TChannel Documentation

Release

Uber Technologies, Inc.

June 11, 2018

Contents

A Python implementation of TChannel.

CHAPTER 1

Getting Started

This guide is current as of version 0.18.0. See the *Upgrade Guide* if you're running an older version.

The code matching this guide is here.

Initial Setup

Create a directory called `keyvalue` to work inside of:

```
$ mkdir ~/keyvalue
$ cd ~/keyvalue
```

Inside of this directory we're also going to create a `keyvalue` module, which requires an `__init__.py` and a `setup.py` at the root:

```
$ mkdir keyvalue
$ touch keyvalue/__init__.py
```

Setup a virtual environment for your service and install the Tornado and Tchannel packages:

```
$ virtualenv env
$ source env/bin/activate
$ pip install 'tchannel<0.19'
```

Thrift Interface Definition

Create a Thrift file under `thrift/keyvalue.thrift` that defines an interface for your service:

```
$ mkdir thrift
$ vim thrift/keyvalue.thrift
$ cat thrift/keyvalue.thrift
```

```
exception NotFoundError {
    1: required string key,
}

service KeyValue {
    string getValue(
        1: string key,
    ) throws (
```

```
        1: NotFoundError notFound,
    )

    void setValue(
        1: string key,
        2: string value,
    )
}
```

This defines a service named `KeyValue` with two functions:

`getValue` a function which takes one string parameter, and returns a string.

`setValue` a void function that takes in two parameters.

Thrift Types

TChannel has some custom behavior so it can't use the code generated by the Apache Thrift code generator. Instead we're going to dynamically generate our Thrift types.

Open up `keyvalue/thrift.py`:

```
$ cat > keyvalue/thrift.py
from tchannel import thrift

service = thrift.load(path='thrift/keyvalue.thrift', service='keyvalue')
```

Let's make sure everything is working:

```
$ python -m keyvalue.thrift
```

You shouldn't see any errors. A lot of magic just happened :)

Python Server

To serve an application we need to instantiate a TChannel instance, which we will register handlers against. Open up `keyvalue/server.py` and write something like this:

```
from __future__ import absolute_import

from tornado import ioloop
from tornado import gen

from tchannel import TChannel

from keyvalue.thrift import service


tchannel = TChannel('keyvalue-server')


@tchannel.thrift.register(service.KeyValue)
def getValue(request):
    pass
```

```
@tchannel.thrift.register(service.KeyValue)
def setValue(request):
    pass


def run():
    tchannel.listen()
    print('Listening on %s' % tchannel.hostport)


if __name__ == '__main__':
    run()
    ioloop.IOLoop.current().start()
```

Here we have created a TChannel instance and registered two no-op handlers with it. The name of these handlers map directly to the Thrift service we defined earlier.

A TChannel server only has one requirement: a name for itself. By default an ephemeral port will be chosen to listen on (although an explicit port can be provided).

(As your application becomes more complex, you won't want to put everything in a single file like this. Good code structure is beyond the scope of this guide.)

Let's make sure this server is in a working state:

```
python -m keyvalue.server
Listening on localhost:8889
^C
```

The process should hang until you kill it, since it's listening for requests to handle. You shouldn't get any exceptions.

Handlers

To implement our service's endpoints let's create an in-memory dictionary that our endpoints will manipulate:

```
values = {}


@tchannel.thrift.register(service.KeyValue)
def getValue(request):
    key = request.body.key
    value = values.get(key)

    if value is None:
        raise service.NotFoundError(key)

    return value


@tchannel.thrift.register(service.KeyValue)
def setValue(request):
    key = request.body.key
    value = request.body.value
    values[key] = value
```

You can see that the return value of `getValue` will be coerced into the expected Thrift shape. If we needed to return an additional field, we could accomplish this by returning a dictionary.

This example service doesn't do any network IO work. If we wanted to take advantage of Tornado's asynchronous capabilities, we could define our handlers as coroutines and yield to IO operations:

```
@tchannel.register(service.KeyValue)
@gen.coroutine
def setValue(request):
    key = request.body.key
    value = request.body.value

    # Simulate some non-blocking IO work.
    yield gen.sleep(1.0)

    values[key] = value
```

Transport Headers

In addition to the call arguments and headers, the `request` object also provides some additional information about the current request under the `request.transport` object:

`transport.flags` Request flags used by the protocol for fragmentation and streaming.

`transport.ttl` The time (in milliseconds) within which the caller expects a response.

`transport.headers` Protocol level headers for the request. For more information on transport headers check the Transport Headers section of the protocol document.

Hyperbahn

As mentioned earlier, our service is listening on an ephemeral port, so we are going to register it with the Hyperbahn routing mesh. Clients will use this Hyperbahn mesh to determine how to communicate with your service.

Let's change our *run* method to advertise our service with a local Hyperbahn instance:

```
import json
import os

@gen.coroutine
def run():

    tchannel.listen()
    print('Listening on %s' % tchannel.hostport)

    if os.path.exists('/path/to/hyperbahn_hostlist.json'):
        with open('/path/to/hyperbahn_hostlist.json', 'r') as f:
            hyperbahn_hostlist = json.load(f)
        yield tchannel.advertise(routers=hyperbahn_hostlist)
```

The *advertise* method takes a seed list of Hyperbahn routers and the name of the service that clients will call into. After advertising, the Hyperbahn will connect to your process and establish peers for service-to-service communication.

Consult the Hyperbahn documentation for instructions on how to start a process locally.

Debugging

Let's spin up the service and make a request to it through Hyperbahn. Python provides `tcurl.py` script, but we need to use the Node version for now since it has Thrift support.

```
$ python keyvalue/server.py &
$ tcurl -H /path/to/hyperbahn_host_list.json -t ~/keyvalue/thrift/keyvalue.thrift keyvalue-server Key
$ tcurl -H /path/to/hyperbahn_host_list.json -t ~/keyvalue/thrift/keyvalue.thrift keyvalue-server Key
$ tcurl -H /path/to/hyperbahn_host_list.json -t ~/keyvalue/thrift/keyvalue.thrift keyvalue-server Key
```

Your service can now be accessed from any language over Hyperbahn + TChannel!

Python Client

Let's make a client call from Python in `keyvalue/client.py`:

```
from tornado import gen, ioloop
from tchannel import TChannel, thrift

tchannel = TChannel('keyvalue-consumer')
service = thrift.load(
    path='examples/guide/keyvalue/service.thrift',
    service='keyvalue-server',
    hostport='localhost:8889',
)


@gen.coroutine
def run():

    yield tchannel.thrift(
        service.KeyValue.setValue("foo", "Hello, world!"),
    )

    response = yield tchannel.thrift(
        service.KeyValue.getValue("foo"),
    )

    print response.body


if __name__ == '__main__':
    ioloop.IOLoop.current().run_sync(run)
```

CHAPTER 2

API Documentation

TChannel

class `tchannel.`**`TChannel`**(*name*, *hostport=None*, *process_name=None*, *known_peers=None*, *trace=True*, *reuse_port=False*, *context_provider=None*, *tracer=None*)

Manages connections and requests to other TChannel services.

