链滴

# 我把 deno 官方文档抄了一遍，以确保，知晓有哪些功能了！

作者：devcui

原文链接：https://ld246.com/article/1626071041619

来源网站：链滴

# 1. installation

- 1. shell-mac/linux `curl -fsSL https://deno.land/x/install/install.sh | sh`
- 2. powerShell-win `iwr https://deno.land/x/install/install.ps1 -useb | iex`
- 3. scoop-win `scoop install deno`
- 4. choco-win `choco install deno`
- 5. homebrew-mac `brew install deno`
- 6. nix-mac/linux `nix-shell -p deno`
- 7. source using Cargo `cargo install deno --locked`
- 8. docker https://github.com/denoland/deno_docker

# 2. env

- 1. DENO_DIR defaults to $HOME/.cache/deno
- 2. NO_COLOR turn off/on color output

# 3. auto compleate

bash

```
deno completions bash > /usr/local/etc/bash_completion.d/deno.bash
source /usr/local/etc/bash_completion.d/deno.bash
```

zsh

```
# create a folder to save your completions. it can be anywhere
mkdir ~/.zsh
```

```
deno completions zsh > ~/.zsh/_deno

# write this into .zshrc
fpath=(~/.zsh $fpath)
autoload -Uz compinit
compinit -u
```

zsh + oh-my-zsh

```
mkdir ~/.oh-my-zsh/custom/plugins/deno
deno completions zsh > ~/.oh-my-zsh/custom/plugins/deno/_deno
```

powershell

```
deno completions powershell >> $profile
```

vscode: Deno initalize workspace configuration

More IDE's config https://deno.land/manual@main/getting_started/setup_your_environment

# 4. first steps

- 1. curl

```
const url = Deno.args[0];
const res = await fetch(url);
const body = new Uint8Array(await res.arrayBuffer());
await Deno.stdout.write(body);
```

Deno is a runtime which is secure by default, using --alow-net please

deno run --allow-net=example.com https://deno.land/std@0.100.0/examples/curl.ts https://example.com

- 2. file --allow-read

```
const filenames = Deno.args;

for (const filename of filenames) {
  const file = await Deno.open(filename);
  await Deno.copy(file, Deno.stdout);
  file.close();
}
```

deno run --allow-read https://deno.land/std@0.100.0/examples/cat.ts /etc/passwd

- 3. tcp --allow-net

```
const hostname = "0.0.0.0";
const port = 8080;
const listener = Deno.listen({ hostname, port });
console.log(`Listening on ${hostname}:${port}`);
for await (const conn of listener) {
  Deno.copy(conn, conn);
}
```

deno run --allow-net https://deno.land/std@0.100.0/examples/echo_server.ts

# 5. command line interface

```
# subcommand
deno help
# shot flag
deno -h
# full flag
deno --help


SUBCOMMANDS:
    bundle         Bundle module and dependencies into single file
    cache          Cache the dependencies
    compile        UNSTABLE: Compile the script into a self contained executable
    completions    Generate shell completions
    coverage       Print coverage reports
    doc            Show documentation for a module
    eval           Eval script
    fmt            Format source files
    help           Prints this message or the help of the given subcommand(s)
    info           Show info about cache or info related to source file
    install        Install script as an executable
    lint           Lint source files
    lsp            Start the language server
    repl           Read Eval Print Loop
    run            Run a JavaScript or TypeScript program
    test           Run tests
    types          Print runtime TypeScript declarations
    upgrade        Upgrade deno executable to given version

ENVIRONMENT VARIABLES:
    DENO_AUTH_TOKENS     A semi-colon separated list of bearer tokens and
                         hostnames to use when fetching remote modules from
                         private repositories
                         (e.g. "abcde12345@deno.land;54321edcba@github.com")
    DENO_CERT            Load certificate authority from PEM encoded file
    DENO_DIR             Set the cache directory
    DENO_INSTALL_ROOT    Set deno install's output directory
                         (defaults to $HOME/.deno/bin)
    DENO_WEBGPU_TRACE    Directory to use for wgpu traces
    HTTP_PROXY           Proxy address for HTTP requests
                         (module downloads, fetch)
    HTTPS_PROXY          Proxy address for HTTPS requests
                         (module downloads, fetch)
```

```
    NO_COLOR             Set to disable color
    NO_PROXY             Comma-separated list of hosts which do not use a proxy
                      (module downloads, fetch)
```

