devfest 2022

book-nav-toggle''
dden="" fixed="" aria-label="Hide
Hide side navigation''

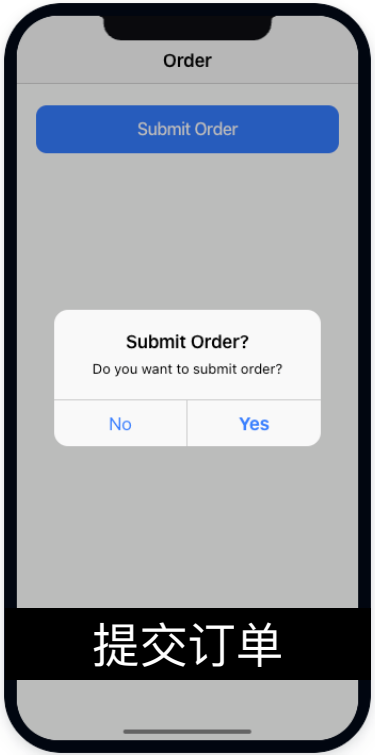
面向TC39标准的前端依赖注入

谢亚东
Google Developer Experts

Google Developer Groups
[Hangzhou]

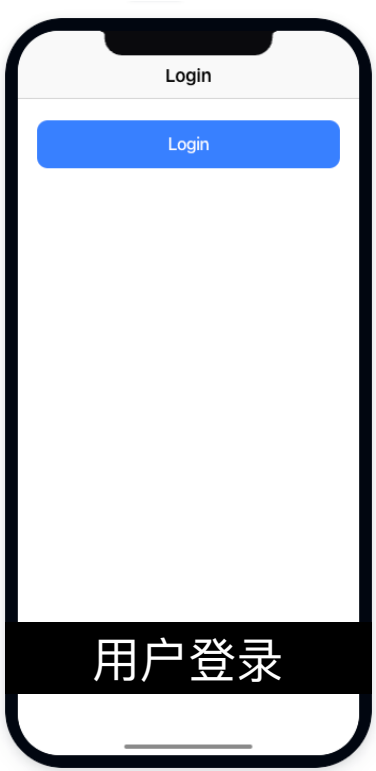
devfest
2022

业务场景

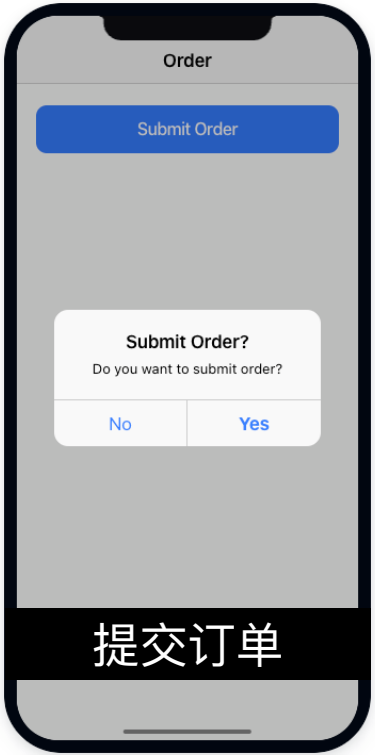


提交订单

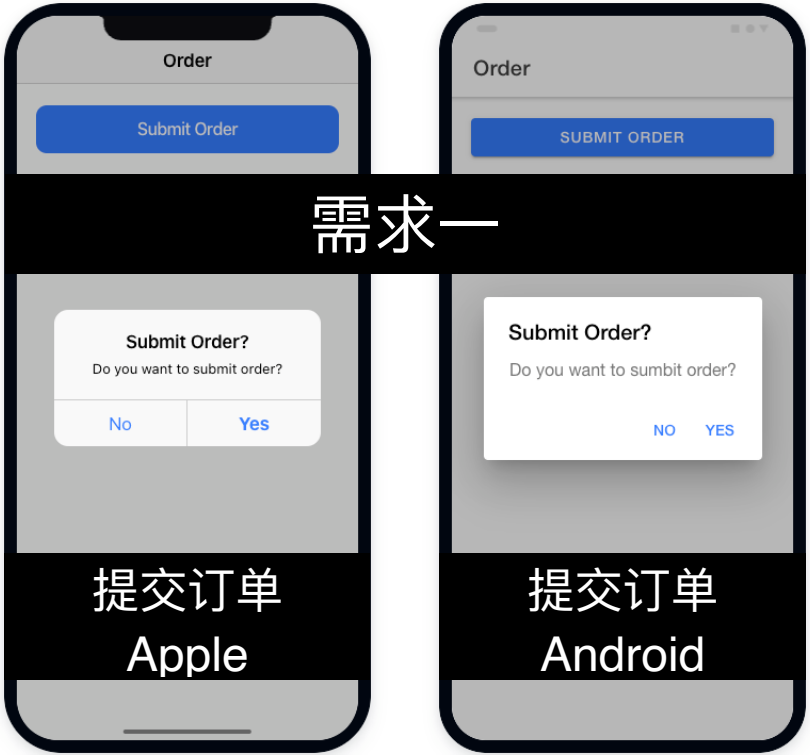
购物应用



用户登录




```tsx
export class AppleDialogService {
  confirm(message: string): void {
    alert(`Apple Design Spec Message: ${message}`);
  }
}

export class OrderService {
  private dialogService = new AppleDialogService();
  confirmOrder(message: string): void {
    this.dialogService.confirm(message);
  }
}

export const Order: FC = () => {
  const orderService = new OrderService();
  return (
    <button onClick={() => orderService.confirmOrder("Submit Order")}>
      Submit Order
    </button>
  );
};
```

