* HTML파일에 AngularJS 적용
  + angular.js파일을 불러오는 script element를 추가함으로써 적용
  + AngularJS는 하나 이상의 module로 구성
    - angular.module method를 호출하여 module 생성
      * 생성할 module name, 필요한 추가 module array를 parameter로 호출
      * var totoApp = **angular.module(“todoApp”, [])**
        + application module name에 App을 첨부하는 명명 관례를 사용해 todoApp이라는 module을 생성
        + 빈 배열을 두번째 parameter로 하여 추가 모듈이 필요 없음을 AngularJS에게 알린다(dependency parameter를 빼면 에러 발생)
  + ng-app attribute를 사용하여 AngularJS가 모듈을 어떻게 적용해야하는지를 알려준다
    - ng-app attribute는 AngularJS가 컴파일하고 처리해야할 모듈이 있음을 알려주는 역할을 한다.
      * 자바스크립트 프레임워크로 AngularJS만 사용하는 경우 관례상 ng-app attribute를 html element에 적용한다
      * AngularJS를 Jquery 같은 다른 기술과 함께 사용하는 경우에는 ng-app attribute를 문서 내 element에 적용함으로써 AngularJS 앱이 차지하는 범위를 제한할 수 있다
    - <html **ng-app=”todoApp”**></html>
  + AngularJS는 새 엘리먼트, 어트리뷰트, 클래스 및 특수 주석을 추가함으로써 HTML을 확장한다.
    - AngularJS 라이브러리는 동적으로 문서 내 HTML을 파싱해 추가 요소의 위치를 파악하고 처리하여 애플리케이션 생성
    - AngularJS 라이브러리는 **브라우저가 컨텐츠를 로드할 때**, HTML 엘리먼트를 해석하고, 표준 DOM API 및 자바스크립트 기능을 사용해 엘리먼트를 추가 또는 제거하고, 이벤트 핸들러 등을 설정
    - AngularJS 개발 과정에서는 명시적인 컴파일 단계가 없고 HTML과 자바스크립트 파일을 수정한 후 브라우저로 로드하는 과정만 있다.
* 데이터 모델 생성
  + AngularJS는 MVC 패턴을 지원한다.
    - 데이터를 데이터 표현 로직과 분리
  + Model
    - application의 데이터
    - 데이터의 저장 및 조회와 관련한 로직 포함
  + Controller
    - 데이터를 처리하는 로직
    - 모델과 뷰 사이에 위치하며 둘을 서로 연결
  + View
    - 데이터를 보여주는 로직
    - 데이터를 formatting하고 사용자에게 보여주는 로직
* 컨트롤러 생성
  + View를 지원하는 데 필요한 비즈니스 로직을 정의
  + 데이터를 처리하는 로직
  + 모델과 뷰 사이에 위치하며 둘을 서로 연결
  + Controller는 사용자 상호작용에 반응하며, 모델 내 데이터를 업데이트하고, 뷰에서 요청한 데이터를 뷰에게 제공
  + angular.module을 호출해 반환되는 Module객체에서 controller method를 호출해 생성
    - 관례상 컨트롤러의 이름은 <이름>Ctrl로 지정(<이름>은 애플리케이션에서 컨트롤러가 하는 역할을 잘 나타내는 이름으로 한다)
    - var todoApp = angular.module(“todoApp”, []);
    - **todoApp.controller(“ToDoCtrl”, function($scope){ $scope.todo = model; });**
    - <body **ng-controller=”ToDoCtrl”**></body>
  + **$scope**
    - View가 항상 전체 모델에 접근할 수 있는 것은 적절하지 않으므로 컨트롤러를 사용하 뷰가 접근할 수 있는 데이터영역을 선택한다
    - **'$'** 로 시작하는 변수명은 AngularJS에서 제공하는 내장 기능을 나타낸다. 주로 내장 **서비스**를 나타냄
    - 서비스는 자체 컴포넌트로 여러 컨트롤러에게 기능을 제공하는 일을 한다
    - $scope은 특수 변수로서, 뷰에게 데이터 및 기능을 노출하는 데 사용한다.
