华南师范大学
SOUTH CHINA NORMAL UNIVERSITY

实验报告

课程名称：_____

实验题目：_____

指导老师：_____

学生姓名：_____

学　　号：_____

学　　院：_____

专　　业：_____

班　　级：_____

时　　间：_____

目录

1. 实验目标

- 搭建服务器端的数据库系统软件，使之可以运行，并开放一定的访问权限。
- 任选一个 NoSQL 数据库，编写简单的客户端程序并通过网络访问。
- 基于 C/S 访问模式开发（客户端和服务器端包含独立 IP）。
- 服务器端要求全部在 Linux 环境下进行部署。

2. 实验内容及要求

- 在 Linux 下搭建服务器端，并部署运行传统数据库，包括但不限于：SCYLLADB、Memcached、Redis、VoltDB、MongoDB、Neo4j。
- 采用 C/C++、Java 或 Web 的方式来开发客户端。
- 实现数据插入、修改或删除等额外功能。
- 实现和该专用数据库相关的特色操作。

3. 基本设计

实验的主体任务分为两个，第一个为在 Linux 平台上搭建一个传统的数据库，并且部署好后端代码控制数据库，响应来自于客户端的请求。第二个任务为搭建一个客户端来访问服务端，并且显示各数据表的信息。在服务端方面，我沿用了第一次的开发环境，即部署了 Python + Flask + Gunicorn + MySQL 的阿里云服务器，Python 语言简单便捷，配合上微框架 Flask，大大提高了开发效率；在客户端方面，我沿用了 Web 的方式进行开发，用到的语言和库有 JavaScript、HTML、CSS、jQuery。

下面给出基本设计的示意图，下面的章节将详细介绍各个部分的情况。

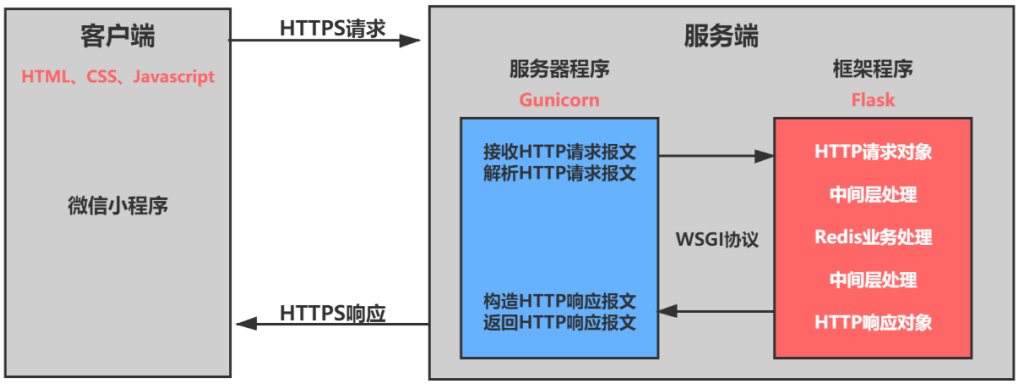

图 1　项目架构图

4. 服务端的基本搭建部署

为了能够让客户端在 24 小时内随时访问到服务端，我们采用了阿里云服务器来部署我们的服务端。选择阿里云服务器来开发有众多优点：（1）服务端具有真实的独立 IP，同时可以随时响应，而使用虚拟机则需要每次都花一段时间来启动。(2)最大程度释放主机的性能，让我们可以用自己的电脑专注于代码开发上。（3）接近真实的工作场景。在以后的工作中，公司的服务端程序也多部署在云服务器上，我们通过 SSH 与服务端进行多方面的交互。如今采用阿里云服务器来开发则可以帮助我们提前体验以后工作的模式。

在阿里云 CentOS8 服务器上配置好 SSH 客户端，并生成公私钥文件进行部署。部署完成后，便可以通过 Windows Terminal 的 SSH 客户端对其进行连接。连接成功的截图如下所示：

