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.

Tipos de funções restritas valida argumentos e retorna tipos de retorno de chamada. Então, se você disser que eu utilizo a função mouseListener como argumento, strictFunctionTypes digitará com mais rigor a correspondência com tudo o que você tentar passar para ela. E vincule, chame, aplique, temos ja olhou que garante que os argumentos passados para a função bind, call e apply e o escopo lexical todos os tipos sejam verificados de maneira apropriada e normal, sem que o último sinalizador seja definido como true, bind, call e apply, você basicamente perde toda a segurança do tipo ao entrar em essas funções Portanto, é aqui que você está tentando se livrar do maior número possível de itens explícitos e substituí-los por tipos mais significativos e deseja evitar a transmissão, que é onde você está forçando o TypeScript a considerar algo como um tipo específico. Isso é feito com a palavra-chave as, como você diria, considere isso como uma string. Quanto mais você introduzir no seu aplicativo, mais seus tipos poderão mentir.

Orador 1: Quero dizer, qual é o benefício de elevar uma base de código existente para o TypeScript?

Resposta: Eles são os mesmos benefícios do uso do TypeScript.

Orador 1: A quantidade de trabalho que você está falando, e isso justifica esses benefícios?

Portanto, depende um pouco de cada caso. Por exemplo, se você está falando sobre alguns desses módulos JavaScript realmente pequenos que fazem uma coisa e apenas uma coisa, o teclado esquerdo seria um bom exemplo disso, certo? Ele está adicionando uma quantidade prescrita de espaço em branco ao início da string. Não sei se há muito valor em converter esse tipo de coisa para TypeScript.

No entanto, se você quiser usar o TypeScript em outro lugar no código que não possui tipos, alguém precisará fornecer essas informações de tipo. Você tem duas opções, você pode, através de algo que eu mostrarei mais tarde. digamos, considere isso enquanto módulo como qualquer um, como qualquer coisa.

Quando eu importar algo deste módulo, deixe-me fazer o que eu quiser. Você poderia fazer isso, mas isso é arriscado pela mesma razão que qualquer um é arriscado em outro lugar. A outra opção é que todo consumidor da sua biblioteca precisa preencher as informações, pelo menos na parte que deseja usar. Como o compilador está realizando uma análise holística do seu aplicativo, que inclui todas as suas dependências. Portanto, falta informações de tipo para dependências que causam fraqueza em qualquer aplicativo que dependa desse código.

# AddressBook Exercise

Vamos aplicar nossa abordagem trifásica recém-aprendida para converter um pedaço de código JavaScript em TypeScript. Então, eu escrevi um exemplo para você. Está na pasta de desafios do catálogo de endereços e há um pequeno erro de digitação aqui que não é index.ts, é index.js. O único arquivo na pasta do catálogo de endereços, espero que possamos descobrir isso.

Vamos dar uma olhada neste pedaço de código para entender os vários aspectos dele e onde podemos precisar pensar em informações de tipo. Então, temos uma classe aqui chamada AddressBook e vamos aprofundar isso em um momento. Temos aqui uma função chamada formatDate, getFullName e getVcardText. Se você nunca ouviu falar do Vcard, pode criar um arquivo Vcard, é apenas um arquivo de texto com informações de contato e pode instalá-lo no aplicativo de contatos ou algo assim. Na classe AddressBook, temos uma matriz que parece ser para contatos. Um método para adicionar contato e, em seguida, localize um contato pelo nome. Portanto, a primeira das três etapas nos renomeia esse arquivo de JS para TS e configura o compilador TypeScript no modo muito flexível até conseguirmos que as coisas passem.

O passo 0 é garantir que os testes sejam aprovados antes e depois da conclusão.

Vamos começar renomeando este arquivo. Estou renomeando-o para ts, vou para o meu tsconfig e adiciono allow implícito, desculpe, noImplicitAny, configurando-o como false. Portanto, esta opção desabilita o ImplicitAnys e desejo permitir. É meio que um duplo negativo. Então, eu vou salvar isso e voltar ao nosso arquivo.