    - $scope.todo = model; → $scope 서비스 객체에 todo라는 속성을 정의하고 model을 대입
  + **ng-controller** attribute 사용하여 컨트롤러가 책임질 HTML 문서 영역 지정
* 뷰 생성
  + 뷰는 컨트롤러에서 제공하는 데이터를 브라우저에서 보여줄 컨텐츠를 생성하게 하는 annotation HTML element와 결합함으로써 생성
  + **Data binding** annotation을 사용
    - **{{~}}**를 사용해 data binding 표현식을 지정
    - 표현식 안의 내용은 자바스크립트에 의해 해석
    - 컨트롤러의 $scope에 지정된 데이터 및 함수로 제한된다
    - AngularJS는 문서 내 HTML을 컴파일하고 ng-controller attribute를 찾은 후, ng-controller로 연결된 함수를 호출해 뷰를 생성하는데 사용할 $scope를 설정한다. 각 데이터 바인딩 표현식을 만날 때마다 AngularJS는 지정된 값을 $scope 객체에서 검사한 후 해당 값을 HTML 문서에 삽입한다.
    - 표현식은 데이터 값을 포여주기 적절하게끔 간단히 수정하는데만 사용해야한다.
    - Data binding은 복잡한 로직을 수행하거나 모델을 조작하는데 사용해서는 안된다.
  + **Directive** 사용
    - 표현식은 Directive와 함께 사용할 수 있다.
    - Directive는 AngularJS에게 컨텐츠를 어떻게 처리해야 할지 알려준다
    - ex :<tr **ng-repeat=”item in todo.items”**><td>{{item.action}}</td></tr> → tr element 및 포함된 element들을 todo.items배열 내 객체별로 생성하고, 배열 내 각 객체를 item이라는 변수에 대입
    - Directive 종류
      * **ng-repeat=”<이름> in <컬렉션>”** → 컬렉션을 차례로 순회하면서 element를 <이름>에 대입한다.
      * **ng-hide=”표현식”** → attribute로 지정된 표현식이 true일 경우, 이 디렉티브가 적용된 엘리먼트(및 컨텐츠 엘리먼트)를 숨겨준다.
      * **ng-class** → 동적으로 클래스명을 바꿀 수 있다.(동적으로 CSS class들을 html에 적용 시킬 수 있다.)
      * **ng-model** → Controller scope 내에서 동적으로 속성을 생성해주고, 사용자 입력을 처리하는 데 사용할 수 있는 동적인 모델 속성을 생성해준다.
      * **ng-click=”표현식”** → click event가 일어날 때 표현식을 실행하는 핸들러를 설정
  + **양방향 바인딩**
    - 단방향 바인딩 : 모델에서 값을 가져와 템플릿 내 엘리먼트를 채우는데 사용
    - 양방향 바인딩
      * 모델은 엘리먼트를 생성하는 데 사용될 뿐 아니라, 엘리먼트 내 변화가 생기면 모델 또한 같이 바뀌게 된다.
      * 양방향 바인딩은 사용자 입력을 받는 엘리먼트에 적용할 수 있다.
      * **ng-model** attribute
        + AngularJS에서 입력 엘리먼트와 데이터 객체의 done 속성 사이에 양방향 바인딩을 적용하게끔 한다.
        + <input type=”checkbox” **ng-model=”item.done”**/>
* 컨트롤러 동작 구현 및 활용
  + 컨트롤러는 scope의 동작을 정의한다.
    - 동작
      * application의 비즈니스 로직을 구현하기 위해 모델 내 데이터에 수행하는 기능
      * 컨트롤러에서 정의한 동작은 데이터를 사용자에게 보여줄 수 있게 뷰를 지원하며, 사용자 상호작용을 기반으로 모델을 업데이트한다.
    - 동작은 컨트롤러 함수로 전달된 $scope 객체에 함수를 추가함으로써 정의
    - $scope 객체에 함수를 첨부하는 데 사용한 속성명은 동작명으로 사용되며, 뷰를 형성하는 HTML 엘리먼트에 컨트롤러를 적용하는 ng-comtroller attribute scope 내에서 호출할 수 있다.('()'를 사용해 호출)
    - 동작으로는 객체를 인자로 넘겨줄 수도 있다.
* 사용자 상호작용 반응
  + 실제 프로젝트에서는 뷰를 위해 데이터를 가져오고 준비하는 동작과 사용자 상호작용에 반응하고 모델을 업데이트하는 동작이 여러개로 분리되어 있는 게 보통이다.