Usage for a JSON client/server:

```
tchannel = TChannel(name='foo')

@tchannel.json.register
def handler(request):
    return {'foo': 'bar'}

response = yield tchannel.json(
    service='some-service',
    endpoint='endpoint',
    headers={'req': 'headers'},
    body={'req': 'body'},
)
```

Variables

- **`thrift`** (`ThriftArgScheme`) – Make Thrift requests over TChannel and register Thrift handlers.
- **`json`** (`JsonArgScheme`) – Make JSON requests over TChannel and register JSON handlers.
- **`raw`** (`RawArgScheme`) – Make requests and register handles that pass raw bytes.

`__init__`(*name*, *hostport=None*, *process_name=None*, *known_peers=None*, *trace=True*, *reuse_port=False*, *context_provider=None*, *tracer=None*)

Note: In general only one `TChannel` instance should be used at a time. Multiple `TChannel` instances are not advisable and could result in undefined behavior.

Parameters

- **`name`** (*`string`*) – How this application identifies itself. This is the name callers will use to make contact, it is also what your downstream services will see in their metrics.

- **hostport** (*string*) – An optional host/port to serve on, e.g., `"127.0.0.1:5555`. If not provided an ephemeral port will be used. When advertising on Hyperbahn you callers do not need to know your port.

call(**args*, ***kwargs*)
Make low-level requests to TChannel services.

Note: Usually you would interact with a higher-level arg scheme like *tchannel.schemes.JsonArgScheme* or *tchannel.schemes.ThriftArgScheme*.

advertise(*routers=None*, *name=None*, *timeout=None*, *router_file=None*, *jitter=None*)
Advertise with Hyperbahn.

After a successful advertisement, Hyperbahn will establish long-lived connections with your application. These connections are used to load balance inbound and outbound requests to other applications on the Hyperbahn network.

Re-advertisement happens periodically after calling this method (every minute). Hyperbahn will eject us from the network if it doesn't get a re-advertise from us after 5 minutes.

This function may be called multiple times if it fails. If it succeeds, all consecutive calls are ignored.

Parameters

- **routers** (*list*) – A seed list of known Hyperbahn addresses to attempt contact with. Entries should be of the form `"host:port"`.
- **name** (*string*) – The name your application identifies itself as. This is usually unneeded because in the common case it will match the `name` you initialized the `TChannel` instance with. This is the identifier other services will use to make contact with you.
- **timeout** – The timeout (in sec) for the initial advertise attempt. Defaults to 30 seconds.
- **jitter** – Variance allowed in the interval per request. Defaults to 5 seconds. The jitter applies to the initial advertise request as well.
- **router_file** – The host file that contains the routers information. The file should contain a JSON stringified format of the routers parameter. Either routers or router_file should be provided. If both provided, a ValueError will be raised.

Returns A future that resolves to the remote server's response after the first advertise finishes.

Raises **TimeoutError** – When unable to make our first advertise request to Hyperbahn. Subsequent requests may fail but will be ignored.

class `tchannel.singleton.`**TChannel**
Maintain a single TChannel instance per-thread.

tchannel_cls
alias of `TChannel`

classmethod prepare(**args*, ***kwargs*)
Set arguments to be used when instantiating a TChannel instance.

Arguments are the same as *tchannel.TChannel.__init__()*.

classmethod reset(**args*, ***kwargs*)
Undo call to prepare, useful for testing.

classmethod get_instance()
Get a configured, thread-safe, singleton TChannel instance.

Returns tchannel.TChannel

class tchannel.**Request**(*body=None*, *headers=None*, *transport=None*, *endpoint=None*, *service=None*, *timeout=None*)

A TChannel request.

This is sent by callers and received by registered handlers.

Variables

- **body** – The payload of this request. The type of this attribute depends on the scheme being used (e.g., JSON, Thrift, etc.).
- **headers** – A dictionary of application headers. This should be a mapping of strings to strings.
- **transport** – Protocol-level transport headers. These are used for routing over Hyperbahn.

  The most useful piece of information here is probably `request.transport.caller_name`, which is the identity of the application that created this request.
- **service** – Name of the service being called. Inside request handlers, this is usually the name of "this" service itself. However, for services that simply forward requests to other services, this is the name of the target service.
- **timeout** – Amount of time (in seconds) within which this request is expected to finish.

class tchannel.**Response**(*body=None*, *headers=None*, *transport=None*, *status=None*)

A TChannel response.

This is sent by handlers and received by callers.

Variables

- **body** – The payload of this response. The type of this attribute depends on the scheme being used (e.g., JSON, Thrift, etc.).
- **headers** – A dictionary of application headers. This should be a mapping of strings to strings.
- **transport** – Protocol-level transport headers. These are used for routing over Hyperbahn.

Serialization Schemes

Thrift

class tchannel.schemes.**ThriftArgScheme**(*tchannel*)

Handler registration and serialization for Thrift.

Use *tchannel.thrift.load()* to parse your Thrift IDL and compile it into a module dynamically.

```
from tchannel import thrift

keyvalue = thrift.load('keyvalue.thrift', service='keyvalue')
```

To register a Thrift handler, use the `register()` decorator, providing a reference to the compiled service as an argument. The name of the service method should match the name of the decorated function.

```
tchannel = TChannel(...)

@tchannel.thrift.register(keyvalue.KeyValue)
def setValue(request):
    data[request.body.key] = request.body.value
```

Use methods on the compiled service to generate requests to remote services and execute them via `TChannel.thrift()`.

```
response = yield tchannel.thrift(
    keyvalue.KeyValue.setValue(key='foo', value='bar')
)
```

__call__(**args*, ***kwargs*)

Make a Thrift TChannel request.

Returns a `Response` containing the return value of the Thrift call (if any). If the remote server responded with a Thrift exception, that exception is raised.

Parameters

- **request** (*string*) – Request obtained by calling a method on service objects generated by *tchannel.thrift.load()*.
- **headers** (*dict*) – Dictionary of header key-value pairs.
- **timeout** (*float*) – How long to wait (in seconds) before raising a `TimeoutError` - this defaults to `tchannel.glossary.DEFAULT_TIMEOUT`.
- **retry_on** (*string*) – What events to retry on - valid values can be found in `tchannel.retry`.
- **retry_limit** (*int*) – How many attempts should be made (in addition to the initial attempt) to re-send this request when retryable error conditions (specified by `retry_on`) are encountered.

  Defaults to `tchannel.retry.DEFAULT_RETRY_LIMIT` (4).