Agora, na verdade, estamos bem aqui. Obviamente, existem muitos ImplicitAnys, mas não encontramos muitos problemas aqui. Vamos ver, contatos, é uma variedade de qualquer. Esta é uma pista em torno dessa variedade de anys contra a variedade de nevers, estamos em um modo frouxo, então parece que realmente terminamos.

Vamos executar nosso teste novamente.

Tudo bem, e é aqui que fazemos um commit, abrimos uma solicitação de pesquisa e consideramos isso uma posição no mundo do TypeScript.

Alguém se lembra da etapa 2.

Proibir ImplicitAnys, tornando nossos anys explícitos.

Nisto, vamos ter um pouco de trabalho.

Agora podemos ver em nossa sarjeta aqui o TypeScript nos dizendo que há muitos lugares em que precisamos resolver as coisas.

Então, eu gostaria de começar com formatDate, então precisamos encontrar um tipo para data. Alguém tem uma idéia do que pode ser isso?

Ok, agora isso parece ser feliz. Portanto, isso acaba sendo uma espécie de matriz de strings que filtramos para qualquer coisa que falta. Isso pode ser tudo opcional, essa filtragem me dá uma pista de que há algo falso lá que está tentando eliminar.

Voltaremos se começar a nos incomodar.

Tudo bem, agora este parece um pouco retorcido, então eu realmente escolheria esperar até o fim.

O problema que estamos enfrentando aqui, se olharmos para o contato, estamos acessando todo tipo de coisas com isso. Portanto, isso está usando uma quantidade razoável de qualquer estrutura, isso está usando muito e há a nossa data, formatDate passando, mas provavelmente deveríamos estar passando em profundidade, se ainda é bom, as datas definem em algum lugar. Não, está vindo daqui, estamos bem.

Tudo bem, vamos olhar para o AddressBook, então contatos.

Então, obviamente, isso precisa ser do mesmo tipo que o array aqui em cima, certo?

Todo esse método é como uma conveniência que nos permite operar na matriz subjacente.

Aqui estamos filtrando e há c, então c tem firstName e lastName.

Portanto, isso se parece muito com a estrutura de dados com a qual trabalhamos na função abaixo, o nome partes.

Este é um ponto em que eu diria que parece que estamos usando isso em vários lugares.

Eu provavelmente quero criar uma interface para isso.

Então eu vou em frente e faço isso.

Aí está, então vamos chamá-lo de Pessoa, e no topo, e diríamos que os contatos provavelmente devem estar em uma série de Pessoas, e isso envolverá uma única pessoa.

O filtro tem uma propriedade, é firstName e lastName.

Então, novamente, eu não identifiquei que isso é necessariamente, poderia ser uma Pessoa.

Quero dizer, está verificando se estão indefinidos. Se encaixa por enquanto, não encontramos um motivo para não se encaixar. Eu gostaria de tomar nota de que provavelmente deveria voltar, posso estar restringindo demais isso.

Se são apenas algumas propriedades de uma Pessoa, e agora que estou pensando nisso provavelmente, você não gostaria de exigir que isso seja uma coisa completa.

Essas verificações indefinidas me fazem querer tornar firstName e lastName opcionais. Ok, então este pedaço de código está feliz. Na verdade, todo o nosso AddressBook parece estar feliz.

Agora, estamos na área gnarly e, por conveniência, e sendo capaz de não rolar por todo o lugar, quero mover isso para baixo, para que fique por perto.

Tudo bem, temos uma saudação, saudação voltando a um fio vazio.

Essa é uma pista de que uma string provavelmente funcionaria aqui e o fato de eu precisar de um substituto sugere que isso pode ser falso.

Eu acho que talvez possa ser uma string vazia, mas isso deve estar bem.

Ok, então temos telefones e endereços.

Alguém tem uma ideia para isso?

Isso sugere uma forma específica desse valor para alguém?

Então, algo assim, certo?

É isso que Object.keys quer, quer um objeto.

Vamos ver, esse erro de tipo, parece que estou alcançando esse objeto e transmitindo pHName.

Esta mensagem de erro é muito boa a seguir.

O que sugere que faremos a seguir?

O que estamos perdendo?

Alto-falante 4: e phName à propriedade do seu telefone.

phName é apenas uma string.

