

AngularJS

한국소프트웨어산업협회
신용권



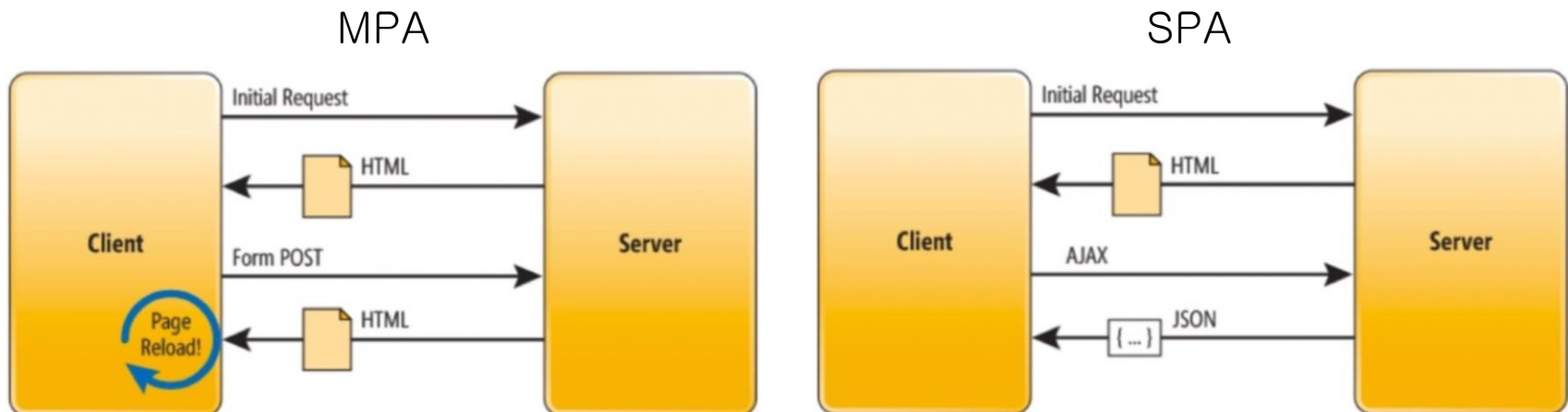
❖ 멀티 페이지와 싱글 페이지 애플리케이션

➤ MPA (Multi Page Application)

- UI 변경시 새로운 HTML으로 DOM 생성 (멀티 페이지)
- 서버 사이드에서 HTML을 렌더링하기 때문에 브라우저는 최소한의 기능만 요구
- 많은 종류의 브라우저 지원만 응답 시간이 지연될 수 있음
- 접속하는 브라우저가 많을 경우 대규모 서버사이드 인프라 스트럭처 필요

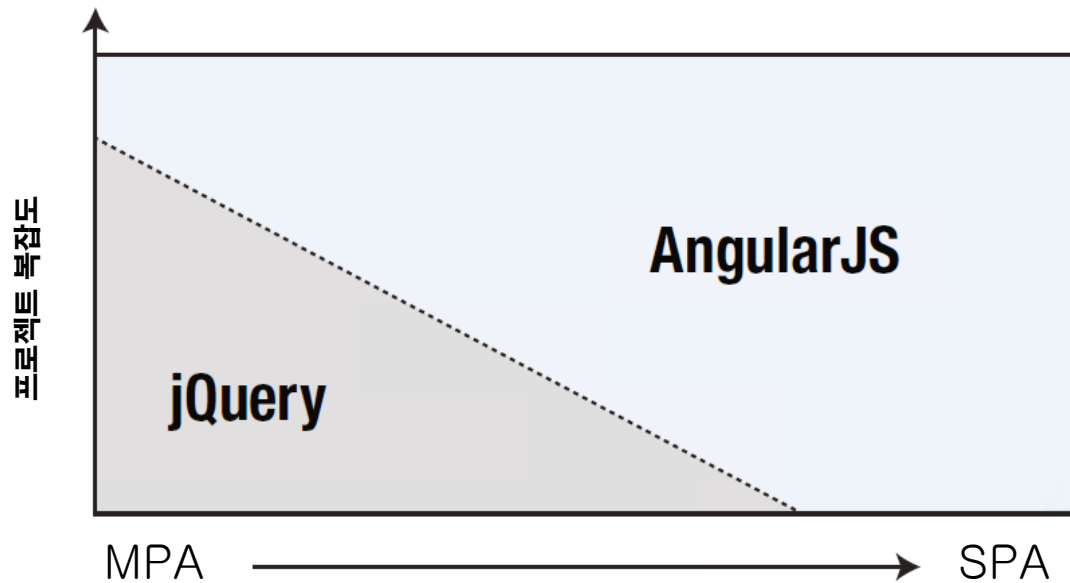
➤ SPA (Single Page Application)

- 최초 HTML을 받고 DOM 생성 (싱글 페이지)
- 이후 UI 변경은 AJAX로 DOM의 일부만 추가, 수정, 삭제
- 프레임워크를 이용해서 일관된 코드 작성이 필요(AngularJS, React, ...)
- 자바스크립트 코드량이 증가, 초기 로딩 속도가 늦음



➤ 모던 웹 애플리케이션

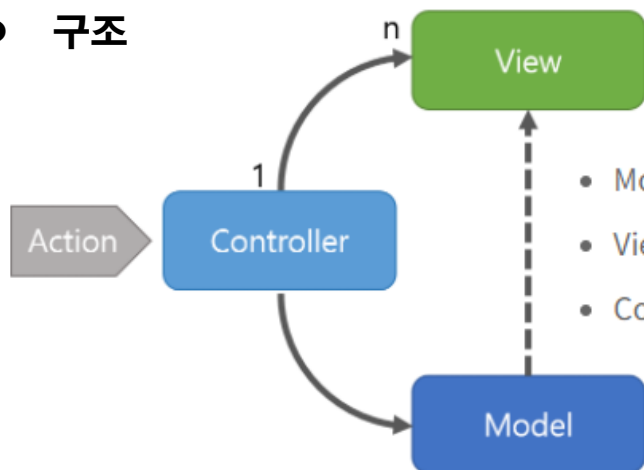
- 기본적으로 MPA 모델을 사용
- 사용자 액션이 많은 페이지는 SPA 모델을 사용



❖ 다양한 프레임워크 아키텍처

➤ MVC(Model-View-Controller)

● 구조



- Model : 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분입니다.
- View : 사용자에서 보여지는 UI 부분입니다.
- Controller : 사용자의 입력(Action)을 받고 처리하는 부분입니다.

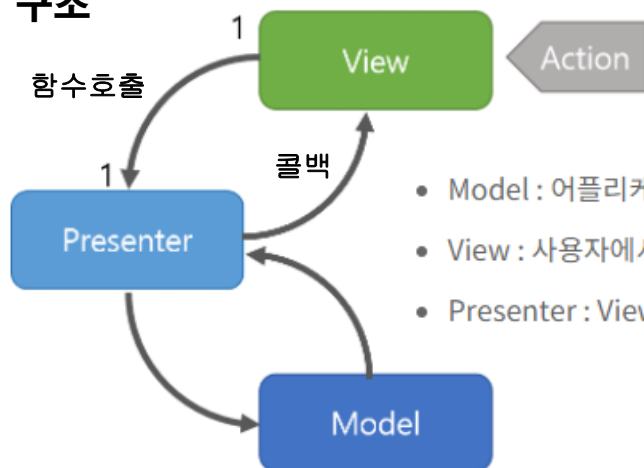
MVC는 Model + View + Controller를 말합니다.

● 동작

1. 사용자의 Action들은 Controller에 들어오게 됩니다.
2. Controller는 사용자의 Action를 확인하고, Model을 업데이트합니다.
3. Controller는 Model을 나타내줄 View를 선택합니다.
4. View는 Model을 이용하여 화면을 나타냅니다.

➤ MVP(Model-View-Presenter)

● 구조



- Model : 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분입니다.
- View : 사용자에서 보여지는 UI 부분입니다.
- Presenter : View에서 요청한 정보로 Model을 가공하여 View에 전달해 주는 부분입니다.

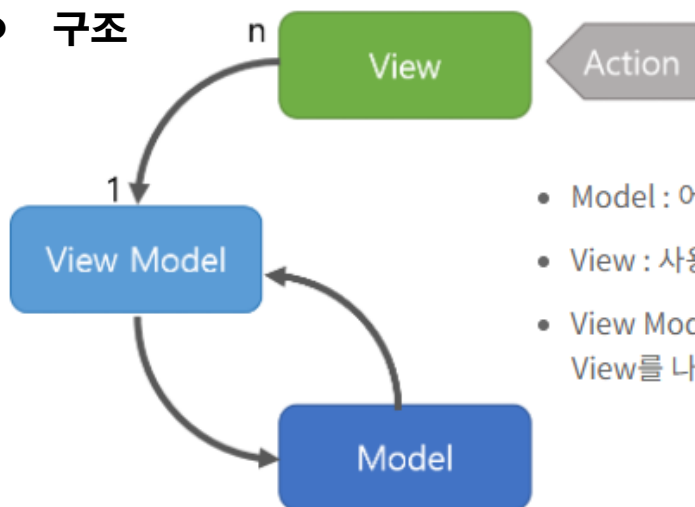
MVP는 Model + View + Presenter를 말합니다.

● 동작

1. 사용자의 Action들은 View를 통해 들어오게 됩니다.
2. View는 데이터를 Presenter에 요청합니다.
3. Presenter는 Model에게 데이터를 요청합니다.
4. Model은 Presenter에서 요청받은 데이터를 응답합니다.
5. Presenter는 View에게 데이터를 응답합니다.
6. View는 Presenter가 응답한 데이터를 이용하여 화면을 나타냅니다.

