异步化

在前面的例子中，我们并没有对`RequestHandler`中的`get`或`post`方法进行异步处理，这就意味着，一旦在`get`或`post`方法中出现了耗时间的操作，不仅仅是当前请求被阻塞，按照Tornado框架的工作模式，其他的请求也会被阻塞，所以我们需要对耗时间的操作进行异步化处理。

在Tornado稍早一些的版本中，可以用装饰器实现请求方法的异步化或协程化来解决这个问题。

- 给`RequestHandler`的请求处理函数添加`@tornado.web.asynchronous`装饰器，如下所示：

```
class AsyncReqHandler(RequestHandler):

    @tornado.web.asynchronous
    def get(self):
       http = httpclient.AsyncHTTPClient()
       http.fetch("http://example.com/", self._on_download)

    def _on_download(self, response):
       do_something_with_response(response)
       self.render("template.html")
```

- 给`RequestHandler`的请求处理函数添加`@tornado.gen.coroutine`装饰器，如下所示：

```
class GenAsyncHandler(RequestHandler):

    @tornado.gen.coroutine
    def get(self):
        http_client = AsyncHTTPClient()
        response = yield http_client.fetch("http://example.com")
        do_something_with_response(response)
        self.render("template.html")
```

- 使用`@return_future`装饰器，如下所示：

```
@return_future
def future_func(arg1, arg2, callback):
    # Do stuff (possibly asynchronous)
    callback(result)

async def caller():
    await future_func(arg1, arg2)
```

在Tornado 5.x版本中，这几个装饰器都被标记为**deprcated**（过时），我们可以通过Python 3.5中引入的`async`和`await`（在Python 3.7中已经成为正式的关键字）来达到同样的效果。当然，要实现异步化还得靠其他的支持异步操作的三方库来支持，如果请求处理函数中用到了不支持异步操作的三方库，就需要靠自己写包装类来支持异步化。

下面的代码演示了在读写数据库时如何实现请求处理的异步化。我们用到的数据库建表语句如下所示：

```
create database hrs default charset utf8;

use hrs;
```

```
/* 创建部门表 */
create table tb_dept
(
    dno     int not null comment '部门编号',
    dname   varchar(10) not null comment '部门名称',
    dloc    varchar(20) not null comment '部门所在地',
    primary key (dno)
);

insert into tb_dept values
    (10, '会计部', '北京'),
    (20, '研发部', '成都'),
    (30, '销售部', '重庆'),
    (40, '运维部', '深圳');
```

我们通过下面的代码实现了查询和新增部门两个操作。

```
import json

import aiomysql
import tornado
import tornado.web

from tornado.ioloop import IOLoop
from tornado.options import define, parse_command_line, options

define('port', default=8000, type=int)


async def connect_mysql():
    return await aiomysql.connect(
        host='120.77.222.217',
        port=3306,
        db='hrs',
        user='root',
        password='123456',
    )


class HomeHandler(tornado.web.RequestHandler):

    async def get(self, no):
        async with self.settings['mysql'].cursor(aiomysql.DictCursor) as cursor:
            await cursor.execute("select * from tb_dept where dno=%s", (no, ))
            if cursor.rowcount == 0:
                self.finish(json.dumps({
                    'code': 20001,
                    'mesg': f'没有编号为{no}的部门'
                }))
                return
            row = await cursor.fetchone()
            self.finish(json.dumps(row))

    async def post(self, *args, **kwargs):
        no = self.get_argument('no')
        name = self.get_argument('name')
```

```
        loc = self.get_argument('loc')
        conn = self.settings['mysql']
        try:
            async with conn.cursor() as cursor:
                await cursor.execute('insert into tb_dept values (%s, %s, %s)',
                                     (no, name, loc))
            await conn.commit()
        except aiomysql.MySQLError:
            self.finish(json.dumps({
                'code': 20002,
                'mesg': '添加部门失败请确认部门信息'
            }))
        else:
            self.set_status(201)
            self.finish()


def make_app(config):
    return tornado.web.Application(
        handlers=[(r'/api/depts/(.*)', HomeHandler), ],
        **config
    )


def main():
    parse_command_line()
    app = make_app({
        'debug': True,
        'mysql': IOLoop.current().run_sync(connect_mysql)
    })
    app.listen(options.port)
    IOLoop.current().start()


if __name__ == '__main__':
    main()
```

上面的代码中，我们用到了`aiomysql`这个三方库，它基于`pymysql`封装，实现了对MySQL操作的异步化。操作Redis可以使用`aioredis`，访问MongoDB可以使用`motor`，这些都是支持异步操作的三方库。