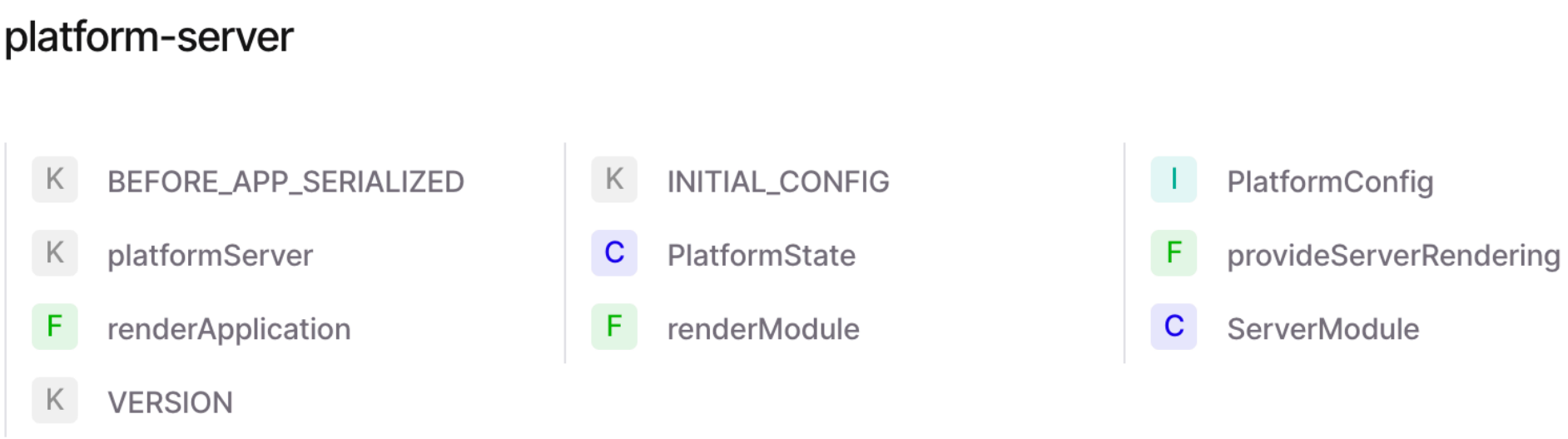
platform-server 是什么

是 Angular 中的一个模块，它提供了服务器端渲染（Server-Side Rendering，SSR）的支持。服务器端渲染是一种技术，用于在服务器上动态生成 HTML 页面，并将其发送给客户端，以便快速加载和显示内容，同时有助于搜索引擎优化（SEO）和提高性能。

都有些什么



PlatformState
ServerModule
renderApplication
renderModule
provideServerRendering

ServerModule

将一些 angular 在 浏览器要到的东西重新引入到一个新的 Class 里并在后端应用这个 Class 就可调用在浏览器渲染页面的方法

```
import {BrowserModule} from '@angular/platform-browser';
import {NoopAnimationsModule} from '@angular/platform-browser/animations';
import {HttpClientModule} from '@angular/common/http';

@NgModule({
  exports: [BrowserModule],
  imports: [HttpClientModule, NoopAnimationsModule],
  providers: PLATFORM_SERVER_PROVIDERS,
})
export class ServerModule {}
```

PlatformState

```
export class PlatformState {
  constructor(@Inject(DOCUMENT) private _doc: any) {}

  /**
   * Renders the current state of the platform to string.
   */
  renderToString(): string {
    return serializeDocument(this._doc);
  }

  /**
   * Returns the current DOM state.
   */
  getDocument(): any {
    return this._doc;
  }
}
```

renderModule

使用模块化的 angular 应用实例，并将页面内容序列化为字符串

```
// NgModule 使用 renderModule


// app.server.module.ts
// 创建一个 app.server.module.ts  将AppModule 和 platform-server 的 ServerModule 都引入进来
import { NgModule } from '@angular/core';
import { ServerModule } from '@angular/platform-server';

import { AppModule } from './app.module';
import { AppComponent } from './app.component';

@NgModule({
  imports: [
    AppModule,
    ServerModule,
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule {}


// 在 server.ts 中添加 使用 renderModule 渲染页面并返回给前端
    import { AppServerModule } from './src/main.server';
    import { renderModule } from '@angular/platform-server';

    // 读取打包好的 dist 中 静态文件
    const FOLDER = join(process.cwd(), 'dist/v18-ssr-demo/browser');
    server.use(express.static(FOLDER));
    // 读取 Angular 应用的 index.html 文件
    const renderHtml = readFileSync(join(FOLDER, 'index.html'), 'utf8');
    server.get('/renderModule', async (req, res) => {
        // const html = '<div>ppp</div>'
        const renderedHtml = await renderModule(AppServerModule, {
            document: renderHtml,
            url: req.url
        });
        res.status(200).send(renderedHtml);
    });


export async function renderModule<T>(
  moduleType: Type<T>,
  options: {document?: string | Document; url?: string; extraProviders?: StaticProvider[]},
): Promise<string> {
  const {document, url, extraProviders: platformProviders} = options;
  const platformRef = createServerPlatform({document, url, platformProviders});
  const moduleRef = await platformRef.bootstrapModule(moduleType);
  const applicationRef = moduleRef.injector.get(ApplicationRef);
  return _render(platformRef, applicationRef);// html
}
```

renderApplication

引导 angular 程序实例并将页面内容序列化为字符串

```
// standalone  使用  renderApplication

// app.config.ts
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideClientHydration } from '@angular/platform-browser';

export const appConfig: ApplicationConfig = {
  providers: [provideZoneChangeDetection({ eventCoalescing: true }), provideRouter(routes), provideClientHydration()]
};



//app.server.config.ts

import { mergeApplicationConfig, ApplicationConfig } from '@angular/core';
import { provideServerRendering } from '@angular/platform-server';
import { appConfig } from './app.config';

const serverConfig: ApplicationConfig = {
  providers: [
    // provideServerRendering()
  ]
};

export const config = mergeApplicationConfig(appConfig, serverConfig);



// app.server.ts
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';
import { config } from './app/app.config.server';

const bootstrap = () => bootstrapApplication(AppComponent, config);

export default bootstrap;


// server.ts
import bootstrap from './src/main.server';

server.get('/renderApplication', async (req, res) => {
    try {
        const renderedHtml = await renderApplication(bootstrap,{
            document: renderHtml,
            url: req.url
          });
          res.status(200).send(renderedHtml);
    } catch (error) {
        console.error('Server-side rendering error:', error);
        res.status(500).send('Internal Server Error');
    }
});


export async function renderApplication<T>(
  bootstrap: () => Promise<ApplicationRef>,
  options: {document?: string | Document; url?: string; platformProviders?: Provider[]},
): Promise<string> {
  const platformRef = createServerPlatform(options);
  const applicationRef = await bootstrap();
  return _render(platformRef, applicationRef);
}
```

provideServerRendering

设置必要的 providers 程序以启用应用程序的服务器渲染功能。

```
// standalone  中使用 provideServerRendering


import { mergeApplicationConfig, ApplicationConfig } from '@angular/core';
import { provideServerRendering } from '@angular/platform-server';
import { appConfig } from './app.config';

const serverConfig: ApplicationConfig = {
  providers: [
    provideServerRendering()
  ]
};

export const config = mergeApplicationConfig(appConfig, serverConfig);



// angular/core
provideServerRendering(): EnvironmentProviders {
  return makeEnvironmentProviders([provideNoopAnimations(), ...PLATFORM_SERVER_PROVIDERS]);
}
```