➤ MVVM(Model-View-View Model)

● 구조



- Model : 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분입니다.
- View : 사용자에서 보여지는 UI 부분입니다.
- View Model : View를 표현하기 위해 만든 View를 위한 Model입니다.
View를 나타내 주기 위한 Model이자 View를 나타내기 위한 데이터 처리를 하는 부분

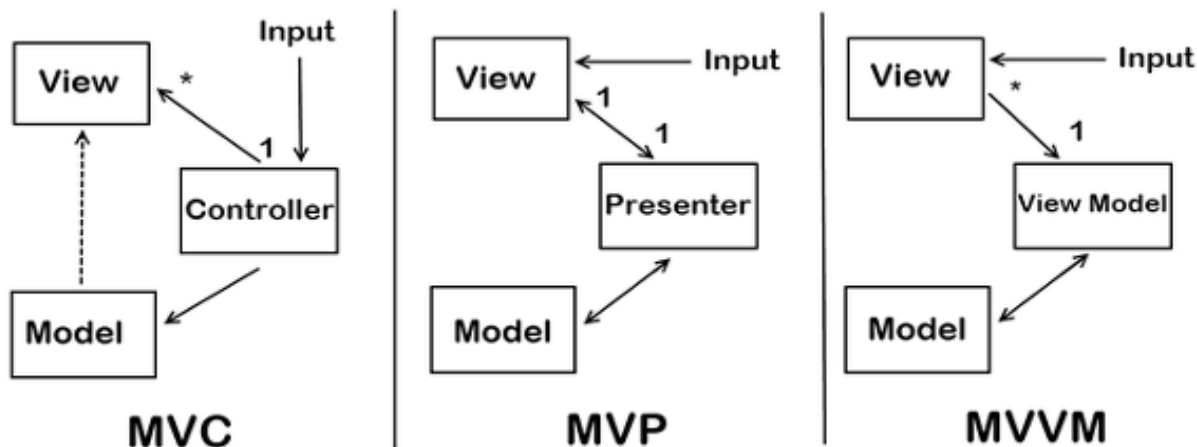
MVVM는 Model + View + View Model를 말합니다.

● 동작

1. 사용자의 Action들은 View를 통해 들어오게 됩니다.
2. View에 Action이 들어오면, Command 패턴으로 View Model에 Action을 전달합니다.
3. View Model은 Model에게 데이터를 요청합니다.
4. Model은 View Model에게 요청받은 데이터를 응답합니다.
5. View Model은 응답 받은 데이터를 가공하여 저장합니다.
6. View는 View Model과 Data Binding하여 화면을 나타냅니다.

➤ MVW(Model-View-Whatever)

- MVC 또는 MVVM 어떤 아키텍처라도 구현 가능



- 가장 효율적인 패턴을 적용할 수 있도록 유연함
- 대표적 프레임워크: AngularJS

❖ AngularJS 라이브러리 설치

➤ 기본 라이브러리

```
<html>
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>HTML Page</title>

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>

<body>
</body>
</html>
```


➤ 추가 라이브러리

- `<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/파일명" >`
`</script>`
- <https://angularjs.org>

Download AngularJS

Branch

1.8.x (latest)

1.2.x (legacy)

Build

Minified

Uncompressed

Zip

CDN

https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js

Bower

bower install angular#1.8.2

npm

npm install angular@1.8.2

Extras

Browse additional modules

Previous Versions

Download

❖ URL 라우트

➤ URL 라우트이란

- \$location의 url() 또는 path() 메소드로 설정한 경로의 뷰 파일을 렌더링
- ngRoute 모듈의 \$route 서비스 이용

➤ ngRoute 모듈 로딩

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/  
angular-route.min.js"></script>
```

➤ 뷰 렌더링 위치 지정: <ng-view/>

- URL 라우팅된 뷰 파일의 렌더링 위치 지정
- 현재 라우팅된 뷰의 DOM만 메모리에 존재
- ngRoute 모듈의 \$routeProvider로 라우트 구성(URL과 뷰 매핑)
- \$location.url()을 이용해서 ng-view 디렉티브에 보여줄 뷰 선택

➤ HTML5 모드 활성화

- 라우트시 이동 경로에서 #! 제거시킴
`$location.url("/user/loginForm");`
=> <http://127.0.0.1:5500/service/exam05/#!/user/loginForm>
=> <http://127.0.0.1:5500/service/exam05/user/loginForm>

- HTML5 히스토리 API를 이용하도록 변경

```
angular.module("app", [])  
.config(function($locationProvider) {  
  //if(window.history && history.pushState) {  
    $locationProvider.html5Mode({  
      enabled: true,  
      requireBase: true  
    });  
  }  
})
```

- 기준 경로 설정

```
<!-- <base href="/front-end/" /> -->  
<base href="/" />
```

- 서버 페이지 요청 및 이동하는 원래 기능이 제거되고
SPA 내부에서 Route 경로로 이동함

➤ URL 라우트 정의

```
angular.module("app", ["ngRoute"])
.config(function ($routeProvider) {
  //라우트 설정
  $routeProvider
    .when("/", { templateUrl: "views/home.html" })

    .when("/exam01_library_loading", {
      templateUrl: "views/exam01_library_loading.html"
    })

    .when("/exam24_builtin_service/boards", {
      templateUrl: "views/exam24_builtin_service/boards.html",
      controller: "exam24Controller"
    })