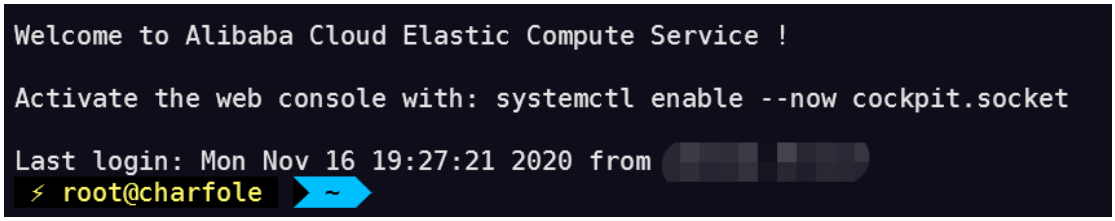

图 2　SSH 连接阿里云服务器

同样地，为了能够快捷高效地在服务器上进行后端代码的开发，我同时也在 VSCode 上连接了阿里云服务器，实现在 VSCode 中远程编辑服务器中的文件。

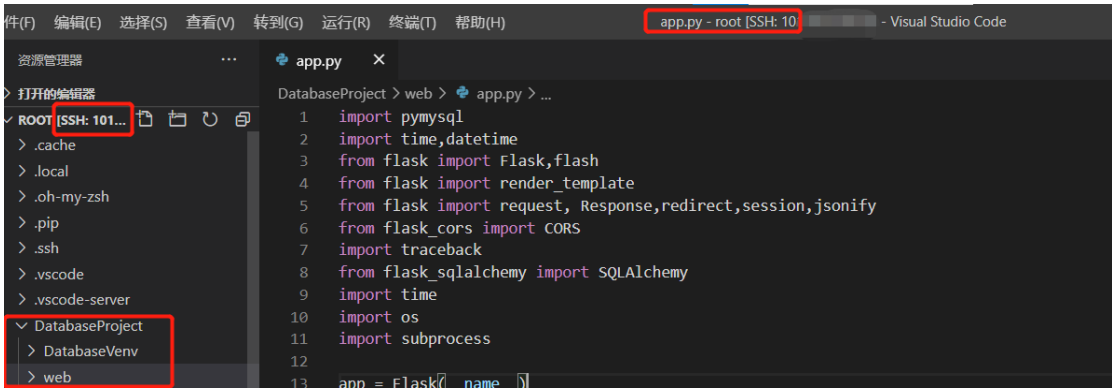

图 3　VSCode 远程连接服务器

经过这两步后，便可以开始准备搭建数据库和后端程序了。在 NoSQL 数据库这一方面，我选择了 Redis，主要是因为它具有以下这几个优点：（1）性能极高——Redis 能支持超过 100K+每秒的读写频率。（2）丰富的数据类型——Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。（3）原子性——Redis 的所有操作都是原子性的，同时 Redis 还支持对几个操作全并后的原子性执行。（4）丰富的特性——Redis 还支持 publish/subscribe, 通知,

key 过期等等特性。经过简单的几步操作后，便在服务器中部署好了 Redis，其版本信息如下所示：

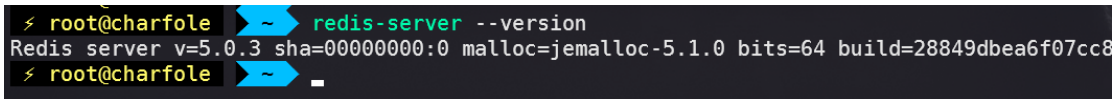

图 4　查看 Redis 版本信息

在部署好数据库后，还需要有一个交互工具来对其进行更好的操作与管理。在这里我选用了 Redis 自带的工具——redis-cli 进行操纵交互。

图 5　redis-cli 管理 Redis

在后端开发的框架方面，我选择了 Python 的 Flask 框架。Flask 框架的优点如下：（1）Flask 是一个微框架，十分轻巧简洁，易于学习。（2）Flask 可扩展性强，Python 语言的加持让它能够随时引入各类强大的库。（3）Flask 内置开发服务器和调试器。在阿里云服务器上按教程配置好 Flask 后，其版本信息如下所示：

