/\*\*

\* The `dns.promises` API provides an alternative set of asynchronous DNS methods

\* that return `Promise` objects rather than using callbacks. The API is accessible

\* via `require('dns').promises` or `require('dns/promises')`.

\* @since v10.6.0

\*/

declare module 'dns/promises' {

import {

LookupAddress,

LookupOneOptions,

LookupAllOptions,

LookupOptions,

AnyRecord,

CaaRecord,

MxRecord,

NaptrRecord,

SoaRecord,

SrvRecord,

ResolveWithTtlOptions,

RecordWithTtl,

ResolveOptions,

ResolverOptions,

} from 'node:dns';

/\*\*

\* Returns an array of IP address strings, formatted according to [RFC 5952](https://tools.ietf.org/html/rfc5952#section-6),

\* that are currently configured for DNS resolution. A string will include a port

\* section if a custom port is used.

\*

\* ```js

\* [

\* '4.4.4.4',

\* '2001:4860:4860::8888',

\* '4.4.4.4:1053',

\* '[2001:4860:4860::8888]:1053',

\* ]

\* ```

\* @since v10.6.0

\*/

function getServers(): string[];

/\*\*

\* Resolves a host name (e.g. `'nodejs.org'`) into the first found A (IPv4) or

\* AAAA (IPv6) record. All `option` properties are optional. If `options` is an

\* integer, then it must be `4` or `6` – if `options` is not provided, then IPv4

\* and IPv6 addresses are both returned if found.

\*

\* With the `all` option set to `true`, the `Promise` is resolved with `addresses`being an array of objects with the properties `address` and `family`.

\*

\* On error, the `Promise` is rejected with an `Error` object, where `err.code`is the error code.

\* Keep in mind that `err.code` will be set to `'ENOTFOUND'` not only when

\* the host name does not exist but also when the lookup fails in other ways

\* such as no available file descriptors.

\*

\* `dnsPromises.lookup()` does not necessarily have anything to do with the DNS

\* protocol. The implementation uses an operating system facility that can

\* associate names with addresses, and vice versa. This implementation can have

\* subtle but important consequences on the behavior of any Node.js program. Please

\* take some time to consult the `Implementation considerations section` before

\* using `dnsPromises.lookup()`.

\*

\* Example usage:

\*

\* ```js

\* const dns = require('dns');

\* const dnsPromises = dns.promises;

\* const options = {

\* family: 6,

\* hints: dns.ADDRCONFIG | dns.V4MAPPED,

\* };

\*

\* dnsPromises.lookup('example.com', options).then((result) => {

\* console.log('address: %j family: IPv%s', result.address, result.family);

\* // address: "2606:2800:220:1:248:1893:25c8:1946" family: IPv6

\* });

\*

\* // When options.all is true, the result will be an Array.

\* options.all = true;

\* dnsPromises.lookup('example.com', options).then((result) => {

\* console.log('addresses: %j', result);

\* // addresses: [{"address":"2606:2800:220:1:248:1893:25c8:1946","family":6}]

\* });

\* ```

\* @since v10.6.0

\*/

function lookup(hostname: string, family: number): Promise<LookupAddress>;

function lookup(hostname: string, options: LookupOneOptions): Promise<LookupAddress>;

function lookup(hostname: string, options: LookupAllOptions): Promise<LookupAddress[]>;

function lookup(hostname: string, options: LookupOptions): Promise<LookupAddress | LookupAddress[]>;

function lookup(hostname: string): Promise<LookupAddress>;

/\*\*

\* Resolves the given `address` and `port` into a host name and service using

\* the operating system's underlying `getnameinfo` implementation.

\*

\* If `address` is not a valid IP address, a `TypeError` will be thrown.

\* The `port` will be coerced to a number. If it is not a legal port, a `TypeError`will be thrown.

\*

\* On error, the `Promise` is rejected with an `Error` object, where `err.code`is the error code.