    .otherwise({ redirectTo: "/" });
});
```

➤ URL 라우트 파라미터

- 경로 및 쿼리문자열의 파라미터 정의

```
$routeProvider.when( "/boards/:bno" , {templateUrl: " ..." });
```

```
<a href= "boards/3" >
```

```
$location.url( "/boards/3" );
```

```
$routeProvider.when( "/boards" , {templateUrl: " ..." });
```

```
<a href= "boards?pageNo=1" >
```

```
$location.url( "/boards?pageNo=1" );
```

- 컨트롤러에서 \$routeParams 서비스를 주입받고 파라미터값 얻기

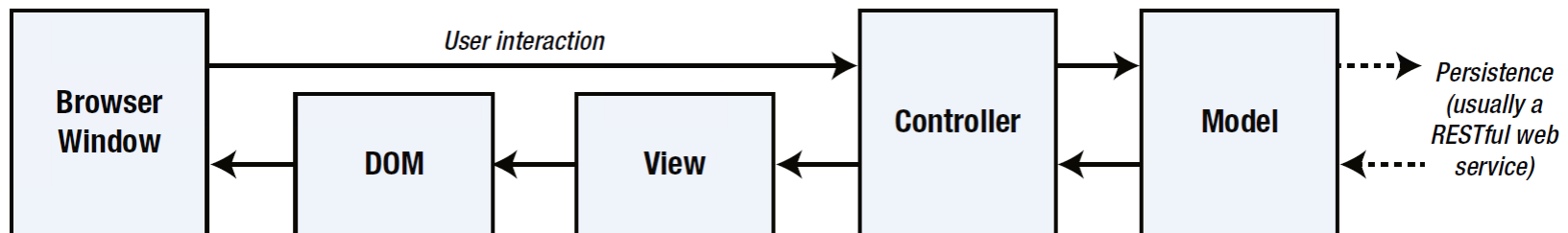
```
angular.module("app")
.controller("exam24Controller", function($routeParams) {
  $scope.$on("$routeChangeSuccess", () => {
    console.log($routeParams);
    console.log($routeParams.bno);
    console.log($routeParams.pageNo);
  });
});
```

❖ AngularJS MVC 패턴

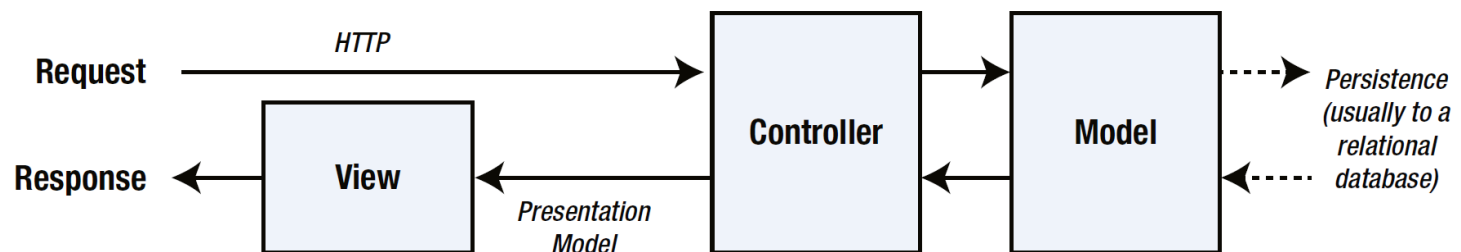
➤ 관심사(Model, View, Controller)를 분리해서 코드를 작성

- Model: 데이터 및 데이터 CRUD에 필요한 로직(RESTful 서비스 호출 코드) 담당
- View: 사용자 인터페이스(UI) 생성을 위한 마크업 생성 담당
- Controller: Model과 View를 연결짓고, 사용 흐름 제어 담당

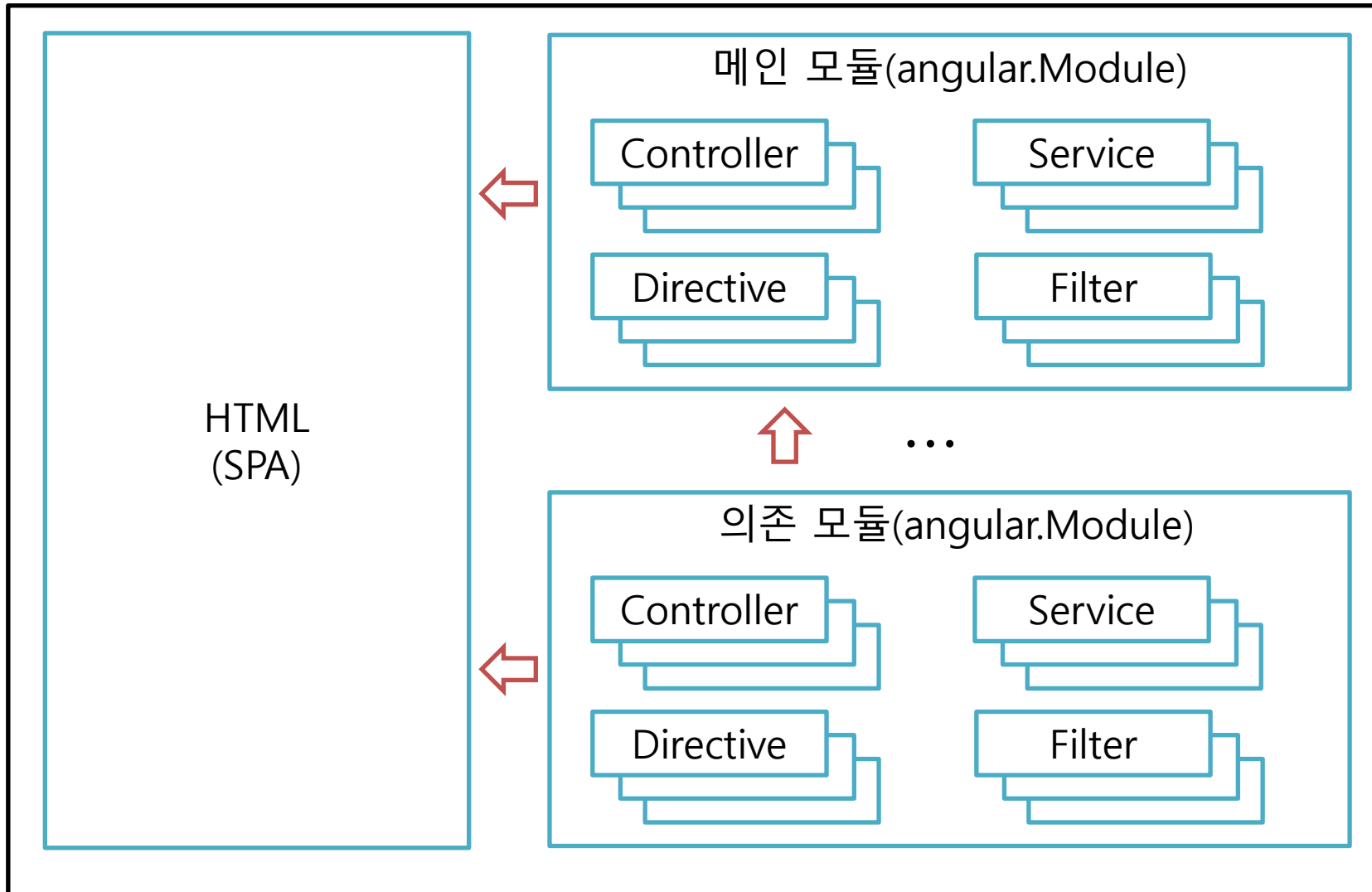
➤ AngularJS의 MVC 패턴



➤ [참고] 서버 사이드 MVC 패턴



❖ AngularJS 애플리케이션을 구성하는 컴포넌트



➤ Module

- AngularJS 애플리케이션에서 최상위 자바스크립트 컴포넌트
- HTML 문서와 ng-app 디렉티브로 연결해서 최초 구동

➤ Controller

- 모델과 뷰를 연결
- 모델의 데이터를 뷰에 제공
- 뷰의 이벤트 처리

➤ Service

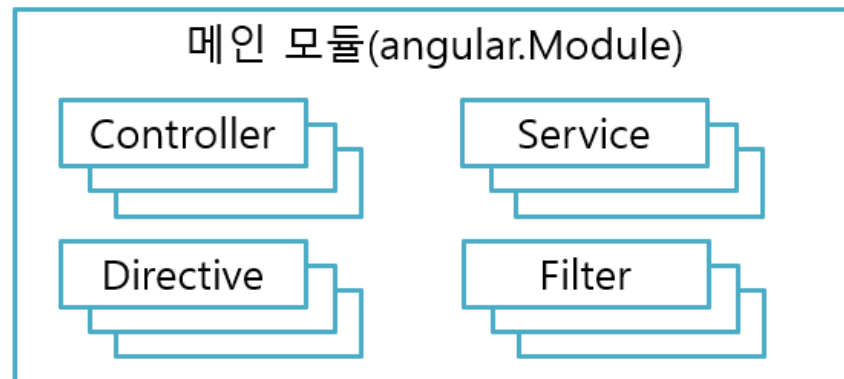
- 재사용 가능한 비즈니스 로직 제공
- 백-엔드 RESTful 서비스와 연동

➤ Directive

- 태그 및 속성으로 기술하고 뷰 생성과 관련 기능 제공

➤ Filter

- 데이터를 필터링하거나, 변형 기능 제공



❖ 모듈 생성 및 HTML과 연결

➤ 모듈 생성

- `var app = angular.module("app" , ["의존모듈이름 ", "의존모듈이름 ", ...]);`
- 의존 모듈이 없어도 []는 생략할 수 없음

➤ 모듈 찾기

- `var app = angular.module("app");`
- []는 기술하면 안됨

➤ HTML 문서와 연결

- `<html ng-app= "app" >`

```
//모듈 생성
var myModule1 = angular.module("myModule", []);

//모듈 찾기
var myModule2 = angular.module("myModule");

if(myModule1 === myModule2) {
    console.log("같은 모듈");
} else {
    console.log("다른 모듈");
}
```

❖ 복수 모듈 정의 및 로딩

➤ 모듈 의존성

- AngularJS 모듈은 다른 모듈에 정의된 컴포넌트에 의존할 수 있다.
- 복잡한 애플리케이션일 경우 모듈을 분리해서 개발 고려

➤ 외부 모듈 로딩 및 현재 모듈의 의존성 설정

- `<script src= "otherModule.js" ></script>`
- `<script>angular.module("app" , ["otherModuleName"]);</script>`

- EX: exam03Module.js

```
angular.module("exam03Module", [])  
  .controller("exam03ModuleController", function($scope) {  
    $scope.controllerName = "exam03Module.exam03ModuleController";  
  });
```

//index.html 수정

```
//<script src="modules/exam03Module.js"></script>
```

//app.js 수정

```
//angular.module("app", ["ngRoute", "exam03Module"])
```

❖ 모듈의 콜백 메소드

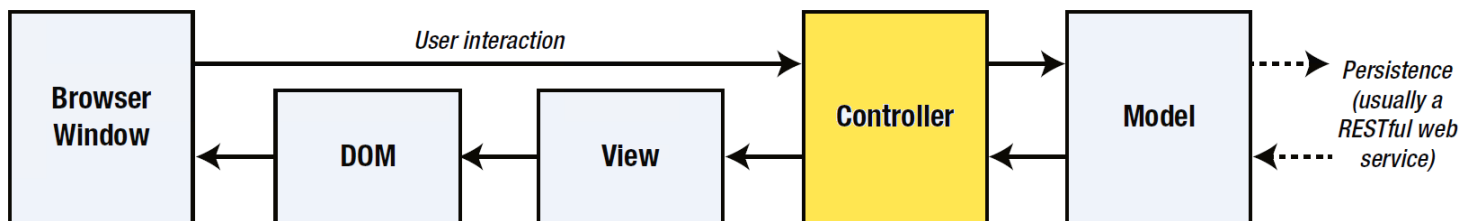
- **config(function(xxxProvider, ...) { ... }) 메소드**
 - 매개변수로 서비스의 Provider를 주입받고 서비스를 초기화할 목적
 - 로그 레벨, HTML5 활성화, 라우팅 설정
 - 의존 모듈의 config()가 먼저 콜백되고, 현재 모듈의 config()가 콜백됨
- **run(function(\$rootScope, xxxService, ...) {...}) 메소드**
 - AngularJS 애플리케이션의 전역 범위 데이터 및 기능 정의
 - 매개변수를 이용해서 다양한 서비스를 주입 받을 수 있음
 - 의존 모듈 및 현재 모듈의 config()가 각각 콜백된 이후,
 - 의존 모듈의 run()이 먼저 콜백되고 현재 모듈의 run()이 콜백됨

```
angular.module("app", [])  
  .config(function ($routeProvider) {  
    | console.log("app - config callback");  
  | })  
  
  .run(function ($rootScope) {  
    | console.log("app - run callback");  
  | })  
  ;
```

❖ 컨트롤러 선언 및 뷰 적용

➤ 컨트롤러 역할:

- 모델(서비스)와 뷰를 연결해주는 역할
- 뷰(브라우저)에서 발생하는 이벤트(사용자의 상호작용)를 처리



➤ 컨트롤러 정의

- `var module = module.controller("xxxController" , function($scope) {...});`
- 두번째 함수는 컨트롤러를 생성하기 위한 생성자임
- `$scope`에 정의된 속성 및 메소드만 표현식과 바인딩에 사용할 수 있음

➤ 컨트롤러의 뷰 적용

- `<태그 ng-controller= "xxxController" >` 컨트롤러의 뷰 적용 범위 `</태그>`
- `ng-controller` 디렉티브는 컨트롤러 생성자로부터 객체를 생성
- `ng-controller` 디렉티브 수만큼 컨트롤러 객체 생성(싱글톤이 아님)
 - EX: `<body ng-controller="xxxController"> ... </body>`
 - EX: `<div ng-controller="xxxController"> ... </div>`

- 컨트롤러는 `$scope` 객체의 속성 및 메소드를 통해 뷰와 상호작용
 - 컨트롤러는 `$scope` 객체의 속성 및 메소드를 통해 뷰와 상호작용
 - 바인딩 표현식 `{{ ... }}` 또는 디렉티브(`ng-xxx`)에서 사용됨