  Note that the maximum possible time elapsed for a request is thus `(retry_limit + 1) * timeout`.
- **shard_key** (*string*) – Set the `sk` transport header for Ringpop request routing.
- **trace** (*int*) – Flags for tracing.
- **hostport** (*string*) – A 'host:port' value to use when making a request directly to a TChannel service, bypassing Hyperbahn. This value takes precedence over the `hostport` specified to *tchannel.thrift.load()*.
- **routing_delegate** – Name of a service to which the request router should forward the request instead of the service specified in the call req.
- **caller_name** – Name of the service making the request. Defaults to the name provided when the TChannel was instantiated.

Return type *Response*

`tchannel.thrift.`**load**(*path*, *service=None*, *hostport=None*, *module_name=None*)

Loads the Thrift file at the specified path.

The file is compiled in-memory and a Python module containing the result is returned. It may be used with `TChannel.thrift`. For example,

```
from tchannel import TChannel, thrift

# Load our server's interface definition.
donuts = thrift.load(path='donuts.thrift')

# We need to specify a service name or hostport because this is a
# downstream service we'll be calling.
coffee = thrift.load(path='coffee.thrift', service='coffee')

tchannel = TChannel('donuts')

@tchannel.thrift.register(donuts.DonutsService)
@tornado.gen.coroutine
def submitOrder(request):
    args = request.body

    if args.coffee:
        yield tchannel.thrift(
            coffee.CoffeeService.order(args.coffee)
        )

    # ...
```

The returned module contains, one top-level type for each struct, enum, union, exeption, and service defined in the Thrift file. For each service, the corresponding class contains a classmethod for each function defined in that service that accepts the arguments for that function and returns a `ThriftRequest` capable of being sent via `TChannel.thrift`.

For more information on what gets generated by `load`, see thriftrw.

Note that the `path` accepted by `load` must be either an absolute path or a path relative to the *the current directory*. If you need to refer to Thrift files relative to the Python module in which `load` was called, use the `__file__` magic variable.

```
# Given,
#
#   foo/
#     myservice.thrift
#     bar/
#       x.py
#
# Inside foo/bar/x.py,

path = os.path.join(
    os.path.dirname(__file__), '../myservice.thrift'
)
```

The returned value is a valid Python module. You can install the module by adding it to the `sys.modules` dictionary. This will allow importing items from this module directly. You can use the `__name__` magic variable to make the generated module a submodule of the current module. For example,

```
# foo/bar.py

import sys
from tchannel import thrift

donuts = = thrift.load('donuts.thrift')
sys.modules[__name__ + '.donuts'] = donuts
```

This installs the module generated for `donuts.thrift` as the module `foo.bar.donuts`. Callers can then import items from that module directly. For example,

```
# foo/baz.py

from foo.bar.donuts import DonutsService, Order

def baz(tchannel):
    return tchannel.thrift(
        DonutsService.submitOrder(Order(..))
    )
```

Parameters

- **service** (*str*) – Name of the service that the Thrift file represents. This name will be used to route requests through Hyperbahn.
- **path** (*str*) – Path to the Thrift file. If this is a relative path, it must be relative to the current directory.
- **hostport** (*str*) – Clients can use this to specify the hostport at which the service can be found. If omitted, TChannel will route the requests through known peers. This value is ignored by servers.
- **module_name** (*str*) – Name used for the generated Python module. Defaults to the name of the Thrift file.

tchannel.**thrift_request_builder**(**args*, ***kwargs*)

Provide TChannel compatibility with Thrift-generated modules.

The service this creates is meant to be used with TChannel like so:

```
from tchannel import TChannel, thrift_request_builder
from some_other_service_thrift import some_other_service

tchannel = TChannel('my-service')

some_service = thrift_request_builder(
    service='some-other-service',
    thrift_module=some_other_service
)

resp = tchannel.thrift(
    some_service.fetchPotatoes()
)
```

Deprecated since version 0.18.0: Please switch to *tchannel.thrift.load()*.

Warning: This API is deprecated and will be removed in a future version.

Parameters

- **service** (*string*) – Name of Thrift service to call. This is used internally for grouping and stats, but also to route requests over Hyperbahn.
- **thrift_module** – The top-level module of the Apache Thrift generated code for the service that will be called.
- **hostport** (*string*) – When calling the Thrift service directly, and not over Hyperbahn, this 'host:port' value should be provided.

- **thrift_class_name** (*string*) – When the Apache Thrift generated Iface class name does not match thrift_module, then this should be provided.

JSON

class tchannel.schemes.**JsonArgScheme**(*tchannel*)
Semantic params and serialization for json.

__call__(**args*, ***kwargs*)
Make JSON TChannel Request.

Parameters

- **service** (*string*) – Name of the service to call.
- **endpoint** (*string*) – Endpoint to call on service.
- **body** (*string*) – A raw body to provide to the endpoint.
- **headers** (*dict*) – Dictionary of header key-value pairs.
- **timeout** (*float*) – How long to wait (in seconds) before raising a `TimeoutError` - this defaults to `tchannel.glossary.DEFAULT_TIMEOUT`.
- **retry_on** (*string*) – What events to retry on - valid values can be found in `tchannel.retry`.
- **retry_limit** (*int*) – How many attempts should be made (in addition to the initial attempt) to re-send this request when retryable error conditions (specified by `retry_on`) are encountered.

  Defaults to `tchannel.retry.DEFAULT_RETRY_LIMIT` (4).

  Note that the maximum possible time elapsed for a request is thus `(retry_limit + 1) * timeout`.
- **hostport** (*string*) – A 'host:port' value to use when making a request directly to a TChannel service, bypassing Hyperbahn.
- **routing_delegate** – Name of a service to which the request router should forward the request instead of the service specified in the call req.
- **caller_name** – Name of the service making the request. Defaults to the name provided when the TChannel was instantiated.

Return type *Response*

Raw

class tchannel.schemes.**RawArgScheme**(*tchannel*)
Semantic params and serialization for raw.

__call__(**args*, ***kwargs*)
Make a raw TChannel request.

The request's headers and body are treated as raw bytes and not serialized/deserialized.

The request's headers and body are treated as raw bytes and not serialized/deserialized.

Parameters

- **service** (*string*) – Name of the service to call.

- **endpoint** (*string*) – Endpoint to call on service.
- **body** (*string*) – A raw body to provide to the endpoint.
- **headers** (*string*) – A raw headers block to provide to the endpoint.
- **timeout** (*float*) – How long to wait (in seconds) before raising a `TimeoutError` - this defaults to `tchannel.glossary.DEFAULT_TIMEOUT`.
- **retry_on** (*string*) – What events to retry on - valid values can be found in `tchannel.retry`.
- **retry_limit** (*int*) – How many attempts should be made (in addition to the initial attempt) to re-send this request when retryable error conditions (specified by `retry_on`) are encountered.

  Defaults to `tchannel.retry.DEFAULT_RETRY_LIMIT` (4).

