Definitely Typed

> Репозиторий для *высококачественных* определений типов TypeScript.

Также посетите веб-сайт definitelytyped.org, хотя информация в этом README более свежая.

Вы также можете прочитать этот README на английском, испанском, корейском и китайском.

Содержание

Текущее состояние

Этот раздел отслеживает состояние репозитория и процесс публикации. Это может быть полезно для участников, испытывающих любые проблемы с PR'ами и пакетами.

- Самая последняя сборка прошла проверку-типов/линтинг полностью:
  Azure Pipelines succeeded
- Все пакеты проходят проверку-типов/линтинг полностью на `typescript@next`:
  Azure Pipelines failed
- Все пакеты публикуются на npm в течении часа: Azure Pipelines succeeded
- typescript-bot проявляет активность на Definitely Typed
  Azure Pipelines succeeded

Если что-то здесь кажется неправильным или что-либо из вышеперечисленного не работает, пожалуйста, поднимите проблему на канале DefiniteTyped Gitter.

gitter join chat

Что такое файлы декларации (файлы описания/объявления типов)?

Смотрите руководство по TypeScript.

Как их получить?

npm

Это предпочтительный метод. Например:

```
npm install --save-dev @types/node
```

Компилятор должен автоматически подключить типы. Вам может понадобиться добавить связь `types`, если вы не используете модули:

```
/// <reference types="node" />
```

Подробнее смотрите в справочнике.

Для NPM пакета `foo`, описания будут находиться в `@types/foo`. Если вы не можете найти необходимый вам пакет, ищите его в TypeSearch.

Если вы все еще не можете найти его, проверьте включает ли пакет собственную типизацию. Обычно это отражается в поле `"types"` или `"typings"` файла `package.json`, или просто ищите любые файлы «.d.ts» в пакете и вручную включайте их с помощью `/// <reference path="" />`.

Typescript 2.9 и старее

Начиная с ноября 2019 года, Definitely Typed тестирует пакеты только на версиях Typescript, которым меньше двух лет. Если вы используете Typescript от 2.0 до 2.9, вы все равно можете попробовать установить пакеты `@types` - большинство пакетов не используют новые функции Typescript. Но нет гарантии, что они будут работать.

График обновлений:

Версия	Релиз	Окончание поддержки
2.8	Март 2018	Март 2020
2.9	Май 2018	Май 2020
3.0	Июль 2018	Июль 2020
3.1	Сентябрь 2018	Сентябрь 2020
3.2	Ноябрь 2018	Ноябрь 2020
3.3	Январь 2019	Январь 2021
3.4	Март 2019	Март 2021
3.5	Май 2019	Май 2021
3.6	Август 2019	Август 2021
3.7	Ноябрь 2019	Ноябрь 2021
3.8	Февраль 2020	Февраль 2022

Версия	Релиз	Окончание поддержки
3.9	Май 2020	Май 2022

Пакеты, которые существовали до ноября 2019 года, могут иметь более старые версии, которые явно помечены как совместимые с более старыми версиями Typescript; используйте тег "ts2.6" для Typescript 2.6, например.

Например, если вы запустите `npm dist-tags @types/react`, вы увидите следующую таблицу, которая показывает, что у react@16.4 есть типы для Typescript 2.6:

Tag	Version
`latest`	`16.9.11`
`ts2.0`	`15.0.1`
...	...
`ts2.6`	`16.4.7`
...	...

Typescript 1.8 и старше

- Typings
- ~~NuGet~~ (используйте предпочтительные альтернативы, публикация типа nuget DT отключена)
- Вручную загрузите из ветки `master` этого репозитория

Возможно, вам придется добавить ручные ссылки (references).

Как я могу внести свой вклад?

Definitely Typed работает только благодаря вкладу таких пользователей, как вы!

Тестирование

Прежде чем поделиться своим улучшением с миром, используйте его сами.

Тестирование редактирования существующего пакета

Для добавления новых функций вы можете использовать разрешение модулей. Вы также можете напрямую редактировать типы в `node_modules/@types/foo/index.d.ts`, или скопировать их оттуда и выполнить следующие шаги.

