



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Dependency Injection Deep Dive

ANGULARarchitects.io

Content

- Classic Providers
- Tree-shakable Providers
- Scopes
- Constants as Tokens
- Services and Lazy Loading
- Multi Providers



Classic Providers

Service

```
@Injectable()  
export class FlightService {  
  
    [...]  
  
}
```

Classic Providers

```
@NgModule({  
  imports: [  
    BrowserModule, HttpClientModule, FormsModule  
  ],  
  declarations: [  
    AppComponent, FlightSearchComponent  
  ],  
  providers: [  
    FlightService  
  ],  
  bootstrap: [  
    AppComponent  
  ]  
})  
export class AppModule {  
}
```

Injecting Service into Consumer

```
@Component({  
    selector: 'flight-search',  
    templateUrl: 'flight-search.html'  
})  
export class FlightSearchComponent {  
    von: string;  
    nach: string;  
    fluege: Array<Flight>;  
  
    constructor(flightService: FlightService) { ... }  
  
    flightSearch() { [...] }  
    selectFlight(flight) { [...] }  
}
```



Abstraction

```
export abstract class FlightService {  
    abstract find(from: string, to: string): Observable<Flight[]>;  
}
```

```
@Injectable()  
export class DefaultFlightService implements FlightService {  
    find(from: string, to: string): Observable<Flight[]> { ... }  
}
```

"Forward" with useClass

```
@NgModule({  
  imports: [  
    BrowserModule, HttpClientModule, FormsModule  
  ],  
  declarations: [  
    AppComponent, FlightSearchComponent  
  ],  
  providers: [  
    { provide: FlightService, useClass: DefaultFlightService }  
  ],  
  bootstrap: [  
    AppComponent  
  ]  
})  
export class AppModule {  
}
```


"Forward" with useClass

```
@NgModule({
  imports: [
    BrowserModule, HttpClientModule, FormsModule
  ],
  declarations: [
    AppComponent, FlightSearchComponent
  ],
  providers: [
    { provide: FlightService, useClass: DefaultFlightService }
  ],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {
}
```



"Forward" with useClass

```
@NgModule({
  imports: [
    BrowserModule, HttpClientModule, FormsModule
  ],
  declarations: [
    AppComponent, FlightSearchComponent
  ],
  providers: [
    { provide: FlightService, useClass: DefaultFlightService }
  ],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {
}
```

Token ↑

Service ↑



Tokens

- Interfaces **cannot** be used as tokens
- However, you can use abstract classes instead

useFactory

```
const DEBUG = true;
[...]  
  
providers: [{  
  provide: FlightService,  
  useFactory: (http: HttpClient) => {  
    if (DEBUG) {  
      return dummyFlightService;  
    }  
    else {  
      return new FlightService(http);  
    }  
  },  
  [...]  
}]
```



useFactory

```
const DEBUG = true;
[...]  
  
providers: [{  
  provide: FlightService,  
  useFactory: (http: HttpClient) => {  
    if (DEBUG) {  
      return dummyFlightService;  
    }  
    else {  
      return new FlightService(http);  
    }  
  },  
  deps: [HttpClient]  
}]
```



DEMO

Types of Providers

useClass

useValue

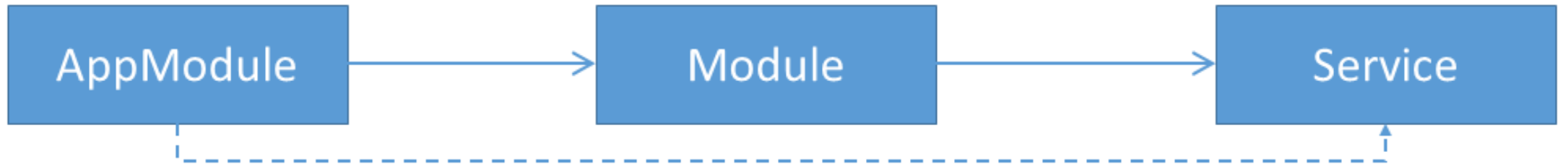
useFactory

useExisting

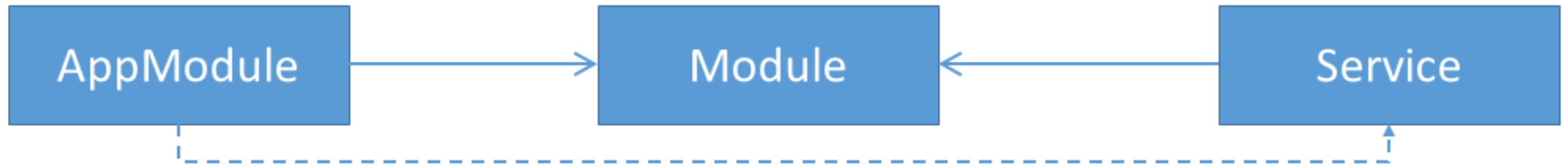


Tree-Shakable Provider

Traditional Providers are not Tree-shakable



Tree-shakable Providers



Example

Globaler Scope



```
@Injectable({ providedIn: 'root' })  
export class FlightService {  
  
    [...]  
  
}
```