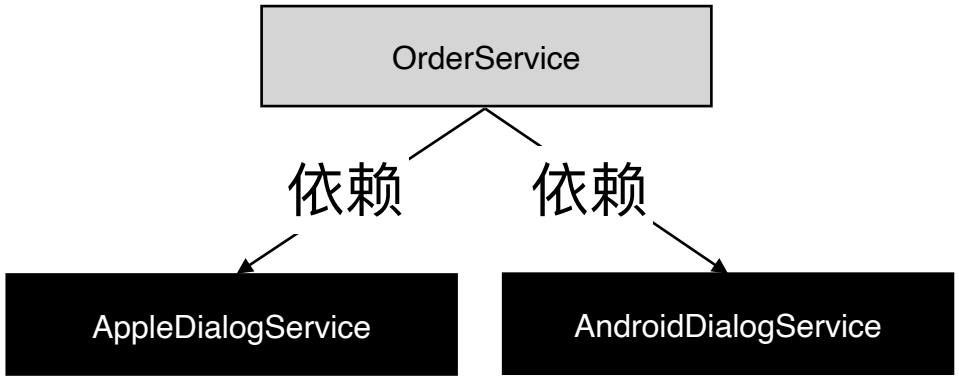


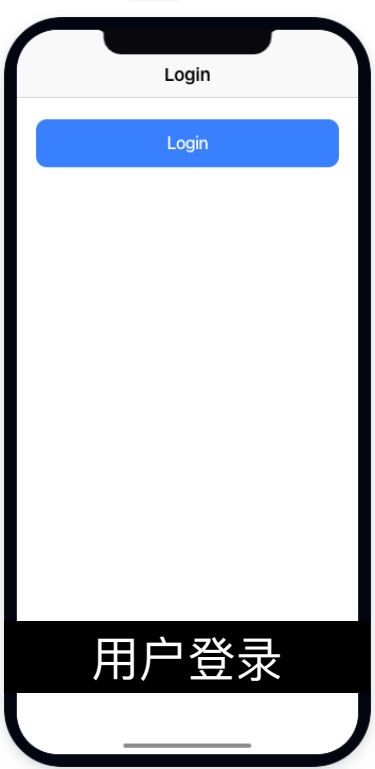
需求一

```tsx
export class AppleDialogService {
  confirm(message: string): void {
    alert(`Apple Design Spec Message: ${message}`);
  }
}

export class AndroidDialogService {
  confirm(message: string): void {
    alert(`Android Design Spec Message: ${message}`);
  }
}

export class OrderService {
  private appleDialogService = new AppleDialogService();
  private androidDialogService = new AndroidDialogService();
  private env = "android";
  confirmOrder(message: string): void {
    if (this.env === "apple") {
      this.appleDialogService.confirm(message);
    } else if (this.env === "android") {
      this.androidDialogService.confirm(message);
    }
  }
}

export const Order: FC = () => {
  const orderService = new OrderService();
  return (
    <button onClick={() => orderService.confirmOrder("Submit Order")}>
      Submit Order
    </button>
  );
};
```