  Note that the maximum possible time elapsed for a request is thus `(retry_limit + 1) * timeout`.
- **hostport** (*string*) – A 'host:port' value to use when making a request directly to a TChannel service, bypassing Hyperbahn.
- **routing_delegate** – Name of a service to which the request router should forward the request instead of the service specified in the call req.
- **caller_name** – Name of the service making the request. Defaults to the name provided when the TChannel was instantiated.

Return type *Response*

Exception Handling

Errors

`tchannel.errors.`**TIMEOUT = 1**
The request timed out.

`tchannel.errors.`**CANCELED = 2**
The request was canceled.

`tchannel.errors.`**BUSY = 3**
The server was busy.

`tchannel.errors.`**BAD_REQUEST = 6**
The request was bad.

`tchannel.errors.`**NETWORK_ERROR = 7**
There was a network error when sending the request.

`tchannel.errors.`**UNHEALTHY = 8**
The server handling the request is unhealthy.

`tchannel.errors.`**FATAL = 255**
There was a fatal protocol-level error.

exception `tchannel.errors.`**TChannelError**(*description=None*, *id=None*, *tracing=None*)
Bases: `exceptions.Exception`

A TChannel-generated exception.

Variables **code** – The error code for this error. See the Specification for a description of these codes.

classmethod from_code(*code*, ***kw*)
Construct a TChannelError instance from an error code.

This will return the appropriate class type for the given code.

exception tchannel.errors.**RetryableError**(*description=None*, *id=None*, *tracing=None*)
Bases: *tchannel.errors.TChannelError*

An error where the original request is always safe to retry.

It is always safe to retry a request with this category of errors. The original request was never handled.

exception tchannel.errors.**MaybeRetryableError**(*description=None*, *id=None*, *tracing=None*)
Bases: *tchannel.errors.TChannelError*

An error where the original request may be safe to retry.

The original request may have reached the intended service. Hence, the request should only be retried if it is known to be idempotent.

exception tchannel.errors.**NotRetryableError**(*description=None*, *id=None*, *tracing=None*)
Bases: *tchannel.errors.TChannelError*

An error where the original request should not be re-sent.

Something was fundamentally wrong with the request and it should not be retried.

exception tchannel.errors.**ReadError**(*description=None*, *id=None*, *tracing=None*)
Bases: tchannel.errors.FatalProtocolError

Raised when there is an error while reading input.

exception tchannel.errors.**InvalidChecksumError**(*description=None*, *id=None*, *tracing=None*)
Bases: tchannel.errors.FatalProtocolError

Represent invalid checksum type in the message

exception tchannel.errors.**NoAvailablePeerError**(*description=None*, *id=None*, *tracing=None*)
Bases: *tchannel.errors.RetryableError*

Represents a failure to find any peers for a request.

exception tchannel.errors.**AlreadyListeningError**(*description=None*, *id=None*, *tracing=None*)
Bases: tchannel.errors.FatalProtocolError

Raised when attempting to listen multiple times.

exception tchannel.errors.**OneWayNotSupportedError**(*description=None*, *id=None*, *tracing=None*)
Bases: tchannel.errors.BadRequestError

Raised when a one-way Thrift procedure is called.

exception tchannel.errors.**ValueExpectedError**(*description=None*, *id=None*, *tracing=None*)
Bases: tchannel.errors.BadRequestError

Raised when a non-void Thrift response contains no value.

exception tchannel.errors.**SingletonNotPreparedError**(*description=None*, *id=None*, *tracing=None*)
Bases: *tchannel.errors.TChannelError*

Raised when calling get_instance before calling prepare.

exception tchannel.errors.**ServiceNameIsRequiredError**
Bases: exceptions.Exception

Raised when service name is empty or None.

Retry Behavior

These values can be passed as the retry_on behavior to *tchannel.TChannel.call()*.

tchannel.retry.**CONNECTION_ERROR = u'c'**
Retry the request on failures to connect to a remote host. This is the default retry behavior.

tchannel.retry.**NEVER = u'n'**
Never retry the request.

tchannel.retry.**TIMEOUT = u't'**
Retry the request on timeouts waiting for a response.

tchannel.retry.**CONNECTION_ERROR_AND_TIMEOUT = u'ct'**
Retry the request on failures to connect and timeouts after connecting.

tchannel.retry.**DEFAULT_RETRY_LIMIT = 4**
The default number of times to retry a request. This is in addition to the original request.

Synchronous Client

class tchannel.sync.**TChannel**(*name*, *hostport=None*, *process_name=None*, *known_peers=None*, *trace=False*, *threadloop=None*)
Make synchronous TChannel requests.

This client does not support incoming requests – it is a uni-directional client only.

The client is implemented on top of the Tornado-based implementation and offloads IO to a thread running an IOLoop next to your process.

Usage mirrors the TChannel class.

```
from tchannel.sync import TChannel

tchannel = TChannel(name='my-synchronous-service')

# Advertise with Hyperbahn.
# This returns a future. You may want to block on its result,
# particularly if you want you app to die on unsuccessful
# advertisement.
tchannel.advertise(routers)

# keyvalue is the result of a call to ``tchannel.thrift.load``.
future = tchannel.thrift(
    keyvalue.KeyValue.getItem('foo'),
    timeout=0.5,  #  0.5 seconds
)

result = future.result()
```

Fanout can be accomplished by using as_completed from the concurrent.futures module:

```
    from concurrent.futures import as_completed

    from tchannel.sync import TChannel

    tchannel = TChannel(name='my-synchronous-service')

    futures = [
        tchannel.thrift(service.getItem(item))
        for item in ('foo', 'bar')
    ]

    for future in as_completed(futures):
        print future.result()
```

(`concurrent.futures` is native to Python 3; `pip install futures` if you're using Python 2.x.)

advertise(*routers=None*, *name=None*, *timeout=None*, *router_file=None*, *jitter=None*)

Advertise with Hyperbahn.

After a successful advertisement, Hyperbahn will establish long-lived connections with your application. These connections are used to load balance inbound and outbound requests to other applications on the Hyperbahn network.

Re-advertisement happens periodically after calling this method (every minute). Hyperbahn will eject us from the network if it doesn't get a re-advertise from us after 5 minutes.

This function may be called multiple times if it fails. If it succeeds, all consecutive calls are ignored.

Parameters

- **routers** (*list*) – A seed list of known Hyperbahn addresses to attempt contact with. Entries should be of the form `"host:port"`.
- **name** (*string*) – The name your application identifies itself as. This is usually unneeded because in the common case it will match the `name` you initialized the `TChannel` instance with. This is the identifier other services will use to make contact with you.
- **timeout** – The timeout (in sec) for the initial advertise attempt. Defaults to 30 seconds.
- **jitter** – Variance allowed in the interval per request. Defaults to 5 seconds. The jitter applies to the initial advertise request as well.
- **router_file** – The host file that contains the routers information. The file should contain a JSON stringified format of the routers parameter. Either routers or router_file should be provided. If both provided, a ValueError will be raised.

Returns A future that resolves to the remote server's response after the first advertise finishes.

