Utilize JavaScript in Magento

Describe different types and uses of JavaScript modules. Which JavaScript modules are suited for which tasks?

- plain requirejs module

  Good for executing regular JS code, maybe some legacy code that doesn't require active interactions with other existing JS components on the page.

```
<script type="text/x-magento-init">
{
    "*": {
        "mage/cookies": {
            "expires": null,
            "path": "/",
            "domain":
".m22ee.local",
            "secure": false,
            "lifetime": "3600"
        }
    }
}
</script>
```

  Element selector syntax is good for passing container element, so module can bind event listeners only within given element instead of querying DOM directly. Makes code tidier.

```
<script type="text/x-magento-init">
{
    "[data-gallery-role=gallery-placeholder]": {
        "mage/gallery/gallery": {
            "mixins":["magnifier/magnify"],
            "magnifierOpts":
{"fullscreenzoom":"20","top":"","left":"","width":"","height":"","eventType":"hover","enabled":"fa
            "data": [/*...*/],
            "options": {
                // ...
            },
            "fullscreen": {
                // ...
            },
            "breakpoints": {"mobile":{"conditions":{"max-width":"767px"},"options":{"options":
{"nav":"dots"}}}}            }
    }
}
</script>
```

- action - set some data, call AJAX
  - Magento_Checkout/js/action/select-payment-method.js
  - Magento_Customer/js/action/login.js - when called, calls AJAX, when done executes callbacks and adds messages
- shared model for data access - ko.observable properties
  - Magento_Customer/js/model/customer
  - Magento_Checkout/js/model/quote
  - some state - is loading etc. Magento_Checkout/js/model/shipping-address/form-popup-state.js
  - validation rules Magento_Ui/js/lib/validation/rules.js
- view - ui component
  - form Magento_Ui/js/form/form
  - grid Magento_Ui/js/grid/listing
  - minicart Magento_Checkout/js/view/minicart
- jQuery widget

```
{
  $.widget('mage.shippingCart', {
    ... // this.config,
this.element
  })
  return $.mage.shippingCart;
}
$(element).shippingCart(...)
```

Describe UI components.

Base class hierarchy:

- uiClass
- uiElement + uiEvents + links
- uiCollection

uiClass - Magento_Ui/js/lib/core/class

About the uiClass library

The uiClass is an abstract class from which all components are extended. The uiClass is a low-level class and is rarely used as direct parent for UI components' classes.

Adds support for extending classes `extend()` and calling parent methods `_super()`.

methods:

- initialize, initConfig

properties:

- contructor.defaults

uiElement - Magento_Ui/js/lib/core/element/element.js

About the uiElement class

Powers most of UI components internals. Adds support for features:

- automatic data linking between models
- automatic module loading
- working with observable properties
- stateful properties - stored automatically
- events - on/off/trigger

methods:

- initialize, initObservable, initModules properties:
- tracks, modules, source, provider,

initLinks:

- imports
- exports
- links == same value in imports and exports

```
imports: {
    totalRecords: '${ $.provider
}:data.totalRecords'
}
```

same as

```
imports: {
    totalRecords:
'example.example_data_source:data.totalRecords'
}
```

```
exports: {'visible': '${ $.provider }:visibility'}
imports: {'visible': '${ $.provider }:visibility'}
links: {'visible': '${ $.provider }:visibility'}
links: {value: '${ $.provider }:${ $.dataScope
}'}
```

```
listens: {'${ $.namespace }.${ $.namespace }:responseData': 'setParsed'}
listens: {'${ $.provider }:data.overload': 'overload reset validate'} -- call
multiple callbacks
listens:
  'index=create_category:responseData' => 'setParsed',
  'newOption' => 'toggleOptionSelected'
```

$.someProperty - property of the UI component in the scope of this component

conditions in string literals:

‘${ $.provider }:${ $.customScope ? $.customScope + “.” : “”}data.validate’: ‘validate’

initObservable

- observe([{Boolean} isTracked,] {String|Array|Object} listOfProperties)
- observe(‘var1 var2 var3’)
- observe([‘var1’, ‘var2’, ‘var3’])
- observe(true, ‘var1 var2 var3’) – isTracked, use property accessors
- observe(false, ‘var1 var2 var3’) – not tracked, use observable properties
- observe({var1: ‘value’, var2: true}) – initial values from object

initModules

```
defaults: {
    modules: {
        '%myProperty%': '%linkToTheComponent%',
        externalSource: '${ $.externalProvider
}',
        street: '${ $.parentName }.street',
        city: '${ $.parentName }.city',
        country: '${ $.parentName }.country_id',
        productForm: 'product_form.product_form'
    }
}
```

uiCollection - Magento_Ui/js/lib/core/collection.js

About the uiCollection class

Enables components to have nested child components:

- `elems` observable property
- initElement
- getChild, insertChild, removeChild, destroyChildren
- regions

Module_Ui/js/core/app, uiLayout - Module_Ui/js/core/renderer/layout

The uiLayout service object

Renders UI collection components structure, instantiates classes, sets children etc.

- app -> layout()

- run
- iterator
- process
- addChild - current node to parent
- manipulate - appendTo, prependTo, insertTo
- initComponent
    - loadDeps
    - loadSource (source='uiComponent')
    - initComponent
        - long async -> global function var component = new Constr(_.omit(node, 'children'));

In which situation would you use UiComponent versus a regular JavaScript module?

Overview of UI components

UI components work well together: they communicate with each other via the uiRegistry service that tracks their asynchronous initialization.

Therefore, if we need to extend something that has already been implemented as a hierarchy of UI components or add a new feature that should interact with other UI components, it's easier and more effective to use a UI component.

Describe the use of requirejs-config.js, x-magento-init, and data-mage-init.

requirejs-config.js

We are interested in:

- map as alias - same as defining virtual type preference in DI

```
var config = {
    map: {
        '*': {
            'Magento_Swatches/js/swatch-renderer' : 'Custom_Module/js/swatch-
renderer'
        }
    }
};
```

- map as preference - same as preference in DI, replace one class with another

```
var config = {
    map: {
        '*': {
            uiElement:
'Magento_Ui/js/lib/core/element/element',
        }
    }
};
```

- mixins - same as around plugins in DI. This is Magento customization over requireJS

```
var config = {
    config: {
        mixins: {
            'Magento_Checkout/js/action/place-order': {
                'Magento_CheckoutAgreements/js/model/place-order-mixin':
true
            }
        }
    }
};
```

Use `wrapper` model to implement around functionality for functions.

Place order mixin body:

```
define([
    'jquery',
    'mage/utils/wrapper',
    'Magento_CheckoutAgreements/js/model/agreements-assigner'
], function ($, wrapper, agreementsAssigner) {
    'use strict';

    return function (placeOrderAction) {

        /** Override default place order action and add agreement_ids to request
*/
        return wrapper.wrap(placeOrderAction, function (originalAction,
paymentData, messageContainer) {
            agreementsAssigner(paymentData);

            return originalAction(paymentData, messageContainer);
        });
    };
});
```

Plain requirejs objects can be extended directly.

Example - mixin over Magento_Checkout/js/model/quote:

```
define([
    'jquery',
    'ko'
], function($, ko) {
    'use strict';

    var checkoutMethod =
ko.observable(null);

    return function(quote) {
        return $.extend(quote, {
            checkoutMethod: checkoutMethod
        });
    };
});
```

Text/x-magento-init and Data-mage-init

Alan Storm - Javascript Init Scripts

Following 3 snippets are completely equivalent:

```
<div class="selector" data-mage-init="{'Custom_Module/js/something': {option1:
'value1', option2: 'value2'}}"></div>
```

```
<div class="selector"></div>
<script type="text/x-magento-
init">
{
  '.selector': {
    'Custom_Module/js/something': {
       option1: 'value1',
       option2: 'value2'
    }
  }
}
</script>
```

```
<div class="selector"></div>
<script>
require(['Custom_Module/js/something'], function(fn) {
    fn({option1: 'value1', option2: 'value2'},
DOMElement('.selector'));
});
</script>
```