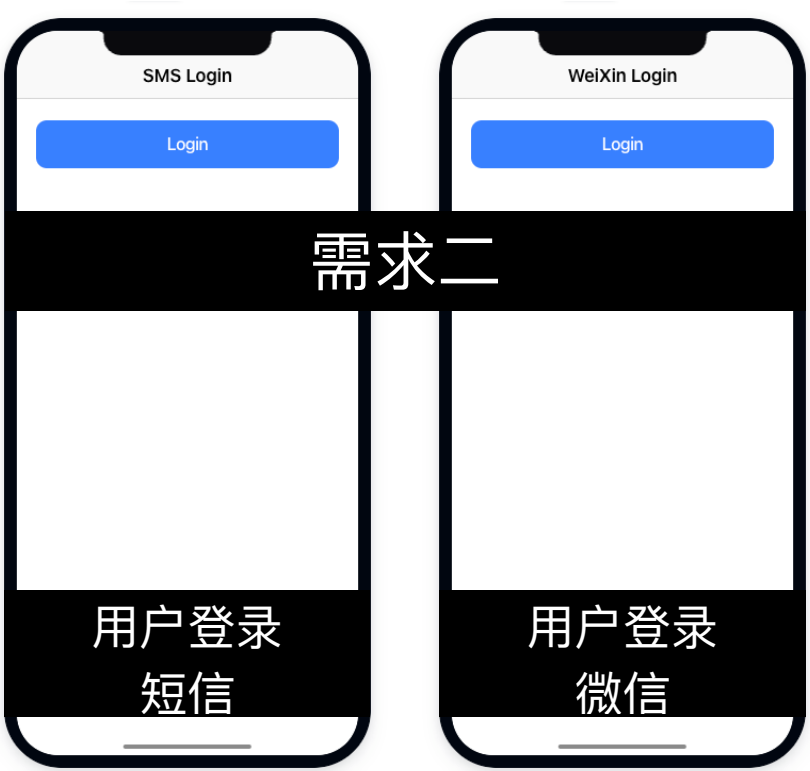
困境一 OrderService 依赖于具体的实现

```
export class OrderService {
  private appleDialogService = new AppleDialogService();
  private androidDialogService = new AndroidDialogService();
  private env = "android";
  confirmOrder(message: string): void {
    if (this.env === "apple") {
      this.appleDialogService.confirm(message);
    } else if (this.env === "android") {
      this.androidDialogService.confirm(message);
    }
  }
}
```






```ts
export class AuthService {
  name = "SMS";
  authenticated = true;
}

export class LoginService {
  private authService = new AuthService();
  login(): void {
    alert(
      `Login from ${this.authService.name} ${
        this.authService.authenticated ? "Success" : "Failed"
      }`
    );
  }
}

export const Login: FC = () => {
  const loginService = new LoginService();
  return <button onClick={() => loginService.login()}>Login</button>;
};
```

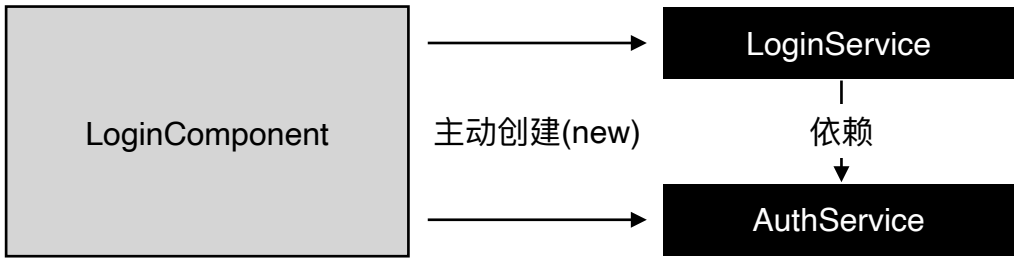



```ts
export abstract class AuthService {
  abstract authenticated: boolean;
  abstract name: string;
}
export class SMSAuthService implements AuthService {
  name = "SMS";
  authenticated = true;
}

export class WeiXinAuthService implements AuthService {
  name = "WeiXin";
  authenticated = false;
}

export class LoginService {
  constructor(private authServiceInterface: AuthService) {}
  login(): void {
    alert(
      `Login from ${this.authServiceInterface.name} ${
        this.authServiceInterface.authenticated ? "Success" : "Failed"
      }`
    );
  }
}

export const SMSLogin: FC = () => {
  const authService = new SMSAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};

export const WeiXinLogin: FC = () => {
  const authService = new WeiXinAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};
```

困境二

被依赖的模块需要手动创建

```
export const SMSLogin: FC = () => {
  const authService = new SMSAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};

export const WeiXinLogin: FC = () => {
  const authService = new WeiXinAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};
```

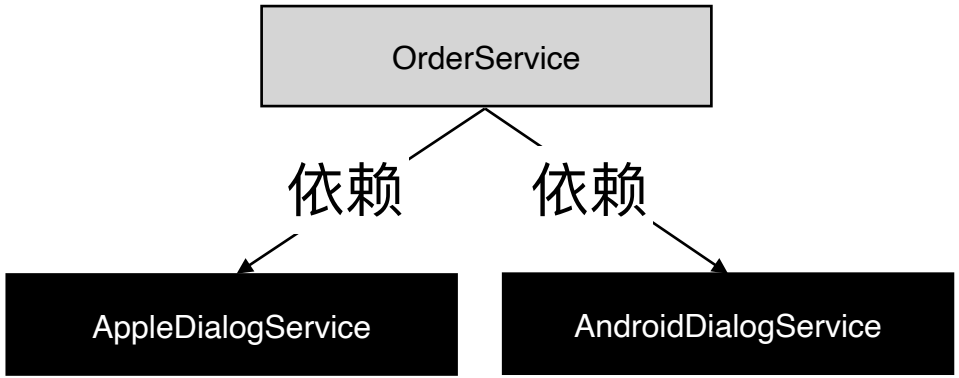


devfest
2022

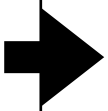
解决方案

困境一 OrderService 依赖于具体的实现

```
export class OrderService {
  private appleDialogService = new AppleDialogService();
  private androidDialogService = new AndroidDialogService();
  private env = "android";
  confirmOrder(message: string): void {
    if (this.env === "apple") {
      this.appleDialogService.confirm(message);
    } else if (this.env === "android") {
      this.androidDialogService.confirm(message);
    }
  }
}
```