Raises **TimeoutError** – When unable to make our first advertise request to Hyperbahn. Subsequent requests may fail but will be ignored.

call(**args*, ***kwargs*)

Make low-level requests to TChannel services.

Note: Usually you would interact with a higher-level arg scheme like *tchannel.schemes.JsonArgScheme* or *tchannel.schemes.ThriftArgScheme*.

class tchannel.sync.singleton.**TChannel**

tchannel_cls

alias of `TChannel`

classmethod **get_instance**()
Get a configured, thread-safe, singleton TChannel instance.

Returns tchannel.sync.TChannel

prepare(**args*, ***kwargs*)
Set arguments to be used when instantiating a TChannel instance.

Arguments are the same as *tchannel.TChannel.__init__()*.

reset(**args*, ***kwargs*)
Undo call to prepare, useful for testing.

Testing

CHAPTER 3

FAQ

Can I register an endpoint that accepts all requests?

The fallback endpoint is the endpoint called when an unrecognized request is received. By default, the fallback endpoint simply returns a `BadRequestError` to the caller. This behavior may be changed by registering an endpoint with `TChannel.FALLBACK`.

```
from tchannel import TChannel

server = TChannel(name='myservice')

@server.register(TChannel.FALLBACK)
def handler(request):
    # ...
```

This may be used to implement a TChannel server that can handle requests to all endpoints. Note that for the fallback endpoint, you have access to the raw bytes of the headers and the body. These must be serialized/deserialized manually.

Why do I keep getting a "Cannot serialize MyType into a 'MyType"' error?

You are trying to mix code generated by Apache Thrift with the module generated by *`tchannel.thrift.load()`*. These are two separate ways of using Thrift with TChannel and the classes generated by either cannot be mixed and matched. You should be using only one of these approaches to interact with a specific service.

CHAPTER 4

Changelog

Changes by Version

1.3.2 (unreleased)

- Nothing changed yet.

1.3.1 (2018-06-11)

- Fixed a bug which caused servers to send requests to peers that sent requests to them.

1.3.0 (2017-11-20)

- Added OpenTracing client interceptor support for outbound requests.

1.2.0 (2017-10-19)

- Hook methods can now be implemented as coroutines.
- Added a new event (*before_serialize_request_headers*) that can be hooked. This is intended to allow application headers to be modified before requests are sent.

1.1.0 (2017-04-10)

- Added messages with ttl, service, and hostport information to TimeoutErrors

1.0.2 (2017-03-20)

- Fixed a race condition where the on_close callback for tchannel connections would not be called if the connection was already closed.
- Fixed a bug where the reference to the *next* node would not be cleared when nodes were pulled from message queues (Introducing a potential memory leak).

1.0.1 (2016-12-14)

- Add *str* functions to Peer and PeerClientOperation for easier debugging in exc_info
- Updated internal APIs to no longer depend on the PeerGroup *add* function and to use the *get* function for creating new peers instead.
- Fixed a bug where choosing a hostport directly for a downstream call would add that peer to the "core" peers which are used for regular calls. Now choosing the hostport directly will create a peer but will exclude it from selection.

1.0.0 (2016-11-17)

- Committing to existing API. We're calling this a 1.0.

0.30.6 (2016-11-14)

- Fixed a bug which would cause handshake timeouts to bubble up to the caller rather than retry a different peer.

0.30.5 (2016-11-10)

- Fixed a bug which would cause assertion errors if a connection to a peer disconnected shortly after a handshake.

0.30.4 (2016-11-03)

- Time out handshake attempts for outgoing connections after 5 seconds.
- Fixed a regression where large requests would block small requests until they were completely written to the wire.
- Propagate message sending errors up to the caller. This should greatly reduce the number of `TimeoutError: None` issues seen by users and show the root cause instead.
- Fail `TChannel` instantiation if the service name is empty or None.

0.30.3 (2016-10-24)

- Revert 0.30.2. The previous release may have introduced a memory leak.

0.30.2 (2016-10-12)

- Propagate message sending errors up to the caller. This should greatly reduce the number of `TimeoutError: None` issues seen by users and show the root cause instead.
- Fail `TChannel` instantiation if the service name is empty or None.

0.30.1 (2016-10-05)

- Relax opentracing upper bound to next major.
- Never send requests to ephemeral peers.

0.30.0 (2016-09-29)

- Pass span.kind tag when calling start_span(), not after the span was started.
- Add jitter argument to advertise().

0.29.1 (2016-10-05)

- Never send requests to ephemeral peers.
- Relax opentracing upper bound to next major.

0.29.0 (2016-09-12)

- Change default setting for tracing to be enabled.
- You can now specify an override for a request's `cn` transport header using the `caller_name` argument of the `call()`, `json()`, `raw()`, and `thrift()` methods of `TChannel`.

0.28.3 (2016-10-05)

- Never send requests to ephemeral peers.
- Relax opentracing upper bound to next major.

0.28.2 (2016-09-12)

- Bug fix: Tracing headers will no longer be added for raw requests if the headers are unparsed.

0.28.1 (2016-08-19)

- Ignore tracing fields with empty/zero trace ID.

0.28.0 (2016-08-17)

- Don't send more Hyperbahn advertise requests if an existing request is ongoing.
- Add jitter between Hyperbahn consecutive advertise requests.
- If the initial advertise request fails, propagate the original error instead of a timeout error.

0.27.4 (2016-10-05)

- Never send requests to ephemeral peers.
- Relax opentracing upper bound to next major.

0.27.3 (2016-08-19)

- Ignore tracing fields with empty/zero trace ID.

0.27.2 (2016-08-17)

- VCR should ignore tracing headers when matching requests. This will allow replaying requests with or without tracing regardless of whether the original request was recorded with it.

0.27.1 (2016-08-10)

- Bug fix: set Trace.parent_id to 0 if it's None

0.27.0 (2016-08-08)

- Native integration with OpenTracing (for real this time)
- Replace tcollector and explicit trace reporting with OpenTracing

0.26.1 (2016-10-05)

- Never send requests to ephemeral peers.

0.26.0 (2016-07-13)

- VCR: `use_cassette` now uses cached copies of cassettes if their contents have not changed. This should improve performance for large cassette files.

0.25.2 (2016-10-05)

- Never send requests to ephemeral peers.

0.25.1 (2016-06-30)

- Fixed a bug where the application error status code was not being copied into Response objects.

0.25.0 (2016-06-16)

- Support for OpenTracing.

0.24.1 (2016-10-05)

- Never send requests to ephemeral peers.

0.24.0 (2016-04-19)

- Added `TChannel.host` and `TChannel.port`.
- Added `TChannel.close()` and `TChannel.is_closed()`.

0.23.2 (2016-10-05)

- Never send requests to ephemeral peers.

0.23.1 (2016-04-14)

- Fixed tornado version constraint causing `reuse_port` to be missing, updated constraint to `tornado>=4.3,<5`.
- Only pass `reuse_port` to `bind_sockets` if it's set to True.

