In real project development, you may need data flow solutions such as Redux or MobX. Ant Design React is a UI library that can be used with data flow solutions and application frameworks in any React ecosystem. Based on the business scenario, we launched a pluggable enterprise-level application framework Umi, which is recommended for use in the project.

And Umi is a routing-based framework that supports next.js-like conventional routing and various advanced routing functions, such as routing-level on-demand loading. With a complete plugin system that covers every life cycle from source code to build product, Umi is able to support various functional extensions and business needs; meanwhile Umi UI is provided to enhance the development experience and development efficiency through Visual Aided Programming (VAP).

> *You may also be interested in Ant Design Pro, an Out-of-box UI solution for enterprise applications based on Umi and antd.*

This article will guide you to create a simple application from zero using Umi and antd.

Install Umi

It is recommended to use yarn to create an application and execute the following command.

```
$ mkdir myapp && cd myapp
$ yarn create @umijs/umi-app
$ yarn
```

> *If you use npm, you can execute `npx @umijs/create-umi-app` with the same effect.*

> *And if you want to use a fixed version of antd, you can install additional antd dependency in your project, and the antd dependencies declared in package.json will be used first.*

Create Routes

We need to write an application displaying the list of products. The first step is to create a route.

If you don't have npx, you need to install it first to execute the commands under node_modules.

```
$ yarn global add npx
```

Then create a `/products` route,

```
$ npx umi g page products --typescript

Write: src/pages/products.tsx
Write: src/pages/products.css
```

In `.umirc.ts` configured in routing, if there is need to internationalization, can configure `locale` enable antd internationalization:

```
import { defineConfig } from 'umi';

export default defineConfig({
+ locale: { antd: true },
```

```
  routes: [
    { path: '/', component: '@/pages/index' },
+   { path: '/products', component: '@/pages/products' },
  ],
});
```

run `yarn start` then open http://localhost:8000/products in your browser and you should see the corresponding page.

Write UI Components

As your application grows and you notice you are sharing UI elements between multiple pages (or using them multiple times on the same page), in Umi it's called reusable components.

Let's create a `ProductList` component that we can use in multiple places to show a list of products.

Create `src/components/ProductList.tsx` by typing:

```
import { Table, Popconfirm, Button } from 'antd';

const ProductList = ({ onDelete, products }) => {
  const columns = [
    {
      title: 'Name',
      dataIndex: 'name',
    },
    {
      title: 'Actions',
      render: (text, record) => {
        return (
          <Popconfirm title="Delete?" onConfirm={() => onDelete(record.id)}>
            <Button>Delete</Button>
          </Popconfirm>
        );
      },
    },
  ];
  return <Table dataSource={products} columns={columns} />;
};

export default ProductList;
```

Simple data management solution

`@umijs/plugin-model` is a simple data flow scheme based on the hooks paradigm, which can replace dva to perform global data flow in the middle stage under certain circumstances. We agree that the files in the `src/models` directory are the model files defined by the project. Each file needs to export a function by default, the function defines a hook, and files that do not meet the specifications will be filtered out.

The file name corresponds to the name of the final model, and you can consume the data in the model through the API provided by the plug-in.

Let's take a simple table as an example. First we create a new file `src/services/product.ts` for remote API.

```
/*
export function queryProductList() {
  return fetch('/api/products').then(res => res.json());
}
*/
// mock request service by setTimeout
export function queryProductList() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve([
        {
          id: 1,
          name: 'dva',
        },
        {
          id: 2,
          name: 'antd',
        },
      ]);
    }, 2000);
  });
}
```

Then you need to create a new file `src/models/useProductList.ts` .

```
import { useRequest } from 'umi';
import { queryProductList } from '@/services/product';

export default function useProductList(params: { pageSize: number; current: number
}) {
  const msg = useRequest(() => queryUserList(params));

  const deleteProducts = async (id: string) => {
    try {
      await removeProducts(id);
      message.success('success');
      msg.run();
    } catch (error) {
      message.error('fail');
    }
  };

  return {
    dataSource: msg.data,
    reload: msg.run,
    loading: msg.loading,
    deleteProducts,
  };
}
```