依赖具体实现



依赖业务抽象

高阶模块

OrderService

依赖

AppleDialogService

具体实现

高阶模块

OrderService

依赖

DialogService

业务抽象

依赖

具体实现

AppleDialogService

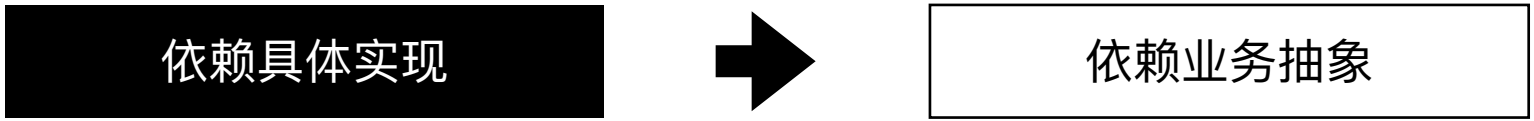
AndroidDialogService

设计原则一 依赖反转（DIP）

Dependency-Inversion Principle

解除高阶模块与低阶模块的耦合关系，使高阶模块不再依赖于低阶模块

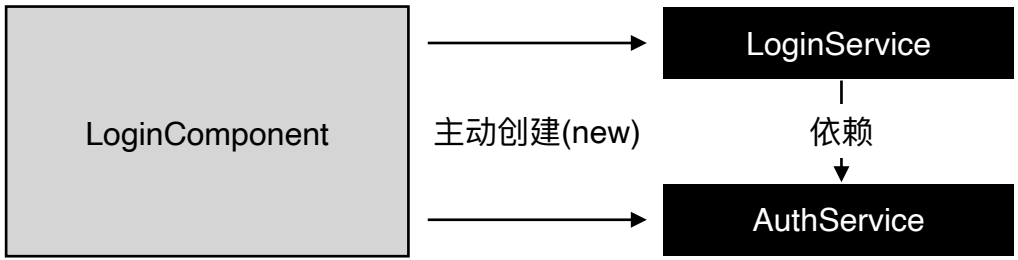
1. 高阶模块不应当依赖于低阶模块，两者都应当依赖于抽象
2. 抽象不应当依赖具体的实现方式
3. 实现方式应当依赖于抽象



困境二

被依赖的模块需要手动创建

```
export const SMSLogin: FC = () => {
  const authService = new SMSAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};

export const WeiXinLogin: FC = () => {
  const authService = new WeiXinAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};
```

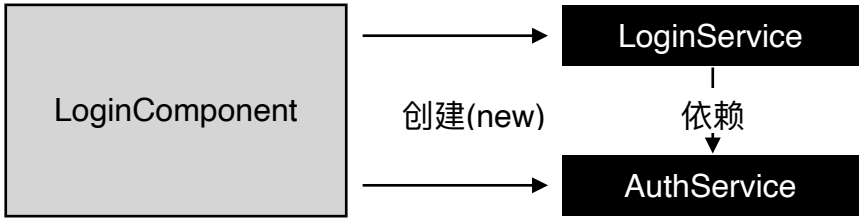


设计原则二　控制反转（IoC）

Inversion of Control

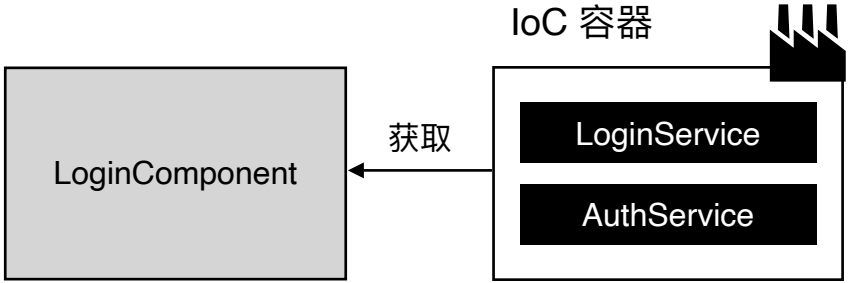
反转获得依赖对象的控制流程

主动创建




```
export const WeiXinLogin: FC = () => {
  const authService = new WeiXinAuthService();
  const loginService = new LoginService(authService);
  return <button onClick={() => loginService.login()}>Login</button>;
};
```

被动获取



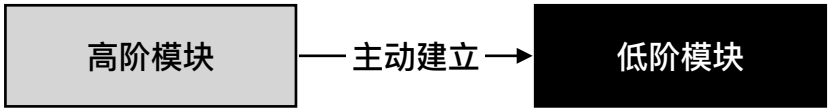

```
export const Login: FC = () => {
  const loginService = useInject(LoginService);
  return <button onClick={() => loginService.login()}>Login</button>;
});
```