0.23.0 (2016-04-14)

- Added an opt-in feature to use the `SO_REUSEPORT` socket option for TChannel servers. Use `reuse_port=True` when instantiating a `TChannel`.

0.22.4 (2016-10-05)

- Never send requests to ephemeral peers.

0.22.3 (2016-04-07)

- Fixed a bug where type mismatch for timeouts could cause a crash.

0.22.2 (2016-04-06)

- VCR now respects the timeout specified on the original request. Timeouts in making the requests while recording now propagate as `TimeoutError` exceptions rather than `RemoteServiceError`.
- Reduced a warning for unconsumed error messages to info.
- Made UnexpectedError's message a little more debuggable.

0.22.1 (2016-04-06)

- Added a timeout to the VCR proxy call.
- Fixed a bug where tests would time out if the VCR server failed to start. The VCR server failure is now propagated to the caller.

0.22.0 (2016-03-31)

- Peer selection is now constant time instead of linear time. This should significantly reduce CPU load per request.
- Fixed a bug where certain errors while reading requests would propagate as TimeoutErrors.
- Attempting to register endpoints against a synchronous TChannel now logs an INFO level message.
- Reduced default advertisement interval to 3 minutes.

0.21.10 (2016-03-17)

- Zipkin traces now include a server-side 'cn' annotation to identify callers.
- Reduced "unconsumed message" warnings to INFO. These are typically generated when Hyperbahn garbage collects your process due to a timed-out advertisement.
- Handshake timeouts were incorrectly being surfaced as StreamClosedError but are now raised as NetworkError.
- Reduced default tracing sample rate from 100% to 1%.

0.21.9 (2016-03-14)

- Fixed a bug that caused silent failures when a write attempt was made to a closed connection.
- Reduce `StreamClosedError` log noisiness for certain scenarios.
- Make `TChannel.advertise` idempotent and thread-safe.

0.21.8 (2016-03-10)

- Reduce read errors due to clients disconnecting to INFO from ERROR.

0.21.7 (2016-03-08)

- Fixed an unhelpful stack trace on failed reads.

0.21.6 (2016-03-08)

- Fixed a logging error on failed reads.

0.21.5 (2016-03-08)

- Tornado 4.2 was listed as a requirement but this was corrected to be 4.3 which introduced the locks module.
- Fixed in issue where clients could incorrectly time out when reading large response bodies. This was due to response fragments being dropped due to out-of-order writes; writes are now serialized on a per-connection basis.

0.21.4 (2016-02-15)

- Fixed noisy logging of late responses for requests that timed out locally.

0.21.3 (2016-01-22)

- Attempting to register endpoints against a synchronous TChannel is now a no-op instead of a crash.

0.21.2 (2016-01-05)

- The synchronous client will no longer start a thread when the `TChannel` instance is initialized. This resolves an issue where an application could hang indefinitely if it instantiated a synchronous `TChannel` at import time.

0.21.1 (2015-12-29)

- Fixed a bug in Zipkin instrumentation that would cause CPU spikes due to an infinite loop during downstream requests.

0.21.0 (2015-12-10)

- Add support for zipkin trace sampling.
- `tchannel.TChannel.FALLBACK` may now be used to register fallback endpoints which are called for requests with unrecognized endpoints. For more information, see *Can I register an endpoint that accepts all requests?*
- Expose `timeout` and `service` attributes on `Request` objects inside endpoint handlers.
- Disable the retry for all zipkin trace submit.
- Fix Thrift service inheritance bug which caused parent methods to not be propagated to child services.
- VCR recording should not fail if the destination directory for the cassette does not exist.
- Fix bug which incorrectly encoded JSON arg scheme headers in the incorrect format.
- Add support for `rd` transport header.
- **BREAKING** - Support unit testing endpoints by calling the handler functions directly. This is enabled by changing `tchannel.thrift.register` to return the registered function unmodified. See Upgrade Guide for more details.

0.20.2 (2015-11-25)

- Lower the log level for Hyperbahn advertisement failures that can be retried.
- Include the full stack trace when Hyperbahn advertisement failures are logged.
- Include the error message for unexpected server side failures in the error returned to the client.

0.20.1 (2015-11-12)

- Fix bug which prevented requests from being retried if the candidate connection was previously terminated.

0.20.0 (2015-11-10)

- Support thriftrw 1.0.
- Drop explicit dependency on the `futures` library.

0.19.0 (2015-11-06)

- Add tchannel version & language information into init message header when initialize connections between TChannel instances.

0.18.3 (2015-11-03)

- Reduced Hyperbahn advertisement per-request timeout to 2 seconds.
- Removed an unncessary exception log for connection failures.

0.18.2 (2015-10-28)

- Reduced Hyperbahn advertisement failures to warnings.

0.18.1 (2015-10-28)

- Improved performance of peer selection logic.
- Fixed a bug which caused the message ID and tracing for incoming error frames to be ignored.
- Prefer using incoming connections on peers instead of outgoing connections.

0.18.0 (2015-10-20)

- Deprecated warnings will now sound for `tchannel.thrift.client_for`, `tchannel.thrift_request_builder`, and `tchannel.tornado.TChannel` - these APIs will be removed soon - be sure to move to `tchannel.thrift.load` in conjunction with `tchannel.TChannel`.
- Added singleton facility for maintaining a single TChannel instance per thread. See `tchannel.singleton.TChannel`, `tchannel.sync.singleton.TChannel`, or check the guide for an example how of how to use. Note this feature is optional.
- Added Thrift support to `tcurl.py` and re-worked the script's arguments.
- Specify which request components to match on with VCR, for example, 'header', 'body', etc. See `tchannel.testing.vcr.use_cassette`.
- Removed `tchannel.testing.data` module.
- Changed minimum required version of Tornado to 4.2.
- `tchannel.tornado.TChannel.close` is no longer a coroutine.
- **BREAKING** - headers for JSON handlers are not longer JSON blobs but are instead maps of strings to strings. This mirrors behavior for Thrift handlers.
- Fixed bug that caused server to continue listening for incoming connections despite closing the channel.
- Explicit destinations for `ThriftArgScheme` may now be specified on a per-request basis by using the `hostport` keyword argument.
- Only listen on IPv4, until official IPv6 support arrives.

0.17.11 (2015-10-19)

- Fix a bug that caused `after_send_error` event to never be fired.
- Request tracing information is now propagated to error responses.

0.17.10 (2015-10-16)

- Support thriftrw 0.5.

0.17.9 (2015-10-15)

- Fix default timeout incorrectly set to 16 minutes, now 30 seconds.

0.17.8 (2015-10-14)

- Revert timeout changes from 0.17.6 due to client incompatibilities.

0.17.7 (2015-10-14)

- Network failures while connecting to randomly selected hosts should be retried with other hosts.

0.17.6 (2015-10-14)

- Fixed an issue where timeouts were being incorrectly converted to seconds.

