GraphQL

تیم CTO/سید علی معصوم زاده

1399/06/15-1399/07/19

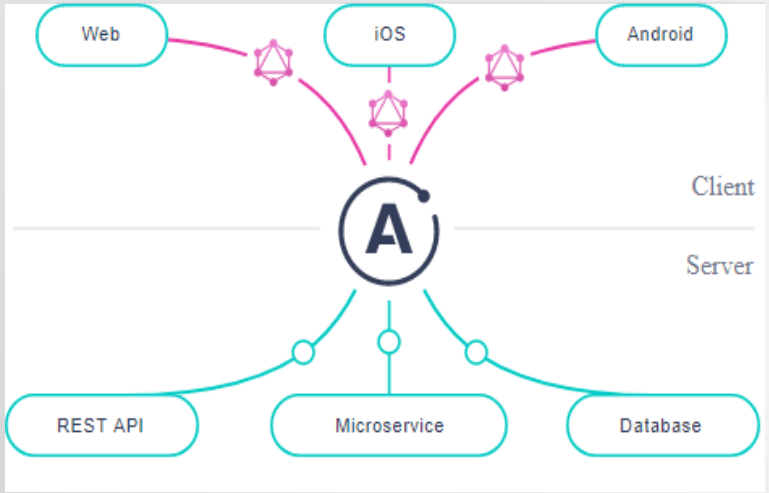
برای پیاده سازی GraphQL در سمت کلاینت از کتابخانه های زیر استفاده می شود

- Apollo-Client : جمعی از برنامه نویس ها ارائه دادن و انعطاف پذیر و آسان
- Relay : کتابخانه ای که فیسبوک ارائه داده و دارای ساختار و ستخگیرانه .

- GraphQL-Request
- GraphQL- CLI
- GraphQL-Compose
- @octokit

Apollo Client

یک کتابخانه open-source با GraphQL سازگار است که باعث می شود GraphQL کلاینت را به سرور متصل نماید .



ویژگی ها :

واکشی داده های اعلامی : یک queryبنویسید و داده ها را دریافت کنید .

تجربه عالی توسعه دهنده : از ابزار های مفید chrome DevToolsو TypeScriptو VS Code لذت ببرید .

به طور فزاینده ای قابل قبول : میتوانید در هر برنامه javaScript از Apollo استفاده کنید .

مبتنی بر انجمن : به لطف انجمن بصورت open-sourceمی باشد .

کتابخانه **Apollo** با زبان ها و فریم ورک های زیر سازگاری دارد

JavaScript

Angular

Vue

Ember

React

Mobile

Native IOS

Native Android

پیاده سازی

در ابتدا شما باید پکیج های زیر را در npm یا yarn نصب کنید :

```
npm install apollo-boost apollo-client apollo-link-ws apollo-link-http apollo-link apollo-utilities
apollo-cache-inmemory react-apollo graphql
```

```
yarn add apollo-boost apollo-client apollo-link-ws apollo-link-http apollo-link apollo-utilities
apollo-cache-inmemory react-apollo graphql
```

- از Apollo-boost استفاده نکردیم به این دلیل که subscriptions را پشتیبانی نمی کند .

سپس در کامپونت App.js کتابخانه ApolloClient را ایمپورت میکنید :

```
Import ApolloClient from 'apollo-boost'
```

ApolloClient یک client می سازه که دسترسی داشته باشه به سرور .

بعد clinet خود را می سازید :

```
const client = new ApolloClient ({
  uri: 'http://localhost:8082/graphql'
});
```

سپس در کامپونت App.js کتابخانه ApolloClient را ایمپورت میکنید :

```
import { WebSocketLink } from 'apollo-link-ws';
import { HttpLink } from 'apollo-link-http';
import { split } from 'apollo-link';
import { getMainDefinition } from 'apollo-utilities';
import ApolloClient from "apollo-client";
import { InMemoryCache } from 'apollo-cache-inmemory';
import { ApolloProvider } from 'react-apollo';
```

WebSocketLink : با آن عملیات GraphQL را انجام دهید و از Subscriptions ها پشبانی میکند .

