Birkbeck
UNIVERSITY OF LONDON

jQuery Mobile Events

Mobile Application Development

Session 7

Events

- There are two types of events we need to consider when developing jQuery Mobile apps, and mobile apps in general:
  - Physical events
  - Page events
- Physical events are initiated by some form of user action.
- Page events are initiated by some or other change in the status of a page (e.g. load; create; transition or show)).

Physical Events

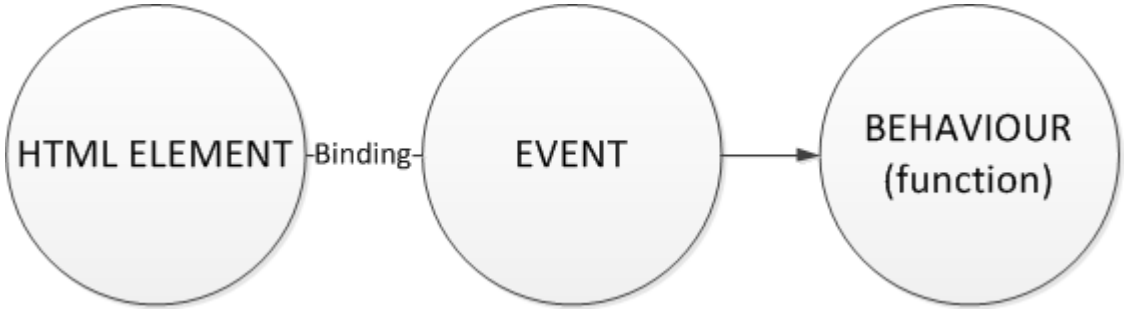
- The hardware used in desktop machines determines the need for a set of desktop specific, physical events (e.g. mouseover, click, keypress).
- In mobile application development, however, we need a set of events which are specific to mobile hardware, most notably dual aspect touchscreens.
- jQuery Mobile provides such a set of mobile specific events: e.g. tap, taphold swipe, scroll and orientation change.

Tap Events

- There are two tap events in jQM: *tap* and *taphold*.
- We should think of a tap as equating to a desktop left mouse click, and a taphold as equating to a desktop right mouse click.
- A tap event triggers after a short, complete touch of a screen that occurs on a single page element.
- A taphold event triggers after a sustained touch event, also on a single page element.

Event Binding

- jQM events need to be handled using *event handlers* in the same way as desktop events.
- We need to *bind* an event to a page element, and then state what behaviour should occur when the event fires.
- The behavior is usually stated in the form of an anonymous function



Binding a Tap Event

- In the following example, we use the jQuery *bind* method, to associate a tap event with an HTML element.
- *bind* takes two arguments:
  - The event being bound
  - The anonymous function that will run when the event is triggered

```
$("#tapme" ).bind( "tap", function (event) {
    var myText = $("#confirm").text();
    if (myText.length === 0) {
      $("#confirm").text("Confirmation of tap event" )
    } else {
      $("#confirm").text("");
}
```

Example

Tap Hold Event

- In this example, a *taphold* event is bound to the <body> element. This means that if we taphold anywhere on any page in our application the associated script will run.
- Example

```
$("body").bind( "taphold", function (event) {
  $( "#savepopup" ).popup( "open" )
})
```

Swipe Events

- Swipe events are sometimes used to transition between pages as part of an application navigation (e.g. when moving through a series of images).
- A swipe occurs when a horizontal finger drag of 30px or more occurs within 1 second duration.
- jQM has three swipe events:
  - swipe: triggered with either a left or right swipe
  - swipeleft: triggered only by a left swipe
  - swiperight: triggered only by a right swipe.

