Writing JavaScript in ActionScript

FlexJS World Tour

San Francisco, California
April 4, 2016

OmPrakash Muppirala
Apache Flex™

Summary

- Why cross-compile AS to JS?
- Manipulate HTML DOM
- SVG too
- Animation
- Externs (Jquery, Angular, Material Design)
- Web Components: Custom Elements
- Questions

Why cross-compile ActionScript to JavaScript?

- True Object Oriented Programming
  - Classes
  - Interfaces
  - Encapsulation
  - Polymorphism
  - Design Patterns

Why cross-compile ActionScript to JavaScript?

- Statically Typed
  - Type checking done by compiler
  - Catch ‘stupid’ bugs during compile time
    - No spelling mistakes
    - No accidentally creating global variables
  - Cut down unit tests that exist only to test stupid bugs
  - Optional dynamic typing

Why cross-compile ActionScript to JavaScript?

- IDE Support
  - Use any of the supported IDEs: Flash Builder, IntelliJ Idea, FDT, MoonShine, etc.
  - Code completion (more in a bit)
  - Debug via SWF/AIR runtime

Manipulate HTML DOM

```
var container:HTMLDivElement =
document.createElement('div') as
HTMLDivElement;
container.style.width = '100%';
container.style.height = '100%';
document.body.appendChild(container);
```

SVG!

```
//Create SVG element
var svg : SVGElement =
document.createElementNS(SVG_NAMESPACE_URI, "svg") as
SVGElement;
svg.setAttribute("width", "200");
svg.setAttribute("height", "200");
document.body.appendChild(svg);

//Create Circle element
var circle:SVGCircleElement =
document.createElementNS(SVG_NAMESPACE_URI, "circle") as
SVGCircleElement;
circle.setAttributeNS(null,"cx", "50");
circle.setAttributeNS(null,"cy", "50");
circle.setAttributeNS(null,"r", "50");
circle.setAttributeNS(null,"fill", "green");
svg.appendChild(circle);
svg.addEventListener("click", button_clickListener, false);
```

Let's Animate!

```
private var circleRadius:Number = 50;
private function animateUp():void {
    circleRadius += 1;
    if(circleRadius > 100)
    {
        return;
    }
    circle.setAttributeNS(null,"r",
    circleRadius);
    requestAnimationFrame(animateUp);
}
```

Externs

- Externs are API signatures (interfaces) of third party libraries that can be directly accessed from ActionScript during compile time
- Runtime implementation comes from the 3rd party library itself

Externs

- FlexJS has a built in extern: js.swc
  - Provides all the APIs necessary to access and manipulate the HTML(5)/SVG DOM
- Tooling available (outside of Apache Flex) to create extern files for any third party JavaScript library (Check out Josh Tynjala’s nextgenactionscript.com)

Externs - JQuery

```
$(circle).animate({opacity: 0.25,
                   r: "toggle"});
$(circle).fadeIn();
$(circle).click(handleCircleClick);
```

Externs - AngularJS

```
//Define the Angular App
//Add dependencies
var app:IModule = angular.module("app",["ngMaterial"]);

//Add an AngularJS controller
app.controller("MyController", ["$scope", "$mdDialog",
MyController]);

//Set ng-app attribute on the body element
document.body.setAttribute("ng-app", "app");
```

Your AngularJS app is ready!

Externs - AngularJS

```
//AngularJS Controller class (yes, a proper class!)
public class MyController {
       //$scope and $mdDialog gets injected by AngularJS
       private var $scope:IScope;
       private var $mdDialog:MDDialogService;

       public function
MyController(scope:IScope,mdDialog:MDDialogService) {
       this.$scope = scope;
       this.$mdDialog = mdDialog;

//Anything added to $scope is available for databinding from
html
       this.$scope["handleBtnClick"] = this.handleBtnClick;
       this.$scope["close"] = this.close;
       this.$scope["myDate"] = new Date();
       this.$scope["btnLabelStr"] = "Click me";
}
```

Externs – Material Design

- Just add a Material Design directive to the DOM
- AngularJS + Material Design takes care of the rest

```
div.innerHTML += '<md-button
                   id="myBtn"
                   class="md-primary md-raised"
                   ng-click="handleBtnClick()">
                         {{btnLabelStr}}
                   </md-button>';
```

Web Components: Custom Elements

- **Custom Elements** allow web developers to define new types of HTML elements
- The spec is one of several new API primitives landing under the **Web Components** umbrella
- Lets us create custom HTML elements
- Supported by js.swc extern library
- It’s changing constantly, so this stuff might not work in a few months!

Web Components: Custom Elements

```
public class WebComponent extends HTMLElement implements
IWebComponent {

        protected var shadowRoot : ShadowRoot;

        //Lifecycle method
        public function createdCallback() : void {
                shadowRoot = this['createShadowRoot']();
                setupComponent();
        }

        public function setupComponent() : void {
        //override in subclass
        }
}
```

Web Components: Custom Elements

```
public class MDButton extends WebComponent
{
…
       override public function setupComponent():void {
              createLabel();
              createIcon();
       }

       protected function createLabel():void
       {
              label = ownerDocument.createTextNode("");
              shadowRoot.appendChild(Node(label));
       }
…
}
```

Everything comes together

Demo

Questions?

- Wiki page: s.apache.org/flexjswiki
- Mailing list : s.apache.org/flex-dev-forum

Twitter: @bigosmallm

That's all, folks!