How Magento executes scripts:

- mage/bootstrap.js
- lib/web/mage/apply/scripts.js
  - parses init scripts `document.querySelectorall('script[type="text/x-magento-init"]')`
  - converts to data-attribute format on according element, e.g. `<div data-mage-init="...">`
  - scripts with `*` selector (virtuals, no node to assign to) are collected separately
- mage/apply/main.js
  - parses data-attribute init scripts `document.querySelectorAll('[data-mage-init]')` - including converted `<script type="text/x-magento-init">`
  - merges virtuals from above (`*` selector)
  - creates components from parsed definitions

In other words:

- we return component-function
- magento gets it via require and invokes with arguments. E.g. uiComponent(config, node)
- node '*' -> false
- selector matching multiple nodes - multiple module instances (like jQuery plugin)

Benefits:

- top-level selector is defined outside js module
- initial values from server are defined outside modules

Example:

```
<div class="selector-one" data-mage-init="{'Custom_Module/js/something':
{option1:'', option2:''}}"></div>
<div id="selector-two"></div>

<script type="text/x-magento-init">
{
  '#selector-two': {
    'Custom_Module/js/component': {
       mixins: [],
       option1: value1,
       option2: value2
    }
  }
}
</script>

<script type="text/x-magento-init">
{
    "*": {
        "Magento_Ui/js/core/app": {
            "components": {
                "customer": {
                    "component": "Magento_Customer/js/view/customer"
                }
            }
        }
    }
}
</script>
```

Transforms into following:

```
<div class="selector-one" data-mage-init="{'Custom_Module/js/something':
{option1:'', option2:''}}"></div>
<div id="selector-two" data-mage-init="{'Custom_Module/js/component': {mixins: [],
option1: value1, option2: value2}}"></div>
<script>
// somewhere in closures...
virtuals[] = {
  el: false,
  data: '{"Magento_Ui/js/core/app": {"components": {"customer": {"component":
"Magento_Customer/js/view/customer"}}}}'
}
<script>
```

And then JS is executed like this:

```
require(['Custom_Module/js/something'], function(fn) {
  // your component should return function-constructor with 2 params: (config,
$element)
  fn({option1:'', option2:''}, DOMElement('.selector-one'));
});
require(['Custom_Module/js/something'], function(fn) {
  fn({option1: value1, option2: value2}, DOMElement('#selector-two'));
});
require(['Magento_Ui/js/core/app'], function(appFn) {
  // element = false when selector '*'
  appFn({"components": {"customer": {"component":
"Magento_Customer/js/view/customer"}}}, false);
})
```

There’s interesting possibility to return jQuery widget instead of contructor function. Magento detects this and calls widget accordingly.

Here’s how:

```
require([component], function (fn) {
    if (typeof fn === 'object') {
        fn = fn[component].bind(fn);
    }

    if (_.isFunction(fn)) {
        fn(config, el);
    } else if ($(el)[component]) {
        $(el)[component](config);
    }
});
```

Mixins example

```
<script type="text/x-magento-init">
    {
        "[data-gallery-role=gallery-placeholder]":
{
            "mage/gallery/gallery": {
                "mixins":["magnifier/magnify"],
                "magnifierOpts": ...,
                "data": ...,
                "options": ...
            }
        }
    }
</script>
```

```
element = $('[data-gallery-role=gallery-placeholder]');
config = {magnifierOpts:..., data:..., options:...}

require(['mage/gallery/gallery'], function(gallery) {
  require(["magnifier/magnify"], function(magnify) {
    var result = magnify(config, element); // call for each mixin. Magnify will
use config.magnifierOpts
    extend(config, result); // magnify returns same config, config not changed
  })
  gallery(config, element);
});
```