设计模式 依赖注入（DI）

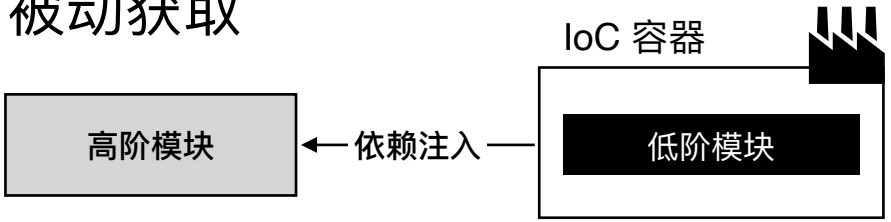
Dependency Injection

一种遵循控制反转原则的设计模式

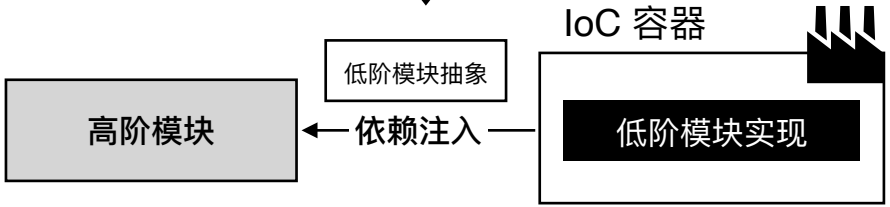
主动创建

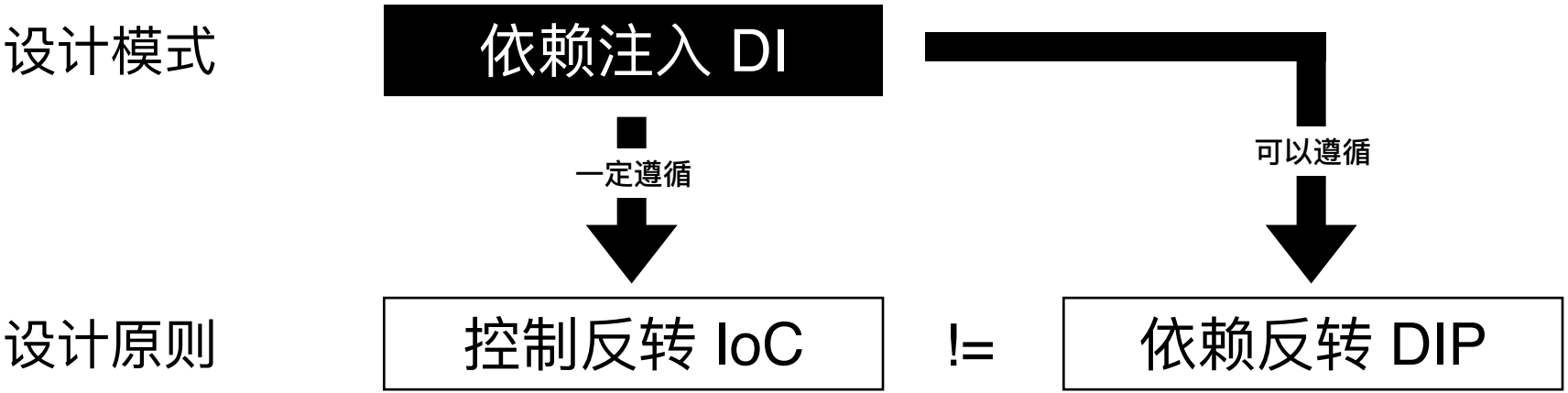


被动获取



依赖反转思想




设计模式
依赖注入 DI
一定遵循
可以遵循
设计原则
控制反转 IoC
!=
依赖反转 DIP

devfest
2022

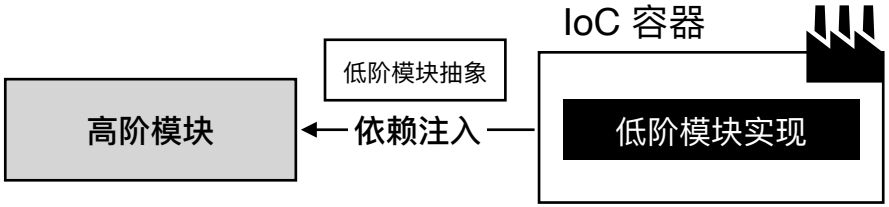
技术实现


Angular
Angular - Angular 中的依赖注入
angular.cn/guide/dependency-injection
无痕模式
ANGULAR
特性
文档
资源
活动
译者博客
关于中文版
搜索
介绍
快速上手
理解 Angular
概览
组件
模板
指令
依赖注入
Angular 依赖注入
DI 提供者
开发指南
最佳实践
Angular 工具
Angular 中的依赖注入
依赖项是指某个类执行其功能所需的服务或对象。依赖项注入（DI）是一种设计模式，在这种设计模式中，类会从外部源请求依赖项而不是创建它们。
Angular 的 DI 框架会在实例化某个类时为其提供依赖。可以使用 Angular DI 来提高应用程序的灵活性和模块化程度。
包含本指南中代码片段的可工作范例，参阅现场演练 / 下载范例。
创建可注入服务
要想在 src/app/heroes 目录下生成一个新的 HeroService 类，请使用下列 Angular CLI 命令。
Angular 中的依赖注入
创建可注入服务
注入服务
在其他服务中使用这些服务
下一步呢?