deno run

```
deno-run
Run a JavaScript or TypeScript program

By default all programs are run in sandbox without access to disk, network or
ability to spawn subprocesses.

  deno run https://deno.land/std/examples/welcome.ts

Grant all permissions:

  deno run -A https://deno.land/std/http/file_server.ts

Grant permission to read from disk and listen to network:

  deno run --allow-read --allow-net https://deno.land/std/http/file_server.ts

Grant permission to read allow-listed files from disk:

  deno run --allow-read=/etc https://deno.land/std/http/file_server.ts

Deno allows specifying the filename '-' to read the file from stdin.

  curl https://deno.land/std/examples/welcome.ts | target/debug/deno run -

USAGE:
    deno run [OPTIONS] <SCRIPT_ARG>...

OPTIONS:
    -A, --allow-all
            Allow all permissions

        --allow-env=<allow-env>
            Allow environment access

        --allow-hrtime
            Allow high resolution time measurement

        --allow-net=<allow-net>
            Allow network access

        --allow-plugin
            Allow loading plugins

        --allow-read=<allow-read>
            Allow file system read access

        --allow-run=<allow-run>
            Allow running subprocesses
```

```
    --allow-write=<allow-write>
        Allow file system write access

    --cached-only
        Require that remote dependencies are already cached

    --cert <FILE>
        Load certificate authority from PEM encoded file

-c, --config <FILE>
        Load tsconfig.json configuration file

-h, --help
        Prints help information

    --import-map <FILE>
        Load import map file from local file or remote URL.
        Docs: https://deno.land/manual/linking_to_external_code/import_maps
        Specification: https://wicg.github.io/import-maps/
        Examples: https://github.com/WICG/import-maps#the-import-map
    --inspect=<HOST:PORT>
        Activate inspector on host:port (default: 127.0.0.1:9229)

    --inspect-brk=<HOST:PORT>
        Activate inspector on host:port and break at start of user script

    --location <HREF>
        Value of 'globalThis.location' used by some web APIs

    --lock <FILE>
        Check the specified lock file

    --lock-write
        Write lock file (use with --lock)

-L, --log-level <log-level>
        Set log level [possible values: debug, info]

    --no-check
        Skip type checking modules

    --no-remote
        Do not resolve remote modules

    --prompt
        Fallback to prompt if required permission wasn't passed

-q, --quiet
        Suppress diagnostic output
        By default, subcommands print human-readable diagnostic messages to stderr.
        If the flag is set, restrict these messages to errors.
-r, --reload=<CACHE_BLOCKLIST>
        Reload source code cache (recompile TypeScript)
```

```
        --reload
          Reload everything
        --reload=https://deno.land/std
          Reload only standard modules
        --reload=https://deno.land/std/fs/utils.ts,https://deno.land/std/fmt/colors.ts
          Reloads specific modules
    --seed <NUMBER>
        Seed Math.random()

    --unstable
        Enable unstable features and APIs

    --v8-flags=<v8-flags>
        Set V8 command line options (for help: --v8-flags=--help)

    --watch
        UNSTABLE: Watch for file changes and restart process automatically.
        Only local files from entry point module graph are watched.

ARGS:
    <SCRIPT_ARG>...
        Script arg
```

# 6. WebAssembly

```
const wasmCode = new Uint8Array([
  0, 97, 115, 109, 1, 0, 0, 0, 1, 133, 128, 128, 128, 0, 1, 96, 0, 1, 127,
  3, 130, 128, 128, 128, 0, 1, 0, 4, 132, 128, 128, 128, 0, 1, 112, 0, 0,
  5, 131, 128, 128, 128, 0, 1, 0, 1, 6, 129, 128, 128, 128, 0, 0, 7, 145,
  128, 128, 128, 0, 2, 6, 109, 101, 109, 111, 114, 121, 2, 0, 4, 109, 97,
  105, 110, 0, 0, 10, 138, 128, 128, 128, 0, 1, 132, 128, 128, 128, 0, 0,
  65, 42, 11
]);
const wasmModule = new WebAssembly.Module(wasmCode);
const wasmInstance = new WebAssembly.Instance(wasmModule);
const main = wasmInstance.exports.main as CallableFunction
console.log(main().toString());
```