Eu estou olhando para isso, não tem assinatura de índice.

Vamos adicionar uma assinatura de índice, porque estamos usando uma string arbitrária.

Só sabemos o que é a string em tempo de execução, certo?

Estamos apenas iterando todas as chaves, passando a chave.

É assim que o acesso ao dicionário se parece.

Estamos apenas iterando todas as chaves, passando a chave.

Onde podemos passar uma chave e obter um valor de volta.

Então é aqui que precisamos.

Algo assim, e aposto que os endereços são semelhantes.

Poderíamos dar uma olhada, mas AddrName, vamos provar isso para nós mesmos.

Aqui está, alcance o objeto de endereços, passe a chave, obtenha o valor, certo?

Sinal indicador de um dicionário ou de um associado da matriz, passando a chave no valor.

Então agora precisamos descobrir o que são essas coisas. Então, temos o endereço, número da casa, rua, cidade, país, estado, código postal.

Vou apenas fazer algumas suposições razoáveis para essas bases sobre o que acho que deve ser o tipo.

houseNumber, número, rua, string,

city é uma string, state é uma string, postalCode parece que um número pode ser uma string, isso realmente não importa.

Todos esses são apenas tipos de strings ou coisas interpoladas.

Ok, e o contato tem uma propriedade de e-mail e todo o meu vermelho na minha barra lateral desapareceu aqui.

Só para que eu possa mostrar mais um lugar onde essas informações aparecem no VS Code, existe essa guia Problemas e porque eu tenho aberto e fechado muitos arquivos, alguns deles não estão relacionados ao nosso exemplo aqui.

Mas isso fornece alguns pontos problemáticos em seu código, locais onde o compilador está encontrando erros.

Então, lá vamos nós, e isso seria outro commit. Como temos que executar nosso teste, não podemos esquecer isso.

Não espero nenhum problema, porque acho que estávamos apenas adicionando informações de tipo.

Há um problema.

Acho que os telefones têm o tipo errado para o valor.

Obrigado, os telefones têm o tipo errado para o valor.

Vejamos isso, veja nos dados de fixação que o postalCode deve ser um número.

Portanto, estamos recebendo um erro de tipo nos próprios testes.

Vamos abordar isso, e seus telefones, não há nada aqui.

Você sugeriu que deveria ser um número?

Parece algo estratificável aqui, estou disposto a aceitar o número se os testes forem felizes.

Tudo bem, vamos ver o que mais precisamos fazer.

Tipo string não é atribuível ao número, é uma string.

Lá vamos nós, então agora conhecemos nosso estado normal de retorno e retorno, assumindo que nossa cobertura de teste seja boa.

Você vê o quanto estamos nos apoiando na cobertura do teste para garantir que não alteremos substancialmente o código de maneira a causar problemas e a última fase aqui voltará ao nosso tsconfig, ativando o modo estrito, estrito: true, strictFunctionTypes: true, strictNullChecks: true e não que isso importe, porque não estamos usando BindCallApply.

Mas, para completar, configuraremos isso também e, felizmente, não vejo erros aqui.

Acontece que nosso terceiro passo foi meio que ruim, onde nada de novo apareceu.

Na verdade, chegamos à linha de chegada proibindo apenas entradas implícitas.

Mas seria totalmente possível que novas coisas aparecessem bem no modo loose, e meio que se tornaram verdadeiras violações quando ativamos os sinalizadores do compilador.

Orador 6: Então, se eu adicionasse acidentalmente vírgulas enquanto escrevia minha interface. Parece que isso é bom de se fazer.

São vírgulas entre pares de chave-valor e interfaces.

A razão pela qual prefiro usar ponto-e-vírgula é que isso me ajuda a identificar visualmente a diferença entre e um tipo e valor de objeto. Como você aponta, ambos estão bem.

Eu uso mais bonito para formatar automaticamente as coisas toda vez que eu salvar, e isso substituirá vírgulas por ponto e vírgula.

# Generics

Então, vamos nos aprofundar nos genéricos.

Os genéricos parametrizam tipos da mesma maneira que as funções parametrizam valores.