0.17.5 (2015-10-12)

- Set default checksum to `CRC32C`.

0.17.4 (2015-10-12)

- Updated `vcr` to use `thriftrw`-generated code. This should resolve some unicode errors during testing with `vcr`.

0.17.3 (2015-10-09)

- Fixed uses of `add_done_callback` that should have been `add_future`. This was preventing propper request/response interleaving.
- Added support for `thriftrw` 0.4.

0.17.2 (2015-09-18)

- VCR no longer matches on hostport to better support ephemeral ports.
- Fixed a bug with thriftrw where registering an endpoint twice could fail.

0.17.1 (2015-09-17)

- Made "service" optional for `thrift.load()`. The first argument should be a path, but backwards compatibility is provided for 0.17.0.

0.17.0 (2015-09-14)

- It is now possible to load Thrift IDL files directly with `tchannel.thrift.load`. This means that the code generation step using the Apache Thrift compiler can be skipped entirely. Check the API documentation for more details.
- Accept host file in advertise: `TChannel.advertise()` now accepts a parameter, `router_file` that contains a JSON stringified format of the router list.
- Add `TChannel.is_listening` method to return whether the tchannel instance is listening or not.

0.16.10 (2015-10-15)

- Fix default timeout incorrectly set to 16 minutes, now 30 seconds.

0.16.9 (2015-10-15)

- Network failures while connecting to randomly selected hosts should be retried with other hosts.

0.16.8 (2015-10-14)

- Revert timeout changes from 0.16.7 due to client incompatibilities.

0.16.7 (2015-10-14)

- Fixed an issue where timeouts were being incorrectly converted to seconds.

0.16.6 (2015-09-14)

- Fixed a bug where Zipkin traces were not being propagated correctly in services using the `tchannel.TChannel` API.

0.16.5 (2015-09-09)

- Actually fix status code being unset in responses when using the Thrift scheme.
- Fix request TTLs not being propagated over the wire.

0.16.4 (2015-09-09)

- Fix bug where status code was not being set correctly on call responses for application errors when using the Thrift scheme.

0.16.3 (2015-09-09)

- Make `TChannel.listen` thread-safe and idempotent.

0.16.2 (2015-09-04)

- Fix *retry_limit* in *TChannel.call* not allowing 0 retries.

0.16.1 (2015-08-27)

- Fixed a bug where the 'not found' handler would incorrectly return serialization mismatch errors..
- Fixed a bug which prevented VCR support from working with the sync client.
- Fixed a bug in VCR that prevented it from recording requests made by the sync client, and requests made with `hostport=None`.
- Made `client_for` compatible with `tchannel.TChannel`.
- Brought back `tchannel.sync.client_for` for backwards compatibility.

0.16.0 (2015-08-25)

- Introduced new server API through methods `tchannel.TChannel.thrift.register`, `tchannel.TChannel.json.register`, and `tchannel.TChannel.raw.register` - when these methods are used, endpoints are passed a `tchannel.Request` object, and are expected to return a `tchannel.Response` object or just a response body. The deprecated `tchannel.tornado.TChannel.register` continues to function how it did before. Note the breaking change to the top-level TChannel on the next line.
- Fixed a crash that would occur when forking with an unitialized `TChannel` instance.
- Add `hooks` property in the `tchannel.TChannel` class.
- **BREAKING** - `tchannel.TChannel.register` no longer has the same functionality as `tchannel.tornado.TChannel.register`, instead it exposes the new server API. See the upgrade guide for details.
- **BREAKING** - remove `retry_delay` option in the `tchannel.tornado.send` method.
- **BREAKING** - error types have been reworked significantly. In particular, the all-encompassing `ProtocolError` has been replaced with more granualar/actionable exceptions. See the upgrade guide for more info.
- **BREAKING** - Remove third `proxy` argument from the server handler interface.
- **BREAKING** - `ZipkinTraceHook` is not longer registered by default.
- **BREAKING** - `tchannel.sync.client.TChannelSyncClient` replaced with `tchannel.sync.TChannel`.

0.15.2 (2015-08-07)

- Raise informative and obvious `ValueError` when anything but a map[string]string is passed as headers to the `TChannel.thrift` method.
- First param, request, in `tchannel.thrift` method is required.

0.15.1 (2015-08-07)

- Raise `tchannel.errors.ValueExpectedError` when calling a non-void Thrift procedure that returns no value.

0.15.0 (2015-08-06)

- Introduced new top level `tchannel.TChannel` object, with new request methods `call`, `raw`, `json`, and `thrift`. This will eventually replace the akward `request` / `send` calling pattern.
- Introduced `tchannel.thrift_request_builder` function for creating a request builder to be used with the `tchannel.TChannel.thrift` function.
- Introduced new simplified examples under the `examples/simple` directory, moved the Guide's examples to `examples/guide`, and deleted the remaining examples.
- Added ThriftTest.thrift and generated Thrift code to `tchannel.testing.data` for use with examples and playing around with TChannel.
- Fix JSON arg2 (headers) being returned a string instead of a dict.

0.14.0 (2015-08-03)

- Implement VCR functionality for outgoing requests. Check the documentation for `tchannel.testing.vcr` for details.
- Add support for specifying fallback handlers via `TChannel.register` by specifying `TChannel.fallback` as the endpoint.
- Fix bug in `Response` where `code` expected an object instead of an integer.
- Fix bug in `Peer.close` where a future was expected instead of `None`.

0.13.0 (2015-07-23)

- Add support for specifying transport headers for Thrift clients.
- Always pass `shardKey` for TCollector tracing calls. This fixes Zipkin tracing for Thrift clients.

0.12.0 (2015-07-20)

- Add `TChannel.is_listening()` to determine if `listen` has been called.
- Calling `TChannel.listen()` more than once raises a `tchannel.errors.AlreadyListeningError`.
- `TChannel.advertise()` will now automatically start listening for connections if `listen()` has not already been called.
- Use `threadloop==0.4`.
- Removed `print_arg`.

0.11.2 (2015-07-20)

- Fix sync client's advertise - needed to call listen in thread.

0.11.1 (2015-07-17)

- Fix sync client using `0.0.0.0` host which gets rejected by Hyperbahn during advertise.

0.11.0 (2015-07-17)

- Added advertise support to sync client in `tchannel.sync.TChannelSyncClient.advertise`.
- **BREAKING** - renamed `router` argument to `routers` in `tchannel.tornado.TChannel.advertise`.

0.10.3 (2015-07-13)

- Support PyPy 2.
- Fix bugs in `TChannel.advertise`.

0.10.2 (2015-07-13)

- Made `TChannel.advertise` retry on all exceptions.

0.10.1 (2015-07-10)

- Previous release was broken with older versions of pip.

0.10.0 (2015-07-10)

- Add exponential backoff to `TChannel.advertise`.
- Make transport metadata available under `request.transport` on the server-side.

