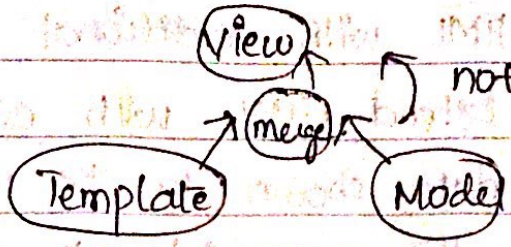
11/28/2016

Monday.

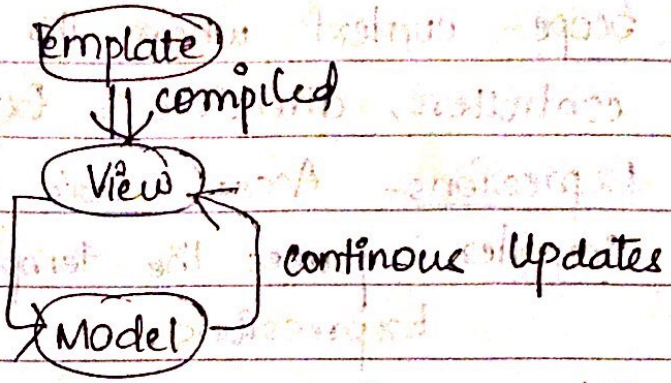
Angularjs Interview Questions & points

- Template- HTML with additional Markup
- Directives- Extend HTML with custom Att's & Elements
- Model- the data shown to the user in the view & with which the user interacts.
- Scope- context where the model is stored so that the controllers, directives & Expressions can access it.
- Expressions- Access var's & fun's from the scope.
- Compiler- parses the templates & instantiates directives & Expressions.
- Filter- formats the value of an Expr for display to the user.
- View- What the user sees (the DOM).
- Data Binding- Sync data b/w model & view.
- Controller- the business logic behind views.
- Dependency Injection- Creates & wires objects & functions
- Injector- dependency Injection container.
- Module- container for diff parts of an app including controllers, services, filters, directives which configures the Injector.
- Service- Reusable business logic independent of views.

ONE-WAY DATA BINDING :-



TWO-WAY :-



View is like projection of the Model.

→ Controller - JS constructor() used as Augment to Ang Scope.

ng-Controller

Use them to :- 1) setup the initial state of $scope
2) Add behavior to $scope.

```
var myApp = angular.module('myApp', []);
myApp.controller('GreetingController', ['$scope', function
                                        ($scope) {
  $scope.greeting = 'Hi!';
}]);
```

•) Scope Inheritance - ng-controller creates a new child scope, we get hierarchy of scopes that inherit from each other. It can have access to prop & methods defined by controllers higher up the hierarchy.

* Service - substitutable obj's that are wired up together using DI.

They are:-

1) lazily instantiated & 2) singletons

(Each component dependent on a service gets a reference to single instance generated by service factory).

→ Creating services:- register & provide service name, serv factory() within Ang module.

* DI - design pattern that deals how components get hold of dependencies. (run, config blocks for a module).

* Templates - written with HTML that contains Angular specific Elements & Attr's.

* Expressions - JS like code placed in interpolation binding.

* one time binding - ::

* Directives are markers on a DOM Element that tells AJS Html compiler (& compile) to Attach a specified behavior to that Dom Element.

11/29/2016

Angularjs -

-> Isolated scope - Directives are declared using isolated scope. When we declare an Element an isolate scope its relation to its parent scope changes. The child iso scope will no longer has access to data on its parent scope. (pass data isolated scope explicitly).

@ & = -> @ pass by value (Attribute binding)

-> & pass by method (Expression binding) & makes a call on something outside scope

-> = pass by reference. (2 way data Binding).

@ used to pass in literal value, such as string / no.
Data can be passed from parent -> child (not backup).
Value <-> Attribute

& pass a method into directive.
Method <-> Expression.