- https://github.com/mgechev/injection-js
- https://github.com/inversify/InversifyJS
- https://github.com/hullis/redi

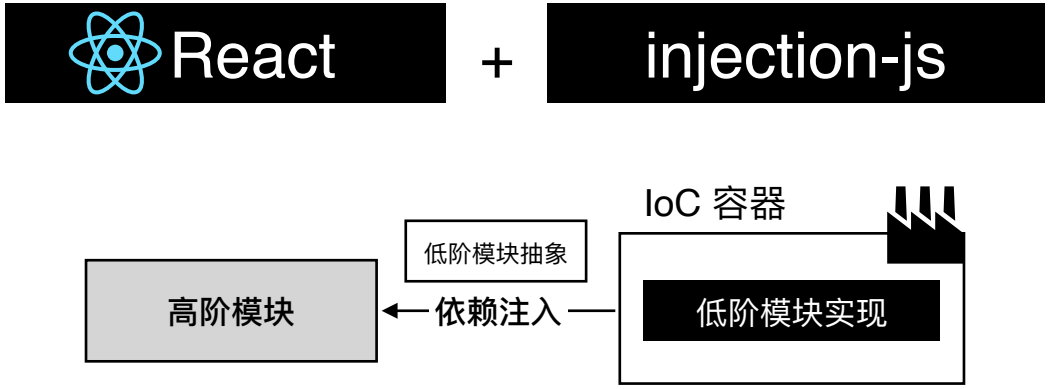
依赖注入（DI）

injection-js



- 声明 IoC 容器及内容：`ReflectiveInjector.resolveAndCreate`
- 模块间依赖关系：`constructor(private service: Service)`
- 手动从 IoC 容器中获得模块：`IoCContainer.get(Service)`