\*

\* ```js

\* const dnsPromises = require('dns').promises;

\* dnsPromises.lookupService('127.0.0.1', 22).then((result) => {

\* console.log(result.hostname, result.service);

\* // Prints: localhost ssh

\* });

\* ```

\* @since v10.6.0

\*/

function lookupService(

address: string,

port: number

): Promise<{

hostname: string;

service: string;

}>;

/\*\*

\* Uses the DNS protocol to resolve a host name (e.g. `'nodejs.org'`) into an array

\* of the resource records. When successful, the `Promise` is resolved with an

\* array of resource records. The type and structure of individual results vary

\* based on `rrtype`:

\*

\* <omitted>

\*

\* On error, the `Promise` is rejected with an `Error` object, where `err.code`is one of the DNS error codes.

\* @since v10.6.0

\* @param hostname Host name to resolve.

\* @param [rrtype='A'] Resource record type.

\*/

function resolve(hostname: string): Promise<string[]>;

function resolve(hostname: string, rrtype: 'A'): Promise<string[]>;

function resolve(hostname: string, rrtype: 'AAAA'): Promise<string[]>;

function resolve(hostname: string, rrtype: 'ANY'): Promise<AnyRecord[]>;

function resolve(hostname: string, rrtype: 'CAA'): Promise<CaaRecord[]>;

function resolve(hostname: string, rrtype: 'CNAME'): Promise<string[]>;

function resolve(hostname: string, rrtype: 'MX'): Promise<MxRecord[]>;

function resolve(hostname: string, rrtype: 'NAPTR'): Promise<NaptrRecord[]>;

function resolve(hostname: string, rrtype: 'NS'): Promise<string[]>;

function resolve(hostname: string, rrtype: 'PTR'): Promise<string[]>;

function resolve(hostname: string, rrtype: 'SOA'): Promise<SoaRecord>;

function resolve(hostname: string, rrtype: 'SRV'): Promise<SrvRecord[]>;

function resolve(hostname: string, rrtype: 'TXT'): Promise<string[][]>;

function resolve(hostname: string, rrtype: string): Promise<string[] | MxRecord[] | NaptrRecord[] | SoaRecord | SrvRecord[] | string[][] | AnyRecord[]>;

/\*\*

\* Uses the DNS protocol to resolve IPv4 addresses (`A` records) for the`hostname`. On success, the `Promise` is resolved with an array of IPv4

\* addresses (e.g. `['74.125.79.104', '74.125.79.105', '74.125.79.106']`).

\* @since v10.6.0

\* @param hostname Host name to resolve.

\*/

function resolve4(hostname: string): Promise<string[]>;

function resolve4(hostname: string, options: ResolveWithTtlOptions): Promise<RecordWithTtl[]>;

function resolve4(hostname: string, options: ResolveOptions): Promise<string[] | RecordWithTtl[]>;

/\*\*

\* Uses the DNS protocol to resolve IPv6 addresses (`AAAA` records) for the`hostname`. On success, the `Promise` is resolved with an array of IPv6

\* addresses.

\* @since v10.6.0

\* @param hostname Host name to resolve.

\*/

function resolve6(hostname: string): Promise<string[]>;

function resolve6(hostname: string, options: ResolveWithTtlOptions): Promise<RecordWithTtl[]>;

function resolve6(hostname: string, options: ResolveOptions): Promise<string[] | RecordWithTtl[]>;

/\*\*

\* Uses the DNS protocol to resolve all records (also known as `ANY` or `\*` query).

