tRPCの紹介

Press Space for next page →

自己紹介

- 📝 飯野陽平（wheatandcat）
- 🏢 法人設立（合同会社UNICORN 代表社員）
- 💻 Work: シェアフル株式会社CTO
- 📚 Blog: https://www.wheatandcat.me/
- 🛠️ 今までに作ったもの
  - memoir
  - ペペロミア
  - MarkyLinky
  - Atomic Design Check List

tRPCとは？

- スキーマやコード生成なしで型安全なAPIの連携が行えるPRCフレームワーク
- 比較対象としては、GraphQLやOpenAPIなどの技術に該当
- TypeScriptに特化しており、非常にシンプルに実装できる
- BFFとして利用されることも多い

そもそもRPCとは？

- RPC (Remote Procedure Call) とは、ネットワークで繋がっている別のコンピュータのプログラムを実行できるようにする技術
- クライアント側からサーバー側への疎通を行う際に、具体的な通信手段やプロトコルを意識することなく、関数の呼び出しを行うことがでやり取りが行える仕組み
- Googleが開発したRPCのプロトコルなので、**gRPC**という命名になっている
- **tRPC**のtはTypeScriptを表している

学ぶモチベーション

- OpenAPIやGraphQLだとスキーマ定義が必須でコード以外の実装が入りファットになりがち
  - またフロントエンド側を型安全にするためには、さらに別途でコード生成の対応が必要
- 気軽にかつ型安全を保ちながらAPIを実装したい
- なので、tRPCを使ってみた

サーバー側の実装

```
import { createHTTPServer } from "@trpc/server/adapters/standalone";
import { initTRPC } from "@trpc/server"

const t = initTRPC.create();

const appRouter = t.router({
  helloWorld: t.procedure.query(() => {
    return "Hello World";
  }),
});

export type AppRouter = typeof appRouter;

const server = createHTTPServer({
  router: appRouter,
});

server.listen(8000);
```

- 参考:quickstart

サーバー側の実装

```
import { createHTTPServer } from "@trpc/server/adapters/standalone";
import { initTRPC } from "@trpc/server"

const t = initTRPC.create();

const appRouter = t.router({
  helloWorld: t.procedure.query(() => {
    return "Hello World";
  }),
});

export type AppRouter = typeof appRouter;

const server = createHTTPServer({
  router: appRouter,
});

server.listen(8000);
```

- 参考:quickstart

サーバー側の実装

```
import { createHTTPServer } from "@trpc/server/adapters/standalone";
import { initTRPC } from "@trpc/server"

const t = initTRPC.create();

const appRouter = t.router({
  helloWorld: t.procedure.query(() => {
    return "Hello World";
  }),
});

export type AppRouter = typeof appRouter;

const server = createHTTPServer({
  router: appRouter,
});

server.listen(8000);
```

- 参考:quickstart

サーバー側の実装

```
import { createHTTPServer } from "@trpc/server/adapters/standalone";
import { initTRPC } from "@trpc/server"

const t = initTRPC.create();

const appRouter = t.router({
  helloWorld: t.procedure.query(() => {
    return "Hello World";
  }),
});

export type AppRouter = typeof appRouter;

const server = createHTTPServer({
  router: appRouter,
});

server.listen(8000);
```