```
angular.module("app")
  .controller("exam06Controller", function($scope) {
    $scope.cities = ["서울", "뉴욕", "런던"];
    $scope.city = "서울";
    $scope.getCountry = () => {
      switch ($scope.city) {
        case "서울":
          return "한국";
        case "뉴욕":
          return "미국";
        case "런던":
          return "영국";
      }
    };
  });
```

```
<div class="form-group">
  <label>도시 선택:</label>
  <select class="form-control"
    ng-options="city for city in cities" ng-model="city" style="width:200px"></select>
</div>
<div>
  <p>선택한 도시: <span class="text-primary">{{city}}</span></p>
  <p>도시의 나라: <span class="text-primary">{{getCountry() || "Unknown"}}</span></p>
</div>
```

컨트롤러 및 스코프

➤ 암시적 모델 속성

- ng-model의 값이 \$scope의 존재하는 속성이 아닌 경우
- 암시적으로 \$scope의 속성으로 자동 생성됨
- 하지만 최초 값이 변경되지 않으면 속성이 자동 생성되지 않음
- 따라서 검사 코드 필요

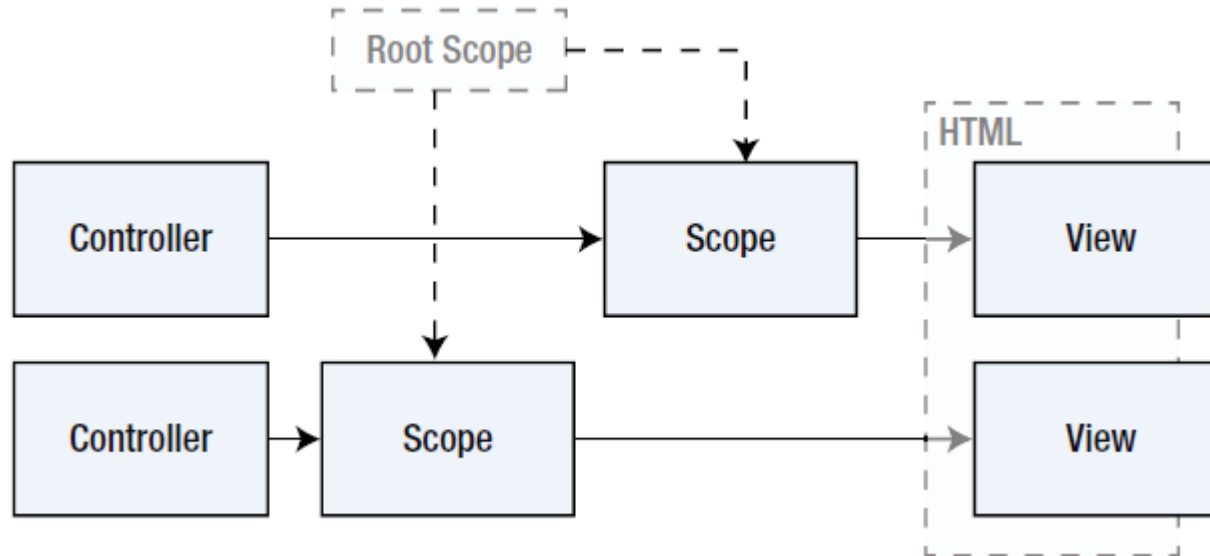
```
<div class="form-group">
  <label for="actionText">Action:</label>
  <input type="text" class="form-control" ng-model="newTodo.action" />
</div>
<div class="form-group">
  <label for="actionLocation">Location:</label>
  <select class="form-control" ng-model="newTodo.location">
    <option>Home</option>
    <option>Office</option>
    <option>Mall</option>
  </select>
</div>
<button class="btn btn-primary btn-block" ng-click="addNewItem()">Add</button>
```

```
$scope.todos.push({
  action: $scope.newTodo.action + " (" + $scope.newTodo.location + ")",
  complete: false,
});
```

❖ \$rootScope

➤ \$rootScope

- \$rootScope 서비스 객체를 시작으로 \$scope는 계층구조를 갖는다.
- 각 컨트롤러에는 \$rootScope의 자식인 새 \$scope 객체가 부여된다.



- \$rootScope 서비스 객체는 컨트롤러 생성자 함수의 매개값으로 주입될 수 있다.

```
var app = app.controller( "mainController" , ($rootScope, $scope) => {  
    ...  
});
```
- \$rootScope의 속성과 메소드는 뷰에서 사용할 수 있다.

```
.run(function ($rootScope) {  
    $rootScope.uid = "winter";  
    $rootScope.authToken = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiJ1c2VyMSIsIm";  
})
```

```
<!-- app.js에서 run() 에 데이터 저장 -->  
<p>uid: <spsn class="text-primary">{{uid}}</spsn></p>  
<p>authToken: <spsn class="text-primary">{{authToken}}</spsn></p>
```


➤ 이벤트 중계를 통한 스코프 통신

- \$rootScope 및 \$scope 객체는 이벤트로 통신을 할 수 있다.
- \$broadcast(): 위에서 아래 방향으로 이벤트 전달 (\$rootScope -> \$scope)
- \$emit(): 아래에서 위 방향으로 이벤트 전달 (\$scope -> \$rootScope)
- \$on(): 이벤트 수신

```
angular.module("app")
  .controller("exam09Controller", function($rootScope, $scope) {
    $scope.broadcast = () => {
      //전달할 내용
      const message = {
        from: "홍길동",
        data: "AngularJS 방송은 모든 수신자에게 전달됩니다."
      };

      //방송하기
      $rootScope.$broadcast("messageArrived", message);
    };

    //수신자 등록
    $scope.$on("messageArrived", (event, message) => {
      $scope.message = message;
    });
  });
```

❖ 스코프 속성 감시

➤ 스코프의 속성 변화에 반응하는 핸들러 등록

● `$watch(expression, handler)`

적용 범위의 표현식을 통해 참조한 값이 바뀔 때 통보 받을 핸들러를 등록

```
$scope.btnDisabled = true;

$scope.$watch("password1", function(password1) {
  if(password1 !== undefined && password1 !== "" && password1 === $scope.password2) {
    $scope.btnDisabled = false;
  } else {
    $scope.btnDisabled = true;
  }
});

$scope.$watch("password2", function(password2) {
  if(password2 !== undefined && password2 !== "" && $scope.password1 === password2) {
    $scope.btnDisabled = false;
  } else {
    $scope.btnDisabled = true;
  }
});
```

```
<form>
  <input type="text" ng-model="password1"/>
  <input type="text" ng-model="password2"/>
  <button id="btn" class="{{btnDisabled?'disabled':''}}">버튼</button>
</form>
```

- **\$watchCollection(object, handler)**

적용 범위의 객체 내 임의의 속성이 바뀔 때 통보 받을 핸들러 등록

```
$scope.btnDisabled = true;
$scope.user = {};
$scope.$watchCollection("user", function(user) {
    if(angular.isDefined(user.password1) &&
        user.password1 !== "" &&
        user.password1 === user.password2) {
        $scope.btnDisabled = false;
    } else {
        $scope.btnDisabled = true;
    }
});
```

```
<form>
  <input type="text" ng-model="user.password1"/>
  <input type="text" ng-model="user.password2"/>
  <button id="btn" class="{{btnDisabled?'disabled':''}}">버튼</button>
</form>
```

❖ 컨트롤러 외부에서 스코프 변화 일으키기

➤ ng-controller가 적용된 엘리먼트 찾기

- `angular.element(angularRegion)`;
 <!-- id는 전역변수로 자동 생성되므로 매개값으로 사용가 -->
 <body id= "**angularRegion**" ng-controller= "**mainController**" >
- jQuery가 로딩되어 있다면 `$(#angularRegion)`와 동일
- jQuery가 로딩되어 있지 않다면 내장된 `jqLite(jQuery Lite)` 구현체 사용

➤ `$scope`에 접근해서 스코프 함수 호출하기

- `angular.element(angularRegion).scope().$apply("스코프의 fn()");`
- 컨트롤러 비적용 범위의 변경 사항을 컨트롤러 적용 범위로 주입

```
<script>
function handleBtn() {
    var content = $("#txtSend").val();
    $(angularRegion).scope().$apply(`setContent("${content}")`);
}
</script>
```

❖ 디렉티브는 언제, 왜 사용하나

➤ 언제 왜 사용하나?

- 데이터 바인딩
- 템플릿 생성
- 폼 유효성 검증
- 이벤트 처리
- HTML 조작 기능

➤ 디렉티브 형태

- <태그>에 애플리케이션 기능을 추가해 준다.
- 태그명 또는 어트리뷰트명으로 사용된다.
- AngularJS의 내장 디렉티브는 ng-* 이름을 가진다.

EX:

```
<html ng-app= “...” >  
<div ng-controller= “...” >  
<div ng-repeat= “...” >  
<ng-include= “...” >
```

❖ 바인딩 관련 디렉티브

➤ 단방향 바인딩

- 표현식: {{ }},
- 디렉티브: ng-bind

➤ 처리되지 않은 템플릿 바인딩 표현식 숨기기

- {{...}} 대신 ng-bind 이용
- ng-cloak 이용