  + AngularJS가 HTML을 컴파일하다 directive를 만나면, AngularJS는 이벤트 핸들러 코드가 엘리먼트와 분리되게끔 핸들러를 설정해준다.
    - <input type=”button” **ng-click=”addNewItem(actoinText)”** value=”add”/>
    - AngularJS가 이벤트 핸들러 코드를 엘리먼트와 분리되게 해준다
* 모델 데이터의 필터 및 정렬
  + 동작을 구현하지 않고도 뷰에서 보여줄 모델의 데이터를 효과적으로 준비할 수 있다.
  + 동작을 사용하는 것보다 좀 더 범용적인 용도로 사용하고, 애플리케이션 전역에서 재사용하는 특징이 있다.
  + 필터링은 데이터 모델의 어느 부분에나 적용할 수 있다.
  + Filter 종류
    - **filter**
      * 설정된 기준에 따라 객체를 선택
    - **orderBy**
      * 데이터를 정렬
      * 정렬에 사용하려는 속성을 리터럴값으로 지정
      * 리터럴 값으로 지정하지 않으면 scope 속성을 찾음
        + 기본적으로 AngularJS에서는 모든 것을 $scope에 의해 정의된 속성으로 간주하므로 리터럴 값으로 사용하지 않으면 $scope 속성을 찾는다
        + 상수를 지정해야할 때는 리터럴 값 사용
  + 커스텀 필터
    - AngularJS 모듈 객체에서 정의한 filter() method로 filter factory를 생성할 수 있다.
    - Filter factory → 데이터 객체 셋을 필터링하는 데 사용할 함수를 반환
      * angular-module.**filter(“filter-name”, function(){return function(items){}})**
      * items parameter → AngularJS에서 제공하며, 필터링할 객체 셋
      * items parameter외에 parameter 사용가능
        + filter(“filter-name”, function(){return function(items, showComplete){}})
    - 사용
      * **custom-filter:전달인자**
* Ajax를 통한 데이터 수신
  + Ajax 요청을 통해 데이터를 받을 수 있다.
  + Angular-module.**run(function($http){$http.get(“url”).success(function(data){})})**
    - AngularJS 모듈 객체에 정의된 run method에 호출 코드 추가
    - run method는 AngularJS가 **초기 설정을 마치는 시점에 한 번만 실행할 함수**를 parameter로 받는다
    - $http 서비스 객체를 parameter로 지정하여 AngularJS에게 $http 서비스를 사용할 것을 알린다.(dependency injection)
    - $http 서비스 객체
      * 저수준 Ajax 요청에 대한 접근 기능 제공
      * RESTful 웹 서비스에서 사용하는 $resource와 비교할 때 실제로 그렇게 저수준은 아님
    - get() method의 결과는 **Promise 객체**로서, 이 객체는 미래에 완료될 일을 나타내는 데 사용된다.
    - 서버에서 조회한 JSON 데이터는 파싱을 거쳐 자바스크립트 객체가 되고, success 함수의 data parameter로 넘어온다
  + **Dependency Injection**
    - parameter를 사용해 필요한 기능을 AngularJS에게 알려주는 방식
* AngularJS이해
  + AngularJS의 목적
    - Server side 개발 시에만 사용할 수 있었던 툴과 기능을 web client에도 제공함으로써, 풍부하고 복잡한 web application을 개발, 테스트, 유지보수하기 쉽게 해주기 위함
  + AngularJS는 HTML을 확장할 수 있게 해줌으로써 기능을 발휘
  + AngularJS application은 custom element, attibute, class, comment를 통해 기능을 표현
  + AngularJS는 MVC 패턴에서 유래함.
    - AngularJS를 사용할 때 다른 모습으로 많이 바뀌는 만큼 MVW(Model-View-Whatever)라고 불리기도 한다.
  + AngularJS는 server side 개발자들만 사용할 수 있었던 기능을 제공하지만, 이 작업을 모두 브라우저 내에서 처리한다.
    - AngularJS를 적용한 HTML 문서를 로드할 때, AngularJS가 매번 많은 일을 처리해야한다.
