

Routing

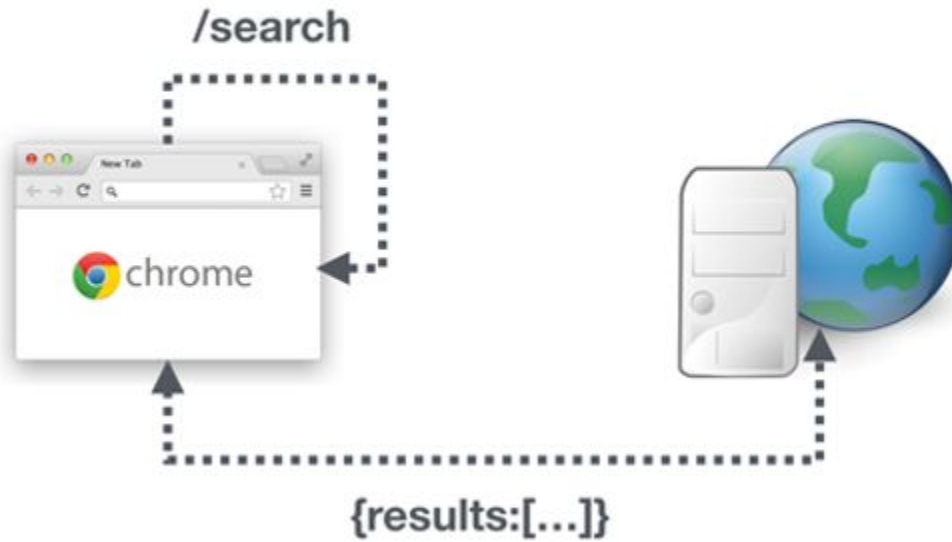
Routing

tradizionale



Routing

SPA



Routing

I vantaggi di una SPA sono:

1. **Può essere più veloce** Invece di fare una richiesta dispendiosa in termini di tempo a un server remoto ogni volta che l' URL cambia, l'app client aggiorna la pagina molto più velocemente.
2. **Meno larghezza di banda richiesta.** Non inviamo una grande pagina html per ogni modifica dell'URL , ma potremmo semplicemente chiamare un'API più piccola che restituisce *solo i dati sufficienti* a rendere la modifica nella pagina.
3. **Convenienza.** Ora un singolo sviluppatore può sviluppare la maggior parte delle funzionalità di un sito invece di suddividere lo sforzo tra uno sviluppatore front-end e uno lato server.

Routing

redirect

```
const routes:Routes = [  
  {path: '', redirectTo: 'home', pathMatch: 'full'}, (1)  
  {path: 'find', redirectTo: 'search'}, (1)  
  {path: 'home', component: HomeComponent},  
  {path: 'search', component: SearchComponent}  
];
```

Routing

utilizzando il percorso `**`, se l' URL non corrisponde a *nessuno* degli altri percorsi che abbinerà questa rotta.




```
const routes:Routes = [  
  {path: '', redirectTo: 'home', pathMatch: 'full'},  
  {path: 'find', redirectTo: 'search'},  
  {path: 'home', component: HomeComponent},  
  {path: 'search', component: SearchComponent},  
  {path: '**', component: HomeComponent} (1)  
];
```

Routing

```
pathMatch: 'full'
```

in modo che Angular sappia che deve corrispondere esattamente alla stringa vuota
e non *parzialmente* alla stringa vuota.

Routing

il percorso , se l' URL non corrisponde a *nessuno* degli altri percorsi che abbinerà questa rotta.



Routing



```
//...  
constructor(private router: Router) {} (2)  
  
goHome() {  
    this.router.navigate(['']); (3)  
}  
  
goSearch() {  
    this.router.navigate(['search']); (3)  
}  
//...
```

Routing



```
let part = "foo";  
this.router.navigate(['search', part, 'moo']);  
...  
//oppure senza parametri  
this.router.navigate(['search', 'foo', 'moo']);
```



```
<nav class="navbar navbar-light bg-faded">
  <a class="navbar-brand"
    [routerLink]="['home']">iTunes Search App
</a>
<ul class="nav navbar-nav">
  <li class="nav-item"
    [routerLinkActive]="['active']">
    <a class="nav-link"
      [routerLink]="['home']">Home
    </a>
  </li>
  <li class="nav-item"
    [routerLinkActive]="['active']">
    <a class="nav-link"
      [routerLink]="['search']">Search
    </a>
  </li>
</ul>
</nav>
```

Routing

RouterLinkActive

Inserisce le
classi tra []
quando
attiviamo la
rotta

Si può inserire
anche in tag
contenitori



Routing

```
const routes: Routes = [  
  {path: '', redirectTo: 'home', pathMatch: 'full'},  
  {path: 'find', redirectTo: 'search'},  
  {path: 'home', component: HomeComponent},  
  {path: 'search', component: SearchComponent},  
  {  
    path: 'artist/:artistId',  
    component: ArtistComponent,  
    children: [  
      {path: '', redirectTo: 'tracks'}, (1)  
      {path: 'tracks', component: ArtistTrackListComponent}, (2)  
      {path: 'albums', component: ArtistAlbumListComponent}, (3)  
    ]  
  },  
  {path: '**', component: HomeComponent}  
];
```

Routing

Guard

CanActivate

Controlla se un utente può visitare un percorso.

CanActivateChild

Controlla se un utente può visitare i percorsi Child.

CanDeactivate

Controlla se un utente può uscire da una rotta.

Resolve

Esegue il recupero dei dati del percorso prima dell'attivazione del percorso.

canLoad

Controlla se un utente può indirizzare un modulo lazy.