# 7. debug

chrome

- 1. using --inspect-brk
- 2. open chrome and input chrome://inspect

vscode

```
{
```

```
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Deno",
      "type": "pwa-node",
      "request": "launch",
      "cwd": "${workspaceFolder}",
      "runtimeExecutable": "deno",
      "runtimeArgs": ["run", "--inspect-brk", "-A", "${file}"],
      "attachSimplePort": 9229
    }
  ]
}
```

# 8. runtime

apis

there has two types of deno's api (Web API,Global API)

apis: https://github.com/denoland/deno/releases/latest/download/lib.deno.d.ts

- --unstable: unlocked the features of Deno that still in the draft phase and in the https://raw githubusercontent.com/denoland/deno/main/cli/dts/lib.deno.unstable.d.ts

---

lifecycle

supports browser compatible lifecycle events: load an unload

```
// imported.ts
const handler = (e: Event): void => {
  console.log(`got ${e.type} event in event handler (imported)`);
};

window.addEventListener("load", handler);
window.addEventListener("unload", handler);

window.onload = (e: Event): void => {
  console.log(`got ${e.type} event in onload function (imported)`);
};

window.onunload = (e: Event): void => {
  console.log(`got ${e.type} event in onunload function (imported)`);
};

console.log("log from imported script");

// main.ts
import "./imported.ts";

const handler = (e: Event): void => {
  console.log(`got ${e.type} event in event handler (main)`);
```

```
};

window.addEventListener("load", handler);

window.addEventListener("unload", handler);

window.onload = (e: Event): void => {
  console.log(`got ${e.type} event in onload function (main)`);
};

window.onunload = (e: Event): void => {
  console.log(`got ${e.type} event in onunload function (main)`);
};

console.log("log from main script");

$ deno run main.ts
# first console was not eventListener
log from imported script
log from main script
# load event listener
got load event in event handler (imported)
got load event in event handler (main)
# onload
got load event in onload function (main)
# unload event listener
got unload event in event handler (imported)
got unload event in event handler (main)
# unload
got unload event in onunload function (main)
```

---

Check, by descriptor, if a permission is granted or not.

```
// Global write permission.
const desc1 = { name: "write" } as const;
// Write permission to `$PWD/foo/bar`.
const desc2 = { name: "write", path: "foo/bar" } as const;
// Global net permission.
const desc3 = { name: "net" } as const;
// Net permission to 127.0.0.1:8000.
const desc4 = { name: "net", host: "127.0.0.1:8000" } as const;
// High-resolution time permission.
const desc5 = { name: "hrtime" } as const;
const desc1 = { name: "read", path: "/foo" } as const;
// out put
// devcui@macross devcui-cli % deno run permission.ts
// Check file:///Users/devcui/Projects/devcui-cli/permission.ts
// PermissionStatus { state: "prompt", onchange: null }
console.log(await Deno.permissions.query(desc1));

const desc2 = { name: "read", path: "/foo/bar" } as const;
console.log(await Deno.permissions.query(desc2));
```

```
const desc3 = { name: "read", path: "/bar" } as const;
console.log(await Deno.permissions.query(desc3));
```

request permission

```
const desc1 = { name: "read", path: "/foo" } as const;
const status1 = await Deno.permissions.request(desc1);
// :warning: Deno requests read access to "/foo". Grant? [g/d (g = grant, d = deny)] g
console.log(status1);
```

revoke permissions

```
const desc = { name: "read", path: "/foo" } as const;
console.log(await Deno.permissions.revoke(desc));
// PermissionStatus { state: "prompt" }
```

---

web platform apis

- 1. fetch
- 2. CustomEvent,EventTarget,EventListener
- 3. WebWorker
- 4. Blob
- 5. Console
- 6. Performance
- 7. setTimeout,setInterval,clearInterval
- 8. Streams API
- 9. URL
-