    - 수행 시간은 HTML문서의 복잡도, 관련 자바스크립트 코드, 브라우저의 품질, 기기의 처리 능력에 좌우된다.
    - 가능한 설정을 덜 수행하고, 설정을 수행할 때 사용자에게 가능한 앱의 많은 부분을 제공해야한다.
  + AngularJS는 SPA(Single Page Architecture)에 가까워질수록 초기 작업 부담을 크게 줄여주는 효과가 있다.
    - 라운드 트립 애플리케이션
      * 브라우저에서 서버에 초기 HTML문서를 요청 → 사용자 상호작용이 일어나면 브라우저는 새 HTML문서를 요청해서 다시 받는다
      * 브라우저가 사실상 HTML 컨텐츠의 랜더링 엔진이며, 모든 애플리케이션 로직 및 데이터는 서버단에 있다.
      * 브라우저는 계속해서 무상태 HTTP요청을 보내고, 서버는 동적으로 HTML문서를 생성해 요청을 처리한다.
      * 브라우저에서 필요한 기능이 적기 때문에 가능한 많은 클라이언트를 지원할 수 있기 때문에 많은 웹 개발이 라운드 트립 애플리케이션으로 이루어지고 있다.
      * 단점
        + 다음 HTML문서를 요청하고 로드할 때까지 사용자를 기다리게 한다.
        + 많은 요청을 처리하고 모든 애플리케이션 상태를 관리할 대규모의 서버사이드 인프라스트럭쳐가 필요
        + 각 HTML문서 내에 모든 내용이 포함돼 있으므로 많은 대역폭이 필요
    - 단일 페이지 애플리케이션
      * 초기 HTML문서가 브라우저로 보내진다 → 사용자 상호작용이 일어나면 Ajax 요청을 통해 HTML이나 데이터 일부를 가져와 기존 엘리먼트에 삽입 후 사용자에게 보여준다
      * 초기 HTML문서는 다시 로드하거나 대체하지 않으며, 사용자는 Ajax 요청이 비동기적으로 처리되는 동안 기존 HTML과 계속 상호작용 할 수 있다.
    - 라운드 트립 모델을 사용하면서 자바스크립트를 사용해 전체 페이지 변경 횟수를 줄이는 개선 방식을 많이 사용
    - 현재 웹 앱 프로젝트는 단일 페이지 모델로 이동하는 추세이다.
  + AngularJS및 유사 프레임워크는 복잡한 웹 애플리케이션이 작성하고 유지보수하기 어렵기 때문에 등장했다.
* AngularJS와 jQuery
  + jQuery
    - 명시적으로 브라우저의 DOM을 조작해 애플리케이션을 생성하는 데 초점을 맞춤
    - 복잡한 대형 애플리케이션을 작성하고 관리하기 어려우며, 완벽한 단위 테스트 작업 또한 매우 어렵다.
    - 단위 테스트가 중요하지 않고, 즉각적으로 결과를 확인하고 싶은, 복잡도가 낮은 앱에 적합
    - 라운드 트립 웹 앱에서 생성한 HTML을 개선하는 데도 이상적임
  + AngularJS
    - 브라우저와 협력해 애플리케이션의 개발 기초를 제공하는 데 초점을 맞춤
    - AngularJS는 jQuery의 핵심 기능을 기반으로 개발 됐다.
      * AngularJS에는 jqLite(jQuery의 축소 버전)가 들어있다.
      * Custom directive를 작성하는 데 jqLite가 사용된다.
      * jQuery를 HTML에 추가하면 AngularJS에서는 이를 자동으로 감지해 jqLite 대신 jQuery를 사용
    - 단점
      * 실제 결과를 보기 전까지 상단한 초기 개발 투자 시간이 필요
    - 충분한 설계 및 기획 시간이 있을 때 복잡한 단일 페이지 웹 앱에 사용해야 한다.
* MVC 패턴
  + MVC 개념은 초기 GUI 애플리케이션을 조직화하기 위한 반안으로 고안됐다.