\* On success, the `Promise` is resolved with an array containing various types of

\* records. Each object has a property `type` that indicates the type of the

\* current record. And depending on the `type`, additional properties will be

\* present on the object:

\*

\* <omitted>

\*

\* Here is an example of the result object:

\*

\* ```js

\* [ { type: 'A', address: '127.0.0.1', ttl: 299 },

\* { type: 'CNAME', value: 'example.com' },

\* { type: 'MX', exchange: 'alt4.aspmx.l.example.com', priority: 50 },

\* { type: 'NS', value: 'ns1.example.com' },

\* { type: 'TXT', entries: [ 'v=spf1 include:\_spf.example.com ~all' ] },

\* { type: 'SOA',

\* nsname: 'ns1.example.com',

\* hostmaster: 'admin.example.com',

\* serial: 156696742,

\* refresh: 900,

\* retry: 900,

\* expire: 1800,

\* minttl: 60 } ]

\* ```

\* @since v10.6.0

\*/

function resolveAny(hostname: string): Promise<AnyRecord[]>;

/\*\*

\* Uses the DNS protocol to resolve `CAA` records for the `hostname`. On success,

\* the `Promise` is resolved with an array of objects containing available

\* certification authority authorization records available for the `hostname`(e.g. `[{critical: 0, iodef: 'mailto:pki@example.com'},{critical: 128, issue: 'pki.example.com'}]`).

\* @since v15.0.0, v14.17.0

\*/

function resolveCaa(hostname: string): Promise<CaaRecord[]>;

/\*\*

\* Uses the DNS protocol to resolve `CNAME` records for the `hostname`. On success,

\* the `Promise` is resolved with an array of canonical name records available for

\* the `hostname` (e.g. `['bar.example.com']`).

\* @since v10.6.0

\*/

function resolveCname(hostname: string): Promise<string[]>;

/\*\*

\* Uses the DNS protocol to resolve mail exchange records (`MX` records) for the`hostname`. On success, the `Promise` is resolved with an array of objects

\* containing both a `priority` and `exchange` property (e.g.`[{priority: 10, exchange: 'mx.example.com'}, ...]`).

\* @since v10.6.0

\*/

function resolveMx(hostname: string): Promise<MxRecord[]>;

/\*\*

\* Uses the DNS protocol to resolve regular expression based records (`NAPTR`records) for the `hostname`. On success, the `Promise` is resolved with an array

\* of objects with the following properties:

\*

\* \* `flags`

\* \* `service`

\* \* `regexp`

\* \* `replacement`

\* \* `order`

\* \* `preference`

\*

\* ```js

\* {

\* flags: 's',

\* service: 'SIP+D2U',

\* regexp: '',

\* replacement: '\_sip.\_udp.example.com',

\* order: 30,

\* preference: 100

\* }

\* ```

\* @since v10.6.0

\*/

function resolveNaptr(hostname: string): Promise<NaptrRecord[]>;

/\*\*

\* Uses the DNS protocol to resolve name server records (`NS` records) for the`hostname`. On success, the `Promise` is resolved with an array of name server

\* records available for `hostname` (e.g.`['ns1.example.com', 'ns2.example.com']`).

\* @since v10.6.0

\*/

function resolveNs(hostname: string): Promise<string[]>;

/\*\*

\* Uses the DNS protocol to resolve pointer records (`PTR` records) for the`hostname`. On success, the `Promise` is resolved with an array of strings

\* containing the reply records.

\* @since v10.6.0

\*/

function resolvePtr(hostname: string): Promise<string[]>;

/\*\*

\* Uses the DNS protocol to resolve a start of authority record (`SOA` record) for

\* the `hostname`. On success, the `Promise` is resolved with an object with the

\* following properties:

\*

\* \* `nsname`

\* \* `hostmaster`

\* \* `serial`

\* \* `refresh`

\* \* `retry`

\* \* `expire`

\* \* `minttl`

\*

\* ```js

\* {

\* nsname: 'ns.example.com',

\* hostmaster: 'root.example.com',

\* serial: 2013101809,

\* refresh: 10000,

\* retry: 2400,

\* expire: 604800,

\* minttl: 3600

\* }

\* ```

\* @since v10.6.0

\*/

function resolveSoa(hostname: string): Promise<SoaRecord>;

/\*\*

\* Uses the DNS protocol to resolve service records (`SRV` records) for the`hostname`. On success, the `Promise` is resolved with an array of objects with

\* the following properties:

\*

\* \* `priority`

\* \* `weight`

\* \* `port`

\* \* `name`

\*

\* ```js

\* {

\* priority: 10,

\* weight: 5,

\* port: 21223,

\* name: 'service.example.com'

\* }

\* ```

\* @since v10.6.0

\*/

function resolveSrv(hostname: string): Promise<SrvRecord[]>;

/\*\*

\* Uses the DNS protocol to resolve text queries (`TXT` records) for the`hostname`. On success, the `Promise` is resolved with a two-dimensional array

\* of the text records available for `hostname` (e.g.`[ ['v=spf1 ip4:0.0.0.0 ', '~all' ] ]`). Each sub-array contains TXT chunks of

\* one record. Depending on the use case, these could be either joined together or

\* treated separately.

\* @since v10.6.0

\*/

function resolveTxt(hostname: string): Promise<string[][]>;

/\*\*

\* Performs a reverse DNS query that resolves an IPv4 or IPv6 address to an

\* array of host names.

\*

\* On error, the `Promise` is rejected with an `Error` object, where `err.code`is one of the DNS error codes.

\* @since v10.6.0

\*/

function reverse(ip: string): Promise<string[]>;

/\*\*

\* Sets the IP address and port of servers to be used when performing DNS

\* resolution. The `servers` argument is an array of [RFC 5952](https://tools.ietf.org/html/rfc5952#section-6) formatted

\* addresses. If the port is the IANA default DNS port (53) it can be omitted.

\*

\* ```js

\* dnsPromises.setServers([

\* '4.4.4.4',

\* '[2001:4860:4860::8888]',

\* '4.4.4.4:1053',

\* '[2001:4860:4860::8888]:1053',

\* ]);

\* ```

\*

\* An error will be thrown if an invalid address is provided.

\*

\* The `dnsPromises.setServers()` method must not be called while a DNS query is in

\* progress.

\*

\* This method works much like [resolve.conf](https://man7.org/linux/man-pages/man5/resolv.conf.5.html).

\* That is, if attempting to resolve with the first server provided results in a`NOTFOUND` error, the `resolve()` method will \_not\_ attempt to resolve with

\* subsequent servers provided. Fallback DNS servers will only be used if the

\* earlier ones time out or result in some other error.

\* @since v10.6.0

\* @param servers array of `RFC 5952` formatted addresses

\*/

function setServers(servers: ReadonlyArray<string>): void;

/\*\*

\* Set the default value of `verbatim` in `dns.lookup()` and `dnsPromises.lookup()`. The value could be:

\*

\* \* `ipv4first`: sets default `verbatim` `false`.

\* \* `verbatim`: sets default `verbatim` `true`.

\*

\* The default is `ipv4first` and `dnsPromises.setDefaultResultOrder()` have

\* higher priority than `--dns-result-order`. When using `worker threads`,`dnsPromises.setDefaultResultOrder()` from the main thread won't affect the

\* default dns orders in workers.

\* @since v16.4.0, v14.18.0

\* @param order must be `'ipv4first'` or `'verbatim'`.

\*/

function setDefaultResultOrder(order: 'ipv4first' | 'verbatim'): void;

class Resolver {

constructor(options?: ResolverOptions);

cancel(): void;

getServers: typeof getServers;

resolve: typeof resolve;

resolve4: typeof resolve4;

resolve6: typeof resolve6;

resolveAny: typeof resolveAny;

resolveCname: typeof resolveCname;

resolveMx: typeof resolveMx;

resolveNaptr: typeof resolveNaptr;

resolveNs: typeof resolveNs;

resolvePtr: typeof resolvePtr;

resolveSoa: typeof resolveSoa;

resolveSrv: typeof resolveSrv;

resolveTxt: typeof resolveTxt;

reverse: typeof reverse;

setLocalAddress(ipv4?: string, ipv6?: string): void;

setServers: typeof setServers;

}

}

declare module 'node:dns/promises' {

export \* from 'dns/promises';

}