```
<div ng-cloak>
  <div>There are <span class="text-danger">{{todos.length}}</span> items</div>
  <div>There are <span class="text-danger" ng-bind="todos.length"></span> items</div>
  <div>
    First Item: <span class="text-danger">{{todos[0].action}}</span> <br/>
    Second Item: <span class="text-danger" ng-bind="todos[1].action"></span>
  </div>
</div>
```

❖ 바인딩 관련 디렉티브

➤ 양방향 바인딩

- 디렉티브: ng-model

```
<input name="firstItem" class="form-control" ng-model="todos[0].action" />
<div class="alert alert-primary mt-2">The first item is: {{todos[0].action}}</div>
<hr/>
<input type="radio" ng-model="job" value="개발자"/>개발자
<input type="radio" ng-model="job" value="디자이너"/>디자이너
<div class="alert alert-primary mt-2">The job is: {{job}}</div>
```

```
<tr ng-repeat="item in todos">
  <td>{{$index + 1}}</td>
  <td>{{item.action}}</td>
  <td>{{item.complete}}</td>
  <td><input type="checkbox" ng-model="item.complete" /></td>
</tr>
```

❖ 엘리먼트 관련 디렉티브

➤ ng-repeat : 반복

- 배열 요소 반복 및 객체 속성 반복
- ng-repeat 내장 변수

Variable	Description
\$index	Returns the position of the current object or property
\$first	Returns true if the current object is the first in the collection
\$middle	Returns true if the current object is neither the first nor last in the collection
\$last	Returns true if the current object is the last in the collection
\$even	Returns true for the even-numbered objects in a collection
\$odd	Returns true for the odd-numbered objects in a collection

```
<tr ng-repeat="item in todos" ng-class="$odd?'odd':'even'">
  <td>{{$index + 1}}</td>
  <td ng-repeat="prop in item">{{prop}}</td>
</tr>
```


➤ <ng-include> : 외부 HTML 삽입

- <ng-include src= “...” onload= “...” ></ng-include>
- 반드시 끝 태그를 작성해야함
- src 리소스는 Ajax 요청을 해서 응답으로 받음
- 응답을 받고 로딩 완료되면 onload 핸들러 실행

```
<ng-include src="'views/exam15_ng-include/' + viewFile()"
| | | | |
| | | | | onload="handleLoad()"></ng-include>

<div ng-include="'views/exam15_ng-include/' + viewFile()"
| |
| | onload="handleLoad()"></div>
```

➤ ng-switch: 조건에 따른 엘리먼트 대체

```
<div ng-switch on="view.mode">
  <div ng-switch-when="table">
    <table class="table">
      | ...
    </table>
  </div>
  <div ng-switch-when="list">
    <ol>
      | ...
    </ol>
  </div>
  <div ng-switch-default>
    | ...
  </div>
</div>
```

❖ 스타일 관련 디렉티브

➤ 엘리먼트 보이기, 숨김, 제거

- DOM에는 추가하고 CSS의 display: none 으로 제어
ng-show= “true | false”
ng-hide= “true | false”
- DOM에 추가하거나 추가하지 않음
ng-if= “true | false”

```
<td>  
  <!-- <span ng-show="!item.complete">미완료</span> -->  
  <!-- <span ng-hide="item.complete">미완료</span> -->  
  <span ng-if="!item.complete">미완료</span>  
</td>
```

➤ CSS 클래스 및 스타일

- `ng-class= “ ‘클래스명’ | 스코프속성 ”`
- `ng-class-odd= “ ‘클래스명’ | 스코프속성 ”`
- `ng-class-even= “ ‘클래스명’ | 스코프속성 ”`

```
<tr ng-repeat="item in todos"
    ng-class="$odd?'odd':'even'" >
```

```
<tr ng-repeat="item in todos"
    ng-class-odd="'odd'"
    ng-class-even="'even'" >
```

- `ng-class= “{ ‘클래스명’ : 조건식, ... } ← 조건식이 true인 클래스가 추가`

```
<div class="alert" ng-class="{ 'alert-primary': item==current }">...</div>
```

- `ng-style= “{ ‘CSS속성명’ : ‘값’ | 스코프속성 }”`

```
<td ng-style="{ 'backgroundcolor': settings.Columns }">...</td>
```

❖ 이벤트 처리 디렉티브

- 역할: 이벤트가 발생하면 표현식을 실행시킴
- 종류

Directive	Applied As	Description
ng-blur	Attribute, class	Specifies a custom behavior for the blur event, which is triggered when an element loses the focus.
ng-change	Attribute, class	Specifies a custom behavior for the change event, which is triggered by form elements when their state of content is changed (a check box being checked, the text in an input element being edited, and so on).
ng-click	Attribute, class	Specifies a custom behavior for the click event, which is triggered when the user clicks the mouse/pointer.
ng-copy ng-cut ng-paste	Attribute, class	Specifies a custom behavior for the copy, cut, and paste events.
ng-dblclick	Attribute, class	Specifies a custom behavior for the dblclick event, which is triggered when the user double-clicks the mouse/pointer.
ng-focus	Attribute, class	Specifies a custom behavior for the focus event, which is triggered when an element gains the focus.
ng-keydown ng-keypress ng-keyup	Attribute, class	Specifies custom behavior for the keydown, keyup, and keypress events, which are triggered when the user presses/releases a key.
ng-mousedown ng-mouseenter ng-mouseleave ng-mousemove ng-mouseover ng-mouseup	Attribute, class	Specifies custom behavior for the six standard mouse events (mousedown, mouseenter, mouseleave, mousemove, mouseover, and mouseup), which are triggered when the user interacts with an element using the mouse/pointer.
ng-submit	Attribute, class	Specifies a custom behavior for the submit event, which is triggered when a form is submitted. See Chapter 12 for details of AngularJS support for forms.

```
<div class="alert alert-primary">
  <button class="btn btn-info mr-3" ng-click="handleBtnClick($event)">사진변경</button>
  
</div>
<table class="table">
  <thead>
    <tr>
      <th>#</th>
      <th>Action</th>
      <th>Done</th>
    </tr>
  </thead>
  <tbody>
    <tr ng-repeat="item in todos"
      ng-mouseenter="handleMouseEvent($event)"
      ng-mouseleave="handleMouseEvent($event)">
      <td>{{ $index + 1 }}</td>
      <td>{{ item.action }}</td>
      <td>{{ item.complete }}</td>
    </tr>
  </tbody>
</table>
```

불리언 값을 갖는 디렉티브

❖ 불리언 어트리뷰트 디렉티브

➤ 역할

- 체크 및 선택, 활성화여부, 읽기전용여부

➤ 종류

Directive	Applied As	Description
ng-checked	Attribute	Manages the checked attribute (used on input elements)
ng-disabled	Attribute	Manages the disabled attribute (used on input and button elements)
ng-open	Attribute	Manages the open attribute (used on details elements)
ng-readonly	Attribute	Manages the readonly attribute (used on input elements)
ng-selected	Attribute	Manages the selected attribute (used on option elements)

```
<div class="alert alert-success">  
  <input type="checkbox" ng-model="checked" /> 활성화  
</div>  
<p><button class="btn btn-info btn-sm" ng-disabled="!checked">버튼</button></p>  
<p><input type="checkbox" ng-checked="checked">체크박스</button></p>  
<p><input type="text" ng-readonly="!checked"/></p>
```

바인딩후 로딩을 하는 디렉티브

❖ 바인딩후 로딩을 하는 디렉티브

➤ ng-href

- <a> 태그에서 href를 대신해서 사용
- 속성값으로 AngularJS 코드를 작성할 수 있다.

`<a ng-href= “{{ pageUri }}” >xxx`

- href= “{{...}}” 은 표현식이 바인딩 되기 전에 사용자가 클릭하면 이동이 되지만(404)
- ng-href= “{{...}}” 은 바인딩 되기 전에 클릭하면 이동되지 않는다.

➤ ng-src

- 태그에서 src를 대신해서 사용
- 속성값으로 AngularJS 코드를 작성할 수 있다.

``

- src= “{{...}}” 은 표현식이 바인딩되기 전에 브라우저가 다운로드를 시도(에러 발생)
- ng-src= “{{...}}” 은 바인딩 된 후에 브라우저가 다운로드를 시도

Form 관련 디렉티브

❖ 양식과 관련된 디렉티브

➤ 텍스트 입력 양식에 사용될 수 있는 디렉티브

- <input type= "text | url | email | number" >
- <textarea>

```
<form name="myForm" novalidate>
  <div class="form-group">
    <label>Text:</label>
    <input name="sample" class="form-control"
      ng-model="inputValue"
      ng-required="requireValue"
      ng-minlength="3" ng-maxlength="10"
      ng-pattern="matchPattern"
    />
  </div>
```

url, email, number는 ng-pattern이
자동 설정되므로 기술하면 안됨

```
.controller("mainController", function ($scope) {
  $scope.requireValue = true;
  $scope.matchPattern = new RegExp("^[a-z]");
});
```

소문자 알파벳으로 시작

Name

ng-model

ng-change

ng-minlength

ng-maxlength

ng-pattern

ng-required

➤ <input type= “checkbox” >와 사용할 수 있는 디렉티브

Name	Description
ng-model	Specifies a two-model binding, as described earlier in this chapter
ng-change	Specifies an expression that is evaluated when the contents of the element are changed, as described in Chapter 11
ng-true-value	Specifies the value that the model binding expression will be set to when the element is checked
ng-false-value	Specifies the value that the model binding expression will be set to when the element is unchecked

```
<form name="myForm" novalidate>
  <div class="form-group">
    <input name="sample" type="checkbox"
      ng-checked="true"
      ng-model="inputValue"
      ng-true-value="'Hurrah!'" ng-false-value="'Boo!'" />
    This is a checkbox
  </div>
  <div class="alert alert-primary">
    <p>Model Value: {{inputValue}}</p>
  </div>
</form>