Тестирование нового пакета

Добавьте к вашему `tsconfig.json`:

```
"baseUrl": "types",
"typeRoots": ["types"],
```

(Вы также можете использовать `src/types`.) Создайте `types/foo/index.d.ts` содержащие объявления для модуля "foo". Теперь вы сможете импортировать из `"foo"` в свой код, и он будет направлен к новому определению типа. Затем запустите сборку (build) *и* запустите код, чтобы убедиться, что ваше определение типа действительно соответствует тому, что происходит во время выполнения. После того как вы проверили свои определения с реальным кодом, создайте Запрос на принятие изменений (PR) и следуйте инструкциям чтобы отредактировать существующий или создать новый пакет.

Запрос на принятие изменений (PR)

После того, как вы проверили ваш пакет, вы можете поделиться им с Definitely Typed.

Во-первых, разветвите этот репозиторий, установите node, и запустите `npm install`.

Изменение существующего пакета

- `cd types/my-package-to-edit`
- Внесите изменения. Не забудьте отредактировать тесты. Если вы вносите критические изменения, не забудьте обновить основную версию.
- Вы также можете добавить себя в раздел "Definitions by" заголовка пакета.
  - Это приведет к тому, что вы будете уведомлены (через ваше имя пользователя GitHub) о том, что кто-то делает запрос на принятие изменений (PR) или проблему с пакетом.
  - Сделайте это, добавив свое имя в конец строки, например `// Definitions by: Alice <https://github.com/alice>, Bob <https://github.com/bob>`.
  - Или, если есть больше людей, это может быть многострочным

```
// Definitions by: Alice <https://github.com/alice>
//                 Bob <https://github.com/bob>
//                 Steve <https://github.com/steve>
//                 John <https://github.com/john>
```

- Если есть `tslint.json`, запустите `npm run lint package-name`. В противном случае запустите `tsc` в директории пакета.

Когда вы создаете PR для редактирования существующего пакета, `dt-bot` должен @-уведомить предыдущих авторов. Если этого не произойдет, вы можете сделать это самостоятельно в комментарии, связанном с PR.

Создание нового пакета

Если вы являетесь автором библиотеки и ваш пакет написан на TypeScript, свяжите автоматически сгенерированные файлы объявлений в вашем пакете, а не публикуйте в Definitely Typed.

Если вы добавляете типизацию для пакета NPM, создайте директорию с тем же именем. Если пакет, для которого вы добавляете типизацию, отсутствует в NPM, убедитесь, что выбранное вами имя не конфликтует с именем пакета в NPM. (Вы можете использовать `npm info foo` чтобы проверить наличие пакета `foo`.)

Ваш пакет должен иметь такую структуру:

Файл	Назначение
index.d.ts	Содержит типизацию для пакета.
foo-tests.ts	Содержит пример кода, который проверяет типизацию. Этот код *не* запускается, но он проверен на тип.
tsconfig.json	Позволяет вам запускать `tsc` внутри пакета.
tslint.json	Включает linting.

Создайте их, запустив `npx dts-gen --dt --name my-package-name --template module` если у вас npm ≥ 5.2.0, `npm install -g dts-gen` и `dts-gen --dt --name my-package-name --template module` в противном случае. Посмотреть все варианты на dts-gen.

Вы можете отредактировать `tsconfig.json` чтобы добавить новые файлы, добавить `"target": "es6"` (необходимо для асинхронных функций), добавить в `"lib"`, или добавить опцию компилятора `"jsx"`.

Члены группы Definitely Typed регулярно следят за новыми PR, но имейте в виду, что количество других PR может замедлить ход событий.

Хороший пример пакета смотрите base64-js.

Распространенные ошибки

- Сначала следуйте советам из справочника handbook.