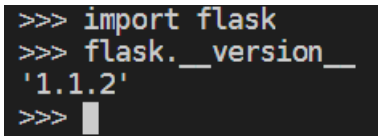

图 6　查看 Flask 版本信息

此外，配合 Gunicorn 服务器，便可以能够快速地将 Flask 程序部署到服务器中。

经过上述的工作后，如今的服务端已成功部署好了 Redis 和 Flask，并实现了利用 Windows Terminal、VSCode 和 redis-cli 对其进行远程管理。

5. Redis 数据表的构造

根据实验的要求，我们需要对数据库进行插入、修改和删除等操作。为了使数据库中的数据更贴切实际，我决定采用 Mock 的方法对数据表进行构造。至于

Mock 的工具选择，我采用了 Mockaroo 对数据表进行构造，Mockaroo 能够支持数据以不同的形式导出，包括 JSON、CSV、SQL 等；此外，其数据的种类非常贴近生活实际，而不是简单且无意义的数据。Mock 完成后将其插入到哈希表中。

下面，我将展示表的概览信息和插入的代码以及插入后的结果。

carId	carType	carModel	carYear
1GD02ZCGXDF501385	Chevrolet	Sportvan G30	1992
1N6AD0CU3AC551673	Ford	Mustang	1965
1FTSW2A51AE584424	Mitsubishi	Galant	1986
1G6DL67A680451443	Pontiac	Torrent	2009
1GD020CG2CF273537	Mitsubishi	Mirage	1989
JHMZE2H36BS492834	Plymouth	Reliant	1981
WAUAF78E67A344316	Jensen	Interceptor	1966
2D4RN3DG9BR929035	Cadillac	XLR	2009

图 7　查看数据表的概览信息

插入 CSV 数据到哈希表的代码：

代码 1　将 CSV 数据插入到哈希表中

输入：CSV

输出：无

```
1.  import csv
2.  import redis
3.  r = redis.StrictRedis(host='127.0.0.1', port=6379, db=0)
4.
5.  with open('./skr.csv', 'rt', encoding='utf-8') as csvfile:
6.      reader = csv.DictReader(csvfile)
7.      i=1
8.
9.      for row in reader:
10.         r.hset(str(i), key='name' , value=row['name'])  #将内容写入 redis
11.         r.hset(str(i), key='email' , value=row['email'])#将内容写入 redis
12.         r.hset(str(i), key='gender' , value=row['gender'])#将内容写入 redis
13.         r.hset(str(i), key='ip' , value=row['ip'])  #将内容写入 redis
14.         i = i+1
```

利用 redis-cli 查看插入后的结果，如下图所示：

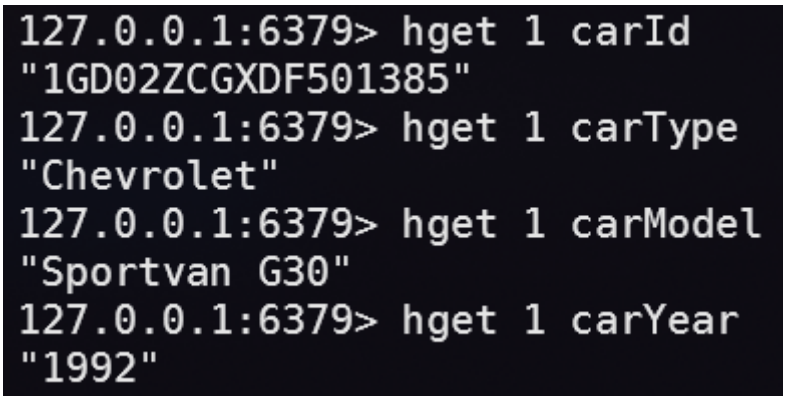

图 8　查询插入后结果

综上，通过 Mockaroo 和 Python 来对数据库进行操作，我们已经顺利在 Redis 中创立了数据表。

6. Flask 操纵 Redis 数据库

Python 的生态十分强大，操纵各个主流数据库的工具也必不可少。如果想要利用 Python 操纵 Redis，则需要借助一个名为 redis 的库，它操作简便，可以直接调用各类函数来对 Redis 进行操作。

由于 Redis 为 KV 数据库，因此所有操作都围绕 name、key、value 来开展，在定义好了数据库对象后，我们便可以对其进行各项操作，下面给出查询函数的代码：