<script>
  angular.module("app", [])
    .controller("mainController", function ($scope) {
      $scope.inputValue = "Hurrah!";
    });
</script>
```

➤ <select> 와 사용할 수 있는 디렉티브

```
<form name="myForm" novalidate>
  <div>
    <select ng-model="selectedItem" ng-required="true">
      <option ng-repeat="item in items">{{item}}</option>
    </select> <br/>
    selectedItem: {{selectedItem || 'None'}}
  </div>

  <div class="mt-3">
    <select ng-model="selectedTodo1" ng-required="true">
      <option ng-repeat="todo in todos">{{todo.action}}</option>
    </select> <br/>
    selected: {{selectedTodo1 || 'None'}}
  </div>

  <div class="mt-3">
    <select ng-model="selectedTodo2" ng-required="true">
      <option ng-repeat="todo in todos" ng-value="todo.id">{{todo.action}}</option>
    </select> <br/>
    selected: {{selectedTodo2 || 'None'}}
  </div>
  <button type="submit" class="btn btn-primary mt-3" ng-disabled="myForm.$invalid">OK</button>
</form>
```

Form 관련 디렉티브

➤ <select> 와 사용할 수 있는 디렉티브

```
<form name="myForm" novalidate>
  <div>
    <select ng-model="selectValue1"
      | ng-required="true"
      | ng-options="item.action for item in todos"></select>
    <p>Selected: {{selectValue1 || 'None'}}</p>
  </div>
  <div>
    <select ng-model="selectValue2" ng-options="item.action for item in todos">
      <option value="">(Pick One)</option>
    </select>
    <p>Selected: {{selectValue2 || 'None'}}</p>
  </div>
  <div>
    <select ng-model="selectValue3" ng-options="item.id as item.action for item in todos">
      <option value="">(Pick One)</option>
    </select>
    <p>Selected: {{selectValue3 || 'None'}}</p>
  </div>
  <div>
    <select ng-model="selectValue4" ng-options="item.action group by item.place for item in todos">
      <option value="">(Pick One)</option>
    </select>
    <p>Selected: {{selectValue4 || 'None'}}</p>
  </div>
  <button type="submit" class="btn btn-primary mt-3"
    | ng-disabled="myForm.$invalid">OK</button>
</form>
```

선택된 항목의 값은 item 객체가 됨

선택된 항목의 값은 item.id가 됨

Form 관련 디렉티브

❖ 폼 유효성 검증

➤ <form name= “...” novalidate> 태그의 필요성

- 양방향 모델 바인딩만 사용할 경우에는 <form> 태그 필요 없음
- 폼 유효성 검증이 필요할 경우에는 반드시 필요
- novalidate: 일관되지 않는 브라우저의 유효성 검증 지원 기능을 비활성화

```
<form name="joinForm" novalidate ng-submit="addUser(user)">
  <div class="card card-body bg-light">
    <div class="form-group">
      <label>Name:</label>
      <input type="text" class="form-control" required ng-model="user.name" />
    </div>
    <div class="form-group">
      <label>Email:</label>
      <input type="email" class="form-control" required ng-model="user.email" />
    </div>
    <div>
      <input type="checkbox" required ng-model="user.agreed" />
      I agree to the terms and conditions
    </div>
    <button type="submit" class="btn btn-primary mt-3" ng-disabled="joinForm.$invalid">OK</button>
  </div>
</form>
```

➤ AngularJS에서 유효성을 검증할 수 있는 input 타입

Type Value	Description
checkbox	Creates a check box (pre-dates HTML5)
email	Creates a text input that accepts an e-mail address (new in HTML5)
number	Creates a text input that accepts a number address (new in HTML5)
radio	Creates a radio button (pre-dates HTML5)
text	Creates a standard text input that accepts any value (pre-dates HTML5)
url	Creates a text input that accepts a URL (new in HTML5)

➤ 폼의 유효성 모니터링

Variable	Description
\$pristine	Returns true if the user has not interacted with the element/form
\$dirty	Returns true if the user has interacted with the element/form
\$valid	Returns true if the contents of the element/form are valid
\$invalid	Returns true if the contents of the element/form are invalid
\$error	Provides details of validation errors



Form 관련 디렉티브

➤ 유효성 검사 결과 표시



- 양식 엘리먼트에 자동으로 클래스를 추가하거나 제거함으로써 검증 결과를 보고
- 유효성 검증과 관련된 CSS 클래스

Variable	Description
ng-pristine	Elements that the user has not interacted are added to this class.
ng-dirty	Elements that the user has interacted are added to this class.
ng-valid	Elements that are valid are in this class.
ng-invalid	Elements that are not valid are in this class.

- 클래스에 CSS를 적용해서 사용자에게 실시간으로 검증 피드백 전달

```
<style>
  form .ng-invalid.ng-dirty { background-color: lightpink; }
  form .ng-valid.ng-dirty { background-color: lightgreen; }
</style>
```

- 특정 유효성 검증 제약에 대한 피드백 전달

```
<style>
  form .ng-invalid-required.ng-dirty { background-color: lightpink; }
  form .ng-invalid-email.ng-dirty { background-color: lightgoldenrodyellow; }
</style>
```

Form 관련 디렉티브

➤ <input type= "text" > 유효성 검사

```
<form name="joinForm" novalidate ng-submit="addUser(user)">
  <!-- text -->
  <div class="form-group">
    <label>아이디:</label>
    <input name="uid" type="text" class="form-control"
      ng-model="user.uid"
      ng-required="true"
      ng-minlength="3" ng-maxlength="10"
      ng-pattern="/^[a-z]/" /> <!-- prefix:/, suffix:/ -->
    <div class="error mt-1" ng-show="joinForm.uid.$invalid && joinForm.uid.$dirty">
      <span ng-show="joinForm.uid.$error.required">필수 입력사항입니다.<br/></span>
      <span ng-show="joinForm.uid.$error.minlength">3자 이상을 입력해주세요.<br/></span>
      <span ng-show="joinForm.uid.$error.maxlength">10자 이하로 입력해주세요.<br/></span>
      <span ng-show="joinForm.uid.$error.pattern">알파벳으로 시작해야합니다.<br/></span>
    </div>
  </div>

  <style>
    form .ng-invalid-required.ng-dirty, form .ng-invalid-minlength.ng-dirty,
    form .ng-invalid-maxlength.ng-dirty { background-color: lightpink; }
    form .ng-valid.ng-dirty { background-color: lightcyan; }
    div.error { color: lightsalmon; font-size: 0.8em; font-weight: bold; }
  </style>
```

url, email, number는 ng-pattern이
자동 설정되므로 기술하면 안됨

소문자 알파벳으로 시작

➤ <input type= “email” > 유효성 검사

```
<div class="form-group">
  <label>Email:</label>
  <input name="email" type="email" class="form-control"
    |   |   |
    |   |   |   ng-model="user.email"
    |   |   |   ng-required="true"/>
  <div class="error" ng-show="joinForm.email.$invalid && joinForm.email.$dirty">
    |   <span ng-show="joinForm.email.$error.required">필수 입력사항입니다.<br/></span>
    |   <span ng-show="joinForm.email.$error.email">이메일 형식이 아닙니다.<br/></span>
  </div>
</div>

<style>
  form .ng-invalid-email.ng-dirty {
    |   background-color:  lightpink;
  }
  form .ng-valid.ng-dirty {
    |   background-color:  lightcyan;
  }
  div.error {
    |   color:  lightsalmon;
    |   font-size: 0.8em;
    |   font-weight: bold;
  }
</style>
```

Form 관련 디렉티브

➤ <textarea> 유효성 검사

```
<div class="form-group">
  <label>Text:</label>
  <textarea name="comment" class="form-control" cols="40" rows="3"
    ng-model="user.comment"
    ng-required="true"
    ng-minlength="2"
    ng-maxlength="3000"></textarea>
  <div class="error mt-1" ng-show="joinForm.comment.$invalid && joinForm.comment.$dirty">
    <span ng-show="joinForm.comment.$error.required">필수 입력사항입니다.<br/></span>
    <span ng-show="joinForm.comment.$error.minlength">2자 이상을 입력해주세요.<br/></span>
    <span ng-show="joinForm.comment.$error.maxlength">3000자 이하로 입력해주세요.<br/></span>
  </div>
</div>
```

➤ <select> 유효성 검사

```
<div class="form-group">
  <select name="city"
    |   |   |   ng-required="true"
    |   |   |   ng-model="user.city">
    |   <option value="">도시 선택</option>
    |   <option ng-repeat="city in cities">{{city}}</option>
  </select>
  <!-- <select name="city"
    |   |   |   ng-required="true"
    |   |   |   ng-model="user.city"
    |   |   |   ng-options="city for city in cities">
    |   <option value="">도시 선택</option>
  </select> -->
  <div class="error mt-1" ng-show="joinForm.city.$invalid && joinForm.city.$dirty">
    |   <span ng-show="joinForm.city.$error.required">필수 입력사항입니다.<br/></span>
  </div>
</div>
```

❖ 필터(filter)

➤ 역할

- 바인딩된 데이터가 뷰에 표시되기 전에 데이터를 변형
- 원본 데이터는 변경하지 않음
- 데이터 포매팅 및 정렬하는데 주로 사용됨

➤ 단일값 필터링

- 소수 이하 자리수 변경: number

```
<tr ng-repeat="p in products">  
  <td>{{p.name}}</td>  
  <td>{{p.category}}</td>  
  <td>{{p.expiry}}</td>  
  <td class="text-right">{{p.price | number:1}}</td>  
</tr>
```

Name	Category	Expiry	Price
Apples	Fruit	10	1.2
Bananas	Fruit	7	2.4

- 통화 문자 표시: currency