- Форматирование: либо используйте все табы, либо всегда используйте 4 пробела.
- `function sum(nums: number[]): number` : используйте `ReadonlyArray` если функция не записывает свои параметры.
- `interface Foo { new(): Foo; }` : Это определяет тип объектов, с методом `new` . Вы, вероятно, хотите объявить `declare class Foo { constructor(); }` .
- `const Class: { new(): IClass; }` : Предпочитайте использовать объявление класса `class Class { constructor(); }` вместо `new` .
- `getMeAT<T>(): T` : Если параметр типа не отображается в типах каких-либо параметров, у вас нет универсальной функции, а просто замаскированное утверждение типа. Предпочитайте использовать утверждение реального типа, например, `getMeAT() as number` . Пример, где допустим параметр типа: `function id<T>(value: T): T;` . Пример, где это недопустимо: `function parseJson<T>(json: string): T;` . Исключение: `new Map<string, number>()` все ОК.
- Использование типов `Function` and `Object` почти никогда не является хорошей идеей. В 99% случаев можно указать более конкретный тип. Примеры: `(x: number) => number` для функций and `{ x: number, y: number }` для объектов. Если нет никакой уверенности в типе, `any` является правильным выбором, а не `Object` . Если единственным известным фактом о типе является то, что это какой-то объект, используйте тип `object` , а не `Object` или `{ [key: string]: any }` .
- `var foo: string | any` : когда `any` используется в типе объединения, результирующий тип все еще `any` . Таким образом, хотя `string` часть аннотации этого типа может *выглядеть* полезной, на самом деле она не предлагает никакой дополнительной проверки типов по сравнению с простым использованием `any` . В зависимости от намерения, приемлемыми альтернативами могут быть `any` , `string` , или `string | object` .

Удаление пакета

Когда пакет объединяет свои собственные типы, типы должны быть удалены из Definitely Typed чтобы избежать путаницы.

Вы можете удалить его, запустив `npm run not-needed -- typingsPackageName asOfVersion sourceRepoURL [libraryName]` .

- `typingsPackageName` : название директории, который нужно удалить.
- `asOfVersion` : заглушка будет опубликована в `@types/foo` с этой версией. Должна быть выше, чем любая опубликованная на данный момент версия
- `sourceRepoURL` : Это должно указывать на репозиторий, который содержит типизации.

- `libraryName` : описательное имя библиотеки, например, "Angular 2" вместо "angular2". (Если опущено, будет идентично "typingsPackageName".)

Любые другие пакеты в Definitely Typed которые ссылаются на удаленный пакет, должны быть обновлены для ссылки на связанные типы. Для этого добавьте в `package.json` ссыклу `"dependencies": { "foo": "x.y.z" }` .

Если пакет никогда не был в Definitely Typed, его не нужно добавлять в `notNeededPackages.json` .

Linter

Все новые пакеты должны быть проанализированы lint. Для этого добавьте `tslint.json` в этот пакет, содержащий

```
{
    "extends": "dtslint/dt.json"
}
```

Это должно быть единственным содержимым в файле `tslint.json` готового проекта. Если `tslint.json` отключает правила, это потому, что это еще не исправлено. Например:

```
{
    "extends": "dtslint/dt.json",
    "rules": {
        // This package uses the Function type, and it will take effort to fix.
        "ban-types": false
    }
}
```

(Чтобы указать, что правило lint действительно не применяется, используйте `// tslint:disable rule-name` или лучше, `//tslint:disable-next-line rule-name` .)

Чтобы проверить, что выражение имеет заданный тип, используйте `$ExpectType` . Чтобы проверить, что выражение вызывает ошибку компиляции, используйте `$ExpectError` .

```
// $ExpectType void
f(1);

// $ExpectError
f('one');
```

Для получения дополнительной информации см. dtslint readme.

Проверка

Протестируйте, запустив `npm run lint package-name` где `package-name` - это имя вашего пакета.

Этот скрипт использует dtslint для запуска компилятора TypeScript на ваших dts файлах.

Часто задаваемые вопросы

Какая связь между этим репозиторием и пакетами `@types` в NPM?

Ветвь `master` автоматически публикуется в область `@types` на NPM благодаря types-publisher.

Я отправил PR. Когда он сольется?