    - 특히 웹 애플리케이션에 매우 적합
  + 최근, MVC 패턴은 날이 갈수록 기능이 풍부해지고, 복잡해지는 클라이언트사이드 웹 개발을 관리하는 방식으로 인식되고 있다.(AngularJS 또한 이런 환경에서 등장)
  + MVC 패턴의 핵심
    - **관심사의 분리**
    - 애플리케이션의 모델은 비즈니스 및 프레젠테이션 로직과 분리
    - 클라이언트사이드 웹 개발에서, 데이터, 데이터를 처리하는 로직, 데이터를 보여주는 데 사용하는 HTML element를 분리해야 한다.
  + 라운드 트립 웹 앱에서 모델을 DB에서 가져오고, 브라우저가 보낸 HTTP 요청을 서비스 하는 것을 애플리케이션의 목적으로 전제 한다.
  + 클라이언트사이드 MVC 패턴 구현체는 데이터를 서버사이드 컨포넌트로부터 가져오며 주로 RESTful 웹 서비스를 사용한다. 컨트롤러와 뷰의 목적은 모델에 들어 있는 데이터를 가지고 DOM 조작을 수행해 사용자가 상호작용할 수 있는 HTML 엘리먼트를 생성하고 관리 하는 것이다. 상호작용은 컨트롤러로 전달됨으로써 interactive application을 구성할 수 있는 고리를 완성한다.
  + AngularJS 클라이언트를 사용하면 AngularJS를사용하지 않을 경우 서버단에 존재할 복잡한 작업을 일부 대신 처리해줄 뿐이다. (서버사이드에서도 MVC 프레임워크를 사용할 수 있다.)
  + AngularJS를 사용하면 서버에서 처리하는 부담이 줄고, 더 적은 서버 성능으로 더 많은 클라이언트를 지원할 수 있다.
  + AngularJS Model-View-Controller
    - Model
      * 사용자가 사용할 데이터가 들어있다.
      * Model의 유형
        + **View model**

Controller에서 View로 전달된 데이터를 나타낸다

* + - * + **Domain Model**

비즈니스 도메인 내 데이터, 작업, 변형, 데이터를 생성, 정령, 조작하는 데 필요한 규칙(통칭 **Model logic**)이 들어 있다.

* + - * AngularJS Model은 로직을 서버사이드로 효과적으로 넘겨주고, RESTful 웹 서비스를 통해 로직을 호출한다. 필요한 데이터는 Ajax를 통해 가져오는게 간편하다.
      * Model이 포함해야할 것
        + Domain data
        + Domain data를 생성, 관리, 수정하는 로직(웹 서비스를 통해 원격 로직을 실행하더라도 포함)
        + Model data를 노출하고 model data 상에서 작업을 수행할 수 있는 정돈된 API를 제공해야 한다.
      * Model이 포함하지 말아야 할 것
        + Model을 가져오고 관리하는 상세 방법(데이터 저장 매커니즘 또는 원격 웹 서비스에 대한 세부 사항을 컨트롤러나 뷰로 노출하지 말아야 한다.)
        + 사용자 상호작용을 기반으로 모델을 변형하는 로직(Controller의 역할)
        + 사용자에게 데이터를 보여주는 로직(View의 역할)
      * 휼륭한 도메인 모델에는 CRUD를 위한 로직이 들어있다.
      * AngularJS Model은 대개 RESTful 웹 서비스를 호출해 서버사이드 DB 작업을 호출하는 로직을 담고 있다.
    - View
      * 개선된 HTML element를 통해 Data binding 및 directive를 사용해 HTML을 생성함으로써 정의
      * AngularJS application의 directive는 HTML element를 통해 동적인 웹 앱을 구현함으로써 유연한 뷰를 만들어준다.
      * View가 포함해야할 것
        + 사용자에게 데이터를 보여주는 데 필요한 로직과 마크업
      * View가 포함하지 말아야할 것
        + 복잡한 로직(controller에 두는 게 적합)
        + 도메인 모델을 생성, 저장, 조작하는 로직
      * 뷰도 로직을 포함할 수 있지만, 최대한 단순해야하며, 가급적 포함하지 않는 게 좋다.
        + 가장 단순한 형태의 메서드 호출이나 표현식 이외의 로직을 View에 넣으면 테스트 및 유지보수가 어려워 질 수 있다.
    - Controller
      * AngularJS 웹 앱에서 데이터 모델과 뷰를 연결
      * Controller는 비즈니스 도메인 로직(동작)을 모델의 일부인 scope에 추가한다.