Então, vamos falar sobre quando é apropriado usar um genérico.

Os meandros dos parâmetros de tipo em geral e como restringir os parâmetros de tipo, por exemplo, dentro de uma função, você tem o que precisa para ter segurança de tipo dentro da função, e o mundo exterior também está satisfeito com as coisas.

Porque os tipos são apenas restrições e representam contratos entre várias entidades.

Então, vamos ver as notas-5. Como eu disse, os genéricos parametrizam tipos como funções parametrizam valores.

Por exemplo, se tivéssemos uma função como esta, e vamos fingir que está escrita em JavaScript, certo?

Sem anotação de tipo, essa função produz um valor agrupado e, dependendo do que a alimentamos, do que atribuímos a x, que determinará o valor da coisa que sai, certo?

Podemos colocar o que quisermos lá, e a função nos permite reutilizar esse trecho de código, se queremos agrupar uma sequência ou número ou ter um valor agrupado dentro de um valor agrupado, o perímetro x é uma espécie de espaço reservado que é usado. E permite reutilizar esse pedaço de código.

Da mesma forma, podemos criar um tipo que nos permita fornecer um parâmetro de tipo, neste caso, X entre esses colchetes angulares. E esse parâmetro de tipo será usado como o tipo para este valor da propriedade.

Então, aqui está um exemplo: se eu passar na matriz de strings, é como passar um argumento para uma função, certo?

Mas estou passando um tipo para um tipo genérico.

Então, por causa das dicas de ferramentas como, isso está correto, mas estou mais interessado no que posso fazer com isso. Se olharmos para val.value, podemos ver que é uma matriz de strings.

Como você pode ver aqui.

Se mudássemos isso e tornássemos apenas uma string, bem, teríamos um erro de tipo aqui, teríamos que fazer disso uma string. E agora, o tipo de valor é string.

Desculpe, ele continua desaparecendo, estou sendo excessivamente zeloso com o mouse.

Mas espero que você possa ver aqui, estamos preenchendo um espaço em branco ou fornecendo um tipo que é então incorporado ao que o WrappedValue deve ser.

Assim, você pode nomear esses parâmetros de tipo como desejar.

Eu posso fazer isso, teclas de função Nome.

Eu posso criar FrontEndMasters, assim como você pode nomear parâmetros de função como quiser.

Esse é um nome completamente local para tudo o que tem acesso a esse tipo de parâmetro, que é meio que, pense nisso como a mesma idéia que um fechamento.

Certo, você tem um escopo, e esta é uma variável local, é quase como um argumento passado para uma função.

O nome desse argumento só tem significado dentro dessa função.

Portanto, a convenção, que você verá muito, é começar com a letra maiúscula T e usar letras maiúsculas como T, U, V ou S ou R.

É uma transição do C ++ que usa parâmetros de modelo que basicamente são conceitualmente idênticos a isso.

É por isso que você verá muitos códigos TypeScript, incluindo meus exemplos, eles usarão apenas a letra maiúscula T.

Mas é como dizer que essa função aceita x e y como argumentos, e podemos nomear isso como quisermos.

Portanto, se estivéssemos usando algo como filtro de matriz, poderíamos digitar uma função que filtraria adequadamente com base nos tipos de coisas que podem estar nessa matriz, certo?

Se estivéssemos filtrando através de uma matriz de strings, para podermos remover qualquer string vazia encontrada, gostaríamos de aceitar como nossa função de filtragem, gostaríamos que ela pegasse uma string como argumento e retornasse um booleano.

E, de fato, é isso que acontece, se tivéssemos uma matriz contendo strings, [''] .filter () e começássemos a implementar retorno de chamada, digamos que seja um filtro que sempre passa, acontece que x tem o valor de uma linha. Certo ?

É exatamente isso que queremos.

Se tivéssemos uma função nomeada que foi projetada, Função nomeada como esta, Que apenas queria receber número, gostaríamos absolutamente que isso dissesse, há um desalinhamento aqui.

Sua função foi projetada para receber números, preciso de uma que seja projetada para receber seqüências de caracteres.

Portanto, esse seria um tipo apropriado usando um genérico que nos permitiria preencher o espaço em branco. Qual é o tipo de entidade que você está filtrando?