= - passing in a value by reference.
2 way communication b/w scopes.
combi of @ & & reference <-> 2 way data binding

- When using @ - use {{}} in directive declaration or pass in as a string directly.
- & - make sure to call the method with an Object inside the directive.
- By adding isolated scope Object to your AJS directive you can insure that "Each Element using that directive has its own scope" - isolated scope.
- Sharing data b/w controllers:-
  use $ stateprovider - configures the states.
  state().
  & create a controller specific to it
  (url, controller, templateUrl).
- & create a service .service(' ', function Greet() {
  ; });
- When the scope property is used the directive is in the so called "isolated scope", it cannot directly access the scope of the parent controller.
- ng-Grid:- Populating Grid structure using ng-Grid in AJS.
  1) Add ref to JQuery & AJS libs
  2) ref to ngGrid's JS & css files
  3) Where you declare your app module, add ngGrid:
     ang.module('myApp', ['ngGrid']);
  4) add

5) Define basic style for grid in .css.

6) In js file,

$scope.myData = [{name: "Mac", age: 50},
{}];

7) Initialize your Grid options (under same code)

$scope.gridOptions = {data: 'myData'};

-> AJS with nodejs, Expressjs framework

-> Create RESTful API using nodejs & Expressjs) & MongoDB(mongoose).

REST- representational state transfer- "Stateless", "Cacheable" "idempotent"

-> Http methods:- GET, PUT, DELETE, PATCH, POST.

-> Base URI

-> URL path

-> Media type

install backEnd components:

MEAN stack: MongoDB, Nodejs & Expressjs.

-> MongoDB with mongoose- npm package that allows you to interact with mongoDB.

mongoose CRUD.

Create, read, Update, Delete.

→ Mongoose - Read & Query data -

Model.find (conditions, [fields], [options], [callback])
- findById (id, [fields], [options], [callback])
- findOne (conditions, [fields], [options], [callback])

→ Update - (batch updates).

Model.update (con, upd, [options], [callback])
- findByIdAndUpdate (id, [up], [op], [call])
- findOneAndUpdate ([cond], [upd], [options], [call])

use findOneAndUpdate - just one upd & returns an object.

→ Delete

Model.remove (cond, [callback])
- findByIdAndRemove (id, [opt], [callback])
- findOneAndRemove (cond, [opt], [callback])

Expressjs :- It has HTML template eolink (jade, ejs, handlebars, hogan.js) css paramompilers (less, stylus, compass)

Middlewares - pluggable processors runs on each req made

use middlewares using app.use. to server.

app.verb - for specific requests

3 param's (request, response, next).

→ Default Express 4.0 middlewares

1) morgan : logger
2) body-parser :
3) cookie-parser :
4) serve-favicon

Express has a seperate package called (Express -generator), to start with API.

→ Connecting Expressjs to MongoDB.

1) Install mongodb.
2) " " driver for Nodejs
3)

→ 3 ways to retrieve, change, delete data from your future API.

1) Curl.
2) Browser & postman (chrome plugin)

→ Wiring up the Apps :-

1) Ajs read with $http.
makes XHR or jsonp req.
which returns a promise (success / Error function).
2) Ajs with $resource (to deal with restful requests)

Initialize:-

$resource(url, [paramDefault], [actions], options);
ngResource.

→ By default Express uses Jade (View Engine).

12/1/2016 * ReactJS *

- ReactJs will allow you to create reusable UI Components.
- It's a Js library.
- It can also render on Server using Node, it can power native app's using React Native.
- One-Way Data Binding

React Features :- 1) JSX - Js Syntax Extension.
2) Components.
3) → data flow & flux
4) License.

Adv :- 1) Uses Virtual DOM (↑ performance)
2) Both Cli & Ser Side
3) Comp & Data patterns improve readability ↑ Larger Apps.
4) Can be used with other frameworks.

Disadv :- 1) Covers only View layer
2) Use of Inline templating & JSX.

- JSX is faster (improves optimization), typesafe, Easier & faster to write templates with HTML.
- Js Expressions - { }
- Styling - CamelCase
- Comments {// end of the comment}
{/* ... */}
-

↗ pure functions
⇒ Stateless components - component which is created needs to be exported. [Eg:-] App ↳ Header → content. (App needs to be exported)

⇒ Stateful - table needs to be loaded dynamically.
Key = {i}, map() - will help react to update only necessary Elements instead of re-rendering Entire list when something change. ↑ performance boost.