代码 2　Flask 实现 Redis 的查询操作

输入：无

输出：Redis 数据库信息

```
1.  @app.route('/Search',methods=['GET'])
2.  def Search():
3.      r = redis.StrictRedis(host='127.0.0.1', port=6379, db=0, decode_respon
    ses=True)
4.      rowsName = sorted(r.keys())
5.      data = []
6.      for i in range(len(rowsName)):
7.          dt_1 = {'rows':rowsName[i]}
8.          dt_2 = r.hgetall(rowsName[i])
9.          dt_1.update(dt_2)
10.         data.append(dt_1)
11.
```

```
    return jsonify(data)
```

在增删查改方面，redis 库也有对应的函数对哈希表进行相应的操作。下面给出这些操作的核心代码：

代码 3　Flask 实现 Redis 的增删查改

输入：实现的操作类别、name、key、value

输出：Redis 执行情况

```
@app.route('/CRUD',methods=['POST'])
def CRUD():
    r = redis.Redis(host='localhost', port=6379, db=0,decode_responses=True)
    action = request.form.get('action')
    name = request.form.get('name')
    key = request.form.get('key')
    value = request.form.get('value')

    if(action=='CREATE'):
        if(r.hexists(name,key) == True):
                return 'This key already exists.'

        else:
            r.hsetnx(name,key,value)
            return 'Sussessfully added.'

    # 这里键的数目可能会出现变化，前端要相应处理
    elif(action=='READ'):
        result = r.hgetall(name)
        rows = {'rows':name}
        result.update(rows)
        if (result != None):
            return jsonify(result)
        else:
            return 'This name doesn\'t exist.'

    elif(action=='UPDATE'):
        if(r.hexists(name,key) != True):
            return 'This key doesn\'t exist.'
        else:
            r.hset(name,key,value)

        return 'Sussessfully updated.'
```

```
34.
35.     elif(action=='DELETE'):
36.         if(r.hexists(name,key) != True):
37.             return 'This key doesn\'t exist.'
38.         else:
39.             r.hdel(name,key)
40.
41.         return 'Sussessfully Deleted.'
42.
43.     return 'The action must be in this list:[CREATE,READ,UPDATE,DELETE]!'
```

在特色操作方面，Redis 提供了一个名为基数查询的功能，即查询所有元素中的基数（此前没有统计过的数）。为了体现 Redis 基数查询快的优点，我实现了常规的查询方法，并与基数查询进行对比。下面给出该函数的核心代码：

代码 4　Flask 实现 Redis 特色功能——基数查询

输入：想要插入的元素总数

输出：常规查询和基数查询的结果

```
1.  @app.route('/Hyperloglog',methods=['POST'])
2.  def Hyperloglog():
3.      redisHyper= redis.StrictRedis(host='127.0.0.1', port=6379, db=1, decod
    e_responses=True)
4.      redisHyper.flushdb()
5.      scale = request.form.get('scale')
6.      scale = int(scale)
7.      _startRedisAdd = time.time()
8.      redisList = [scale]
9.      originalList = [scale]
10.     for i in range(scale):
11.         redisHyper.pfadd(1,i)
12.     _startRedisAddok = time.time()
13.     redisAddTime = _startRedisAddok - _startRedisAdd
14.     redisCount = redisHyper.pfcount(1)
15.     redisSearchTime = time.time() - _startRedisAddok
16.     print('Redis 添加{}个不同元素时间为：{}'.format( scale,redisAddTime) )
17.     print('Redis 查询{}个不同元素的基数值为：{}'.format(scale,redisCount))
18.     print('Redis 查询{}个不同元素基数的时间为：
    {}'.format(scale,redisSearchTime))
19.     redisList.append(redisAddTime)
20.     redisList.append(redisCount)
```

```
    redisList.append(redisSearchTime)

    count = []
    _startOriginalAdd = time.time()
    for i in range(scale):
        count.append(i)
    _startOriginalAddOk = time.time()
    originalAddTime = _startOriginalAddOk - _startOriginalAdd
    originalCount = len(set(count))
    originalSearchTime = time.time() - _startOriginalAddOk
    print('常规添加{}个不同元素时间为：{}'.format( scale, originalAddTime) )
    print('常规查询{}个不同的基数值为：{}'.format( scale, originalCount ))
    print('常规查询{}个不同元素基数的时间为：{}'.format(scale,originalSearchTime))
    originalList.append(originalAddTime)
    originalList.append(originalCount)
    originalList.append(originalSearchTime)

    return jsonify(redisList,originalList)
```

