resolva isso, isso é realmente feliz, é uma interface agora até chegarmos aqui e resolver tudo isso, não sabemos necessariamente o que essa interface pode conter, mas pelo menos podemos nos referir a ela e usá-la como parte de a definição para o alias de tipo.

Assim, você pode criar coisas que são tipos de tipos hierárquicos.

Algumas pessoas chamam isso de tipos recursivos, mas você pode criar coisas assim combinando tipos e interfaces.

Vamos ter certeza de que não estou mentindo.

Portanto, ele pode conter matrizes aninhadas

Pode conter os valores simples em que estamos interessados, tudo parece copacético

espero que agora você entenda a diferença entre esses dois

Os aliases de tipo estão ansiosos, as interfaces são preguiçosas

Os aliases de tipo são extremamente flexíveis, literalmente podem dar um nome a qualquer tipo, qualquer coisa que você usaria como o tipo de variável se encaixa em um alias de tipo.

Interfaces ou o que você gostaria de usar para objetos que incluem funções e matrizes quando digo que são sub-tipos de objetos, quero dizer no sentido Javascript.

Coisas que são coisas imutáveis, certo?

Tudo, exceto o primitivo como string, número, booleano e símbolo e o indefinido

A Microsoft possui uma biblioteca chamada dtslint, que usa tslint e, basicamente, analisa as mensagens de erro de fiapos e as compara com os comentários especiais que você pode deixar nos seus tipos.

Então, isso não é verdade, eu meio que menti quando disse que era um teste Mocha. Esse é apenas um arquivo TypeScript com comentários especiais. E isso basicamente afirma que, se houver um erro, deve haver um erro aqui, eu digo com a mesma facilidade: Você pode escrever casos de teste em torno de coisas que são puramente tipos.

Muitas pessoas não percebem isso e simplesmente pulam os testes completamente quando se trata de interfaces e coisas assim. Mas isso pode ser uma área complicada do seu código.

Onde, se você tem algo que é um pouco abstrato, você realmente quer ter alguma confiança em relação a mudanças não destrutivas.

Então dtslint é o que procurar.

Portanto, os campos de classe podem ter inicializadores. Se você deseja que um inicializador do ParamProp faça o mesmo que faria com argumentos de função, porque é um argumento de função, é o local onde ele mora. Portanto, esse é o valor padrão para o campo de email. Mas se você tiver propriedades que não são itens que você considera como argumentos para um construtor, neste caso, idade, que você pode ver na parte superior, você pode simplesmente fazer o mesmo que faria em JavaScript. Este é efetivamente um valor padrão. Se não de algum modo definido como dentro do construtor aqui. Então, se aqui eu tivesse a idade = 35, agora, assim, nunca entrará em ação. Só é inicializado com isso se ainda não tiver sido inicializado até esse ponto.

Basicamente, isso vai se transformar aqui.

Infere o tipo, se você der um padrão.

Tipo as mesmas regras que uma variável, você está atribuindo a ela seu valor inicial e, portanto, fará o melhor possível para fazer uma estimativa razoável do tipo.

Agora, a única coisa que você gostaria de ter cuidado é se isso poderia ser uma sequência ou um número e você a inicializar em um número, você precisará especificar que aumentará deliberadamente o tipo para incluir a possibilidade de uma corda

sites tentados em que você está tentando gravar nesse valor

Não se engane, através

porque pode haver consumidores Javascript desta biblioteca

E isso não faz nada em termos de realmente impedir direitos a esse valor

É como alugar coisas para pessoas que usam informações de tipo

Então, eu vou usar isso de vez em quando, mas

lembre-se de que nem todo mundo usa informações de tipo

e isso é só fazer uma verificação

Poderíamos mudar isso com a mesma facilidade que um inicializador de propriedades. Mas estou apontando que podemos fazer em qualquer lugar

Além disso, se minhas configurações TypeScript estiverem corretas aqui e elas estiverem

Você notará que estamos sendo gritados aqui, ignore esta primeira mensagem.

Nós vemos muito isso, porque temos muitas classes e coisas que nunca são lidas. Então, o que estamos vendo aqui é que afirmamos que a senha é uma string

E, para qualquer código a jusante, qualquer método, eles poderão acessar o campo privado. Se tentarem acessar a senha, esperam obter uma string de volta e, atualmente, isso não vai acontecer.

Mesmo se tivéssemos uma codificação aqui, onde se disséssemos como se (telefone> 0)

Inicialize isso, ainda receberemos um erro porque, analisando os vários caminhos, podemos percorrer nosso contratado

Uma ou mais foram identificadas onde não acabaríamos definindo senhas com o valor

Existem algumas maneiras de lidar com isso

Uma coisa seria decidir que esse é o problema que sempre devemos sempre inicializar a senha

A outra maneira seria adicionar a possibilidade de que isso seja indefinido

E agora, quem tentar acessar a senha, receberá uma sequência ou uma definição indefinida.

Eles terão que usar um daqueles guardas que mostramos e exemplo de antes.

Então, isso é uma coisa que poderíamos fazer

Aqui está outra coisa que poderíamos fazer

então tudo a jusante deve poder depender da senha estar lá

Nesse caso, usaríamos algo chamado operador de atribuição definida

Estou assumindo a responsabilidade de garantir que este arquivo seja inicializado corretamente

Portanto, é uma área em que quero que você me deixe lidar com isso e não cause um erro neste momento no código

Eu meio que fui revogado para lhe mostrar isso, embora não ache que eu o uso com tanta frequência, aulas abstratas.