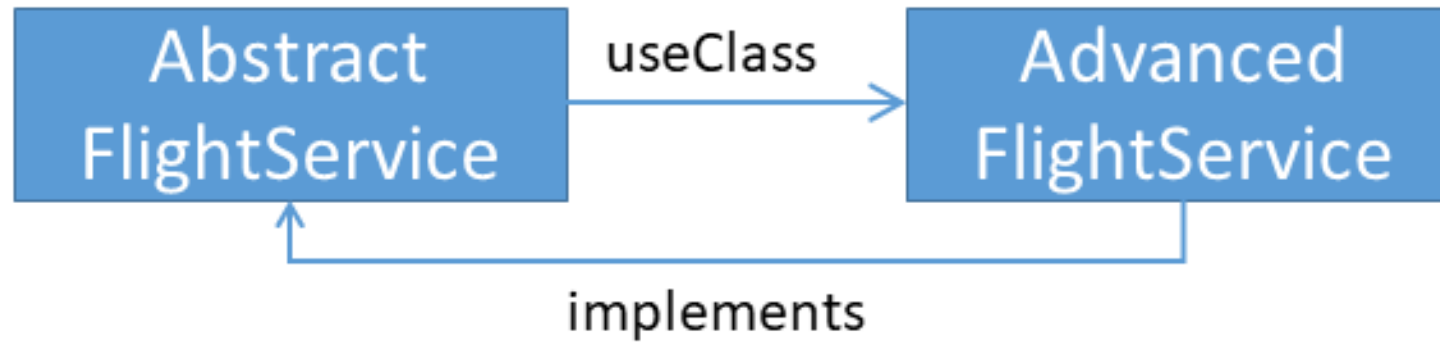
useClass

```
@Injectable({  
    providedIn: 'root',  
    useClass: DefaultFlightService  
})  
export abstract class FlightService {  
    abstract find(from: string, to: string);  
}
```

```
@Injectable()  
export class DefaultFlightService implements FlightService {  
  
    find(from: string, to: string) { ... }  
  
}
```



implements vs. extends



useFactory

```
@Injectable({
  providedIn: 'root',
  useFactory: (http: HttpClient) => {
    return new DefaultFlightService(http);
  },
  deps: [HttpClient]
})
export abstract class FlightService {

  abstract find(from: string, to: string): Observable<Flight[]>;

}
```