Это зависит, но большинство запросов на получение данных будут объединены в течение недели. PR, утвержденные автором, указанным в заголовке определения, обычно объединяются быстрее; PR для новых определений займет больше времени, так как они требуют большего количества проверок от сопровождающих. Каждый PR проверяется членом команды TypeScript или Definitely Typed перед объединением, поэтому будьте терпеливы, так как человеческий фактор может вызвать задержки. Посмотрите на New Pull Request Status Board чтобы увидеть, как сопровождающие работают через открытые PR.

Мой PR слит; когда будет обновлен пакет `@types` NPM?

Пакеты NPM должны обновиться в течение нескольких часов. Если прошло более 24 часов, пингуйте @RyanCavanaugh и @andy-ms в PR, чтобы расследовать.

Я пишу определение, которое зависит от другого определения. Должен ли я использовать `<reference types="" />` или import?

Если модуль, на который вы ссылаетесь, является внешним модулем (использует `export`), используйте import. Если модуль, на который вы ссылаетесь, является окружающим модулем (использует `declare module` , или просто объявляет глобальные переменные), используйте `<reference types="" />` .

Я заметил, что у некоторых пакетов есть `package.json` .

Обычно вам это не нужно. При публикации пакета мы обычно автоматически создаем `package.json`. `package.json` может быть включен для определения зависимостей. Вот пример. Мы не разрешаем определять другие поля, такие как "description", вручную. Кроме того, если вам нужно сослаться на более старую версию типизаций, вы должны сделать это, добавив в `package.json` строки `"dependencies": { "@types/foo": "x.y.z" }`.

В некоторых пакетах отсутствует `tslint.json`, а в некоторых `tsconfig.json` отсутствует `"noImplicitAny": true`, `"noImplicitThis": true`, или `"strictNullChecks": true`.

Тогда они не правы. Вы можете помочь, отправив PR, чтобы исправить их.

Могу ли я запросить определение?

Вот текущие запрошенные определения.

Как насчет определений типов для DOM?

Если типы являются частью веб-стандарта, они должны быть добавлены в TSJS-lib-generator чтобы они могли стать частью `lib.dom.d.ts` по умолчанию.

Пакет использует export `export =`, но я предпочитаю использовать импорт по умолчанию. Могу ли я изменить `export =` на `export default`?

Если вы используете TypeScript 2.7 или более позднюю версию, используйте `--esModuleInterop` в вашем проекте. В противном случае, если импорт по умолчанию работает в вашей среде (например, Webpack, SystemJS, esm), рассмотрите возможность включения опции компилятора `--allowSyntheticDefaultImports`. Не меняйте определение типа, если оно точное. Для пакета NPM, `export =` является точным, если `node -p 'require("foo")'` является экспортом, а `export default` является точным, если `node -p 'require("foo").default'` является экспортом.

Я хочу использовать функции из TypeScript 3.1 или выше.

В таком случае вам нужно будет добавить комментарий к последней строке заголовка вашего определения (после `// Definitions: https://github.com/DefinitelyTyped/DefinitelyTyped`): `// TypeScript Version: 3.1`.

Я хочу добавить DOM API, отсутствующий в TypeScript по умолчанию.

Это может принадлежать TSJS-Lib-Generator. Смотрите инструкции там. Если стандарт все еще является черновиком, добавляйте сюда. Используйте имя, начинающееся с `dom-` и включите ссылку на стандарт в качестве ссылки "Project" в заголовке. Когда он завершает черновой режим, мы можем удалить его из Definitely Typed и объявить устаревшим связанный пакет `@types`.

Я хочу обновить пакет новой старшей версии

Если вы намерены продолжить обновление старой версии пакета, вы можете создать новую подпапку с текущей версией, например, `v2` и скопируйте в него существующие файлы. Если это так, вам необходимо:

1. Обновите относительные пути в `tsconfig.json` а также в `tslint.json` .
2. Добавьте правила сопоставления путей, чтобы убедиться, что тесты выполняются для предполагаемой версии.

