1. 通过该项目，请简要说明 typescript 比 javascript 的优势在哪？

TypeScript 从核心语言方面和类概念的模塑方面对 JavaScript 对象模型进行扩展。

JavaScript 代码可以在无需任何修改的情况下与 TypeScript 一同工作，同时可以使用编译器将 TypeScript 代码转换为 JavaScript。

TypeScript 通过类型注解提供编译时的静态类型检查。

TypeScript 中的数据要求带有明确的类型，JavaScript不要求。

TypeScript 为函数提供了缺省参数值。

TypeScript 引入了 JavaScript 中没有的“类”概念。

TypeScript 中引入了模块的概念，可以把声明、数据、函数和类封装在模块中。

1. 请简述一下支付流程

商品浏览 > 添加购物车 > 结算 > 计算商品总价 > 生成订单 > 选择支付方式 > 支付成功回调

（1）用户在客户端提交订单 向服务器端发送请求

（2）服务器返回支付地址，引导客户端跳转到支付地址

（3）用户支付

（4）支付成功，支付宝重定向到服务端预设的客户端地址，通知用户支付结果。 同时支付宝向服务端发送 post 请求（请求地址是提前设置好的）告诉服务器当前支付结果 ，服务端创建订单，根据支付结果修改订单状态（未支付、已支付）

3.react-redux 的主要作用是什么，常用的 api 有哪些，什么作用？

1.主要作用

react-redux配合redux使用，将redux定义的store数据注入到组件中，可以使组件轻松的拿到全局状态，方便组件间的通信。使react组价与redux数据中心（store）联系起来，调用dispatch函数修改数据状态后，触发通过subscribe注册更新视图的处理逻辑，包括需要渲染的数据和更新数据的函数。

它主要用于在入口处包裹需要用到Redux的组件。本质上是将store放入context里。

conncet方法用于将组件绑定Redux。本质上是HOC，封装掉了每个组件都要写的板式代码。

createStore函数

用来生成store, 使用方法: const store = createStore(reducer, initialState, enhancer)。

1

bindActionCreators 将 action 包装成直接可被调用的函数，用户感知不到dispatch的存在

combineReducers

一个复杂的应用往往state 比较庞大，导致 Reducer 函数也比较庞大，因此如果能把reducer拆分成一个个独立的子Reducer, 最后再把他们合成一个大的reducer，处理起来就比较方便。而combineReducers就是做这件事的，该函数根据state的key去执行响应的子Reducer，并将结果合并到最终的state对象里。

react-redux用来将redux创建的store映射到组件内部，简化redux操作，react-redux 提供了两个重要的函数，Provider 和 connect。

Provider组件 Provider其实是一个React 组件，其原理是通过React组件的context 属性实现store 的传递， 进而拿到整个应用的state。 connect 函数 connect函数是把 redux 的 dispatch 和 state 映射为 react 组件的 props中，将页面里的组件与应用的状态state真正连接起来。 mapStateToProps()、mapDispatchToProps()可以建立一个从（外部的）state，store.dispatch对象到（UI 组件的）props对象的映射关系

2.常用API

Provider组件：提供共享的store，HOC（高阶组件），Provider 返回一个HOC（高阶组件）的函数：conncet。

connect方法：从store中获取数据和方法，注入组件，返回包含数据和方法的高阶组件。接受两个参数：mapStateToProps和mapDispatchToProps。它们定义了 UI 组件的业务逻辑。前者负责输入逻辑，即将state映射到 UI 组件的参数（props），后者负责输出逻辑，即将用户对 UI 组件的操作映射成 Action。 mapDispatchToProps() connect函数的第二个参数，用来建立 UI 组件的参数到store.dispatch方法的映射。它定义了哪些用户的操作应该当作 Action，传给 Store。它可以是一个函数，也可以是一个对象。

useSelector方法：与connect获取数据的作用一样，即获取组件所需的store中的数据

useDispatch方法：与connect获取数据更新方法的作用一样，即获取dispatch方法，用于发送action来更新store中的数据。

applyMiddleware applyMiddleware(…middlewares) 引入中间件，比如我们经常使用的用于处理异步action的redux-thunk 中间件。实际上，中间件是一个函数，对store.dispatch函数进行了改造，在发出action和执行reducer之间，增加了一些其他的功能。

compose compose是一个返回依次执行参数里面的方法的函数， 其内部是通过Array.prototype.reduceRight 函数实现的，一般redux项目使用多个中间件时会用到。

mapStateToProps() mapStateToProps是一个函数，它接受state作为参数，返回一个对象。这个对象有一个todos属性，代表 UI 组件的同名参数，后面的getVisibleTodos也是一个函数，可以从state算出 todos 的值。

3.主要作用

Provider ，让通过props传递进来的store对象挂载到context环境上，并且渲染props.children；

以便在connect(mapStateToProps, mapDispatchToProps ) 返回的HOC中，通过context可以获取到store对象

再通过store.subscribe函数，注册组件更新的逻辑

conncet，输入两个函数作为参数：mapStateToProps、mapDispatchToProps，返回一个HOC；

在HOC中通过this.context获取到Provider中往下传递的store对象；

在store.subscribe中注册视图更新逻辑；

通过store.getState()作为mapStateToProps函数的参数，store.dispatch作为mapDispatchToProps的参数，生成两个对象；

将产生的两个对象，通过props传递给真正的视图组件使用；

1. redux 中的异步如何处理？

redux中的异步处理流程通常如下：

在异步操作开始前，发送action，用来表示要发起异步操作，用户界面应该有所提示

在异步操作结束后，发送action，用来表示异步操作结束，根据异步操作的结果，对store中的数据和用户界面进行更新

redux中通常使用中间件来进行异步处理，常用的中间件包括 redux-thunk 或 redux-saga。

redux-thunk 使用高阶函数实现， 判别action的类型，如果action是函数，就调用这个函数，书写较为简单 。

redux-saga的语义更清晰，耦合更宽松，易于维护和测试，所以复杂项目通常会选用redux-saga来进行异步处理。redux-saga 使用generator函数实现，集中处理了所有的异步操作，异步接口部分一目了然，但是书写较为复杂。在组件中提交的action的type,对应在saga中要处理的type,saga中处理完后,再put提交对应在reducer中的type。