https://stackblitz.com/edit/injection-js-demo



- 声明 IoC 容器及内容：

```
<DIContainer providers={}></DIContainer>
```

- 模块间依赖关系：

```
constructor(private service: Service)
```

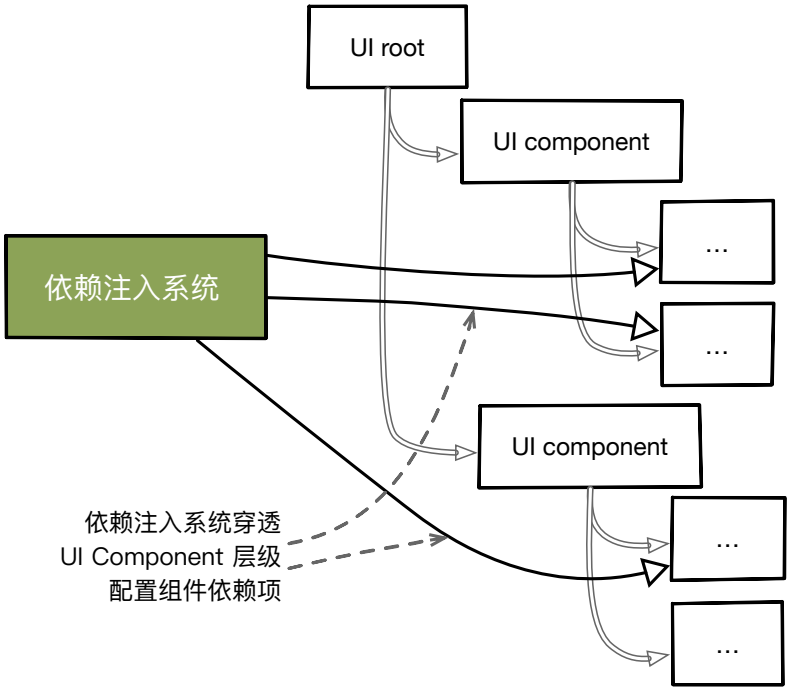
- 手动从 IoC 容器中获得模块：

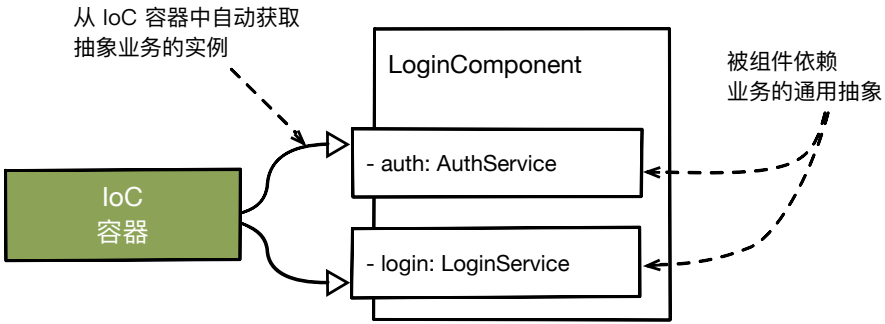
```
useInject(Service)
```

https://github.com/vthinkxie/react-di-demo

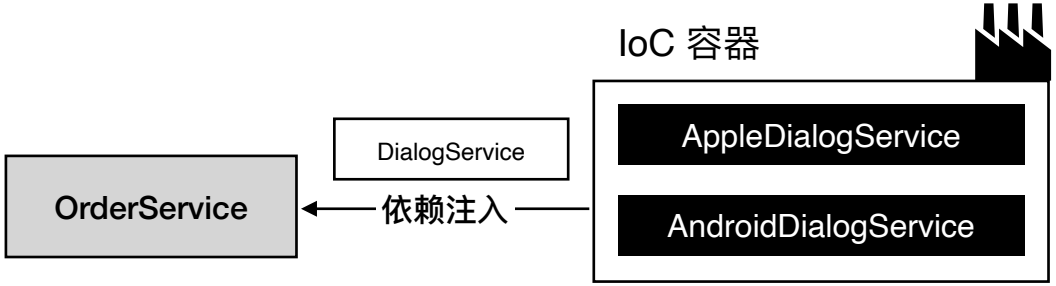
```
<DIContainer
  providers={[
    OrderService,
    LoginService,
    { provide: DialogService, useClass: AppleDialogService },
    { provide: AuthService, useClass: SMSAuthService },
  ]}
>
  <Login></Login>
  <DIContainer
    providers={[
      LoginService,
      { provide: AuthService, useClass: WeiXinAuthService },
    ]}
  >
    <Login></Login>
  </DIContainer>
  <Order></Order>
</DIContainer>
```

https://stackblitz.com/github/vthinkxie/react-di-demo

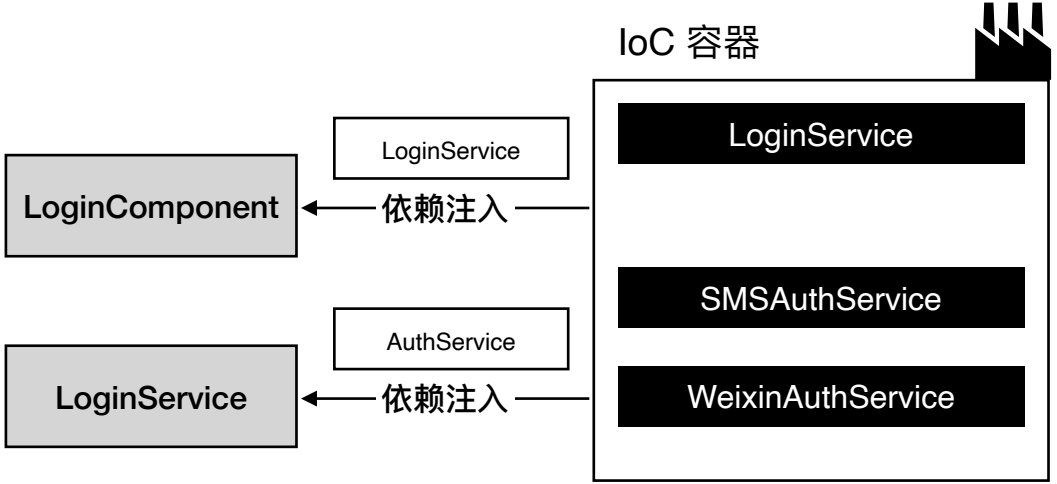

UI root
UI component
...
...
依赖注入系统
UI component
...
...
依赖注入系统穿透
UI Component 层级
配置组件依赖项


从 IoC 容器中自动获取
抽象业务的实例
LoginComponent
被组件依赖
业务的通用抽象
IoC
容器
- auth: AuthService
- login: LoginService

