ArcGIS API for JavaScript Programming Patterns and API Fundamentals

Julian Kissling | Rene Rubalcava

slides: https://git.io/Je0yr
What do I get with the 4x JSAPI?

- Simplified and consistent API
- Write apps in ES6 or TypeScript
- Modern browser support (IE11+)
- Supported in 30+ locales
What are my options?

- Needs?
- Resources?
- Time?
- Customizations?
Why start from scratch?

- App starters
- Widgets
Widgets!

- We'll look at a few widgets
- ~30 Widgets out of the box
- Widgets help make great apps
- Less code for you to write
- Designed with responsive apps in mind
Widgets - Expand

- Clickable button to open container
- Icons
- Group
- Mode
Widgets - Use Portal Content

- Search
- Basemap Gallery
Widgets - Popup

- Menu actions
- Dock
Widgets - Popup Template

- Define fields, charts, custom html content
- Using a function
- Promises
- Arcade
Widgets - Author popup in online

- Web Map or Web Scene
- Layer.fromPortalItem
- App Demo
Widgets - Feature

- Display popup template content
- Hover
Widgets - Architecture

View + View Model

**SearchViewModel**

```javascript
require(['esri/widgets/Search/SearchViewModel'], function(SearchVM) { /* code goes here */ });
```

**Class:** esri/widgets/Search/SearchViewModel

**Inheritance:** SearchViewModel → Accessor

Since: ArcGIS API for JavaScript 4.0

Provides the logic for the Search widget, which performs search operations on locator service(s) and/or map/feature service feature layer(s). If using a locator with a geocoding service, the `findAddressCandidates` operation is used, whereas queries are used on feature layers.
View Models

- Custom View
- Use the view model
  - Additional Examples
const locator = new Locator({ url: url });
locator.addressToLocations({
    address: {
        "singleLine": "380 New York St, Redlands, CA 92373"
    }
});
locator.locationToAddress({ location: point });
const searchVM = new SearchVM();
searchVM.search("380 New York St, Redlands, CA 92373");
searchVM.search(location);
Geocoding

const portal = new Portal(...);
await portal.load();
portal.helperServices.geocode.map(geocoderService => {
    // objects with details on

    // geocode services for your portal
});
Widgets - Styling

Available Themes
Widgets - Styling

- CSS Extension language
- SASS
- Theme Utility
Map and View
const map = new Map({
    basemap: "topo"
});

const mView = new MapView({
    map: map,
    container: "viewDiv"
});

const sView = new SceneView({
    map: map,
    container: "viewDiv"
});
const map = new Map({
    /*
        streets, satellite, hybrid, terrain, topo, gray,
        dark-gray, oceans, national-geographic, osm,
        dark-gray-vector, gray-vector, streets-vector, topo-vector,
        streets-night-vector, streets-relief-vector, streets-navigation-vector
    */
    basemap: "streets"

    /*
        world-elevation
    */
const map = new Map({
  basemap: {
    // Layers drawn at the bottom
    baseLayers: [
      new TileLayer({ url: baselayer })
    ],
    // Layers drawn on top
    referenceLayers: [
      new TileLayer({ url: refUrl })
    ],
  },
  ground: {}
Basemap and Ground
Collections

- esri/core/Collection
Working with Accessor

- Objects are have properties that can be:
  - read and set
  - or read-only
  - constructor arguments
  - watchable
Accessor - property access

```javascript
layer.opacity = 0.5;
layer.title = "My test layer";

// setting multiple values
layer.set({
  opacity: 0.5,
  title: "My test layer"
});

// accessing the value of a deep property
view.get("map.basemap.title");
view.set("map.basemap.title", "new title");
```
Accessor - property watching

```javascript
mapView.watch("scale", (newValue, oldValue, property, target) => {
    console.log(`scale changed: ${newValue}`);
});

mapView.watch("map.basemap.title", (newValue, oldValue, property, target) => {
    console.log(`new basemap title: ${newValue}`);
});

mapView.watch("ready, stationary", (newValue, oldValue, property, target) => {
    console.log(`property ${property}: ${newValue}`);
});

watchUtils
```
Accessor - autocasting and single constructor

```javascript
// 4.x
{
  type: "simple-marker",
  style: 'square',
  color: 'red',
  size: 10,
  outline: {
    color: 'rgba(255, 255, 255, 0.5)
    width: 4
  }
}
```
Promises
Promises

