i18n and L10n

Overview

- Internationalization
  - implementing a software application so it can **potentially** be adapted to various languages **without changes**
    - paraphrased from Wikipedia
  - done once regardless of the number of languages to be supported
  - often shortened to **i18n** because there are 18 letters between the i and n
- Localization
  - adapting a software application for a specific language
    - paraphrased from Wikipedia
  - done once for each supported language
  - often shortened to **L10n** because there are 10 letters between the l and n

What AngularJS Provides

- Applies different formatting to currency amounts, dates, times and numbers based on locale
- Provided by the **`number`**, **`currency`** and **`date`** filters
- Steps to use
  - download a language-specific JavaScript file from extras **`i18n`** directory
    - have names like **`angular-locale_`***`langCode`*`.js`** and **`angular-locale_`***`langCode-countryCode`*`.js`**
  - add **`script`** tag for this JavaScript file to main HTML file
  - add dependency on **`ngLocale`** module to main module
- See http://docs.angularjs.org/guide/i18n

AngularJS Limitation

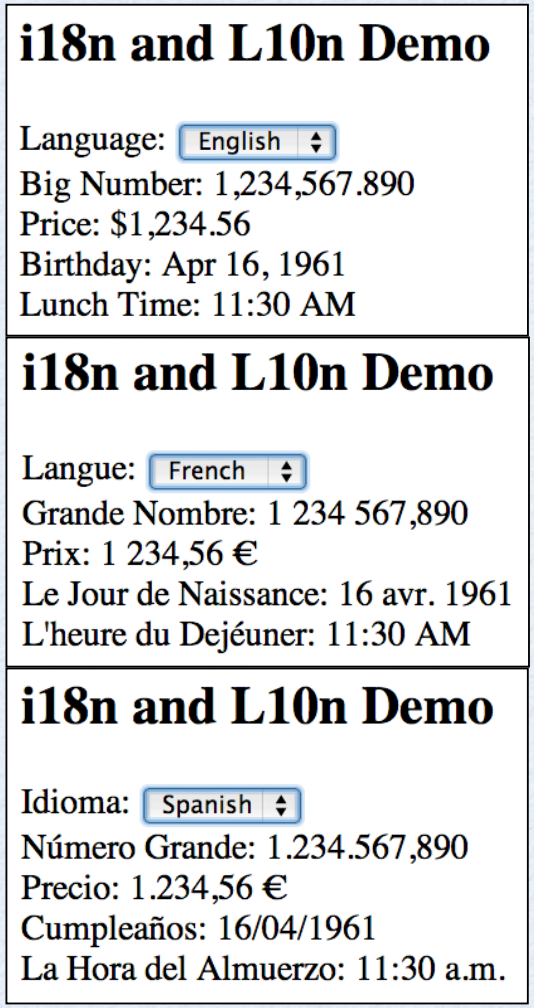
- Must select a single language and cannot change at runtime
  - need a separate **`index.html`** file for each locale
    that includes a **`script`** tag for the property locale **`.js`** file
- Workaround
  - https://github.com/lgalfaso/angular-dynamic-locale
- Steps to use angular-dynamic-locale
  - download language-specific JavaScript files from extras **`i18n`** directory
    for each language to be supported into the subdirectory **`angular/i18n`**
  - download **`tmhDynamicLocale.js`**
  - add **`script`** tag for **`tmhDynamicLocale.js`**
  - add dependency on **`'tmh.dynamicLocale'`** module
  - inject **`tmhDynamicLocale`** service where needed
  - for initial language and each time the language needs to be changed,
    call **`tmhDynamicLocale.set(`*`lang`*`)`**
    where ***`lang`*** is a language code like **`en`**, **`en-us`**, **`es`** or **`fr`**

Locale-based Text Translation

- Not supported in AngularJS,
  but it's easy to write a custom filter that does this
- Example that follows provides this filter
  and bundles use of angular-dynamic-locale

see `locale` directory at
https://github.com/mvolkmann/angularjs-examples

My Locale Service & Filter ...



