

A word from our sponsor

- Dell's Vancouver's software product development team is growing!
- We're a small dynamic team using cutting edge products to develop great software.
- If you have a passion for software, come speak with our team about the opportunities at Dell.
- We have current openings for developers (Front End and Java), QA and Pre-Sales Engineers.

About me

- Tommy Chan
- Polyglot developer interested in web technologies and different languages
- Reach me at tommytcchan@gmail.com
- Github (for slides and demo code):
tommytcchan

Introduction to Knockout.js

The logo for Knockout.js, featuring the word "Knockout." in a white, stylized, cursive font with a thick underline, set against a red-to-orange gradient background.

Agenda

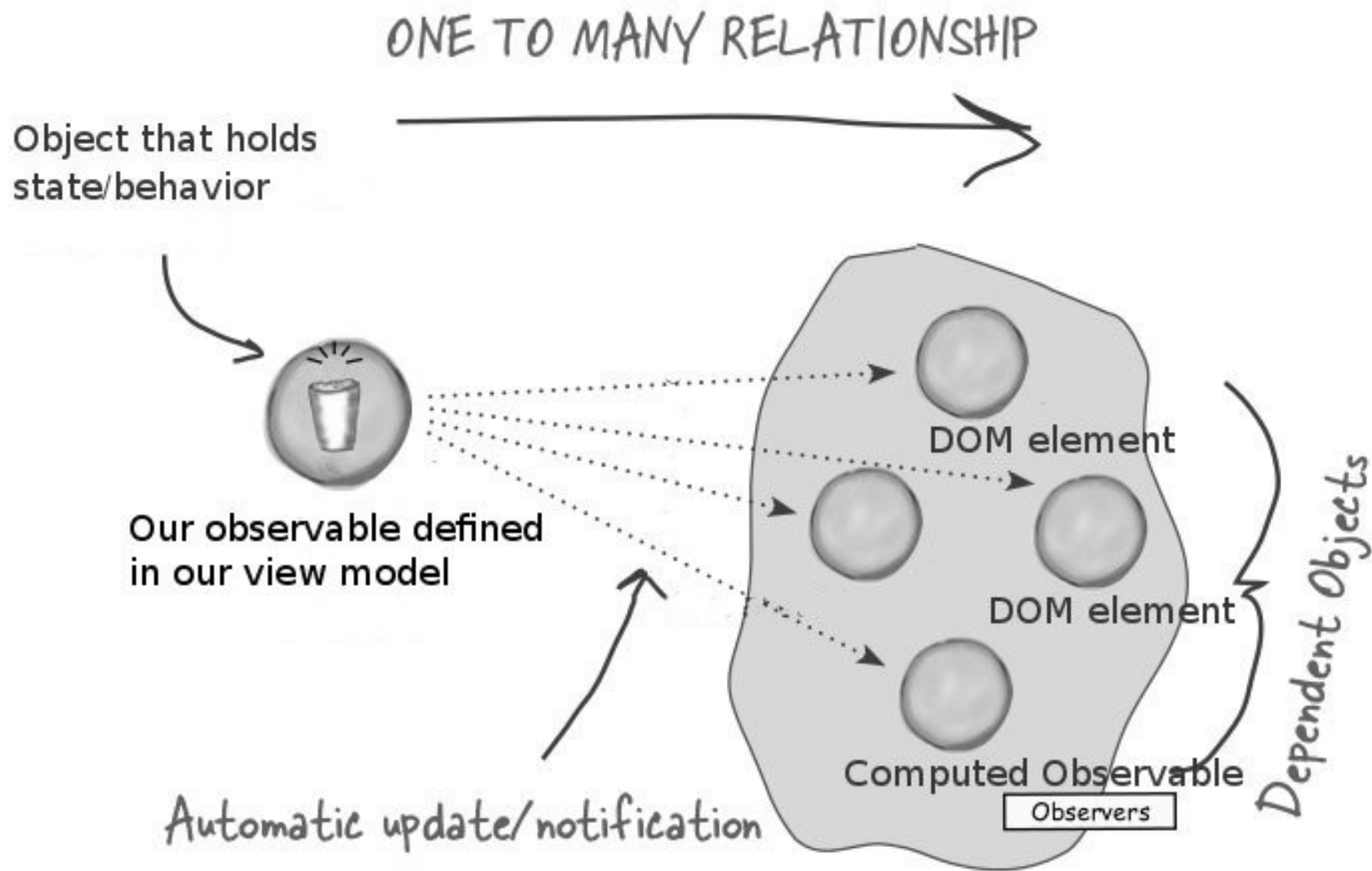
- Review the Observer pattern
- Introduction to the MVVM pattern
- Introduction to Knockout
- Observables/Observable arrays
- Default Bindings
- Custom bindings
- Using the mapping plug-in
- Common pitfalls/Gotchas

Review of the Observable Pattern

- Central to the KnockoutJS framework is the Observer pattern.
- The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

“Don't call me, I'll call you.”

