Class: MockPool

Extends: `undici.Pool`

A mock Pool class that implements the Pool API and is used by MockAgent to intercept real requests and return mocked responses.

`new MockPool(origin, [options])`

Arguments:

- **origin** `string` - It should only include the **protocol, hostname, and port**.
- **options** `MockPoolOptions` - It extends the `Pool` options.

Returns: `MockPool`

Parameter: `MockPoolOptions`

Extends: `PoolOptions`

- **agent** `Agent` - the agent to associate this MockPool with.

Example - Basic MockPool instantiation

We can use MockAgent to instantiate a MockPool ready to be used to intercept specified requests. It will not do anything until registered as the agent to use and any mock request are registered.

```
import { MockAgent } from 'undici'

const mockAgent = new MockAgent()

const mockPool = mockAgent.get('http://localhost:3000')
```

Instance Methods

`MockPool.intercept(options)`

This method defines the interception rules for matching against requests for a MockPool or MockPool. We can intercept multiple times on a single instance.

When defining interception rules, all the rules must pass for a request to be intercepted. If a request is not intercepted, a real request will be attempted.

Matcher type	Condition to pass
`string`	Exact match against string
`RegExp`	Regex must pass
`Function`	Function must return true

Arguments:

- **options** `MockPoolInterceptOptions` - Interception options.

Returns: `MockInterceptor` corresponding to the input options.

Parameter: `MockPoolInterceptOptions`

- **path** `string | RegExp | (path: string) => boolean` - a matcher for the HTTP request path.
- **method** `string | RegExp | (method: string) => boolean` - a matcher for the HTTP request method.
- **body** `string | RegExp | (body: string) => boolean` - (optional) - a matcher for the HTTP request body.
- **headers** `Record<string, string | RegExp | (body: string) => boolean` > - (optional) - a matcher for the HTTP request headers. To be intercepted, a request must match all defined headers. Extra headers not defined here may (or may not) be included in the request and do not affect the interception in any way.

Return: `MockInterceptor`

We can define the behaviour of an intercepted request with the following options.

- **reply** `(statusCode: number, replyData: string | Buffer | object | MockInterceptor.MockResponseDataHandler, responseOptions?: MockResponseOptions) => MockScope` - define a reply for a matching request. You can define this as a callback to read incoming request data. Default for `responseOptions` is `{}`.
- **replyWithError** `(error: Error) => MockScope` - define an error for a matching request to throw.
- **defaultReplyHeaders** `(headers: Record<string, string>) => MockInterceptor` - define default headers to be included in subsequent replies. These are in addition to headers on a specific reply.
- **defaultReplyTrailers** `(trailers: Record<string, string>) => MockInterceptor` - define default trailers to be included in subsequent replies. These are in addition to trailers on a specific reply.
- **replyContentLength** `() => MockInterceptor` - define automatically calculated `content-length` headers to be included in subsequent replies.

The reply data of an intercepted request may either be a string, buffer, or JavaScript object. Objects are converted to JSON while strings and buffers are sent as-is.

By default, `reply` and `replyWithError` define the behaviour for the first matching request only. Subsequent requests will not be affected (this can be changed using the returned `MockScope`).

Parameter: `MockResponseOptions`

- **headers** `Record<string, string>` - headers to be included on the mocked reply.
- **trailers** `Record<string, string>` - trailers to be included on the mocked reply.

Return: `MockScope`

A `MockScope` is associated with a single `MockInterceptor`. With this, we can configure the default behaviour of a intercepted reply.

- **delay** `(waitInMs: number) => MockScope` - delay the associated reply by a set amount in ms.
- **persist** `() => MockScope` - any matching request will always reply with the defined response indefinitely.
- **times** `(repeatTimes: number) => MockScope` - any matching request will reply with the defined response a fixed amount of times. This is overridden by **persist**.

Example - Basic Mocked Request

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

// MockPool
const mockPool = mockAgent.get('http://localhost:3000')
mockPool.intercept({ path: '/foo' }).reply(200, 'foo')

const {
  statusCode,
  body
} = await request('http://localhost:3000/foo')

console.log('response received', statusCode) // response received 200

for await (const data of body) {
  console.log('data', data.toString('utf8')) // data foo
}
```

Example - Mocked request using reply data callbacks

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/echo',
  method: 'GET',
  headers: {
    'User-Agent': 'undici',
    Host: 'example.com'
  }
}).reply(200, ({ headers }) => ({ message: headers.get('message') }))

const { statusCode, body, headers } = await request('http://localhost:3000', {
  headers: {
    message: 'hello world!'
  }
})

console.log('response received', statusCode) // response received 200
console.log('headers', headers) // { 'content-type': 'application/json' }

for await (const data of body) {
```

```
  console.log('data', data.toString('utf8')) // { "message":"hello world!" }
}
```

Example - Mocked request using reply options callback

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/echo',
  method: 'GET',
  headers: {
    'User-Agent': 'undici',
    Host: 'example.com'
  }
}).reply(({ headers }) => ({ statusCode: 200, data: { message:
headers.get('message') }})))

const { statusCode, body, headers } = await request('http://localhost:3000', {
  headers: {
    message: 'hello world!'
  }
})

console.log('response received', statusCode) // response received 200
console.log('headers', headers) // { 'content-type': 'application/json' }

for await (const data of body) {
  console.log('data', data.toString('utf8')) // { "message":"hello world!" }
}
```

Example - Basic Mocked requests with multiple intercepts

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).reply(200, 'foo')

mockPool.intercept({
```

```
  path: '/hello',
  method: 'GET',
}).reply(200, 'hello')

const result1 = await request('http://localhost:3000/foo')

console.log('response received', result1.statusCode) // response received 200

for await (const data of result1.body) {
  console.log('data', data.toString('utf8')) // data foo
}

const result2 = await request('http://localhost:3000/hello')

console.log('response received', result2.statusCode) // response received 200

for await (const data of result2.body) {
  console.log('data', data.toString('utf8')) // data hello
}
```