⇒ State - place where data comes from.

⇒ props - immutable.

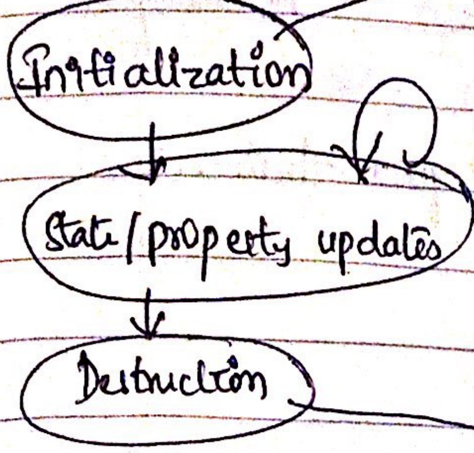
→ "state" can be updated & changed
child components should only pass data from state using props.

→ Set State in parent component & pass it down using props. (render()).

→ App.propTypes - used for props Validation.
" .defaultProps - after specifying validation patterns we set it.
isRequired - for validating PropArray & propBool.

→ React Component APIs :-
1) setState() - updates the state of component.
2) forceUpdate() - for updating the component manually use
3) ReactDOM.findDOMNode() - DOM manipulation

* Life Cycle Methods :-
1) ComponentWillMount - exe before rendering (CS)
2) ComponentDidMount - " after first render (c side) renders
3) componentWillReceiveProps - invoked props are upd bef
4) shouldComponentUpdate - T/F value (True default)
5) componentWillUpdate - just before rendering
6) componentDidUpdate - " after "
7) componentWillUnmount - called after comp is unmounted from DOM.



Lifecycle

* forms - value = { this. state. data } // update state when I/p changes.

onChange() - triggers state update which will be passed to child I/p value & rendered on the screen.

* Events - onClick Event - triggers Update State ()

-> Child Events - When you want to update state of parent comp from its child, you can create Event Handler in parent comp & pass it as a prop (UpdateStateprop) to child comp

-> ref - reference to your Elements.

-> key - useful when working with dynamically created components or when lists are altered by users.

React Key - values keep track of Each Element. unique index {i}.

-> React Router - Routing of your Appln in React.

Scanned by CamScanner

- Flux - pattern - uni-directional, Easy to maintain, app parts are easy to be decoupled.

Action → dispatcher → Store → View

- Redux framework — implementation of flux Architecture.

Actions → Reducers → components

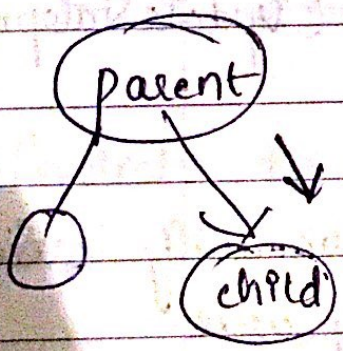
- HOC — Higher Order Components - wrapping normal Comp & provide addnal data I/p. (pure functions)

- SRP (single responsibility principles)

SOLID — principles

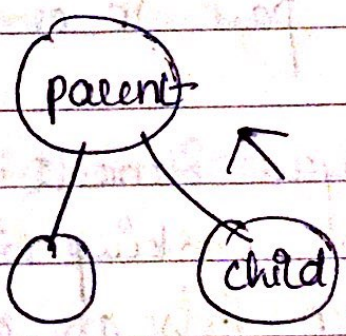
one component should only be responsible for one functionality.

Communication



parent → child

1) props.
2) Ref functions



child - parent

1) func callback
2) Event Bubbling

Sibling - Sibling

1) Parent component.

Any-Any 1) Observer pattern – design pattern – obj sends msg to multiple objects

2) Global vals – use window.

3) Context – similar to props – passed with ( ); to pass data ↑ subtree.