```
<!-- <td class="text-right">{{p.price | currency}}</td> -->  
<td class="text-right">{{p.price | currency:"₩"}}</td>
```

Apples	Fruit	₩1.20	2021.04.06
Bananas	Fruit	₩2.42	2021.04.03

- 날짜 포매팅: date
- 참고 사이트: <https://docs.angularjs.org/api/ng/filter/date>

```
<tr ng-repeat="p in products">  
  <td>{{p.name}}</td>  
  <td>{{p.category}}</td>  
  <td>{{getExpiryDate(p.expiry) | date:"yyyy.MM.dd"}}</td>  
  <td class="text-right">{{p.price | number:1}}</td>  
</tr>
```

Name	Category	Expiry	Price
Apples	Fruit	2021.03.06	1.2
Bananas	Fruit	2021.03.03	2.4

➤ 컬렉션 필터링

- 항목 개수 제한: `limitTo`

```
<tr ng-repeat="p in products | limitTo:5">
  <td>{{p.name}}</td>
  <td>{{p.category}}</td>
  <td>{{p.expiry}}</td>
  <td class="text-right">{{p.price | currency}}</td>
</tr>
```

- 항목 선택: `filter` (맵 객체 또는 `true`를 리턴하는 함수 지정)

```
<tr ng-repeat="p in products | filter:{category:'Fish'}">
  <td>{{p.name}}</td>
  <td>{{p.category}}</td>
  <td>{{p.expiry}}</td>
  <td class="text-right">{{p.price | currency}}</td>
</tr>
```

```
filter:{category:getCategory()}}
```

```
$scope.selectItems = (item) => {
  return item.category === "Fish" || item.name === "Beer";
};
```

```
<tr ng-repeat="p in products | filter:selectItems">
  <td>{{p.name}}</td>
  <td>{{p.category}}</td>
  <td>{{p.expiry}}</td>
  <td class="text-right">{{p.price | currency}}</td>
</tr>
```

- 항목 정렬: orderBy
- 올림, 내림(-) 차순 정렬

```
<tr ng-repeat="p in products | orderBy:'price'">
<!-- <tr ng-repeat="p in products | orderBy:'-price'"> -->
  <td>{{p.name}}</td>
  <td>{{p.category}}</td>
  <td>{{p.expiry}}</td>
  <td class="text-right">{{p.price | currency}}</td>
</tr>
```

- 2차 정렬

```
<tr ng-repeat="p in products | orderBy:['category', '-price']">
  <td>{{p.name}}</td>
  <td>{{p.category}}</td>
  <td>{{p.expiry}}</td>
  <td class="text-right">{{p.price | currency}}</td>
</tr>
```

➤ 필터 체인

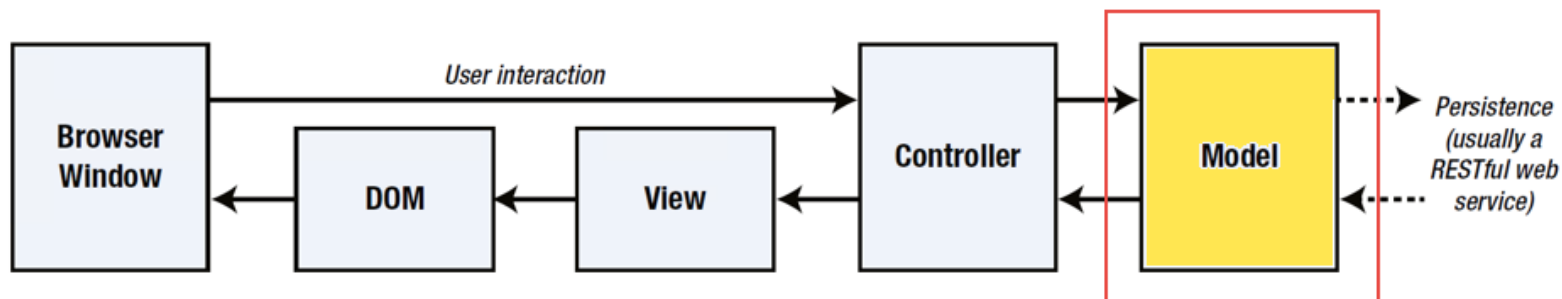
- 여러 개의 필터를 결합

```
<tr ng-repeat="p in products | orderBy:['category', '-price'] | limitTo:5">
  <td>{{p.name}}</td>
  <td>{{p.category}}</td>
  <td>{{p.expiry}}</td>
  <td class="text-right">{{p.price | currency}}</td>
</tr>
```

❖ Service

➤ 역할

- 애플리케이션 전역에서 사용 가능한 공통 기능을 제공
- 싱글톤(Singleton)으로 단 하나의 객체만 생성됨
- RESTful 웹 서비스와 네트워크 연동해서 데이터 입출력



➤ 서비스 선언

- Module의 `factory()`, `service()`, `provider()` 메소드로 커스텀 서비스 선언 가능

➤ 내장 서비스

- AngularJS에서 기본적으로 제공
- 내장 서비스 목록: <https://docs.angularjs.org/api/ng/service>

➤ 서비스 주입

- 컨트롤러 또는 다른 서비스에서 주입해서 사용할 수 있다.

❖ 서비스 선언과 주입

➤ factory(name, providerFunction) 이용

- providerFunction에서 리턴하는 객체가 서비스 객체
- 리턴된 서비스 객체는 재사용됨(싱글톤)
- providerFunction의 로컬 변수는 서비스 객체 내부에서만 사용 가능(private 효과)

```
angular.module("app")
  .factory("counterService", function() {
    let count = 0;
    //서비스 객체 리턴
    return {
      addCount: () => {
        count++;
      },
      getCount: () => {
        return count;
      }
    };
  });
```

```
<script>
  angular.module("app", [])
</script>
<script src="counterService.js"></script>
<script>
  angular.module("app")
    .controller("mainController", function($scope, counterService) {
      $scope.addCount = () => {
        counterService.addCount();
      }

      $scope.getCount = () => {
        return counterService.getCount();
      }
    });
</script>
```

➤ service(name, constructor) 이용

- constructor는 반드시 function() {} 으로 선언 (화살표 함수는 안됨)
- 주어진 constructor를 이용해서 new 연산자로 서비스 객체 생성(싱글톤)

```
angular.module("app")
  .service("counterService", function() { //function 필수
    this.count = 0;
    this.addCount = () => this.count++;
    this.getCount = () => this.count;
  });
```

```
<script>
  angular.module("app", [])
</script>
<script src="counterService.js"></script>
<script>
  angular.module("app")
    .controller("mainController", function($scope, counterService) {
      $scope.addCount = () => {
        counterService.addCount();
      }

      $scope.getCount = () => {
        return counterService.getCount();
      }
    });
</script>
```

- **provider(name, providerType);**
 - providerType은 function() {}으로 반드시 선언(화살표 함수 안됨)
 - providerType에서 리턴하는 것은 프로바이더 객체
 - config() 메소드에서 name**Provider** 라는 이름으로 주입 가능
 - 프로바이더 객체의 \$get 이 리턴하는 것이 서비스 객체임(싱글톤)

```
angular.module("app")
  .provider("counterService", function() { //function 필수
    let count = 0;
    //프로바이더 객체 리턴
    return {
      setCount: (value) => {
        count = value;
      },
      //서비스 객체 리턴
      $get: () => {
        return {
          addCount: () => count++,
          getCount: () => count
        }
      }
    }
  });
```

```
<script>
  angular.module("app", [])
</script>
<script src="counterService.js"></script>
<script>
  angular.module("app")
    .config((counterServiceProvider) => {
      //서비스 설정
      counterServiceProvider.setCount(100);
    })
    .controller("mainController", function($scope, counterService) {
      $scope.addCount = () => {
        counterService.addCount();
      }

      $scope.getCount = () => {
        return counterService.getCount();
      }
    });
</script>
```

❖ 전역 객체 접근을 위한 내장 서비스

➤ \$window

- window 객체를 래핑한 것으로 컨트롤러 및 서비스에 주입될 수 있다.

➤ \$document

- window.document의 jQuery 객체로 컨트롤러 및 서비스에 주입될 수 있다.

➤ \$interval(), \$timeout()

- window.setInterval() 메소드 래핑한 것으로 컨트롤러 및 서비스에 주입될 수 있다.
- window.setTimeout() 메소드 래핑한 것으로 컨트롤러 및 서비스에 주입될 수 있다.