- 参考:quickstart

クライアント側の実装（スクリプト）

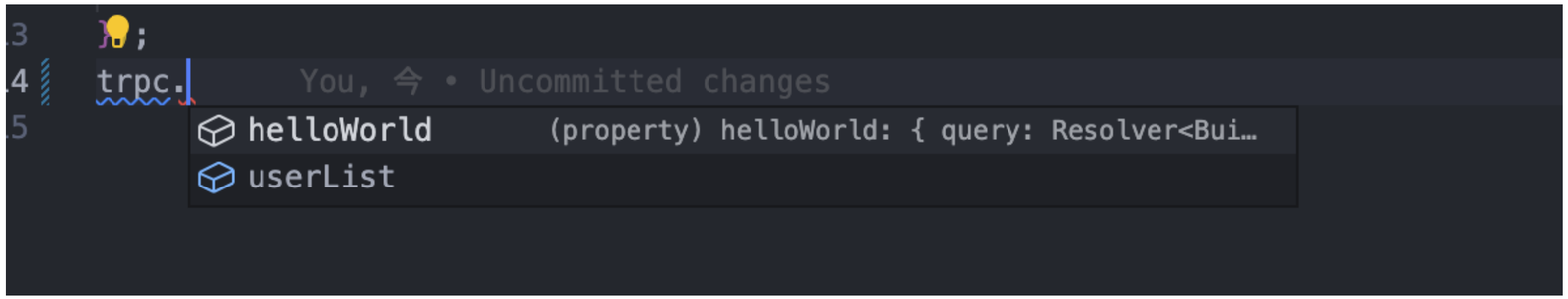
```
import { createTRPCProxyClient, httpBatchLink } from "@trpc/client";
import type { AppRouter } from "./server";

const trpc = createTRPCProxyClient<AppRouter>({
  links: [
    httpBatchLink({
      url: "http://localhost:3000",
    }),
  ],
});

trpc.helloWorld.query().then((res) => {
  // output "Hello World"
  console.log(res);
});
```

クライアント側の実装（スクリプト）

- コードでは以下のように補完が表示される



クライアント側の実装（React)

- 公式でHooksが提供されているので使用して実装
  - https://trpc.io/docs/reactjs/setup

クライアント側の実装（React)

containers/Trpc.tsx

```
import { useState } from "react";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { httpBatchLink } from "@trpc/client";
import { trpc } from "../utils/trpc";

type Props = {
  children: React.ReactNode;
};

function Trpc(props: Props) {
  const [queryClient] = useState(() => new QueryClient());
  const [trpcClient] = useState(() =>
    trpc.createClient({
      links: [
        httpBatchLink({
          url: "http://localhost:8080",
        }),
      ],
    })
  );

  return (
    <trpc.Provider client={trpcClient} queryClient={queryClient}>
      <QueryClientProvider client={queryClient}>
        {props.children}
      </QueryClientProvider>
    </trpc.Provider>
  );
}

export default Trpc;
```

クライアント側の実装（React)

Page.tsx

```
import { trpc } from "./utils/trpc";



export default function IndexPage() {

  const userQuery = trpc.helloWorld.useQuery();



  return <div>{userQuery.data}</div>;

}
```

デモ: React + tRPC + Prismaで実装

- ■ GitHub
  - https://github.com/wheatandcat/tRPC-demo/tree/main/trpc-react-prisma
- デモを見せながらコードを解説
- PrismaでDBの定義を行う
  - server/prisma/schema.prisma
- tRPCとPrismaの連携
  - server/src/router.ts
- Reactとの連携
  - client/src/Page.tsx

補足: Deno + tRPCを試す

- ■ GitHub
  - https://github.com/wheatandcat/deno-trpc-demo
- モチベーション
  - 何だかんだでTypeScriptをデプロイするのが面倒だったので、Denoで行けたら1つで全て済むと思って試してみた
- 結果
  - tRPCの実装はできるけど、Denoで実装したファイルをimportできるのはDenoのみなので、結果フロントエンドもDenoで実装する必要がある
  - こうなってくると旨味が薄れるので、まだ実用段階ではなかったなという感想

補足: 手軽にtRPCを試すならCreate T3 Appがおすすめ

- **T3 Stack**のdemoアプリを簡単に作成できる
  - フロントエンド界隈で新しく提唱されているT3 Stackについて調べてみた
  - 実行方法は以下を参照
    - Introduction • Create T3 App

まとめ

- コードジェネートして型安全をする方法に慣れきっていたが、tRPCはコードジェネートしなくても型安全が実現できるので手軽感がかなりある
- TypeScript周りのツールも大分充実してきて書きやすいので、tRPCを使っていくのも全然有り
- BFF構成なら、GraphQLより筋が良さそう

ご清聴ありがとうございました🎉