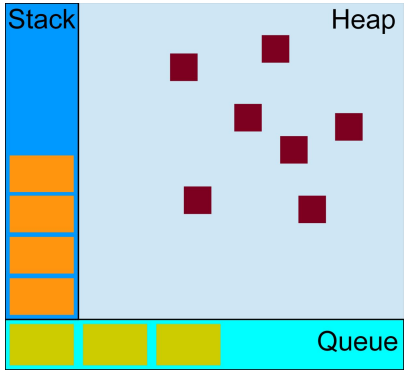
AngularJS Filters

and review of async JS
based on slides by David Herrera

Review - Async JS

What are the three parts of the JavaScript run-time and what purpose does each one serve?

Solution



- The heap, the stack and the queue.
- **Heap:** Dynamically allocates memory
- **Stack:**
  - Main execution engine
  - Functions are pushed to the stack, the stack executes them and then pops them off.
- **Queue:**
  - Takes care of asynchronous requests:
    1. Calls to the *async* API are recognized and their **callbacks** are added to the queue.
    2. When the stack is empty, the **event loop** checks the queue to see whether a response to an async operation has arrived.
    3. Callbacks and responses are passed to the stack to be processed.

- What are *promises*?
- What does *promisifying* mean?
- What are the three states of promises?

Solution

- A *promise* is an object that will eventually contain a result from an asynchronous call.

- *Promisifying* is the process of converting asynchronous functions to make them return a promise.

- The three states of promises are:
  - Pending
  - Fulfilled
  - Failed

Opal Promise Creation Example

```
function requestToServer(request, params)
{
    var deferred = $q.defer();
    var db = firebase.database();
    var key = db.set("request",{"name":request, parameters:params});
    db.ref("response"+"/"+key).once("value", function(snapshot){
        deferred.resolve(snapshot.value());
    }).catch(function(err){
        deferred.reject(err);
    });
    return deferred.promise;
}
```

How Do We Call a Promise?

- Once we have *'promisified'* a function, how do we call it?
  - Use the **then/catch** promise semantics.

```
// Suppose function okToGreet exists
function asyncGreet(name) {
    var deferred = $q.defer();

    setTimeout(function() {
        if (okToGreet(name)) {
            deferred.resolve('Hello, ' + name + '!');
        } else {
            deferred.reject('Greeting ' + name +
                ' is not allowed.');
        }
    }, 1000);
    return deferred.promise;
}
```

```
asyncGreet('Robin Hood')
    .then(function(greeting){
    alert('Success: ' + greeting);
    }).catch(function(error){
    alert('Failed: ' + reason);
});
```

Cases

- **Scenario 1:** One simple async request (shown previously).
- **Scenario 2:** Two or more simple requests that **depend** on one another.
- **Scenario 3:** Two or more simple requests that **do not depend** on one another.
- Every other scenario is a combination of these three.

Scenario 1 - Example

```
fetchUrlContent(imageUrl)
    .then(function(content){

    }).catch(function(error){

    });
```

Scenario 2 - Example

```
// Assume getImages function exists, which fetches
// the images from conversations
requestToServer("GetConversations",{userId:1})
    .then(function(response){

        return getImages(response.data.conversations);
    }).then(function(conversationsWithImages){
        // Handle conversations
    })
    .catch(function(error){ alert(error); });
```

Scenario 3 - Example

```
function getImages(conversations){
    var promiseArray = [];
    for(var i = 0; i< conversations.length; i++)
    {
        promiseArray.push(fetchUrlContent(conversations[i].imageUrl));
    }
    return $q.all(promiseArray).then(function(images){
        images.forEach(function(image,index){
            conversations[index].image = image;
        });
        return images;
    });
}
```

AngularJS Filters

What is a filter?

- In AngularJS, a **filter** is used to format the value of an expression for display to the user or to manipulate an expression in the code.
- Filters allow you to play around with your data and apply different **transformations**.
- Examples:
  - Filtering a list, see filter list.
  - Sorting a list, see orderBy.
  - Applying a function map to an array, for instance, to transform all the dates from text format to JS dates.