Então, aqui está um exemplo de como usamos esse tipo genérico de passagem na string como o parâmetro type, para que este T se transforme em uma string.

E podemos ver que agora, val é string, e estamos verificando se o tipo de val é uma string.

Isso tudo parece ótimo para mim, certo?

O material em tempo de execução, desculpe, esse material em tempo de execução se parece exatamente com o que eu gostaria de verificar se há uma sequência ou não, e isso de fato deve ser uma sequência.

Se tentarmos analisar o argumento errado, obviamente obteremos um erro; caso contrário, se for do tipo certo, ficaremos OK.

# Type Parameters

Também temos aqui algo como esta sintaxe aqui.

Então, eu tenho comparado perímetros de tipos a argumentos de função.

Com base nessa analogia, alguém pode arriscar um palpite sobre o que é igual a qualquer meio nesse contexto?

O que é isso ?

Alto-falante 2: o valor padrão é qualquer um.

se nenhum perímetro de tipo for fornecido, voltaremos a um qualquer.

Esse é o padrão desse parâmetro de tipo.

Então aqui, se passarmos o mouse sobre isso, isso acaba sendo um any e isso é uma espécie de filtro para os valores da verdade.

Tipo de passes X diretamente, passa val diretamente.

E, como resultado, é isso que você poderia esperar.

Você não receberá nenhum erro de tipo ao passar um argumento para isso, porque tudo acontece, certo?

Portanto, é o equivalente a um valor de parâmetro padrão.

É apenas um tipo padrão.

Até agora, em nossos exemplos, como neste exemplo, declaramos que temos esse parâmetro de tipo, T, e descobrimos que estamos usando T diretamente.

Mas você nem sempre precisa fazer isso, não está limitado a isso.

Aqui está um exemplo, um pouco mais complicado, mas que eu retirei do meu próprio código que, Quando passou uma promessa, é como se o tipo de promessa fosse genérico sobre o tipo para o qual resolve.

E quando dizemos genérico sobre isso significa que é, abstrai o tipo para o qual resolve, leva-o como um parâmetro de tipo.

Então você pode ter uma promessa que resolve um número ou uma promessa que resolve uma resposta HTTP, certo?

É isso que você recebe quando chama buscar.

Então T é o parâmetro, mas tomo como argumento Promessa T e inferirei o que T deve ser baseado na promessa que recebi e em seu tipo.

Portanto, são necessárias promessas e argumentos, e um número como argumento, e será uma nova promessa que poderá expirar se o tempo acabar. Ou, se a promessa for resolvida antes do tempo acabar, ela realmente passará pelo valor resultante dessa promessa. E aqui está a implementação, espero que faça sentido.

Chama-se setTimeout.

O que setTimeout retorna é necessário para cancelar o tempo limite mais tarde.

E está tudo pronto, se nada mais acontecer, para rejeitar a promessa.

Se a promessa for resolvida, cancelamos o tempo limite. Não está concluído, o retorno de chamada que passamos para setTimeout pode nunca ser chamado.

E então resolvemos com o valor.

Observe que não preciso passar nada com colchetes angulares aqui.

E o motivo é buscar retorna uma promessa que resolve uma resposta.

E, assim, ao receber esse argumento, digite script descobre que T deve ser uma resposta.

Ele deduz o que T é por meio de arrancá-lo da promessa ou do tipo da promessa.

# Constraints & Scope

Os parâmetros de tipo também podem ter restrições.

Portanto, ao começar a usar parâmetros de tipo, você verá que, se não tiver uma restrição como essa, essa é a parte da restrição.

Isto está dizendo que T deve estender, deve ser atribuível a, este tipo aqui.

A melhor maneira de explicar as restrições para você é removê-lo para revelar a motivação para adicioná-lo. E lá vai você.

Então, estamos recebendo um erro de tipo porque estamos tentando transformar uma matriz do tipo T em um dicionário do tipo T.

E começamos criando um dicionário vazio, iteramos sobre a matriz, tentamos colocar cada valor no dicionário, mas acontece que não podemos acessar esse ID de propriedade de cada membro da matriz.

