在真实项目开发中，你可能会需要 Redux 或者 MobX 这样的数据流方案，Ant Design React 作为一个 UI 库，可以和任何 React 生态圈内的数据流方案以及应用框架搭配使用。我们基于业务场景的场景，推出了可插拔的企业级应用框架 umi，推荐你在项目中使用。

umi 则是一个可插拔的企业级 react 应用框架。Umi 以路由为基础的，支持类 next.js 的约定式路由，以及各种进阶的路由功能，并以此进行功能扩展，比如支持路由级的按需加载。然后配以完善的插件体系，覆盖从源码到构建产物的每个生命周期，支持各种功能扩展和业务需求，同时提供 Umi UI 通过可视化辅助编程（VAP）提高开发体验和研发效率。

> *你可能也会对 Ant Design Pro 感兴趣，这是一个基于 Umi 和 antd 的开箱即用的中台前端/设计解决方案。*

本文会引导你使用 Umi 和 antd 从 0 开始创建一个简单应用。

安装 Umi

推荐使用 yarn 创建 Umi 脚手架，执行以下命令。

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

> *如果你使用 npm，可执行 `npx @umijs/create-umi-app`，效果一致。*

> *默认使用 `"antd": "^4.0.0"`，如果要使用固定版本的 antd，你可以在项目里安装额外的 antd 依赖，`package.json` 里声明的 antd 依赖会被优先使用。*

新建路由

我们要写个应用来先显示产品列表。首先第一步是创建路由，路由可以想象成是组成应用的不同页面。

然后通过命令创建 `/products` 路由，

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

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

在 `.umirc.ts` 中配置路由，如果有国际化需要，可以配置 `locale` 开启 antd 国际化：

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

export default defineConfig({
+ locale: { antd: true },
  routes: [
    { path: '/', component: '@/pages/index' },
+   { path: '/products', component: '@/pages/products' },
  ],
});
```

运行 `yarn start` 然后在浏览器里打开 http://localhost:8000/products，你应该能看到对应的页面。

编写 UI Component

随着应用的发展，你会需要在多个页面分享 UI 元素 (或在一个页面使用多次)，在 Umi 里你可以把这部分抽成 component 。

我们来编写一个 `ProductList` component，这样就能在不同的地方显示产品列表了。

然后新建 `src/components/ProductList.tsx` 文件：

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

const ProductList: React.FC<{ products: { name: string }[]; onDelete: (id: string)
=> void }> = ({
  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;
```

简单数据流方案

`@umijs/plugin-model` 是一种基于 hooks 范式的简单数据流方案，可以在一定情况下替代 dva 来进行中台的全局数据流。我们约定在 `src/models` 目录下的文件为项目定义的 model 文件。每个文件需要默认导出一个 function，该 function 定义了一个 Hook，不符合规范的文件我们会过滤掉。

文件名则对应最终 model 的 name，你可以通过插件提供的 API 来消费 model 中的数据。

我们以一个简单的表格作为示例。首先新建一个 `src/services/product.ts` 存放产品相关的接口。

```
/*
export function queryProductList() {
  return fetch('/api/products').then(res => res.json());
}
```

```
*/
// 先用 setTimeout 模拟一个请求，正常的写法如上所示
export function queryProductList() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve([
        {
          id: 1,
          name: 'dva',
        },
        {
          id: 2,
          name: 'antd',
        },
      ]);
    }, 2000);
  });
}
```

然后新建文件 `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(() => queryProductList(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,
  };
}
```

编辑 `src/pages/products.tsx` ，替换为以下内容：

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

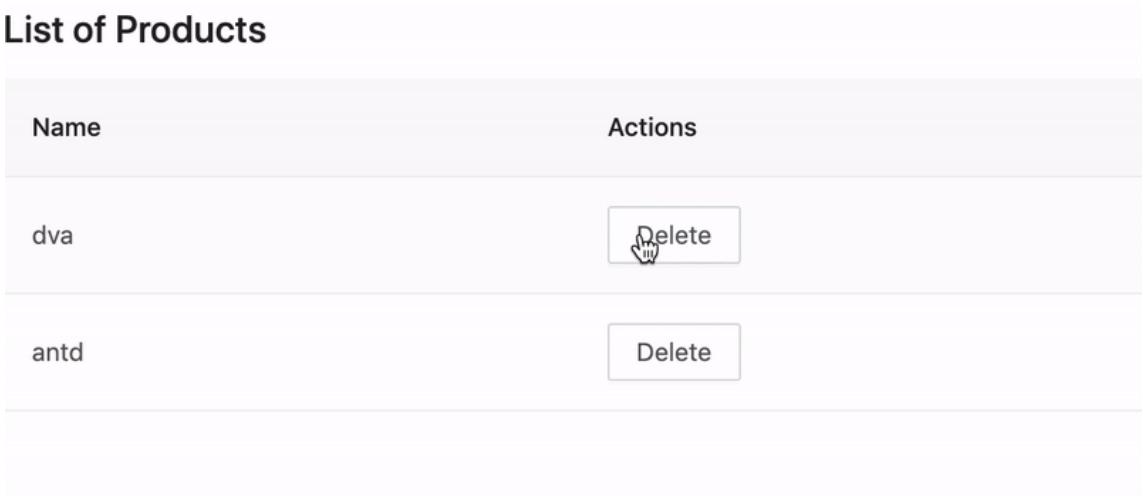
```
const Products = () => {
  const { dataSource, reload, deleteProducts } = useModel('useProductList');
  return (
    <div>
      <a onClick={() => reload()}>reload</a>
      <ProductList onDelete={deleteProducts} products={dataSource} />
    </div>
  );
};