Types of Filters

- Custom-made (see custom filters).
- AngularJS built-in filters (see built-in filters).
  - orderBy
  - filter
  - date
  - lowercase
  - uppercase
  - limitTo
  - json

Filter Calls

- Filters can be called in controllers, views or services.
  - In a **view**, filters can be used to filter, order lists with the ng-repeat attributes, or change display formats.
  - In a **controller**, filters can be used to apply custom sorting based on a specialized function.
  - In a **service**, filters can be used to organize the model data in a certain way while the app is loading.

Filters in Views (HTML)

- **Filter array of conversations by substring:**

```
<ons-list-item class="timeline-li" modifier="tappable"
               ng-repeat="conversation in vm.conversations | filter: vm.searchConversationString"
               ng-click="vm.goToConversation(conversation)">
```

- **Text formatting, reverse (applies to lists too):**

```
<div >
    <input ng-model="greeting" type="text"><br>
    No filter: {{greeting}}<br>
    Reverse: {{greeting|reverse}}<br>
    Reverse + uppercase: {{greeting|uppercase}}<br>
    Date: {{someDate | date: 'MMM d, y h:mm:ss a' }}<br>
</div>
```

Filters in Controllers and Services (JS)

- **Format:**

```
$filter("<name-filter>")(argument1, argument2);
```

- **Example:**

```
var conversations  = $filter("orderBy")(conversations, 'lastMessage',sortFunction);
function sortFunction(lastMessage1, lastMessage2)
{
    //Comparator function, returns boolean
}
```

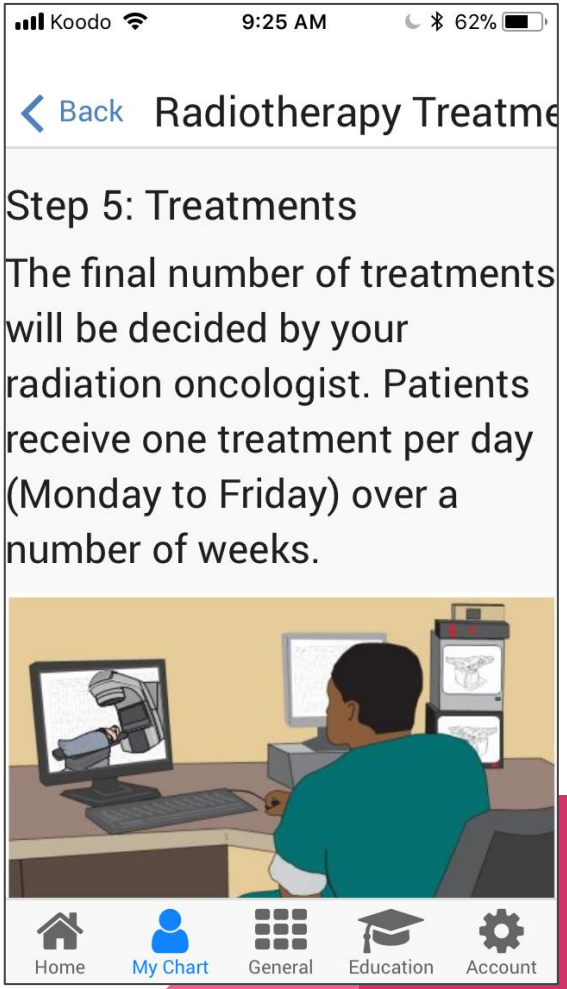
- **Note:** You must inject the $filter dependency into controllers and services.

Custom Filters

- "Radiotherapy Treatment" overfills the header. What can we do?

Solutions:

- Change location of text.
- Recognize this and set a different font-size.
- Use a filter based on length to return a shortened version with "…" at the end.



Custom filters syntax

```
(function(){
    var module = angular.module('messaging-app');
    /**
     * @ngdoc filter
     * @name messaging-app.filter:ellipsis
     * @param {string} text Text to be processed
     * @param {string|number} maxLength Maximum length of text
     * @returns {Function}
     * @description If the text is larger than maxLength,
     *              shortened to maxLength and apply ellipsis
     */
    module.filter("ellipsis", EllipsisFilter);
    EllipsisFilter.$inject = [];
    function EllipsisFilter()
    {
        return function (text, maxLength)
        {
            maxLength = Number(maxLength);
            if(typeof text !== 'string' || isNaN(maxLength)) return text;
            if(text.length < maxLength)
            {
                var tempText = text.substring(0, maxLength);
                return tempText+"...";
            }
        }
    }
})();
```

To be continued...