Observer pattern - visually



MVVM - motivation

- Want to decouple the view and the view's data/behaviors (the view model).
- View defines how it should be displayed.
- The view model defines your data and behaviors.
- Knockout will manage the synchronization between your view and your view model (using the observer pattern).

Introducing the MVVM model

- MVVM stands for Model-View-ViewModel.
- The M and the V stands for the Model and View respectively.
- The difference here is the View Model, which as a model for the view.
- Theoretically this separates our data/behavior on the client side with the view, which is essentially the pattern.

Why use KnockoutJS

- Version 2.1.0
- Open sourced (MIT license)
- Under active development
- Tested/Works with different mainstream browsers
- Awesome tutorial

A sample teaser...

```
<script type="text/javascript" src="js/knockout-2.1.0.js"></script>
</head>
<body>
  <p>Choose your costume:
    <select data-bind="options: availableCostumes, optionsCaption: 'Choose one...', value: selectedCostume"></select>
  </p>
  <div data-bind="with: selectedCostume">
    On Halloween this year, I will be a <span data-bind="text: $root.makeScaryCostume"></span>.
  </div>
</body>
<script type="text/javascript">
  function CostumeViewModel() {
    var self = this;
    self.selectedCostume = ko.observable();
    self.availableCostumes = ko.observableArray(['Pirate', 'Ghost', 'Banana']);
    self.makeScaryCostume = ko.computed(function() {
      return "very scary " + self.selectedCostume();
    });
  };
  model = new CostumeViewModel();
  ko.applyBindings(model);
</script>
</html>
```

Defining the view model

- First thing that we need to do is to define the view model that contains the model/behaviors to be associated with the HTML elements.
- This is done via a view model object.
- The view model can contain functions, variables, and observables.
- Example: `example1.html`

Binding the view model to the view

- KO uses the HTML 5 compliant custom 'data-bind' attribute to bind DOM elements to the view model
- Example usage:
 - Hello, my name is: ``
- Example: `onewaybinding.html`

Binding the view model to the view

- The data-bind attribute can accept multiple bindings.
- For example, a dropdown box can be bound to an observable array to populate its values and an observable so that the selected value is stored.
- Example: `example1.html`

Binding the model with the view

- To associate the view model with the view, a call to `ko.applyBindings(model)` needs to be made.

- Eg.

```
ko.applyBindings(new CostumeViewModel());
```

- This is only required once!

Observables

- In order to take advantage of using Knockout, we need observables.
- Declare an observable like so:

```
var viewModel = {  
    tweetContent: ko.observable();  
}
```
- Whenever the value changes, it will notify everyone who subscribes to it (ie. DOM elements, derived observables).

Observables

- To read the value of an observable, invoke it:
`myViewModel.name();`
- To write the value, set it as a param:
`myViewModel.name('Tommy');`
- To write values to multiple observable properties, can using chaining:
`myViewModel.name('Tommy').costume('pirate');`

Observable Arrays

- Detects and responds to changes of a collection of things.
- Note that KO only tracks the objects in the array, **NOT** the state of those objects.
- Notifies listeners when the objects in the array are added or removed.

Observable arrays

- Property instantiated with the `ko.observableArray(array)` syntax:

```
self.tweets = ko.observableArray([
    new Tweet("tweet1"),
    new Tweet("tweet2")
]);
```

- Observable arrays provides convenience methods:

```
tweet.remove(myOldTweet);
tweet.add(myNewTweet);
```

- Convenience methods work in a consistent way across all browsers, so use them instead of native methods.

Remove() vs Destroy()

- Knockout provides a way for you to conveniently mark objects in arrays to be deleted (so you can send it back to the server), using the `destroy()` method.
- When using `destroy()`, KO will add a flag `_destroy` to each of the items to be removed, instead of actually removing it.
- Destroyed items won't display on the UI, but will show up and objects to be deleted as a `_destroy` flag set.

Computed Observables

- These are functions that are dependent on one or more observables, and will automatically update whenever any of these dependencies change.
- For example:

```
function AppViewModel() {  
    this.firstName = ko.observable("Tommy");  
    this.lastName = ko.observable("Chan");  
    this.fullName = ko.computed(function() {  
        return this.firstName() + " " + this.lastName() + " is scary.";  
    });  
}
```

Bindings

- Bindings define what actions they should perform given the model data.
- KO provides a set of bindings that you can use for showing elements on the UI.
- For example:
 - The default **text/html** binding:
`<div data-bind="html: details"></div>`

Default bindings

- The **visible** binding:

```
<div data-bind="visible:
shouldShowMessage">visible</div>
```

- The **css/style** binding:

```
<div data-bind="style: { color:
currentProfit() < 0 ? 'red' : 'black' }">
Profit Information</div>
```

- The **attr** binding:

```
<a data-bind="attr: { href: url, title:
details }">Report</a>
```

Default bindings

- KO also has bindings to the input/form elements.