Swipe Events

- Swipe events are bound to page elements in the same way as tap events. However, note that in the following example we are binding multiple events to the same element.

```
$("body").bind( "swipeleft swiperight", function (event) {
  var page  = $.mobile.pageContainer.pagecontainer →
  ('getActivePage').attr('id');
  var dir = event.type;
  if (page === "swipepage1" && dir === "swipeleft") {
    $.mobile.pageContainer.pagecontainer("change", "#swipepage2");}
  if (page === "swipepage2" && dir === "swiperight"){
    $.mobile.pageContainer.pagecontainer("change", "#swipepage1");}
})
```

- Example

Orientation Events

- Most mobile devices have two orientations: portrait and landscape. Thus, any application we build must be able to distinguish between these two orientations and make any required changes as it moves between them.
- These changes might include:
  - Changing image dimensions
  - Moving from vertically ordered to horizontally ordered navigation and/or content and visa versa
- Note that CSS media queries are the preferred approach to dealing with orientation changes. But we can also intervene programmatically to deal with layout issues created by orientation changes.

Orientation Events

- To detect an orientation change, we bind the *orientationchange* event to the window object.
- We then include an anonymous function which will determine what will happen when the device orientation changes.
- This function takes an event object as a parameter. The event object returns all jQuery event properties (e.g. event.target, event.type, event.preventDefault, etc.).

```
$(document).on("pagecreate",function(){
  $(window).on("orientationchange",function(event){
    $("#status").text(event.orientation);
  });
});
```

- Example

Page Events

- In jQuery, we are used to working with a single documents which contain single pages. Here, the standard approach is to use *$(document).ready()* to handle events.
- However, for jQM this is problematic as *$(document).ready()* only runs once per HTTP request and not per AJAX call.
- This conflicts with the unique architecture of jQuery Mobile where AJAX is used to load the contents of multiple pages into the DOM structure without a full page reload.
- To overcome this limitation, jQuery Mobile comes with a set of mobile specific page events that allow us to associate events with individual jQM pages at various points in their lifecycle.
- These events can be categorized as follows:
  - Page load events
  - Page create events
  - Page show events

jQuery Mobile Events

Type	Description	To v1.4 (then deprecated. Not supported from v 1.6)	From v 1.4
Load events	These events are fired just before, or at the point, an *external* page is loaded into the DOM.	pagebeforeload pageload pageloadfailed	pagecontainerbeforeload pagecontainerload pagecontainerloadfailed
Create events	These events are triggered just before, or at the point widgets and plugins are added to a page.	pagebeforecreate * pagecreate * pageinit (* = not deprecated)	pagebeforecreate pagecreate
Show events	These events are triggered just before, or at the point, a page is shown or hidden.	pagebeforeshow pageshow pagebeforehide pagehide	pagecontainerbeforeshow pagecontainershow pagecontainerbeforehide pagecontainerhide

jQuery Mobile
Page Events – Binding

- Before version 1.4 all jQuery Mobile events could be bound at *page* level.

```
$(document).on("pagecreate", "#mypage", function() {
 . . .
});
```

- However, since version 1.4, load and show events (but not create events) must be fired at *pagecontainer* level.

```
$(document).on('pagecontainershow', function (event) {
. . .
});
```

- The pagecontainer equates to <body>. It does not equate to any particular page, contained within it.

jQuery Mobile Events – Binding

- Associating pagecontainer events with specific pages must be done programmatically, by singling out a particular page in our script.
- In the example below, *pagecontainer('getActivePage').attr('id')* is used to find the id of the current page. If the page id is equal to the page we wish to associate our script with, then, and only then, does the script run. Example.
- Note that there are also other ways of getting the current page (e.g. using the *toPage* method of the ui object).

```
$(document).on("pagecontainerbeforeshow", function (event) {
  var thisPage = $.mobile.pageContainer.pagecontainer( →
  'getActivePage' ).attr( 'id' );
  if( thisPage == 'personpage') {
     //run script
  }
});
```

Order of Events
(Internal Page to Internal Page)

- Given two pages A and B. If we are navigating from internal A to internal B, the following would represent the order of events:

1. page B---pagebeforecreate
2. page B---pagecreate
3. page A---pageconatinerbeforehide
4. page B---pagecontainerbeforeshow
5. page A---pagecontainerhide
6. page B---pagecontainershow

- Example

Order of Events
(Internal Page to External Page)

- Given two pages A and B. If we are navigating from internal A to external B, the following would represent the order of events:

1. page B---pagecontainerbeforeload
2. page B---pagecontainerload
3. page B---pagebeforecreate
4. page B---pagecreate
5. page A---pagecontainerbeforehide
6. page B---pagecontainerbeforeshow
7. page A---pagecontainerhide
8. page B---pagecontainershow

- Example

Load Events

- Load events are triggered at the point an *external* page is loaded into the DOM, but before the page widgets are created and enhanced, and before any transition takes place.

pagecontainerbeforeload	before page is loaded into DOM.
pagecontainerload	Triggered after a page is successfully loaded and inserted into the DOM.
pagecontainerloadfailed	Triggered if the page load request failed.