10. URLSearchParams

- 

    11. WebSocket

https://github.com/denoland/deno/blob/v1.11.5/cli/dts/lib.deno.shared_globals.d.ts

https://github.com/denoland/deno/blob/v1.11.5/cli/dts/lib.deno.window.d.ts

https://github.com/denoland/deno/blob/v1.11.5/cli/dts/lib.deno.worker.d.ts

---

http server

```
// listen server
const server = Deno.listen({ port: 8080 });

// tls server
const server = Deno.listenTls({
  port: 8443,
  certFile: "localhost.crt",
  keyFile: "localhost.key",
  alpnProtocols: ["h2", "http/1.1"],
});

// handle http
const server = Deno.listen({ port: 8080 });
for await (const conn of server) {
  // ...handle the connection...
}

// accept listener
// close listener
try {
  const conn = await server.accept();
  conn.close();
  // ... handle the connection ...
} catch (err) {
  // The listener has closed
  break;
}

// serving http
const server = Deno.listen({ port: 8080 });
for await (const conn of server) {
  (async () => {
    const httpConn = Deno.serveHttp(conn);
    for await (const requestEvent of httpConn) {
      console.log(requestEvent.method);
    }
    // waiting for the next request
    const requestEvent = await httpConn.nextRequest();
```

```
  })();
}

// request and response
// request
async function handle(conn: Deno.Conn) {
  const httpConn = Deno.serveHttp(conn);
  for await (const requestEvent of httpConn) {
    const url = new URL(requestEvent.request.url);
    console.log(`path: ${url.path}`);
  }
}
// response
async function handle(conn: Deno.Conn) {
  const httpConn = Deno.serveHttp(conn);
  for await (const requestEvent of httpConn) {
    await requestEvent.respondWith(
      // new Response
      new Response("hello world", {
        status: 200,
      })
    );
  }
}

// http2 support
const server = Deno.listenTls({
  port: 8443,
  certFile: "localhost.crt",
  keyFile: "localhost.key",
  // protocol 'h2,http/1.1'
  alpnProtocols: ["h2", "http/1.1"],
});
```

---

location

deno run --location=https://www.baidu.com that equals sevlet-context-path or baseHref ...

```
console.log(location.href);
// https://www.baidu.com

await fetch("./a");
// https://www.baidu.com/a

// deno run --location https://example.com/index.html --allow-net main.ts

const worker = new Worker("./workers/hello.ts", { type: "module" });
// Fetches worker module at "https://example.com/workers/hello.ts".
```

---

web storage

```
localStorage.setItem();
localStorage.getItem();
localStorage.removeItem();
localStorage.clear();
```

---

workers

```
// Good
new Worker(new URL("./worker.js", import.meta.url).href, { type: "module" });

// Bad
new Worker(new URL("./worker.js", import.meta.url).href);
new Worker(new URL("./worker.js", import.meta.url).href, { type: "classic" });
new Worker("./worker.js", { type: "module" });

// this is demo, demo, and demo
const worker = new Worker(new URL("./worker.js", import.meta.url).href, {
  type: "module",
  deno: {
    namespace: true,
  },
});
worker.postMessage({ filename: "./log.txt" });

self.onmessage = async (e) => {
  const { filename } = e.data;
  const text = await Deno.readTextFile(filename);
  console.log(text);
  self.close();
};

// specifying worker permissions
const worker = new Worker(new URL("./worker.js", import.meta.url).href, {
  type: "module",
  deno: {
    namespace: true,
    permissions: {
      net: ["https://deno.land/"],
      read: [
        new URL("./file_1.txt", import.meta.url),
        new URL("./file_2.txt", import.meta.url),
      ],
      write: false,
    },
  },
});

const worker = new Worker(new URL("./worker.js", import.meta.url).href, {
  type: "module",
  deno: {
    namespace: true,
    permissions: "inherit",
```

```
  },
});

const worker = new Worker(new URL("./worker.js", import.meta.url).href, {
  type: "module",
  deno: {
    namespace: true,
    permissions: {
      env: false,
      hrtime: false,
      net: "inherit",
      plugin: false,
      read: false,
      run: false,
      write: false,
    },
  },
});
```

# 9. linking to external code

```
// load third party code
import { assertEquals } from "https://deno.land/std@0.100.0/testing/asserts.ts";

// $ deno run test.ts
// Compile file:///mnt/f9/Projects/github.com/denoland/deno/docs/test.ts
// Download https://deno.land/std@0.100.0/testing/asserts.ts
// Download https://deno.land/std@0.100.0/fmt/colors.ts
// Download https://deno.land/std@0.100.0/testing/diff.ts
// Asserted!
```

the file will cache to

```
On Linux/Redox: $XDG_CACHE_HOME/deno or $HOME/.cache/deno
On Windows: %LOCALAPPDATA%/deno (%LOCALAPPDATA% = FOLDERID_LocalAppData)
On macOS: $HOME/Library/Caches/deno
If something fails, it falls back to $HOME/.deno
```