- uri : یک رشته برای نقطه پایان اتصال .
- options : مجموعه ای از گزینه ها برای انتقال .
- webSocketImpl : برای پیاده سازی سفارشی webSocket .

```
19 const wsLink = new WebSocketLink({
20     uri: `ws://localhost:8082/graphql`,
21     options: {
22         reconnect: true
23     }
24 });
```

HttpLink : با آن عملیات GraphQL را انجام دهید . (query ,mutation)

- uri : یک رشته برای نقطه پایان اتصال .
- includeExtensions : اجازه میدهد فیلد های پسوند به سرور ارسال شود پیش فرض false می باشد .
- fetch : واکشی APIسازگار برای درخواست
- headers : عنوان درخواست
- credentials : رشته ای برای نشان دادن خط مشی اعتبار برای ارتباط های واکشی که شامل : omit, include and same-origin
- fetchOptions : اضافه کردن آرگومان های جدید به ارتباط واکشی.
- useGETForQueries : برای استفاده از روش get برای query ها استفاده شود . برای mutation ها استفاده نمی شود .

```
26 const httpLink = new HttpLink({
27    uri: `http://localhost:8082/graphql`,
28 });
```

split : با آن می توان جریان کنترل مبتی بر عملیات رو انجام داد .

پیوند بین (query Subscriptions, mutation)

```
30 const splitLink =split(
31    ({ query }) => {
32       const { kind, operation } = getMainDefinition(query);
33       return kind === 'OperationDefinition' && operation === 'subscription';
34    },
35    wsLink,
36    httpLink,
37 );
```

getMainDefinition : گرفتن اطلاعات از کوئری مانند عملیات و نوع و ...

ApolloClient : یک client می سازه که دسترسی داشته باشه به سرور.

InMemoryCache : برای کش کردن اطلاعات از ApolloClient .

```
const client = new ApolloClient({
    link:splitLink,
    cache: new InMemoryCache()
});

class App extends Component {
    render() {
        return (
            <ApolloProvider client={client } >
                <Router>
                <div>
                    <Switch>
                        <Route path="/adminDelete/:Id">
                            <AdminDelete />
                        </Route>
                        <Route path="/adminUpdate/:Id">
                            <AdminUpdate />
                        </Route>
                        <Route path="/adminCreate">
                            <AdminCreate />
                        </Route>
                        <Route path="/">
                            <AdminList />
                        </Route>
                    </Switch>
                </div>
                </Router>
            </ApolloProvider>
        );
    }
}
```

```
import {useMutation, useQuery} from '@apollo/react-hooks';
```

```
import {ADMIN_USER_READ_GRID_BY_ID, ADMIN_USER_UPDATE_MUTATION} from "./AdminQueries";
```

برای استفاده از query از ساختار زیر استفاده می کنید :

```
let rowNewId = window.location.pathname.split("/")[2];
const {loading, error, data} = useQuery(ADMIN_USER_READ_GRID_BY_ID,{
    variables:{id:rowNewId}
});
```

برای استفاده از mutation از ساختار زیر استفاده می کنید :

```
//تعریف متغیر ویرایش کننده
const [adminUpdate] = useMutation(ADMIN_USER_UPDATE_MUTATION);

//متد ویرایش کننده اطلاعات طبق داده فرم
const submitUpdate = (event) => {
    adminUpdate({variables: formData})
        .then(({data}) => {
            setFormResult({...formResult,"data":data});
        })
        .catch(error => {
            setFormResult({...formResult,"error":error});
        });
    setFormData(formData);
};
```

ساختار query و mutation :

