附录A 外文资料的调研阅读报告

What is Angular.js?

Angular which is commonly referred to as “Angular” or “Angular.js” is an open-source web application framework. It is mainly maintained by Google and by a community of individuals and corporations to 的deal with many of the challenges encountered in developing single-page application. Aiming at simplifying the development and testing of single-page application, it has many unique features like providing a framework for client-side MVC (model-view-controller) and MVVM(model-view-view-model) architectures. As you can see, I think angular is suitable for developing singe-page application and it is a clear framework for MVC which is an efficient software pattern architecture. Besides, Comparing with the environment when code is written in JavaScript+html+CSS, angular.is can be interpreted as another kind of JavaScript because of the functional similarities.

How does angular work?

The angular.js framework works based on the HTML page and embed additional custom tag attributes to it. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. It is flexible that these JavaScript variables are able to be set within the code or retrieved from static or dynamic JSON resources. So actually the framework adapts and extends traditional HTML to present dynamic content through two-way data-binding that allows for the automatic synchronization of models and views. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. Angular teaches the browser new syntax through a construct we call directives. The normal examples include:

1. data binding in {{}}(ng-model={{xx}})
2. DOM control structures for repeating(ng-repeat)
3. showing and hiding DOM fragments(ng-show/ng-hide/ng-if)
4. support for forms and form validation (implement by directives)

With all the efforts mentioned, angular which is the frontend part of the singe-page application de-emphasizes explicit DOM manipulation. You can see all the details on the official website of angular online.

What are the benefits of angular?

First, it decouple DOM manipulation from application logic. Getting grid of DOM manipulation, it is more convenient for the developers to control elements through angular way and avoid complex function logic. Second, it decouple the client side of an application from the server side. As a result, developers who work for the server side and client side will be able to separated and progress in parallel which means that both sides will be reused in the future work. Third, it provides structure for the journey of building an application: from designing the UI, through writing the business logic, to testing. Besides, angular implements the MVC pattern to separate presentation, data and logic components. Using dependency injection, Angular brings traditionally server-side services to client-side web applications. Consequently, much of the burden on the server can be reduced.

A brief Conceptual overview of angular.js

This section is going to introduce some important parts of angular.

Templates

In Angular, templates are written with HTML that contains Angular-specific elements and attributes. Angular combines the template with information from the model and controller to render the dynamic view that a user sees in the browser.

These are the types of Angular elements and attributes you can use: Directive, Markup, Filter Form controls.

Directive is an attribute or element that augments an existing DOM element or represents a reusable DOM component.

Markup is the double curly brace notation {{ }} to bind expressions to elements is built-in Angular markup.

Filter is used for filtering certain formats data for display.

Form controls is used for validating user input.

Controllers

In Angular, a Controller is defined by a JavaScript constructor function that is used to augment the Angular Scope. When a Controller is attached to the DOM via the “ng-controller” directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function. There is also a child-parent relationship between the controllers which means that child scope which belongs to the child controller coexists with the parent scope. A new child scope will be created and made available as an injectable parameter to the Controller's constructor function as $scope. If the controller has been attached to the app using the controller as syntax then the controller instance will be assigned to a property on the new scope. You can use controllers to set up the initial state of the $scope object and add behavior to the $scope object. “$scope” has many elements which are tightly related to the ones on the HTML. In other words, if you want to initiate and update the data on the UI, just consider to use “$scope”. Not only can you get input values from users but also change data from backend like JavaScript Servlet.

Services

Angular services are substitutable objects that are wired together using dependency injection (DI). So it is lazily instantiated which means that Angular only instantiates a service when an application component depends on it. If you want to use services to organize and share code across your app, you can try services as public functions which can be shared by app. Perhaps services can be regarded as public functions in C++ which can be shared and reused in different parts of a program. The specific example of services is http request which is provided by angular in term “$http”.

Directives

At a high level, directives are markers on a DOM element that tell AngularJS's HTML compiler to attach a specified behavior to that DOM element, or even to transform the DOM element and its children. Angular comes with a set of native directives, like “ng-Bind”, “ng-Model”, and “ng-Class”. Much like you create controllers and services, you can create your own directives for Angular to use. You can learn how to create an efficient directives in angular doc on official website

There are so many important components related to Angular that I don’t want to introduce all of it. You can find more details on official website（<https://angularjs.org/>）

MVC pattern

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces in software engineering. It divides a given software application into three interconnected parts which are model, view and controller, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

Descriptions of the tree parts of MVC

Model directly manages the data, logic and rules of the application. It stores data that is retrieved according to commands from the controller and displayed in the view.

A view can be any output representation of information, such as a chart or a diagram. A view generates new output to the user based on changes in the model. Multiple views of the same information are also available by using different combinations of basic views.

The controller accepts input and converts it to commands for the model or view. It can not only send commands to the model to update the model's state but also send commands to its associated view to change the view's presentation of the model.

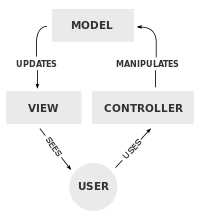
Benefits of MVC？

The separation of three components allows the re-use of the business logic across applications. Multiple User Interfaces can be developed without concerning the code base. For example, by using the controller, we can combine different views together which contributes to different UI for the users. In this step, we don’t need to care about the business logic under each view like how they get data from models.

Similarly, the developers of UI can focus exclusively on the UI screens without caring about business logic. The developer of Model can focus exclusively on the business logic implementations, modifications, updates without concerning the appearance.

Business logic developers can build the classes, while the UI developers can involve in designing UI screens simultaneously, resulting the interdependency issues and time conservation. UI updates can be made without slowing down the business logic process. Business logic rules changes are very less that needs the revision / updates of the UI. So business logic and UI designing are efficiently separated.

Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time. Given the fact that user interfaces tend to change more frequently than business rules and the model does not depend on the views, adding new types of views to the system generally does not affect the model, the scope of change is confined to the view.



REST: Representational State Transfer

Representational state transfer is the software architectural style of the World Wide Web.

The purpose of is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability. What is precise is that REST is an architectural style consisting of a coordinated set of components, connectors, and data elements within a distributed hypermedia system. In this system, instead of impeding by implementation details, developers only focus on component roles and a specific set of interactions between data elements.

When systems conform to the constraints of rest, we call it restful. Restful systems typically communicate over HTTP with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers. REST systems interface with external systems as web resources identified by Uniform Resource Identifiers (URIs) which is introduced in the following.

REST adds three new architectural constraints to B/S pattern which are layered system, code on demand (optional), Uniform interface.

Layered system

In the past, a client could not tell whether it was connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability. Because methods can be used to enable load balancing and provide shared caches and enforce security policies.

Code on demand

Servers can temporarily extend or customize the functionality of a client by the transfer of executable code.

Uniform interface

The uniform interface constraint is fundamental to the design of any REST service. The uniform interface simplifies and decouples the architecture, which enables each part to evolve independently. The four constraints for this uniform interface are Identification of resources, manipulation of resources through these representations, self-descriptive messages and hypermedia as the engine of application state.

Web service APIs

Web service APIs that adhere to the REST architectural constraints are called restful APIs. HTTP-based Restful APIs are defined with the following aspects:

1) base URI

2) an Internet media type for the data. This is often JSON but can be any other valid Internet media type

3) standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE)

4) hypertext links to reference state

5) hypertext links to reference-related resources

参考文献