Example - Mocked request with query body, request headers and response headers and trailers

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo?hello=there&see=ya',
  method: 'POST',
  body: 'form1=data1&form2=data2',
  headers: {
    'User-Agent': 'undici',
    Host: 'example.com'
  }
}).reply(200, { foo: 'bar' }, {
  headers: { 'content-type': 'application/json' },
  trailers: { 'Content-MD5': 'test' }
})

const {
  statusCode,
  headers,
  trailers,
  body
} = await request('http://localhost:3000/foo?hello=there&see=ya', {
    method: 'POST',
    body: 'form1=data1&form2=data2',
    headers: {
```

```
      foo: 'bar',
      'User-Agent': 'undici',
      Host: 'example.com'
    }
  })

console.log('response received', statusCode) // response received 200
console.log('headers', headers) // { 'content-type': 'application/json' }

for await (const data of body) {
  console.log('data', data.toString('utf8')) // '{"foo":"bar"}'
}

console.log('trailers', trailers) // { 'content-md5': 'test' }
```

Example - Mocked request using different matchers

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: /^GET$/,
  body: (value) => value === 'form=data',
  headers: {
    'User-Agent': 'undici',
    Host: /^example.com$/
  }
}).reply(200, 'foo')

const {
  statusCode,
  body
} = await request('http://localhost:3000/foo', {
  method: 'GET',
  body: 'form=data',
  headers: {
    foo: 'bar',
    'User-Agent': 'undici',
    Host: 'example.com'
  }
})

console.log('response received', statusCode) // response received 200

for await (const data of body) {
```

```
  console.log('data', data.toString('utf8')) // data foo
}
```

Example - Mocked request with reply with a defined error

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).replyWithError(new Error('kaboom'))

try {
  await request('http://localhost:3000/foo', {
    method: 'GET'
  })
} catch (error) {
  console.error(error) // Error: kaboom
}
```

Example - Mocked request with defaultReplyHeaders

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).defaultReplyHeaders({ foo: 'bar' })
  .reply(200, 'foo')

const { headers } = await request('http://localhost:3000/foo')

console.log('headers', headers) // headers { foo: 'bar' }
```

Example - Mocked request with defaultReplyTrailers

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
```

```
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).defaultReplyTrailers({ foo: 'bar' })
  .reply(200, 'foo')

const { trailers } = await request('http://localhost:3000/foo')

console.log('trailers', trailers) // trailers { foo: 'bar' }
```

Example - Mocked request with automatic content-length calculation

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).replyContentLength().reply(200, 'foo')

const { headers } = await request('http://localhost:3000/foo')

console.log('headers', headers) // headers { 'content-length': '3' }
```

Example - Mocked request with automatic content-length calculation on an object

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).replyContentLength().reply(200, { foo: 'bar' })

const { headers } = await request('http://localhost:3000/foo')

console.log('headers', headers) // headers { 'content-length': '13' }
```

Example - Mocked request with persist enabled

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).reply(200, 'foo').persist()

const result1 = await request('http://localhost:3000/foo')
// Will match and return mocked data

const result2 = await request('http://localhost:3000/foo')
// Will match and return mocked data

// Etc
```

Example - Mocked request with times enabled

```
import { MockAgent, setGlobalDispatcher, request } from 'undici'

const mockAgent = new MockAgent()
setGlobalDispatcher(mockAgent)

const mockPool = mockAgent.get('http://localhost:3000')

mockPool.intercept({
  path: '/foo',
  method: 'GET'
}).reply(200, 'foo').times(2)

const result1 = await request('http://localhost:3000/foo')
// Will match and return mocked data

const result2 = await request('http://localhost:3000/foo')
// Will match and return mocked data

const result3 = await request('http://localhost:3000/foo')
// Will not match and make attempt a real request
```

`MockPool.close()`

Closes the mock pool and de-registers from associated MockAgent.

Returns: `Promise<void>`

Example - clean up after tests are complete

```
import { MockAgent } from 'undici'

const mockAgent = new MockAgent()
const mockPool = mockAgent.get('http://localhost:3000')

await mockPool.close()
```

`MockPool.dispatch(options, handlers)`

Implements `Dispatcher.dispatch(options, handlers)`.

`MockPool.request(options[, callback])`

See `Dispatcher.request(options [, callback])`.

Example - MockPool request

```
import { MockAgent } from 'undici'

const mockAgent = new MockAgent()

const mockPool = mockAgent.get('http://localhost:3000')
mockPool.intercept({
  path: '/foo',
  method: 'GET',
}).reply(200, 'foo')

const {
  statusCode,
  body
} = await mockPool.request({
  origin: 'http://localhost:3000',
  path: '/foo',
  method: 'GET'
})

console.log('response received', statusCode) // response received 200

for await (const data of body) {
  console.log('data', data.toString('utf8')) // data foo
}
```