Introduction to services and dependency injection

Service is a broad category encompassing any value, function, or feature that an app needs. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

Angular distinguishes components from services to increase modularity and reusability. By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient.

Ideally, a component's job is to enable the user experience and nothing more. A component should present properties and methods for data binding, in order to mediate between the view (rendered by the template) and the application logic (which often includes some notion of a *model*).

A component can delegate certain tasks to services, such as fetching data from the server, validating user input, or logging directly to the console. By defining such processing tasks in an *injectable service class*, you make those tasks available to any component. You can also make your app more adaptable by injecting different providers of the same kind of service, as appropriate in different circumstances.

Angular doesn't *enforce* these principles. Angular does help you *follow* these principles by making it easy to factor your application logic into services and make those services available to components through *dependency injection*.

Service examples

Here's an example of a service class that logs to the browser console.

src/app/logger.service.ts (class)

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

Services can depend on other services. For example, here's a `HeroService` that depends on the `Logger` service, and also uses `BackendService` to get heroes. That service in turn might depend on the `HttpClient` service to fetch heroes asynchronously from a server.

src/app/hero.service.ts (class)

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

Dependency injection (DI)

DI is wired into the Angular framework and used everywhere to provide new components with the services or other things they need. Components consume services; that is, you can *inject* a service into a component, giving the component access to that service class.

To define a class as a service in Angular, use the `@Injectable()` decorator to provide the metadata that allows Angular to inject it into a component as a *dependency*. Similarly, use the `@Injectable()` decorator to indicate that a component or other class (such as another service, a pipe, or an NgModule) *has* a dependency.

- The *injector* is the main mechanism. Angular creates an application-wide injector for you during the bootstrap process, and additional injectors as needed. You don't have to create injectors.
- An injector creates dependencies, and maintains a *container* of dependency instances that it reuses if possible.
- A *provider* is an object that tells an injector how to obtain or create a dependency.

For any dependency that you need in your app, you must register a provider with the app's injector, so that the injector can use the provider to create new instances. For a service, the provider is typically the service class itself.

> A dependency doesn't have to be a service—it could be a function, for example, or a value.

When Angular creates a new instance of a component class, it determines which services or other dependencies that component needs by looking at the constructor parameter types. For example, the constructor of `HeroListComponent` needs `HeroService`.

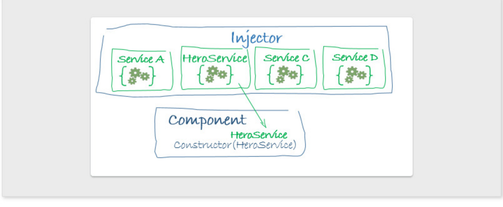
src/app/hero-list.component.ts (constructor)

```
constructor(private service: HeroService) { }
```

When Angular discovers that a component depends on a service, it first checks if the injector has any existing instances of that service. If a requested service instance doesn't yet exist, the injector makes one using the registered provider, and adds it to the injector before returning the service to Angular.

When all requested services have been resolved and returned, Angular can call the component's constructor with those services as arguments.

The process of `HeroService` injection looks something like this.



Providing services

You must register at least one *provider* of any service you are going to use. The provider can be part of the service's own metadata, making that service available everywhere, or you can register providers with specific modules or components. You register providers in the metadata of the service (in the `@Injectable()` decorator), or in the `@NgModule()` or `@Component()` metadata

- By default, the Angular CLI command `ng generate service` registers a provider with the root injector for your service by including provider metadata in the `@Injectable()` decorator. The tutorial uses this method to register the provider of HeroService class definition.

  ```
  @Injectable({
    providedIn: 'root',
  })
  ```

  When you provide the service at the root level, Angular creates a single, shared instance of `HeroService` and injects it into any class that asks for it. Registering the provider in the `@Injectable()` metadata also allows Angular to optimize an app by removing the service from the compiled app if it isn't used.

- When you register a provider with a specific NgModule, the same instance of a service is available to all components in that NgModule. To register at this level, use the `providers` property of the `@NgModule()` decorator,

  ```
  @NgModule({
    providers: [
    BackendService,
    Logger
    ],
    ...
  })
  ```

- When you register a provider at the component level, you get a new instance of the service with each new instance of that component. At the component level, register a service provider in the `providers` property of the `@Component()` metadata.

  src/app/hero-list.component.ts (component providers)

  ```
  @Component({
    selector:    'app-hero-list',
    templateUrl: './hero-list.component.html',
    providers:  [ HeroService ]
  })
  ```

For more detailed information, see the Dependency Injection section.