- Load events are triggered only once when the page is first loaded or refreshed with a page reload.
- In the following example we use *pagecontainerloadfailed* to check if a page has been loaded correctly, and to deal with the page failing to load.
- Example

Create Events

- Create events are triggered after loading, but before a page is shown.

pagebeforecreate	Triggered on the page being created, but before all widgets have had an opportunity to enhance the contained markup.
pagecreate	Triggered when the page has been created in the DOM, and after all widgets have had an opportunity to enhance the contained markup.

- Create events are triggered once (and once only) per page load: the first time the page is requested, or when the page is reloaded.
- In the following example, we use the *pagecreate* event to manipulate the enhancement of a jQM widget. Example.
- Note what happens if you change the event type from *pagecreate* to *pagebeforecreate*. Why does this happen?

Show Events

- Show events fire at the point of transition between two pages (to page and from page), after both pages have been fully loaded into the DOM, and their widgets have been created and enhanced.

pagecontainerbeforeshow	Triggered on the *to page* we are transitioning to, before the actual transition animation is kicked off.
pagecontainershow	Triggered on the *to page* after the transition animation has completed.
pagecontainerbeforehide	Triggered on the *from page* we are transitioning away from, before the actual transition animation is kicked off.
pagecontainerhide	Triggered on the *from page* after the transition animation has completed.

Show Events and Dynamic Data

- One notable difference between show events, and load and create events is that show events trigger each time a page is visited, whereas load and create events trigger only once – when a page is loaded or created.
- Thus, where we have pages that will change dynamically, we need to use show events to handle the update.

Dynamic Data

- jQM pages can be dynamically populated like any other web application pages.
- The external data source can be a file, a database, an RSS feed.
- The following example uses data from simple JSON files to dynamically update a simple page, though the techniques here would work just as well for more data intensive undertakings. Example.

Dynamic Data

- In the example, we use the value of a *querystring* to identify a particular data source to display data from.

```
<p><a href="person.html?id=1">Person1</a></p>
<p><a href="person.html?id=2">Person2</a></p>
```

- Here, the data source is simple JSON file with a number of key/value pairs.

```
{"name":"Martha Lyons","description":"Female"}
```

Dynamic Data

- When the target page is opened, the script extrapolates the querystring from the page URL using $(location).attr('search').
- It then gets the value of the querystring by splitting the query string value from the key. This is done using the JavaScript *split* method.
- *split* returns an array with two parts: [0], the part of the string before the split position; [1], the part of the string after the split position.

```
var thisPage = $.mobile.pageContainer.pagecontainer →
('getActivePage').attr('id');
    if( thisPage === 'personpage') {
     var thisUrl = $(location).attr('search');
     var thisId = thisUrl.split("=")[1];
 . . . .
```

Dynamic Data

- Once we have the querystring value, we then use it to build the path to our target data source.

```
"person" + thisId + ".json"
```

- This path is then used with the jQuery *get* method to open the targeted data source and extract our data.
- The *get* method takes three parameters: the data source to open, an anonymous function to deal with the data returned, and the data type being returned.

```
$.get("person"+thisId+".json", →
function(result, status) {}, "json");
```

Dynamic Data

- The anonymous function that deals with the returned data takes two parameters, *result* and *status*.
- Whatever data is returned from the data source will be contained in *result*.
- We can easily access that data and display it on our page in whatever form we wish. Example

```
$.get("person" + thisId + ".json",
  function(result, status) {
  var person = "<p>Name: " + result.name +"</p>;
  $("#contentArea", "#" + thisPage).html(person);
}, "json");
```