Classes abstratas não podem ser instanciadas diretamente, elas apenas servem como classes base

Portanto, diferentemente da interface, que também não pode ser instanciada, as classes abstratas podem ter implementações

Nesse caso, eu tenho um construtor com algumas propriedades param; portanto, o nome e o email estarão lá, servindo como uma classe base.

Mas eu também tenho um método abstrato aqui, e isso deve ser implementado por qualquer subclasse

Então eu penso nisso como uma espécie de meia classe, meia interface

E isso nos deixa passar

Ainda podemos referir duas coisas por sua classe base abstrata, mas implementações concretas podem parecer muito, muito diferentes

Tudo o que você precisa fazer aqui é tornar a classe abstrata e, em seguida, você pode tornar os campos e métodos abstratos também

E assim as classes são necessárias para implementar aquelas com um modificador de acesso compatível.

Tipo, você pode pegar algo protegido, eu me pergunto se você pode fazer isso

Então, é preciso tornar as coisas públicas, porque seria meio estranho ter coisas abstratas que são privadas.

Você realmente não tem visibilidade das subclasses dessa maneira

Portanto, não cumprimos todos os requisitos que precisamos atender

Compiling in "Loose Mode"

Navegue um, você deseja compilar seu projeto no modo loose e começa certificando-se de que seus testes sejam aprovados e renomeie todos os seus arquivos ou parte deles de Javascript para TypeScript, permitindo implícitos quaisquer tipos. Qualquer um implícito aparece sempre que o compilador TypeScript não pode inferir um tipo mais específico e útil. Um bom exemplo disso seria um argumento de função. Portanto, a maneira como o TypeScript deduz as coisas é apenas em uma direção. É apenas através do recebimento de algum valor e de transmitir isso. Então, apenas para destacar este exemplo aqui.

Este é um implícito e, mesmo que aqui embaixo, se eu tentar dividi-lo para tentar dar a pista de que pode ser uma string, ele ainda permanece implícito. Como as informações de tipo não acontecem, a inferência não flui do ponto de uso para o argumento. Alguns idiomas fazem isso, este não.

Então argumentos funcionais são colocados onde você quase sempre terá um implícito. O único caso em que você não faria isso seria se fosse uma chamada de retorno, e houvesse um tipo específico para essa chamada de retorno, e lembre-se de que não precisamos especificar nada lá. Seu objetivo neste passo é corrigir qualquer coisa que esteja causando erros de compilação TypeScript. Exemplos de onde isso aparece o tempo todo são as classes JavaScript Quando você muda para o TypeScript, você precisa indicar os campos e seus tipos, para saber com antecedência o que é um campo válido e o que não é. Então, você vai corrigir esse tipo de coisa, tomando muito cuidado para evitar alterar qualquer comportamento, e essa é uma solicitação de recebimento. Verifique se seus testes foram aprovados novamente e mesclem isso.

Making Anys Explicit

Etapa 2, você começará com a aprovação dos testes, banirá qualquer um implícito e esta é uma opção do compilador tsconfig, noImplicitAny, true. Portanto, definir isso como verdadeiro significa que, em vez de o TypeScript voltar a lugares em que não é possível entender as coisas por inferência, isso gera um erro. Não lançar um erro, mas ele fornecerá um erro do compilador. Então, você terá basicamente uma lista de tarefas, e ela consistirá principalmente de implícitas que agora se transformaram em erros de compilação e você fornecerá tipos significativos sempre que puder e onde não puder , você usará um any explícito. Você adicionará uma anotação de tipo dizendo: este é um qualquer, estou dizendo explicitamente, o que estiver bem aqui.

Este é um bom momento para destacar o DefinitelyTyped, que é um projeto de código aberto que fornece informações sobre o tipo de ambiente, certo?

Isso é como os arquivos DTS.

Ele fornece pacotes npm separados que você pode trazer e, idealmente, fica perfeitamente sobre a base de código JavaScript subjacente que você está acostumado a usar.

Por exemplo, se você deseja usar o Lodash, que, a menos que algo tenha mudado recentemente, isso não está escrito no TypeScript. Está escrito em JavaScript regular. Você pode importar os tipos Lodash e obter todos os benefícios que obtém do código digitado.

Mas é realmente como a implementação e, em seguida, as informações de tipo que estão por cima. DefinitelyTyped publica para milhares de bibliotecas e todas elas estão no escopo do pacote npm em tipos ou tipos. Então, você verá nos tipos slash lodash e, se olhar no pacote JSON neste projeto de oficina, acho que verá nos tipos slash node. Para que ele tenha o módulo fs e o módulo path, todas as informações de tipo existentes. é um bom momento para começar a trazer isso como uma maneira de combater esses itens implícitos e, em seguida, você fará o teste novamente e essa é uma segunda solicitação de pesquisa independente.

Strict Mode

A terceira coisa que você fará como terceira passagem é em pequenos pedaços, porque não há realmente nenhum benefício em fazer isso em uma única passagem enorme.

Você vai ativar configurações estritas do compilador TypeScript. Portanto, as verificações nulas rigorosas, que fazem com que o nulo não seja considerado, desculpe, eu não quero usar um duplo negativo aqui. Com verificações nulas rigorosas definidas como false, nulo é considerado como um valor válido em qualquer tipo. Não é qualquer tipo, mas você pode ter um número cujo valor é nulo. Portanto, nulo se encaixa em qualquer lugar.

Verificação estrita de nulo, definida como true, faça com que a única coisa que possa conter o nulo seja nula. E é aí que você começará a tremer, ramos do seu código que retornaram nulos e agora você terá que começar a apresentar o O operador de interseção com esta função retorna a cadeia ou nulo.

A verdade é estrita em um ramo das configurações de rigidez, não vou me aprofundar muito nisso.