      * Controller가 포함해야할 것
        + scope를 초기화하는 데 필요한 로직
        + scope의 데이터를 뷰가 표시하는 데 필요한 로직/동작
        + 사용자 상호작용을 기반으로 scope를 업데이트 하는 데 필요한 로직/동작
      * Controller가 포함하지 말아야할 것
        + DOM을 조작하는 로직(View 의 역할)
        + 데이터의 영속화를 관리하는 로직(Model의 역할)
        + scope 밖에 있는 데이터를 조작하지 말아야 한다.
      * View data(View model)
        + Controller는 View data를 생성해 뷰 정의를 단순화한다.
        + View data는 영속적이지 않으며 도메인 모델 데이터의 일부를 합성하거나 사용자 상호작용에 반응해 생성한다.
        + View data는 주로 controller의 scope를 통해 생성하고 접근한다.
* RESTful
  + AngularJS 앱의 도메인 모델 로직은 클라이언트와 서버 사이에 존재.
    - 서버
      * 주로 DB와 같은 영속성 저장소가 있으며, 이를 관리하는 로직이 존재
    - 클라이언트사이드 코드
      * 데이터 저장소에 직접 접근하지 않는게 좋다.
  + 서버를 사용해 데이터 저장소에 대한 접근을 중재
    - 클라이언트와 데이터 저장소 사이의 의존성을 줄 일 수 있다.
    - 클라이언트 코드에서는 서버로부터 데이터를 가져오고 전달하는 책임은 있지만, 데이터의 내부 저장 및 접근 방식에 대한 상세 정보는 모른다.
  + 주로 Ajax 방식으로 클라이언트와 서버 간에 데이터를 주고 받음
  + HTTP요청의 성격을 그대로 사용해 데이터의 CRUD 작업을 수행
    - HTTP방식 및 URL의 조합으로 RESTful 웹 서비스에서 요청하는 작업을 표현
      * GET : URL을 통해 지정한 데이터를 조회
      * PUT : URL을 통해 지정한 데이터를 업데이트
      * POST : 주로 폼데이터 값을 데이터 필드로 사용해 데이터 객체를 생성
      * DELETE : URL을 통해 지정한 데이터를 삭제
      * POST 방식을 통해 객체가 없으면 생성하고 있으면 업데이트 하는 방식을 주로 사용 → PUT 방식을 사용하지 않아도 됨
  + 가능한 한 URL을 통해 데이터 저장소의 내부 구조를 노출하지 않게끔 간단 명료한 URL을 사용하도록 해야함
  + 자주 하기 쉬운 설계 실수
    - 잘못된 곳에 로직 배치
      * 관심사의 분리 방해
        + 비즈니스 로직을 컨트롤러가 아니 뷰에 두기
        + 도메인 로직을 모델이 아니닌 컨트롤러에 두기
        + RESTful 서비스를 사용할 때 데이터 저장 로직을 클라이언트 모델에 두기
      * 로직 배치에 대한 규칙
        + 뷰 로직은 데이터 표현만을 위해 데이터를 준비해야 하며, 모델을 수정해서는 안된다
        + 컨트롤러 로직은 직접 모델로부터 데이터를 생성, 수정, 삭제하지 말아야한다.
        + 클라이언트는 데이터 저장소에 직접 접근하지 말아야한다.
    - 데이터 저장소 데이터 형식의 채택
      * 개발 팀이 서버사이드 데이터 저장소의 특이성에 의존하는 애플리케이션을 개발할 때 주로 문제 발생
      * 잘 설계된 AngularJS 애플리케이션은 RESTful 서비스를 통해 데이터를 가져오고 , 데이터 저장소의 상세 구현 정보를 숨기고 클라이언트로 적당한 데이터 형식을 통해 데이터를 제공하는 것은 서버의 몫이다.
      * 클라이언트에서 데이터를 어떻게 표현해야 하는지 결정하고, 데이터 저장소에서도 이 형식을 사용하게 해야한다
        + 데이터 저장소에서 형식을 지원하지 못한다면 데이터를 변형하는 것 또한 서버의 역할이다.
    - AngularJS 앱에서 jQuery를 통해 DOM을 직접 조작한다면 문제가 있다.