---

if the host of the URL goes down?

```
# Download the dependencies.
DENO_DIR=./deno_dir deno cache src/deps.ts

# Make sure the variable is set for any command which invokes the cache.
DENO_DIR=./deno_dir deno test src

# Check the directory into source control.
git add -u deno_dir
git commit
```

---

reload everything

deno cache --reload my_module.ts

---

checking & lock files

deno cache --lock=lock.json --lock-write src/deps.ts

from lock

deno cache --reload --lock=lock.json src/deps.ts

use --cached-only flag to require that remote dependencies are already cached

---

Proxy configuration is read from environmental variables: HTTP_PROXY, HTTPS_PROXY and O_PROXY.

---

private modules https://deno.land/manual/linking_to_external_code/private
that impl by DENO_AUTH_TOKENS env and the github

---

js map

```
{
  "imports": {
    "fmt/": "https://deno.land/std@0.100.0/fmt/"
  }
}
```

use --import-map=<FILE> cli flag to import js map

# 10. using typeScript

- 1. Deno converts typescript(tsx/jsx) to javascript.
- 2. Build into Deno and Rust library called swc.
- 3. code -> checked & transformed -> cache(js file + meta file + buildinfo file).

```
> deno info
DENO_DIR location: "/path/to/cache/deno"
```

```
Remote modules cache: "/path/to/cache/deno/deps"
TypeScript compiler cache: "/path/to/cache/deno/gen"
```

---

jump type checking

```
deno run --allow-net --no-check my_server.ts
```

---

deno supports the type of file

Media Type / How File is Handled

- application/typescript / TypeScript (with path extension influence)
- text/typescript / TypeScript(with path extension influence)
- video/vnd.dlna.mpeg-tts / TypeScript (with path extension influence)
- video/mp2t / TypeScript (with path extension influence)
- application/x-typescript / TypeScript (with path extension influence)
- application/javascript / JavaScript (with path extensions influence)
- text/javascript / JavaScript (with path extensions influence)
- application/ecmascript / JavaScript (with path extensions influence)
- text/ecmascript / JavaScript (with path extensions influence)
- application/x-javascript / JavaScript (with path extensions influence)
- application/node / JavaScript (with path extensions influence)
- text/jsx / JSX
- text/tsx / TSX
- text/plain / Attempt to determine that path extension, otherwise unknown
- application/octet-stream / Attempt to determine that path extension, otherwise unknown

---

support tsconfig.json to config tsc

```
> deno run --config ./tsconfig.json main.ts
```

https://deno.land/manual@v1.11.5/typescript/configuration

---

how deno uses a configuration file

| Option | Default | Notes |
| --- | --- | --- |
| allowJs | true | This almost never eeds to be changed |
| allowUnreachableCode | false | |

allowUnusedLabels false

checkJs false If true causes Typ
Script to type check JavaScript

experimentalDecorators true
e enable these by default as they are already opt-in in the code and when we skip type check
ng, the Rust based emitter has them on by default. We strongly discourage the use of legacy
ecorators, as they are incompatible with the future decorators standard in JavaScript

jsx react

jsxFactory React.createElement

jsxFragmentFactory React.Fragment

keyofStringsOnly false

lib [ "deno.window" ] The defa
lt for this varies based on other settings in Deno. If it is supplied, it overrides the default. See
elow for more information.

noFallthroughCasesInSwitch false

noImplicitAny true

noImplicitReturns false

noImplicitThis true

noImplicitUseStrict true

noStrictGenericChecks false

noUnusedLocals false

noUnusedParameters false

noUncheckedIndexedAccess false

reactNamespace React

strict true

strictBindCallApply true

strictFunctionTypes true

strictPropertyInitialization true

strictNullChecks true

suppressExcessPropertyErrors false

suppressImplicitAnyIndexErrors false

lib

| **name** | **desc** |
| --- | --- |
| deno.ns | includes all custom deno global nam
space apis plus the Deno additions to import.meta |
| deno.unstable | includes the addition unstable
eno global namespace apis |
| deno.window | includes deno.ns/dom |

| | |
|---|---|
| dom.iterable | standard typescript libraries |
| deno.worker | deno web worker |
| deno.asynciterable | async iterable |
| dom | The main browser global library that shi s with TypeScript |
| dom.iterable | The iterable extensions to the br wser global library |
| scripthost | The library for the Microsoft Wind ws Script Host |
| webworker | The main library for web workers i the browser |
| webworker.importscripts | The library that expo es the importScripts() API in the web worker |
| webworker.iterable | The library that adds itera les to objects within a web worker |

---

when typescript is type checking a file,it only cares about the types for the file, and the tsc c mpiler has a lot of logic to try to resolve those types.