```
.controller("mainController", function($scope, $window, $document, $interval) {
  $scope.openAlert = () => {
    $window.alert("알림 메시지...")
  };
  $scope.findDom = () => {
    $document.find("#content").html("Hello, AngularJS");
  };
  let timerId;
  $scope.startTime = () => {
    timerId = $interval(() => {
      var now = new Date();
      $("#content").html(now.toLocaleDateString() + " " + now.toLocaleTimeString());
    }, 1000);
  }
  $scope.endTime = () => {
    $interval.cancel(timerId);
  };
});
```

❖ URL 접근을 위한 내장 서비스

➤ \$location

- window.location 객체를 랩핑, 현재 URL 정보를 읽거나 변경할 수 있다.
- SPA(Sing Page Application)에서 내부 내비게이션에 사용한다.
- 처럼 다른 외부 문서로의 이동은 할 수 없다.
- 컨트롤러 및 서비스에 주입될 수 있다.

➤ 현재 문서의 경로에서 내부경로로 변경하기

- \$location.url(“/user/loginForm”);
=> <http://127.0.0.1:5500/service/exam05/user/loginForm>
- \$location.url(“/notice/boards”);
=> <http://127.0.0.1:5500/service/exam05/notice/boards>
- \$location.url(“/notice/boards?page=3”);
=> <http://127.0.0.1:5500/service/exam05/notice/boards?page=3>
- \$location.url(“/notice/boards?page=3#top”);
=> <http://127.0.0.1:5500/service/exam05/notice/boards?page=3#top>
- \$location.url(“/notice/boards/1”);
=> <http://127.0.0.1:5500/service/exam05/notice/boards/1>

- **경로 변경 전 이벤트: \$locationChangeStart**
 - URL이 변경되기 전 발생
 - event.preventDefault()로 URL이 변경되는 것을 막을 수 있다.

- **경로 변경 후 이벤트: \$locationChangeSuccess**
 - URL이 변경된 후 발생

- **URL 경로 정보 얻기**
 - \$location.url()
 - \$location.path()
 - \$location.search()
 - \$location.hash()

```
$scope.$on("$locationChangeStart", () => {  
  console.group("$locationChangeStart");  
  urlInfo();  
  console.groupEnd();  
});  
  
$scope.$on("$locationChangeSuccess", () => {  
  console.group("$locationChangeSuccess");  
  urlInfo();  
  console.groupEnd();  
});
```

```
const urlInfo = () => {  
  console.log("$location.url:", $location.url());  
  console.log("$location.path:", $location.path());  
  console.log("$location.search:", $location.search());  
  console.log("$location.hash:", $location.hash());  
};
```

➤ \$anchorScroll

- 단순히 주입만으로 \$location.hash()가 리턴하는 값으로 태그의 id를 찾아 스크롤을 이동
- 수동으로 이동하려면 \$anchorScroll(id)를 호출할 수 있다.

```
<script>
angular.module("app", [])
.config(function($locationProvider, $anchorScrollProvider) {
  $locationProvider.html5Mode({
    enabled: true,
    requireBase: true
  });
})
.controller("mainController", function ($scope, $location, $anchorScroll) {
  $scope.itemCount = 50;
  $scope.items = [];

  for (var i = 0; i < $scope.itemCount; i++) {
    $scope.items[i] = "Item " + i;
  }

  $scope.show = function (id) {
    $anchorScroll(id);
  };
});
</script>

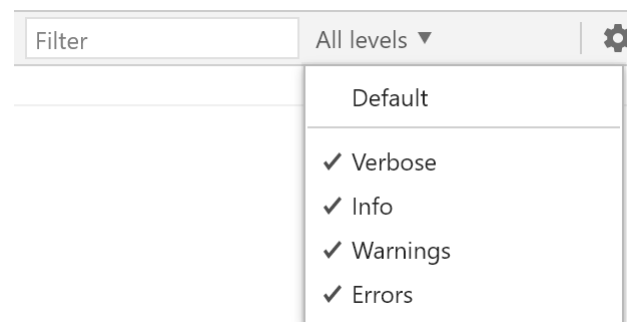
<p id="top">This is the top</p>
<button class="btn btn-primary" ng-click="show('bottom')">Go to Bottom</button>
<ul>
  <li ng-repeat="item in items">{{item}}</li>
</ul>
<p id="bottom">This is the bottom</p>
<button class="btn btn-primary" ng-click="show('top')">Go to Top</button>
```

❖ 로깅 서비스

➤ \$log

- 전역 콘솔 객체(console)를 감싼 래퍼이다.
- 콘솔 객체의 메소드에 대응되는 debug(), info(), warn(), error() 메소드 제공한다.
- 개발자 도구 메뉴에서 로그 레벨을 verbose로 체크해야만 debug 로그를 볼 수 있다.
- \$logProvider.debugEnabled(false)는 debug 로그를 출력을 하지 않는다.
(기본: true)

```
<script>
angular.module("app", [])
.config(function($logProvider) {
  $logProvider.debugEnabled(true);
})
.controller("mainController", function ($scope, $log) {
  $log.error("LogLevel: error");
  $log.warn("LogLevel: warn");
  $log.info("LogLevel: info");
  $log.debug("LogLevel: debug");
});
</script>
```



❖ AngularJS 비동기 통신을 위한 서비스

➤ \$http 서비스

- 비동기 Ajax 요청을 수행
- 표준 HTTP 요청을 수행하는 메소드 제공

Name	Description
get(url, config)	Performs a GET request for the specified URL.
post(url, data, config)	Performs a POST request to the specified URL to submit the specified data.
delete(url, config)	Performs a DELETE request to the specified URL.
put(url, data, config)	Performs a PUT request with the specified data and URL.
head(url, config)	Performs a HEAD request to the specified URL.
jsonp(url, config)	Performs a GET request to obtain a fragment of JavaScript code that is then executed. JSONP, which stands for <i>JSON with Padding</i> , is a way of working around the limitations that browsers apply to where JavaScript code can be loaded from. I do not describe JSONP in this book because it can be incredibly dangerous; see http://en.wikipedia.org/wiki/JSONP for details.

- 이들 메소드의 리턴값은 프라미스(Promise)이다.

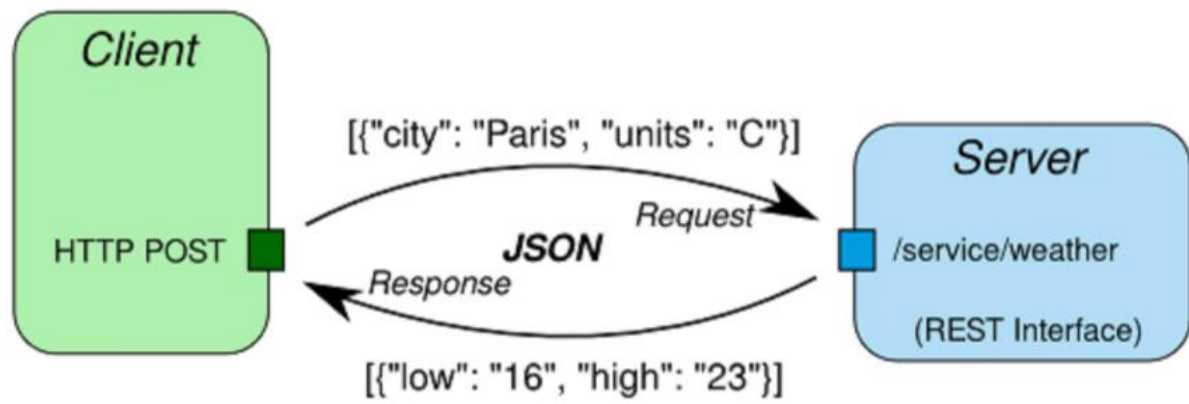
➤ REST

- **RE**presentational **S**tate **T**ransfer
- HTTP 요청 방식 + URL 조합으로 자원 요청
- HTTP 요청 방식에 따라 수행할 작업을 식별
- URL은 작업할 데이터를 식별

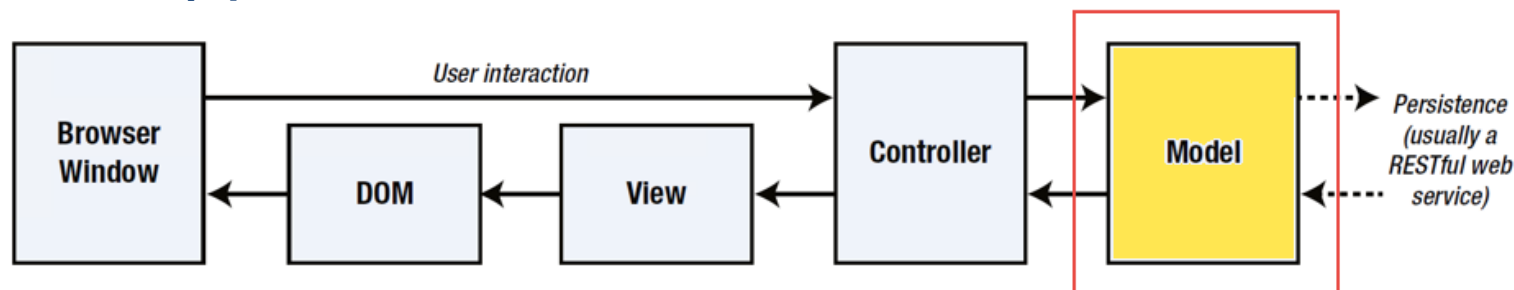
CRUD	HTTP	URI
전체 리소스 조회	GET	/resources
특정 리소스 조회	GET	/resources/:id
리소스 생성	POST	/resources
리소스 전체 수정	PUT	/resources/:id
리소스 일부 수정	PATCH	/resources/:id
특정 리소스 삭제	DELETE	/resources/:id

➤ RESTful 웹 서비스

- REST 스타일을 따르는 API를 제공하는 웹 서비스
- 요청 및 응답 본문의 데이터 포맷은 JSON 사용



➤ Model 서비스 선언



```
angular
.module("app", [])
.controller("mainController", function ($scope, productsModel) {
  $scope.displayMode = "list";
  $scope.currentProduct = null;

  $scope.listProducts = () => {
    productsModel.list()
    .then((response) => {
      $scope.products = response.data;
    });
  };
  $scope.listProducts();

  $scope.createProduct = (product) => {
    productsModel.create(product)
    .then((response) => {
      return productsModel.list();
    })
    .then((response) => {
      $scope.products = response.data;
      $scope.displayMode = "list";
    });
  };
});
```

```
angular.module("app")
.constant("BASE_URL", "http://localhost:8080/products/")
.factory("productsModel", function($http, BASE_URL) {
  return {
    list: () => {
      return $http.get(BASE_URL);
    },

    create: (product) => {
      return $http.post(BASE_URL, product);
    },

    delete: (product) => {
      return $http.delete(BASE_URL + product.pid);
    },

    update: (product) => {
      return $http.put(BASE_URL + product.pid, product);
    }
  };
});
```

수고 하셨습니다.