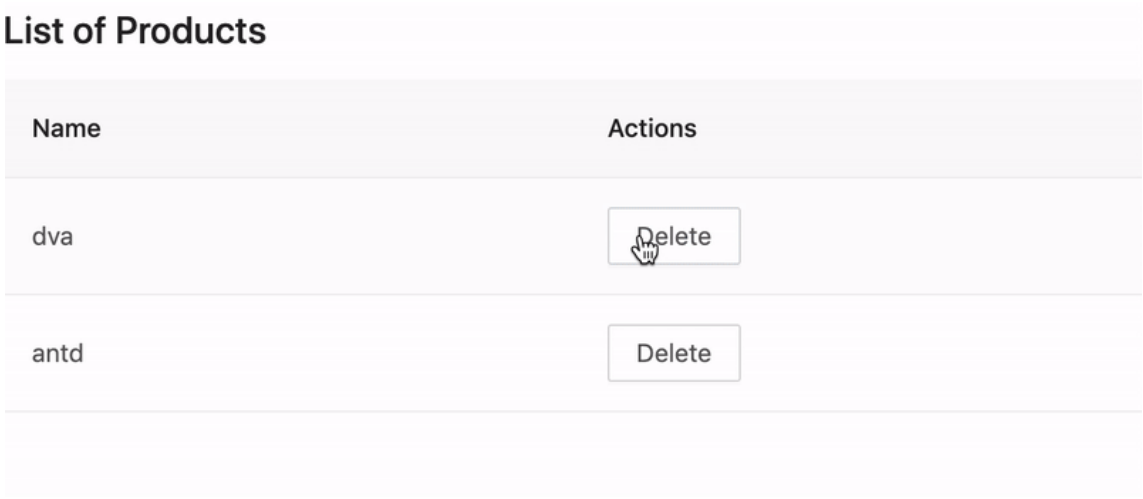
Edit `src/pages/products.tsx` and replace with the following:

```
import { useModel } from 'umi';
import ProductList from '@/components/ProductList';

const Products = () => {
  const { dataSource, reload, deleteProducts } = useModel('useProductList');
  return (
    <div>
      <a onClick={() => reload()}>reload</a>
      <ProductList onDelete={deleteProducts} products={dataSource} />
    </div>
  );
};

export default Products;
```

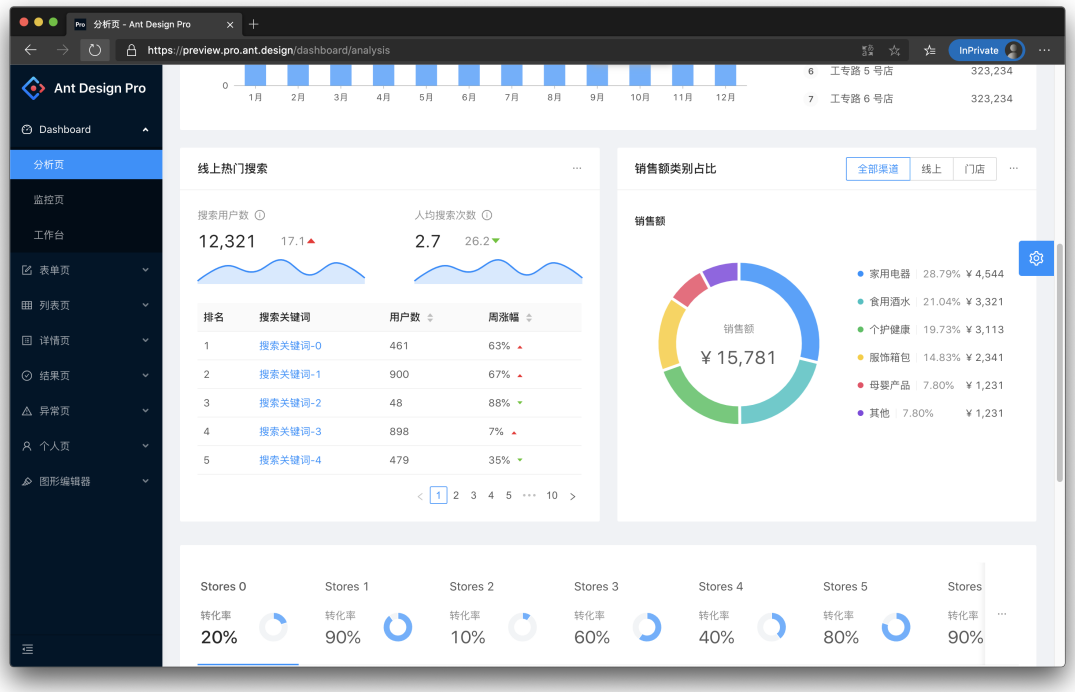
Refresh your browser, you should see the following result:



ProLayout

A standard mid-to-back page generally requires a layout. This layout is often highly similar. ProLayout encapsulates commonly used menus, breadcrumbs, page headers and other functions, provides an independent framework and works out of the box Advanced layout components.

And supports three modes of `side` , `mix` , and `top` , and it also has built-in menu selection, the menu generates breadcrumbs, and automatically sets the logic of the page title. Can help you start a project quickly.



The method of use is also extremely simple, requiring only a few simple settings.

```
import { Button } from 'antd';
import ProLayout, { PageContainer } from '@ant-design/pro-layout';

export default (
  <ProLayout>
    <PageContainer
      extra={[
        <Button key="3">Operating</Button>,
        <Button key="2">Operating</Button>,
        <Button key="1" type="primary">
          Main Operating
        </Button>,
      ]}
      footer={[<Button>reset</Button>, <Button type="primary">submit</Button>]}
    >
      {children}
    </PageContainer>
  </ProLayout>
);
```

Click here Quick Start.

ProTable

Many data in an admin page does not need to be shared across pages, and models are sometimes not needed.

```
import ProTable from '@ant-design/pro-table';
import { Popconfirm, Button } from 'antd';
import { queryProductList } from '@/services/product';

const Products = () => {
  const actionRef = useRef<ActionType>();

  const deleteProducts = async (id: string) => {
    try {
      await removeProducts(id);
      message.success('success');
      actionRef.current?.reload();
    } catch (error) {
      message.error('fail');
    }
  };

  const columns = [
    {
      title: 'Name',
      dataIndex: 'name',
    },
    {
      title: 'Actions',
      render: (text, record) => {
        return (
          <Popconfirm title="Delete?" onConfirm={() => onDelete(record.id)}>
            <Button>Delete</Button>
          </Popconfirm>
        );
      },
    },
  ];

  return (
    <ProTable<{ name: string }>
      headerTitle="Query Table"
      actionRef={actionRef}
      rowKey="name"
      request={(params, sorter, filter) => queryProductList({ ...params, sorter,
filter })}
      columns={columns}
    />
  );
};
```

ProTable provides preset logic to handle loading, pagination and search forms, which can greatly reduce the amount of code, click here ProTable.

Build

Now that we've written our application and verified that it works in development, it's time to get it ready for deployment to our users. To do so, execute the following command:

```
$ yarn build
```

The `build` command packages up all of the assets that make up your application —— JavaScript, templates, CSS, web fonts, images, and more. Then you can find these files in the `dist/` directory.

What's Next

We have completed a simple application, but you may still have lots of questions, such as:

- How to handle onError globally and locally?
- How to handle routes?
- How to mock data?
- How to deploy?
- ant so on...

You can:

- Visit Umi official website
- Know Umi routes
- Know how to deploy Umi application
- Scaffolding out of the box Ant Design Pro
- Advanced Layout ProLayout
- Advanced Table ProTable