- All asynchronous methods return a promise, no more events
- The basic pattern looks like this:

```javascript
layer.queryFeatures(query).then(handleResult).catch(handleError);
```
Promises with async/await

- work with native promises

```javascript
const doQuery = async (query) => {
    const results = await layer.queryFeatures(query);
    const transformedResults = results.map(transformData);

    return transformedResults;
};
```
Promises

- Load resources
- Asynchronously initialized `Layer`, `WebMap`, `WebScene`, `View`

```javascript
const map = new Map({...})

view = new SceneView({
  map: map,
  //...
});

view.when(() => {
  // the view is ready to go
});
```
Promises

view.when() => {
  return view.whenLayerView(map.findLayerById("awesomeLayer"));
}
.then(layerView => {
  return watchUtils.whenFalseOnce(layerView, "updating");
})
.then(result => {
  const layerView = result.target;
  return layerView.queryFeatures();
})
.then(doSomethingWithFeatures)
.catch(errorHandler);

API sample
```javascript
const init = async (doSomethingWithFeatures) => {
  await view.when();
  const layerView = await view.whenLayerView(map.findLayerById("awesomeLayer"));
  const { target as layerView } = await watchUtils.whenFalseOnce(layerView, "updating");
  const features = await layerView.queryFeatures();

  doSomethingWithFeatures(features);
};

try {
  init();
} 
catch(error) {
```

Patterns
Interactivity with view events

- Use view events to interact with the view
- List of events
- You can stop the propagation of the event to prevent the default behavior

```javascript
view.on("drag", event => {
    // user won't be able to drag
    event.stopPropagation();
});
```
Interactivity with view events

- Access the features on click

```javascript
view.on("click", (x, y) => {
  const screenPoint = {x, y};
  view.hitTest(screenPoint)
    .then(response => {
      // do something with the result graphic
      const graphic = response.results[0].graphic;
    });
});
```

- API Sample
goTo() with View

- Sets the view to a given target.
  - Navigate to a geometry/feature/location
- API Sample
Loadables

- brings better control, and scheduling of loading resources.
- extension of `esri/core/Promise`
- in 3.x, instanciating a layer loads it. in 4.0, it's an explicit call
- the views automatically loads the map and its layers
Loadables

• **WebMap/WebScene need to load:**
  ▪ the portal item
  ▪ the layer module
  ▪ the layer's item

• **MapView/SceneView need to load:**
  ▪ the map
  ▪ the layers
// In a single page application, get a feature from a FeatureLayer from a WebMap without displaying:

const webmap = new WebMap({
  portalItem: {
    id: 'affa021c51944b5694132b2d61fe1057'
  }
});

webmap.load().then(() => {
  return webmap.getLayer('myFeatureLayerId').load();
}).then(featureLayer => {
  // Get feature from featureLayer
});
Zoom or Scale

const view = new MapView({
  container: "viewDiv",
  map: map,
  center: [-116.5, 33.80],
  zoom: 14 // what does that really mean?
});

- Zoom = LOD (Level of Details)
- Not all LODs are created equal
Zoom is not Scale

const view = new MapView({
    container: "viewDiv",
    map: map,
    center: [-116.5, 33.80],

    scale: 50000 // I know what that means!
});

• Scale is portable
• Scale has meaning
• We still snap to closest LOD/zoom
WebMap is still a Map

const map = new WebMap({
  basemap: { ... },
  layers: [ ... ]
});

- Still acts like a regular Map
- Has some advantages
WebMap is still a Map
Sublayer to FeatureLayer

- You can extract a FeatureLayer from MapImageLayer Sublayer
- `sublayer.createFeatureLayer()`
- Can use capabilities not normally available with Sublayer
Sublayer to FeatureLayer
createQuery

- When you can do `layer.createQuery()`
  - `query` object will already have the layers filters and layer definitions
  - more consistent
- Use `new Query()` when you don't want predefined filters to be applied
createQuery
MapImageLayer

- If you want to modify Sublayers, do it after you load the layer
- Defining them upfront overrides the defaults
  - May not be what you want
MapImageLayer
LayerViews

- Renders the Layer
- When is it done though?
  - *hotly debated topic!*
  - When can you actually use it!!
  - Behavior different with optimized FeatureLayer
LayerViews
Please Take Our Survey on the App

Download the Esri Events app and find your event

Select the session you attended

Scroll down to find the feedback section

Complete answers and select “Submit”