- The **value** binding

```
<p>Login name: <input data-bind="value: userName" /></p>
```

- The **submit** binding

```
<form data-bind="submit: doSomething">
```

```
    ... form contents go here ...
```

```
<button type="submit">Submit</button>
```

```
</div>
```

- The **event** binding

```
<div data-bind="event: { mouseover: enableDetails,  
mouseout: disableDetails }">Mouse over me</div>
```

Control Flow

- Knockout supports a few other bindings related to control flow:

- **Foreach**, **if/ifnot**, and **with**:

```
<tbody data-bind="foreach: people">
```

```
  <tr><td data-bind="text: firstName"></td></tr>
```

```
</tbody>
```

```
<div data-bind="if: displayMessage">Here is a message.  
Astonishing.</div>
```

```
<div data-bind="ifnot: loggedIn">Please login.</div>
```


With binding

- Dynamically add/remove view elements depending on whether the associated value is null/undefined or not.
- Used to display areas of your HTML... good for single page apps.
- Example: example1.html

Let's go back to our example

- With our knowledge, let us revisit the example earlier.
- Example: example1.html

Custom bindings

- Custom binding provide flexibility for more complicated use cases.
- Let's take a look at how a custom binding looks like...
- Example: `custom-binding.html`

Custom bindings

- A couple of things to note:
- 'update' function should be provided.
- Optional 'init' function can filled in as well.

Mapping JSON data as observables

- There is a cool plugin called the Mapping plugin, that will automatically bind JSON objects to observables.
- Essentially takes this:

```
var data = getDataUsingAjax();  
viewModel.serverTime(data.serverTime);  
viewModel.numUsers(data.numUsers);
```

Mapping JSON data as observables

- There is a cool plugin called the Mapping plugin, that will automatically bind JSON objects to observables.
- Essentially takes this:

```
var data = getDataUsingAjax();  
viewModel.serverTime(data.serverTime);  
viewModel.numUsers(data.numUsers);
```

Mapping Plugin

- ..and turn it to this:

```
var viewModel = ko.mapping.fromJS(data);
```

- Then every time data is sent from the server, you can update the UI like so

```
ko.mapping.fromJS(data, viewModel);
```

Quiz #1

- Time for a giveaway. What is the output in the text field for this example?
- Example: quiz1.html

Best practices

- Store all the application data in one place

```
<p>First name: <strong data-bind="text: firstName"></strong>Tommy</p>
```

```
this.firstName = ko.observable();
```

Do this instead:

```
<p>First name: <strong data-bind="text: firstName"></strong></p>
```

```
this.firstName = ko.observable('Tommy');
```

- With the first way, KO does not know the value!

Common sources of errors

```
update: function(element, valueAccessor) {  
    var currentValue = valueAccessor();  
    $(element).button("option", "disabled", currentValue.enable ===  
        false);  
}
```

```
update: function(element, valueAccessor) {  
    var observable = valueAccessor();  
    $("span", element).each(function(index) {  
        $(this).toggleClass("chosen", index < observable());  
    });  
}
```

- In one case the valueAccessor is a return that returns a value, whereas in the other case the valueAccessor returns a function.

Common source of errors

- It is a mistake to call `ko.applyBindings()` more than once.
- Common for people new to the framework to call `applyBindings()` for each ajax request.
- An example of why you shouldn't do this...
- See [applybindings-twice.html](#)

Gotcha: Observable Arrays

- Consider this example with observable arrays:

```
tweets = ko.observableArray({id: 1,  
author: 'tommytcchan'}, {id: 2, author:  
'somebody'});
```

```
tweets.remove({id: 1, author:  
'tommytcchan'});
```

- What will happen??

Fix: Observable Arrays

- The solution is to pass a filter to the remove() method:

```
tweets.remove(function(item) {  
    return item.id === 1;  
});
```

- KO will iterate through the tweets collection and remove the one with the matching id.

Common sources of errors

- Consider this example:

```
<div data-bind="with: selectedCostume">  
    On Halloween this year, I will be a <span  
    data-bind="text: makeScaryCostume"></span>.  
</div>
```

- Error is thrown by KO – it creates a context inside the DOM tree structure, so it is unable to reference the view model directly within the context.
- Use the built-in \$root instead to refer to the main model. (ie. \$root.makeScaryCostume)

A note about manual DOM injection and KO

- Try to not manipulate the DOM on your own natively (using jQuery or any other libraries).
- Knockout uses the observable pattern for the view and view model, so use that mechanism instead.
- There is logic that KO runs when it thinks a bound HTML element is no longer part of the DOM.

Not so positives

- Documentation on the ko.util methods are not so great...have to read the source code.
- One can argue that it's not a clear separation between view and view model with bindings and logic in the view.

Questions?

- Q + A.
- Comments?
- Slides and examples posted on github.