L10n/en.json

```
{
}
```

L10n/fr.json

```
{
  "Big Number": "Grande Nombre",
  "Birthday": "Le Jour de Naissance",
  "Language": "Langue",
  "Lunch Time": "L'heure du Dej\\u00e9uner",
  "Price": "Prix"
}
```

L10n/es.json

```
{
  "Big Number": "N\\u00famero Grande",
  "Birthday": "Cumplea\\u00f1os",
  "Language": "Idioma",
  "Lunch Time": "La Hora del Almuerzo",
  "Price": "Precio"
}
```

... My Locale Service & Filter ...

index.html

```
<!DOCTYPE html>
<html ng-app="Demo">
  <head>
    <script src="../angular.min.js"></script>
    <script src="tmhDynamicLocale.js"></script>
    <script src="locale.js"></script>
    <script src="demo.js"></script>
  </head>
  <body>
    <h2>i18n and L10n Demo</h2>
    <div ng-controller="DemoCtrl">
      <div>
        <label>{{'Language' | L10n}}:</label>
        <select ng-model="lang">
          <option value="en">English</option>
          <option value="fr">French</option>
          <option value="es">Spanish</option>
        </select>
      </div>
      <div><label>{{'Big Number' | L10n}}:</label> {{bigNumber | number}}</div>
      <div><label>{{'Price' | L10n}}:</label> {{price | currency}}</div>
      <div><label>{{'Birthday' | L10n}}:</label> {{birthday | date}}</div>
      <div><label>{{'Lunch Time' | L10n}}:</label> {{lunchTime | date:'h:mm a'}}</div>
    </div>
  </body>
</html>
```

defines `localeSvc` service and `L10n` filter

if translation includes binding expressions, pass scope to the `L10n` filter with `| L10n:this`

... My Locale Service & Filter ...

demo.js

```
(function () {
  'use strict';
  var app = angular.module('Demo', ['Locale']);

  app.controller('DemoCtrl', function ($scope, localeSvc) {
    $scope.$watch('lang', function (lang) { // causes digest cycle
      localeSvc.setLang(lang);
    });

    $scope.lang = localeSvc.getDefaultLang();

    $scope.bigNumber = 1234567.8901234;
    $scope.price = 1234.56;
    $scope.birthday = new Date(1961, 3, 16);
    var lunchTime = new Date();
    lunchTime.setHours(11);
    lunchTime.setMinutes(30);
    $scope.lunchTime = lunchTime;
  });
})();
```

not worrying about minimizing code in this example

... My Locale Service & Filter ...

locale.js

```
(function () {
  'use strict';
  var module = angular.module('Locale', ['tmh.dynamicLocale']);
  var currentLang, translations = {};

  module.factory('localeSvc', function ($http, $interpolate, tmhDynamicLocale) {
    var svc = {};

    function loadTranslations(lang) {
      var url = 'L10n/' + lang + '.json';
      $http.get(url, {ContentType: 'application/json'}).
        success(function (data) {
          translations[lang] = data;
          sessionStorage.translations = JSON.stringify(translations);
        }).
        error(function (err) {
          throw new Error('Failed to load language translations for "' +
            lang + '".');
        });
    }

    svc.getDefaultLang = function () {
      var lang = navigator.language || navigator.userLanguage;
      return lang ? lang.split('-')[0] : 'en'; // default
    };
```

using **sessionStorage** to restore the language and translations when user refreshes browser

after HTTP response is processed, a digest cycle is triggered

better to retain country code and use a translation file with that in its name if it exists

... My Locale Service & Filter ...

locale.js

```
  svc.setLang = function (lang) {
    if (lang !== currentLang) {
      // Change i18n language.
      tmhDynamicLocale.set(lang); // causes digest cycle

      // Change L10n language.
      if (!translations[lang]) loadTranslations(lang); // causes digest cycle
      currentLang = sessionStorage.currentLang = lang;
    }
  };

  svc.translate = function (phrase, scope) {
    var t = translations[currentLang];
    var result = t ? t[phrase] : null;
    if (scope && result) result = $interpolate(result)(scope);
    return result || phrase;
  };

  return svc;
}); // end of localeSvc
```

using `$interpolate` to allow translations to contain binding expressions

… My Locale Service & Filter

locale.js

```
  module.filter('L10n', function (localeSvc) {
    return function (phrase, scope) {
      if (!currentLang) {
        // This occurs when the user refreshes a page.
        // Get the current language and translations
        // from sessionStorage.
        currentLang = sessionStorage.currentLang;
        translations = JSON.parse(sessionStorage.translations);
      }

      return localeSvc.translate(phrase, scope);
    };
  });
})();
```

`scope` is an optional directive parameter that is only needed for translations that contain binding expressions

Summary

- Relatively simple approach that has been shown to work well in a fairly large app
- Amazing that reevaluating every visible label in every digest cycle is not a performance issue!
- Other options
  - angular-translate - https://github.com/angular-translate/angular-translate
  - more?