export default Products;
```

执行启动命令：

```
$ yarn start
```

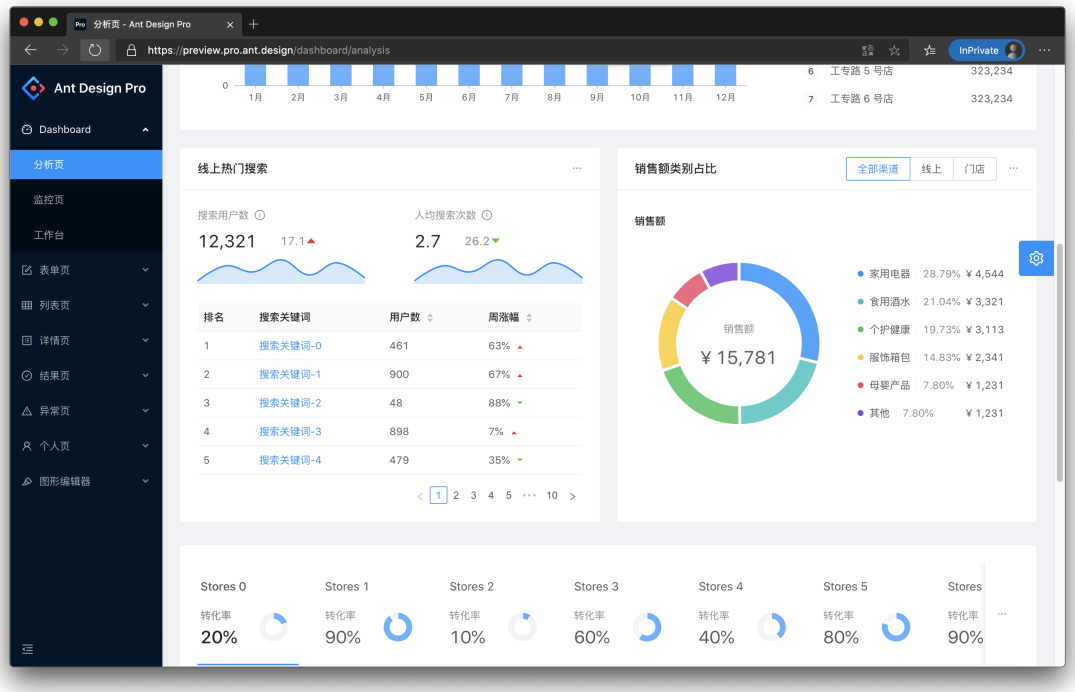
访问 http://localhost:8000，应该能看到以下效果：



ProLayout

一个标准的中后台页面，一般都需要一个布局，这个布局很多时候都是高度雷同的，ProLayout 封装了常用的菜单，面包屑，页头等功能，提供了一个不依赖的框架且开箱即用的高级布局组件。

并且支持 `side`，`mix`，`top` 三种模式，更是内置了菜单选中，菜单生成面包屑，自动设置页面标题的逻辑。可以帮助你快速的开始一个项目。



使用方式也是极为简单，只需要进行几个简单的设置。

```
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>
);
```

点击这里快速开始。

ProTable

一个中后台页面中很多数据都不需要跨页面共享，models 在一些时候也是不需要的。

```
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="查询表格"
      actionRef={actionRef}
      rowKey="name"
      request={(params, sorter, filter) => queryProductList({ ...params, sorter,
filter })}
      columns={columns}
    />
  );
};
```

ProTable 提供了预设逻辑来处理 loading，分页 和搜索表单，可以大大减少代码量，点击这里快速开始。

构建应用

完成开发并且在开发环境验证之后，就需要部署给我们的用户了，执行以下命令：

```
$ yarn build
```

构建会打包所有的资源，包含 JavaScript, CSS, web fonts, images, html 等。你可以在 `dist/` 目录下找到这些文件。

下一步

我们已经完成了一个简单应用，你可能还有很多疑问，比如：

- 如何统一处理出错?
- 如何处理更多路由，比如动态路由，嵌套路由，权限路由等?
- 如何 mock 数据?
- 如何部署?
- 等等

你可以：

- 访问 Umi 官网
- 理解 Umi 的路由
- 理解 如何部署 Umi 应用
- 开箱即用的脚手架 Ant Design Pro
- 高级布局 ProLayout
- 高级表格 ProTable