascript: The Good Parts by Douglas Crockford
- Javascript Bible 7th Ed. Appendix A
  http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470526912,descCd-DOWNLOAD.html