E a razão é que não declaramos que existe um requisito mínimo que T deve atender e, como resultado, há muito pouco que podemos fazer com valores do tipo T dentro da função.

É realmente flexível em termos do que pode ser passado para a função, mas você só pode fazer as coisas que solicitou explicitamente.

Se precisarmos acessar o ID, devemos declarar que T deve ser um objeto e deve ter uma propriedade chamada ID, e é assim que o faríamos.

Aceitamos um tipo T, que é pelo menos um objeto com um ID de propriedade cujo valor é uma sequência. E agora, isso seria uma string.

Então, isso é o equivalente a fazer algo assim, estou pegando uma matriz, De objetos que se parecem com isso. E nós poderíamos tentar mudar isso.

Certo, em vez de T lá, podemos substituir o T aqui.

Nós vamos encontrar outro problema.

Então, eu basicamente reduzi isso aos meus requisitos mínimos.

Eu disse que transformar essa matriz em um dicionário, mas não há mais nenhum parâmetro de tipo.

Aqui está o problema que encontraremos.

Número um, estamos recebendo erros por transmitir quaisquer propriedades que são mais do que simplesmente ter um ID cujo valor é uma string.

O outro problema que temos é que, quando tentamos acessar coisas, simplificamos bastante esse tipo de maneira ruim, certo?

Perdemos essas coisas extras que foram passadas.

O que queremos é ser capaz de passar qualquer coisa que tenha um ID que seja uma string e fazer um rearranjo disso de uma matriz para um dicionário.

Mas não perdemos nenhuma fidelidade nos tipos, ainda temos acesso a tudo o que colocamos.

E é aí que vamos dizer, tudo bem, o que quer que você me dê uma matriz do tipo T, eu vou lhe dar um dicionário do tipo T ou, certo?

Você continuará trabalhando com o que quer que passe, ainda o recuperará na forma de dicionário.

Não vou reduzi-lo apenas a objetos com ID, você terá como me deu. E então a restrição de tipo, já passamos por cima.

Isso é o equivalente a dizer, olha, eu não vou aceitar, é uma maneira de declarar seus requisitos para trabalhar com esse tipo dentro da função. E por que não está tudo feliz?

Eu esqueci uma última coisa. Agora estamos bem. Portanto, apenas para recapitular, tornando esse resumo, adicionando essa idéia de um genérico, podemos manter os detalhes do objeto em que passamos, mesmo através dessa função que estava fazendo o trabalho, ele não sabe nada sobre essas propriedades extras que podem ser há.

Mas como nos deu um espaço em branco que podemos preencher, do lado de fora, podemos dizer, com o objetivo de chamar essa função agora, estou fornecendo objetos enormes com muitas propriedades em um ID e você obterá nesses mesmos tipos, você não terá que lidar com o menor denominador comum, porque isso o coloca em uma posição em que você precisa descobrir uma maneira de ampliar seus tipos novamente para tornar isso mais útil.

Como argumentos de função,

Os parâmetros de tipo estão associados a um escopo.

Então, aqui temos uma função que retorna uma função, inicia a tupla, termina a tupla. E estamos usando uma técnica que Kyle Simpson em seu curso Frontend Masters sobre programação funcional de luzes.

Ele chama essa programação de ponto zero porque estamos retornando uma função e depois chamando a função que ela retorna imediatamente.

Então, passamos um argumento, uma matriz de strings, um segundo argumento, um número e o tipo da tupla que obtemos é esse.

Então, assim como A e B, certo?

Os valores, esses são parâmetros de função regulares, aqui podemos acessar A e B. E, devido a essa analogia em que os genéricos são como funções, também podemos acessar U e T.

Agora aqui na linha 94, podemos acessar B?

Não, e também não podemos acessar U, eles estão associados a escopos da mesma maneira. E isso se encaixa de maneira semelhante a ter um parâmetro de tipo em uma classe e depois em um método dentro dessa classe, certo?

Escopo externo e interno. Você poderia ter uma classe, um método dentro da classe, uma função que retorna, ela funciona exatamente como você esperaria, porque eu achei essa uma analogia muito eficaz.

# Use Cases