By the default '.ts' -> '.d.ts' -> '.js'

---

providing types when importing

if you have javascript module and .d.ts,use @deno-types to import type checking file

```
// @deno-types="./coolLib.d.ts"
import * as coolLib from "./coolLib.js";
```

---

providing types when hosting

Using the triple-slash reference directive

```
/// <reference types="./coolLib.d.ts" />
import * as coolLib from "http://xxx.xxx.coolLib.js";
```

---

checking web workers

```
/// <reference no-default-lib="true" />
/// <reference lib="deno.worker" />
```

or using tsconfig.json

```
{
```

```
  "compilerOptions": {
    "target": "esnext",
    "lib": ["deno.worker"]
  }
}
```

that will use X-TypeScript-Types header

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=UTF-8
Content-Length: 648
X-TypeScript-Types: ./coolLib.d.ts
```

---

Type checking JavaScript

Deno supports using the TypeScript type checker to type check JavaScript. You can mark any ndividual file by adding the check JavaScript pragma to the file:

```
// @ts-check
```

or

tsconfig

```
{
  "compilerOptions": {
    "checkJs": true
  }
}
```

---

doc

```
/** @type {string []} */
const a = [];
```

---

pass checking

```
// @ts-nocheck
```

---

rum time compiler apis

The runtime compiler API is unstable (and requires the --unstable flag to be used to enable it).

```
function emit(
  rootSpecifier: string | URL,
  options?: EmitOptions
): Promise<EmitResult>;
```

```
interface EmitOptions {
  /** Indicate that the source code should be emitted to a single file
   * JavaScript bundle that is a single ES module (`"module"`) or a single
   * file self contained script we executes in an immediately invoked function
   * when loaded (`"classic"`). */
  bundle?: "module" | "classic";
  /** If `true` then the sources will be typed checked, returning any
   * diagnostic errors in the result.  If `false` type checking will be
   * skipped.  Defaults to `true`.
   *
   * *Note* by default, only TypeScript will be type checked, just like on
   * the command line.  Use the `compilerOptions` options of `checkJs` to
   * enable type checking of JavaScript. */
  check?: boolean;
  /** A set of options that are aligned to TypeScript compiler options that
   * are supported by Deno. */
  compilerOptions?: CompilerOptions;
  /** An [import-map](https://deno.land/manual/linking_to_external_code/import_maps#impo
t-maps)
   * which will be applied to the imports. */
  importMap?: ImportMap;
  /** An absolute path to an [import-map](https://deno.land/manual/linking_to_external_code
import_maps#import-maps).
   * Required to be specified if an `importMap` is specified to be able to
   * determine resolution of relative paths. If a `importMap` is not
   * specified, then it will assumed the file path points to an import map on
   * disk and will be attempted to be loaded based on current runtime
   * permissions.
   */
  importMapPath?: string;
  /** A record of sources to use when doing the emit.  If provided, Deno will
   * use these sources instead of trying to resolve the modules externally. */
  sources?: Record<string, string>;
}

interface EmitResult {
  /** Diagnostic messages returned from the type checker (`tsc`). */
  diagnostics: Diagnostic[];
  /** Any emitted files.  If bundled, then the JavaScript will have the
   * key of `deno:///bundle.js` with an optional map (based on
   * `compilerOptions`) in `deno:///bundle.js.map`. */
  files: Record<string, string>;
  /** An optional array of any compiler options that were ignored by Deno. */
  ignoredOptions?: string[];
  /** An array of internal statistics related to the emit, for diagnostic
   * purposes. */
  stats: Array<[string, number]>;
}
```

example

```
> deno run mod.ts
```

You could do something similar with Deno.emit():

```
try {
  const { files } = await Deno.emit("mod.ts");
  for (const [fileName, text] of Object.entries(files)) {
    console.log(`emitted ${fileName} with a length of ${text.length}`);
  }
} catch (e) {
  // something went wrong, inspect `e` to determine
}
```

or improve typescript's source

