Connector

Undici creates the underlying socket via the connector builder. Normally, this happens automatically and you don't need to care about this, but if you need to perform some additional check over the currently used socket, this is the right place.

If you want to create a custom connector, you must import the `buildConnector` utility.

Parameter: `buildConnector.BuildOptions`

Every Tls option, see here. Furthermore, the following options can be passed:

- **socketPath** `string | null` (optional) - Default: `null` - An IPC endpoint, either Unix domain socket or Windows named pipe.
- **maxCachedSessions** `number | null` (optional) - Default: `100` - Maximum number of TLS cached sessions. Use 0 to disable TLS session caching. Default: 100.
- **timeout** `number | null` (optional) - Default `10e3`
- **servername** `string | null` (optional)

Once you call `buildConnector`, it will return a connector function, which takes the following parameters.

Parameter: `connector.Options`

- **hostname** `string` (required)
- **host** `string` (optional)
- **protocol** `string` (required)
- **port** `number` (required)
- **servername** `string` (optional)

Basic example

```
'use strict'

import { Client, buildConnector } from 'undici'

const connector = buildConnector({ rejectUnauthorized: false })
const client = new Client('https://localhost:3000', {
  connect (opts, cb) {
    connector(opts, (err, socket) => {
      if (err) {
        cb(err)
      } else if (/* assertion */) {
        socket.destroy()
        cb(new Error('kaboom'))
      } else {
        cb(null, socket)
      }
    })
  }
})
```

Example: validate the CA fingerprint

```
'use strict'

import { Client, buildConnector } from 'undici'

const caFingerprint = 'FO:OB:AR'
const connector = buildConnector({ rejectUnauthorized: false })
const client = new Client('https://localhost:3000', {
  connect (opts, cb) {
    connector(opts, (err, socket) => {
      if (err) {
        cb(err)
      } else if (getIssuerCertificate(socket).fingerprint256 !== caFingerprint) {
        socket.destroy()
        cb(new Error('Fingerprint does not match or malformed certificate'))
      } else {
        cb(null, socket)
      }
    })
  }
})

client.request({
  path: '/',
  method: 'GET'
}, (err, data) => {
  if (err) throw err

  const bufs = []
  data.body.on('data', (buf) => {
    bufs.push(buf)
  })
  data.body.on('end', () => {
    console.log(Buffer.concat(bufs).toString('utf8'))
    client.close()
  })
})

function getIssuerCertificate (socket) {
  let certificate = socket.getPeerCertificate(true)
  while (certificate && Object.keys(certificate).length > 0) {
    // invalid certificate
    if (certificate.issuerCertificate == null) {
      return null
    }

    // We have reached the root certificate.
    // In case of self-signed certificates, `issuerCertificate` may be a circular
reference.
    if (certificate.fingerprint256 === certificate.issuerCertificate.fingerprint256)
```

```
{
      break
    }

    // continue the loop
    certificate = certificate.issuerCertificate
  }
  return certificate
}
```