Например history v2 `tsconfig.json` looks like:

```
{
    "compilerOptions": {
        "baseUrl": "../../",
        "typeRoots": ["../../"],
        "paths": {
            "history": ["history/v2"]
        }
    },
    "files": ["index.d.ts", "history-tests.ts"]
}
```

Если в Definitely Typed есть другие пакеты, несовместимые с новой версией, вам нужно будет добавить сопоставления путей к старой версии. Вам также нужно будет сделать это для пакетов в зависимости от пакетов в зависимости от старой версии.

Например, `react-router` зависит от `history@2` , поэтому react-router `tsconfig.json` есть сопоставление пути с `"history": [ "history/v2" ]` ; транзитивно `react-router-bootstrap` (который зависит от `react-router`) также добавляет отображение пути в свой tsconfig.json.

Также, `/// <reference types="..." />` не будет работать с отображением пути, поэтому зависимости должны использовать `import` .

Как мне написать определения для пакетов, которые могут использоваться и глобально и в качестве модуля?

Руководство TypeScript содержит отличную общую информацию о написании определений, а также этот пример файла определения , в котором показано, как создать определение с использованием синтаксиса модуля в стиле ES6, а также указаны объекты, доступные для глобальной области. Этот метод демонстрируется практически в определении для definition for big.js, библиотекой, которую можно загружать глобально с помощью тега скрипта на веб-странице или импортировать с помощью импорта по требованию или в стиле ES6.

Чтобы проверить, как ваше определение может использоваться как при глобальных ссылках, так и в качестве импортированного модуля, создайте тестовую папку `test`, и поместите туда два тестовых файла. Назовите один `YourLibraryName-global.test.ts` а другой `YourLibraryName-module.test.ts`. *Глобальный* тестовый файл должен использовать определение в соответствии с тем, как он будет использоваться в скрипте, загруженном на веб-страницу, где библиотека доступна в глобальной области видимости - в этом сценарии не следует указывать оператор импорта. Тестовый файл *модуля* должен использовать определение в соответствии с тем, как оно будет использоваться при импорте (включая оператор(ы) `import`). Если вы указали свойство `files` в файле `tsconfig.json`, обязательно включите оба тестовых файла. Практический пример этого также доступен в определении big.js.

Обратите внимание, что не требуется полностью использовать определение в каждом тестовом файле - достаточно протестировать только глобально доступные элементы в глобальном тестовом файле и полностью выполнить определение в тестовом файле модуля, или наоборот.

А как насчет областных пакетов?

Типы для пакета с областью `@foo/bar` должны указываться в `types/foo__bar`. Обратите внимание на двойное подчеркивание.

Когда `dts-gen` используется для компоновки пакета с областью действия, свойство `paths` должно быть вручную адаптировано в сгенерированном файле `tsconfig.json` для правильной ссылки на пакет с областью действия:

```
{
    "paths": {
        "@foo/bar": ["foo__bar"]
    }
}
```

История файлов в GitHub выглядит неполной.

GitHub не поддерживает историю файлов для переименованных файлов. Вместо этого используйте `git log --follow`.

Должен ли я добавить пустой namespace в пакет, который не экспортирует модуль для использования импорта в стиле ES6?

Некоторые пакеты, такие как chai-http, экспортируют функцию.

Импорт этого модуля с импортом в стиле ES6 в форме `import * as foo from "foo";` приводит к ошибке:

> error TS2497: Module 'foo' resolves to a non-module entity and cannot be imported using this construct

Эту ошибку можно устранить, объединив объявление функции с пустым namespace'ом с тем же именем, но это не рекомендуется. Это часто цитируемый ответ с Stack Overflow по этому вопросу.

Более целесообразно импортировать модуль, используя `import foo = require("foo");` синтаксис или использовать импорт по умолчанию, такой как `import foo from "foo";` при использовании флага `--allowSyntheticDefaultImports`, если среда выполнения вашего модуля поддерживает схему взаимодействия для модулей не-ECMAScript как таковых.

Лицензия

Этот проект лицензирован по лицензии MIT.

Авторские права на файлы определений принадлежат каждому участнику, указанному в начале каждого файла определения.