gql : برای نوشتن کوئری ها در جاوااسکریپت.

```
import gql from "graphql-tag";
```

```
const ADMIN_USER_READ_GRID_BY_ID = gql `
query  common_adminUser_readById($id:Int!){
   common_adminUser_readById(id: $id) {
      username,
      firstName,
      lastName,
      gender_id,
      defaultAdminUserContact_city_id,
      defaultAdminUserContact_address,
      id
   }
}
`;
```

```
const ADMIN_USER_UPDATE_MUTATION = gql `
   mutation Update(
   $username:String!,
   $firstName:String,
   $lastName:String,
   $defaultAdminUserContact_address:String,
   $skillList:[AdminUserSkillModelInput],
   $gender_id:Int!,
   $defaultAdminUserContact_city_id:Int!,
   $id:Int!
   ) {
  update(adminUserModel:{
  username:$username,
  firstName:$firstName,
  lastName:$lastName,
  defaultAdminUserContact_address:$defaultAdminUserContact_address,
  skillList:$skillList,
   defaultAdminUserContact_city_id:$defaultAdminUserContact_city_id,
  gender_id:$gender_id,
  id:$id
  }) {
   id,
   username,
   firstName,
   lastName,
   gender_id,
  }
}
`;
```

```
import {useSubscription} from '@apollo/react-hooks';
```

برای استفاده از Subscriptions از ساختار زیر استفاده می کنید :

```
14  const [notifyCounter, setNotifyCounter] = React.useState(0);
15
16  useSubscription(
17     BRIEF_MESSEAGES_SUBSCRIPTION,
18     {
19        variables: {"userId": 2},
20        onSubscriptionData: ({ subscriptionData: { data } }) => {
21           setNotifyCounter(data.briefMesseages.notifyCounter);
22        }
23     }
24  );
```

ساختار Subscriptions :

```
4   const BRIEF_MESSEAGES_SUBSCRIPTION = gql`
5    subscription BriefMesseages($userId: Int!) {
6     briefMesseages(userId: $userId) {
7      notifyCounter
8     }
9    }
10  `;
```

بررسی خطا ها :

- **GraphQL Errors** : خطا های خود GraphQL که با آنها مواجه می شوید .
- **Server Errors** : خطا هایی که از سمت سرور با آنها مواجه می شوید .
- **Transaction Errors** : خطا هایی که در انتقال درخواست در mutation اتفاق میافتد .
- **UI Errors** : خطا هایی که در خود کامپونت ها اتفاق می افتد .
- **Apollo Client Errors** : خطا هایی که داخل خود سرور کتابخانه مورد نظر اتفاق می افتد .

مواردی که در پاسخ درخواست های GraphQL از سرور به کلاینت که از کتابخانه Apollo استفاده شده است به صورت زیر است :

- **operation** : عملیاتی که خطا داده است.
- **response** : پاسخ از سمت سرور .
- **graphQLErrors** : آرایه ای از خطا های نقط پایان درخواست.
- **networkError** : هر گونه خطا ارتباط با سرور.

بـررسی خطا ها :

```
function ResultHandling(props) {

    let message = "";
    if (props.result.error === "") {
        if (props.result.data !== "") {
            messageStyle = classes.modalBodySuccess;
        }
    } else {
        let dataError = props.result;
        if (props.result.error !== undefined) {
            dataError = props.result.error;
        }
        if (dataError.graphQLErrors !== undefined && dataError.graphQLErrors !== null) {
            message = `پیام خطا : ${dataError.graphQLErrors[0].message}`;
        }
        if (dataError.networkError !== undefined && dataError.networkError !== null) {
            message = `پیام خطا : ${dataError.networkError}`;
        }
    }

    let body = (<div className={messageStyle}>
        <CloseIcon onClick={handleClose} className={classes.closeButton}/>
        <h3 id="simple-modal-title">{titleModal}</h3>
        <p id="simple-modal-description">
            {message}
        </p>
    </div>);

    return (
        <div>
            <Modal
              className={classes.modal}
               open={open}
               onClose={handleClose}
               aria-labelledby="simple-modal-title"
              aria-describedby="simple-modal-description"
            >
                {body}
            </Modal>
        </div>
    );

}
```

رفرنس ها :

- نشانی وب سایت:

https://graphql.org/

https://www.apollographql.com/

https://hasura.io/blog/moving-from-apollo-boost-to-graphql-subscriptions-with-apollo-client-cc0373e0adbo/

yarn add --save apollo-client apollo-link-ws apollo-link-http apollo-link apollo-utilities apollo-cache-inmemory

- نشانی بخش آموزش:

https://graphql.org/learn/

https://www.apollographql.com/docs

- زبان هایی که پشتیبانی می کنند :

https://graphql.org/code/