0.9.1 (2015-07-09)

- Use threadloop 0.3.* to fix main thread not exiting when `tchannel.sync.TChannelSyncClient` is used.

0.9.0 (2015-07-07)

- Allow custom handlers for unrecognized endpoints.
- Released `tchannel.sync.TChannelSyncClient` and `tchannel.sync.thrift.client_for`.

0.8.5 (2015-06-30)

- Add port parameter for `TChannel.listen`.

0.8.4 (2015-06-17)

- Fix bug where False and False-like values were being treated as None in Thrift servers.

0.8.3 (2015-06-15)

- Add `as` attribute to the response header.

0.8.2 (2015-06-11)

- Fix callable `traceflag` being propagated to the serializer.
- Fix circular imports.
- Fix `TimeoutError` retry logic.

0.8.1 (2015-06-10)

- Initial release.

Upgrade Guide

Migrating to a version of TChannel with breaking changes? This guide documents what broke and how to safely migrate to newer versions.

From 0.20 to 0.21

- `tchannel.thrift.register` returns the original function as-is instead of the wrapped version. This allows writing unit tests that call the handler function directly.

  Previously, if you used the `tchannel.thrift.register` decorator to register a Thrift endpoint and then called that function directly from a test, it would return a `Response` object if the call succeeded or failed with an expected exception (defined in the Thrift IDL). For example,

```
# service KeyValue {
#   string getValue(1: string key)
#       throws (1: ValidationError invalid)
# }

@tchannel.thrift.register(kv.KeyValue)
def getValue(request):
    key = request.body.key
    if key == 'invalid':
        raise kv.ValidationError()
    result = # ...
    return result

response = getValue(make_request(key='invalid'))
if response.body.invalid:
    # ...
else:
    result = response.body.success
```

  With 0.21, we have changed `tchannel.thrift.register` to return the unmodified function so that you can call it directly and it will behave as expected.

```
@tchannel.thrift.register(kv.KeyValue)
def getValue(request):
    # ...

try:
    result = getValue(make_request(key='invalid'))
except kv.ValidationError:
    # ...
```

From 0.19 to 0.20

- No breaking changes.

From 0.18 to 0.19

- No breaking changes.

From 0.17 to 0.18

- `request.headers` in a JSON handler is no longer a JSON blob. Instead it is a dictionary mapping strings to strings. This matches the Thrift implementation. If your headers include richer types like lists or ints, you'll need to coordinate with your callers to no longer pass headers as JSON blobs. The same applies to JSON requsts; rich headers will now fail to serialize.
- If you were accessing `request_cls` or `response_cls` directly from a service method in a module generated by `tchannel.thrift.load`, you can no longer do that. The `request_cls` and `response_cls` attributes are internal details of the implementation and have been changed to protected. You should only ever use the service method directly.

  Before:

```
my_service.doSomething.request_cls(..)
```

  After:

```
my_service.doSomething(..)
```

  Note that `request_cls` gives you just an object containing the method arguments. It does not include any of the other information needed to make the request. So if you were using it to make requests, it wouldn't have worked anyway.

From 0.16 to 0.17

- No breaking changes.

From 0.15 to 0.16

- `tchannel.TChannel.register` no longer mimicks `tchannel.tornado.TChannel.register`, instead it exposes the new server API like so:

  Before:

```
from tchannel.tornado import TChannel

tchannel = TChannel('my-service-name')

@tchannel.register('endpoint', 'json')
def endpoint(request, response, proxy):
    response.write({'resp': 'body'})
```

After:

```
from tchannel import TChannel

tchannel = TChannel('my-service-name')

@tchannel.json.register
def endpoint(request):
    return {'resp': 'body'}

    # Or, if you need to return headers with your response:
    from tchannel import Response
    return Response({'resp': 'body'}, {'header': 'foo'})
```

- `TChannelSyncClient` has been replaced with `tchannel.sync.TChannel`. This new synchronous client has been significantly re-worked to more closely match the asynchronous `TChannel` API. `tchannel.sync.thrift.client_for` has been removed and `tchannel.thrift_request_builder` should be used instead (`tchannel.thrift.client_for` still exists for backwards compatibility but is not recommended). This new API allows specifying headers, timeouts, and retry behavior with Thrift requests.

  Before:

```
from tchannel.sync import TChannelSyncClient
from tchannel.sync.thrift import client_for

from generated.thrift.code import MyThriftService

tchannel_thrift_client = client_for('foo', MyThriftService)

tchannel = TChannelSyncClient(name='bar')

future = tchannel_thrift_client.someMethod(...)

result = future.result()
```

  After:

```
from tchannel import thrift_request_builder
from tchannel.sync import TChannel
from tchannel.retry import CONNECTION_ERROR_AND_TIMEOUT

from generated.thrift.code import MyThriftService

tchannel_thrift_client = thrift_request_builder(
    service='foo',
    thrift_module=MyThriftService,
)

tchannel = TChannel(name='bar')
```

```
    future = tchannel.thrift(
        tchannel_thrift_client.someMethod(...)
        headers={'foo': 'bar'},
        retry_on=CONNECTION_ERROR_AND_TIMEOUT,
        timeout=1000,
    )

    result = future.result()
```

- `from tchannel.tornado import TChannel` is deprecated.
- Removed `retry_delay` option from `tchannel.tornado.peer.PeerClientOperation.send` method.

  Before: `tchannel.tornado.TChannel.request.send(retry_delay=300)`

  After: no more `retry_delay` in `tchannel.tornado.TChannel.request.send()`
- If you were catching `ProtocolError` you will need to catch a more specific type, such as `TimeoutError`, `BadRequestError`, `NetworkError`, `UnhealthyError`, or `UnexpectedError`.
- If you were catching `AdvertiseError`, it has been replaced by `TimeoutError`.
- If you were catching `BadRequest`, it may have been masking checksum errors and fatal streaming errors. These are now raised as `FatalProtocolError`, but in practice should not need to be handled when interacting with a well-behaved TChannel implementation.
- `TChannelApplicationError` was unused and removed.
- Three error types have been introduced to simplify retry handling:
  - `NotRetryableError` (for requests should never be retried),
  - `RetryableError` (for requests that are always safe to retry), and
  - `MaybeRetryableError` (for requests that are safe to retry on idempotent endpoints).

From 0.14 to 0.15

- No breaking changes.

From 0.13 to 0.14

- No breaking changes.

From 0.12 to 0.13

- No breaking changes.

From 0.11 to 0.12

- Removed `print_arg`. Use `request.get_body()` instead.

From 0.10 to 0.11

- Renamed `tchannel.tornado.TChannel.advertise` argument `router` to `routers`. Since this is a required arg and the first positional arg, only clients who are using as kwarg will break.

  Before: `tchannel.advertise(router=['localhost:21300'])`

  After: `tchannel.advertise(routers=['localhost:21300'])`

t

Symbols

A

B

C

D

F

G

I

J

L

M

N

O

P

R

S

T

U

V