DEMO

Scopes

"Locale Service": Own Scope

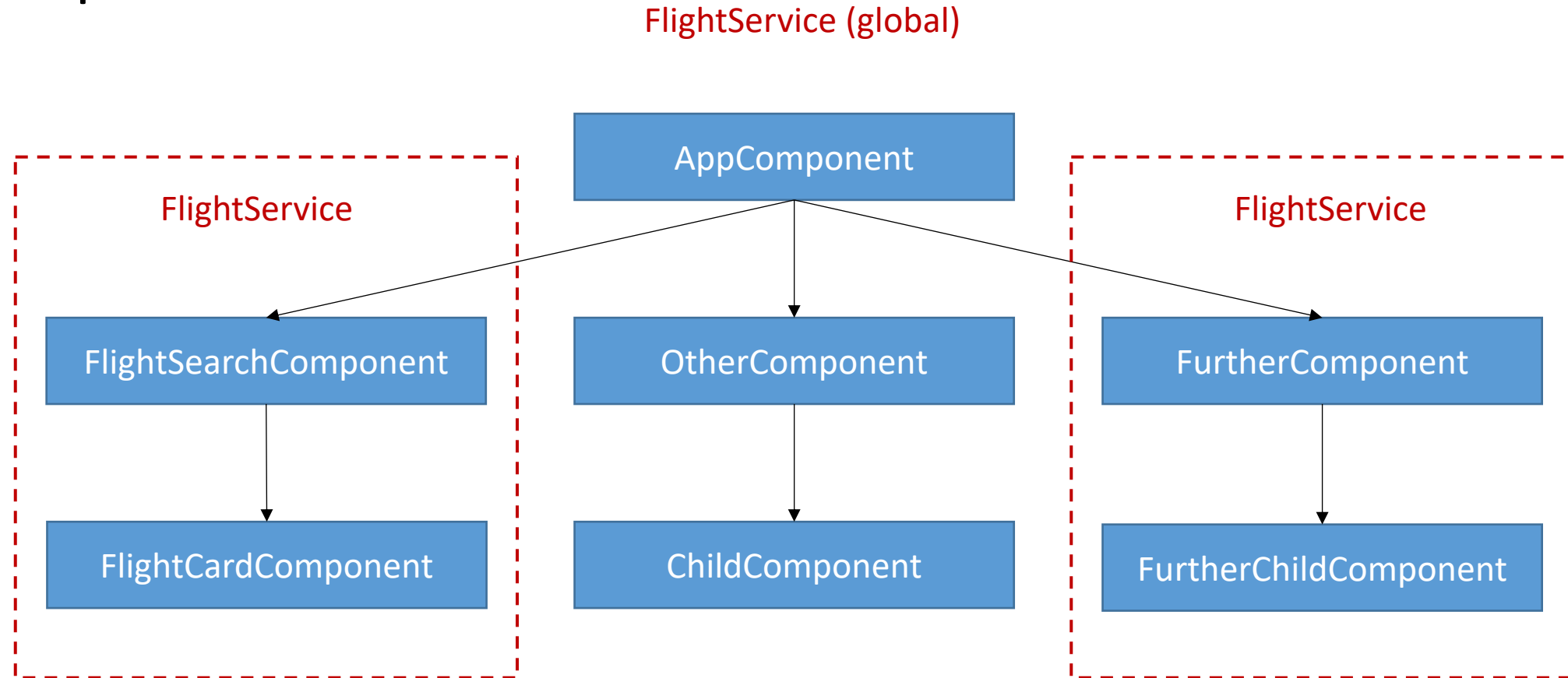
```
@Component({
  selector: 'flight-search',
  templateUrl: 'app/flight-buchen/flight-buchen.html',
  providers: [{ provide: FlightService, useClass: FlightService}]
})
export class FlightSearchComponent {

  [...]

}
```

↑
Can be used in current component and below!

Scopes



DEMO

Lazy Modules: Own Scope

```
@Injectable({ providedIn: LazyApiModule })  
export class FlightService {  
  
    [...]  
  
}
```

Constants as Tokens (Classic Providers)

Why Constants?

- There may not be a suitable type for a concept
 - Strings with configuration data
 - Tokens that refer to functions
 - Several "flavors" of one service
 - Token for Arrays
- Angular uses this option internally too



Classic Providers

```
import { InjectionToken } from "@angular/core";  
  
export const BASE_URL = new InjectionToken<string>("BASE_URL");
```

Classic Providers

```
@NgModule({  
  [...],  
  providers: [  
    [...]  
    { provide: BASE_URL, useValue: 'http://...' }  
  ]  
})  
  
export class AppModule {  
}
```


Injecting the Dependency

```
@Injectable()
export class FlightService {

    flights: Array<Flight> = [];

    constructor(
        @Inject(BASE_URL) private baseUrl: string,
        private http: Http
    ) {
        [...]
    }

    [...]
}
```

DEMO

Constants as Tokens (Tree-shakable Providers)



InjectionToken

```
export const BASE_URL =  
  new InjectionToken<string>('BASE_URL', {  
    providedIn: 'root',  
    factory: () => 'http://www.angular.at/api' } );
```

InjectionToken

```
export const FLIGHT_SERVICE =  
  new InjectionToken<FlightService>('FLIGHT_SERVICE', {  
    providedIn: 'root',  
    factory: () => new FlightService(inject(HttpClient)) } );
```

InjectionToken

```
export const FLIGHT_SERVICE =  
  new InjectionToken<FlightService>('FLIGHT_SERVICE', {  
    providedIn: 'root',  
    factory: () => new FlightService(inject(HttpClient)) } );
```

Multi Providers

Multi

```
[...]  
providers: [  
    { provide: FlightService, useClass: LufthansaFlightService, multi: true },  
    { provide: FlightService, useClass: AustrianFlightService, multi: true }  
]  
[...]
```

```
export class AppComponent {  
  
    constructor(  
        @Inject(FlightService) private flightServices: FlightService[]) {  
    }  
    [...]  
}
```


Multi and Constants

```
[...]  
providers: [  
    { provide: FLIGHT_SERVICES, useClass: LufthansaFlightService, multi: true },  
    { provide: FLIGHT_SERVICES, useClass: AustrianFlightService, multi: true }  
]  
[...]
```

```
export class AppComponent {  
  
    constructor(  
        @Inject(FLIGHT_SERVICES) private flightServices: FlightService[]) {  
    }  
    [...]  
}
```

LAB

Summary

- forRoot vs. in component
- Classic providers vs. tree-shakable providers
- Factories
- Constants
- Multi Providers