```
const { files } = await Deno.emit("/mod.ts", {
  sources: {
    "/mod.ts": `import * as a from "./a.ts";\nconsole.log(a);\n`,
    "/a.ts": `export const a: Record<string, string> = {};\n`,
  },
});
```

type checking and emitting and diagnostics

```
const { files, diagnostics } = await Deno.emit("./mod.js", {
  compilerOptions: {
    checkJs: true,
  },
});
// there is something that impacted the emit
console.warn(Deno.formatDiagnostics(diagnostics));
```

bundling

```
const { files, diagnostics } = await Deno.emit("./mod.ts", {
  bundle: "module",
});
```

import maps

```
const { files } = await Deno.emit("mod.ts", {
  bundle: "module",
  importMap: {
    imports: {
      "lodash": "https://deno.land/x/lodash",
    },
  },
  importMapPath: "file:///import-map.json",
});
```

skip type checking/transpiling only

```
const { files } = await Deno.emit("./mod.ts", {
  check: false,
});
```

# 11. standard library

https://deno.land/std/

# 12. testing

- 1.assert(expr: unknown, msg = ""): asserts expr
- 2.assertEquals(actual: unknown, expected: unknown, msg?: string): void
- 3.assertExists(actual: unknown,msg?: string): void
- 4.assertNotEquals(actual: unknown, expected: unknown, msg?: string): void
- 5.assertStrictEquals(actual: unknown, expected: unknown, msg?: string): void
- 6.assertStringIncludes(actual: string, expected: string, msg?: string): void
- 7.assertArrayIncludes(actual: unknown[], expected: unknown[], msg?: string): void
- 8.assertMatch(actual: string, expected: RegExp, msg?: string): void
- 9.assertNotMatch(actual: string, expected: RegExp, msg?: string): void
- 10.assertObjectMatch( actual: Record<PropertyKey, unknown>, expected: Record<Property ey, unknown>): void
- 11.assertThrows(fn: () => void, ErrorClass?: Constructor, msgIncludes = "", msg?: string): Err r
- 12.assertThrowsAsync(fn: () => Promise <void>, ErrorClass?: Constructor, msgIncludes = "" msg?: string): Promise<Error>

# 13. tools

installer, look like go get

deno install --allow-net --allow-read -n serve https://deno.land/std@0.100.0/http/file_server.ts

---

formatter

```
# format all JS/TS files in the current directory and subdirectories
deno fmt
# format specific files
deno fmt myfile1.ts myfile2.ts
# check if all the JS/TS files in the current directory and subdirectories are formatted
deno fmt --check
# format stdin and write to stdout
cat file.ts | deno fmt -
```

---

read-eval-print-loop

pass

---

bundler

```
deno bundle https://deno.land/std@0.100.0/examples/colors.ts colors.bundle.js
```

---

compiling executables

deno compile [--output <OUT>] <SRC> will compile the script into a self-contained executable.

```
deno compile --allow-read --allow-net https://deno.land/std/http/file_server.ts
```

---

documentation generator

```
/**
 * Adds x and y.
 * @param {number} x
 * @param {number} y
 * @returns {number} Sum of x and y
 */
export function add(x: number, y: number): number {
  return x + y;
}

deno doc add.ts

function add(x: number, y: number): number
  Adds x and y. @param {number} x @param {number} y @returns {number} Sum of x and y
```

---

dependency inspector

```
deno info https://deno.land/std@0.67.0/http/file_server.ts

deno info https://deno.land/std@0.67.0/http/file_server.ts
Download https://deno.land/std@0.67.0/http/file_server.ts
...
local: /home/deno/.cache/deno/deps/https/deno.land/f57792e36f2dbf28b14a75e2372a479c392780d4712d76698d5031f943c0020
type: TypeScript
compiled: /home/deno/.cache/deno/gen/https/deno.land/f57792e36f2dbf28b14a75e2372a49c6392780d4712d76698d5031f943c0020.js
deps: 23 unique (total 139.89KB)
https://deno.land/std@0.67.0/http/file_server.ts (10.49KB)
├─┬ https://deno.land/std@0.67.0/path/mod.ts (717B)
│ ├── https://deno.land/std@0.67.0/path/_constants.ts (2.35KB)
│ ├─┬ https://deno.land/std@0.67.0/path/win32.ts (27.36KB)
│ │ ├── https://deno.land/std@0.67.0/path/_interface.ts (657B)
│ │ ├── https://deno.land/std@0.67.0/path/_constants.ts *
│ │ ├─┬ https://deno.land/std@0.67.0/path/_util.ts (3.3KB)
│ │ │ ├── https://deno.land/std@0.67.0/path/_interface.ts *
│ │ │ └── https://deno.land/std@0.67.0/path/_constants.ts *
│ │ └── https://deno.land/std@0.67.0/_util/assert.ts (405B)
```