提交订单

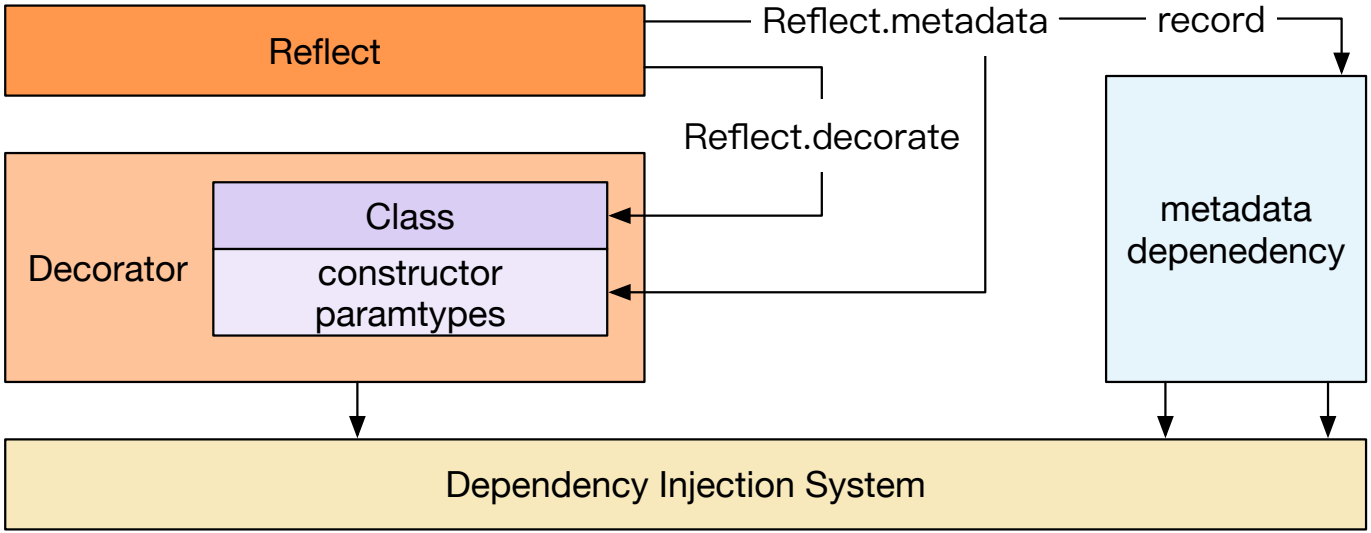

IoC 容器
AppleDialogService
AndroidDialogService
DialogService
依赖注入
OrderService

用户登录


IoC 容器
LoginService
SMSAuthService
WeixinAuthService
LoginService
依赖注入
LoginComponent
AuthService
依赖注入
LoginService

devfest
2022

方案优化



问题

- 依赖 polyfill ： 影响 bundle size
- 非纯函数：影响 tree-shaking 优化
- 非标准：TypeScript Decorator 标准即将 legacy

原理

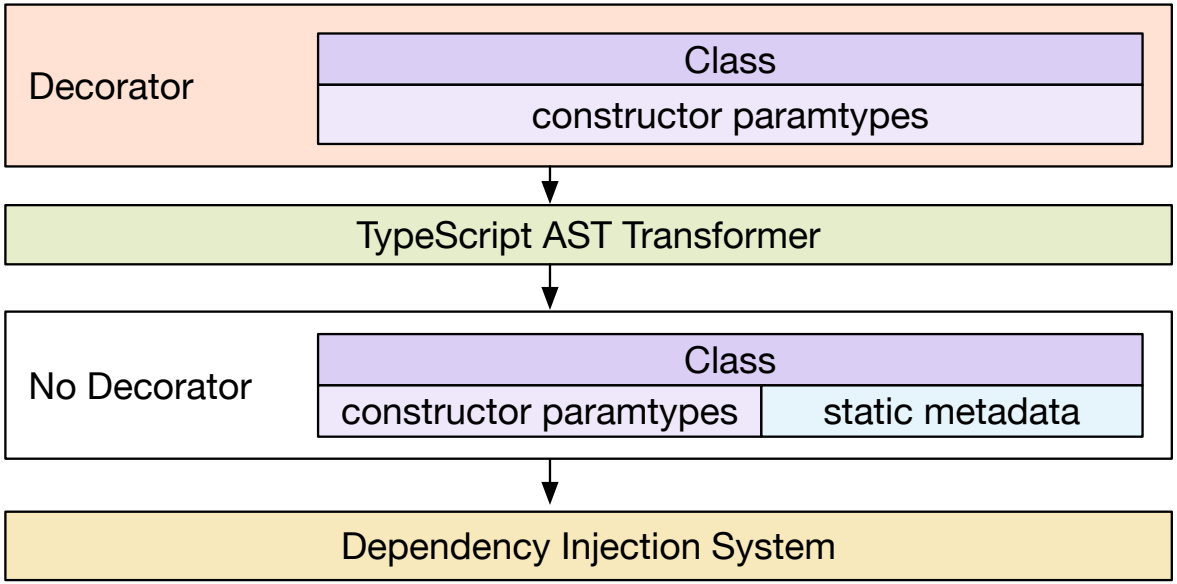
@Injectable
emitDecoratorMetadata
reflect-metadata

https://stackblitz.com/edit/injection-js-prototype
https://www.typescriptlang.org/play

标准化 + TreeShaking

@Injectable
emitDecoratorMetadata
reflect-metadata

https://github.com/tc39/proposal-decorators
https://github.com/evanw/esbuild/issues/257
https://github.com/microsoft/TypeScript/pull/48669/files



优势

- 不依赖 polyfill：减少 2MB reflect-metadata 依赖
- AOT 后代码为纯函数：支持 tree-shaking 优化
- 面向未来标准：不依赖 emitDecoratorMetadata

优化方案

static parameters TypeScript transformer

https://stackblitz.com/edit/injection-js-static-demo
https://astexplorer.net/
https://github.com/angular/angular/blob/main/packages/compiler-cli/src/ngtsc/tsc_plugin.ts

优化方案

injection-js-transformer
use-inject