```
│ ├─┬ https://deno.land/std@0.67.0/path/posix.ts (12.67KB)
│ │ ├── https://deno.land/std@0.67.0/path/_interface.ts *
│ │ ├── https://deno.land/std@0.67.0/path/_constants.ts *
│ │ └── https://deno.land/std@0.67.0/path/_util.ts *
│ ├─┬ https://deno.land/std@0.67.0/path/common.ts (1.14KB)
│ │ └─┬ https://deno.land/std@0.67.0/path/separator.ts (264B)
│ │   └── https://deno.land/std@0.67.0/path/_constants.ts *
│ ├── https://deno.land/std@0.67.0/path/separator.ts *
│ ├── https://deno.land/std@0.67.0/path/_interface.ts *
│ └─┬ https://deno.land/std@0.67.0/path/glob.ts (8.12KB)
│   ├── https://deno.land/std@0.67.0/path/_constants.ts *
│   ├── https://deno.land/std@0.67.0/path/mod.ts *
│   └── https://deno.land/std@0.67.0/path/separator.ts *
├─┬ https://deno.land/std@0.67.0/http/server.ts (10.23KB)
│ ├── https://deno.land/std@0.67.0/encoding/utf8.ts (433B)
│ ├─┬ https://deno.land/std@0.67.0/io/bufio.ts (21.15KB)
│ │ ├── https://deno.land/std@0.67.0/bytes/mod.ts (4.34KB)
│ │ └── https://deno.land/std@0.67.0/_util/assert.ts *
│ ├── https://deno.land/std@0.67.0/_util/assert.ts *
│ ├─┬ https://deno.land/std@0.67.0/async/mod.ts (202B)
│ │ ├── https://deno.land/std@0.67.0/async/deferred.ts (1.03KB)
│ │ ├── https://deno.land/std@0.67.0/async/delay.ts (279B)
│ │ ├─┬ https://deno.land/std@0.67.0/async/mux_async_iterator.ts (1.98KB)
│ │ │ └── https://deno.land/std@0.67.0/async/deferred.ts *
│ │ └── https://deno.land/std@0.67.0/async/pool.ts (1.58KB)
│ └─┬ https://deno.land/std@0.67.0/http/_io.ts (11.25KB)
│   ├── https://deno.land/std@0.67.0/io/bufio.ts *
│   ├─┬ https://deno.land/std@0.67.0/textproto/mod.ts (4.52KB)
│   │ ├── https://deno.land/std@0.67.0/io/bufio.ts *
│   │ ├── https://deno.land/std@0.67.0/bytes/mod.ts *
│   │ └── https://deno.land/std@0.67.0/encoding/utf8.ts *
│   ├── https://deno.land/std@0.67.0/_util/assert.ts *
│   ├── https://deno.land/std@0.67.0/encoding/utf8.ts *
│   ├── https://deno.land/std@0.67.0/http/server.ts *
│   └── https://deno.land/std@0.67.0/http/http_status.ts (5.93KB)
├─┬ https://deno.land/std@0.67.0/flags/mod.ts (9.54KB)
│ └── https://deno.land/std@0.67.0/_util/assert.ts *
└── https://deno.land/std@0.67.0/_util/assert.ts *
```

---

linter

```
# lint all JS/TS files in the current directory and subdirectories
deno lint
# lint specific files
deno lint myfile1.ts myfile2.ts
# print result as JSON
deno lint --json
# read from stdin
cat file.ts | deno lint -
```

rules https://lint.deno.land/

# 14. embedding

Deno consists of multiple parts, one of which is deno_core. This is a rust crate that can be use to embed a JavaScript runtime into your rust application. Deno is built on top of deno_core.

The Deno crate is hosted on crates.io.

https://crates.io/crates/deno_core

You can view the API on docs.rs.

https://docs.rs/deno_core/0.92.0/deno_core/

# 15.OVER