7. 基于 Web 的数据库客户端开发

在客户端方面，我选择了 Web 开发来实现，借助 HTML、CSS、JavaScript 和 jQuery 库打造的 Web 页面对比其 Windows 应用有着可移植性强、访问简单、开发便捷的优点。

在客户端的功能设计方面，主要分为三个部分，第一个部分为基本的 Redis 增删查改模块框，对应着 Redis 的基本功能。下面以增添数据为例，给出该部分的前端核心代码：

代码 5　Redis 基本功能实现前端代码

输入：实现的操作类别、name、key、value

输出：Redis 执行情况

```
<!-- 发送 ajax CREATE 请求 -->
<script type="text/javascript">
    function clickC() {
        // console.log($(".button").val());
        $.ajax({
            //几个参数需要注意一下
```

```
            type: "POST", //方法类型
            // dataType: "json", //预期服务器返回的数据类型
            url: "http://101.xxx.xxx.xx:5000/CRUD", //url
            data: "action=" + $(".button").val() + "&" + $('#form1').seria
    lize(),
            success: function(data) {
                // console.log(result); //打印服务端返回的数据(调试用)
                // console.log(event.status);
                // console.log(data);
                alert(data);
            },
            error: function() {
                alert("异常！");
            }
        });
    }
</script>
```

第二个部分为基数查询功能，对应有一个输入插入总元素个数的按钮，其后在服务器中进行该规模数据的插入，并以模态框的形式返回 Redis 基数查询和常规基数查询的时间对比结果。下面给出该部分的前端代码：

代码 6　Redis 基数查询前端代码

输入：插入元素的个数

输出：基数查询的结果

```
<!-- send 函数 -->
  <script type="text/javascript">
      function send() {
          // console.log($(".button").val());
          $.ajax({
              //几个参数需要注意一下
              type: "POST", //方法类型
              // dataType: "json", //预期服务器返回的数据类型
              url: "http://101.xxx.xxx.xx:5000/Hyperloglog", //url
              data: $('.sendNumber').serialize(),
              success: function(data) {
                  // console.log(result); //打印服务端返回的数据(调试用)
                  // console.log(event.status);
                  // alert(data);
                  // console.log(data[0]);
```

```
                    // console.log(data[1]);
                    var msg = '';
                    if (data[0][3] < data[1][3]) {
                        msg += 'Redis 查询速度更快，' + '快了
    ' + ((data[1][3] / data[0][3]) - 1).toFixed(2) + '倍';

                    } else {
                        msg += '常规查询速度更快' + '快了
    ' + ((data[0][3] / data[1][3]) - 1).toFixed(2) + '倍';
                    }
                    alert('Redis 添加 1000 个不同元素时间为：
    ' + data[0][1] +
                        '\n' + 'Redis 查询 1000 个不同元素的基数值为：
    ' + data[0][2] +
                        '\n' + 'Redis 查询 1000 个不同元素基数的时间为：
    ' + data[0][3] +
                        '\n' + '常规添加 1000 个不同元素时间为：
    ' + data[1][1] +
                        '\n' + '常规查询 1000 个不同的基数值为：
    ' + data[1][2] +
                        '\n' + '常规查询 1000 个不同元素基数的时间为：
    ' + data[1][3] +
                        '\n' + msg
                    );
                },
                error: function() {
                    alert("异常！");
                }
            });
        }
    </script>
```

以上就是客户端部分的核心代码，下面给出客户端的截图。

图 9　客户端页面概览

8. 功能展示

8.1 基本功能

8.1.1 创建键值对

在 Web 页面中填入需创建的(name、key、value)：(1002、carColor、red)。执行后，可在客户端中查看到键值对已创建成功。

图 10　创建键值对示意图

8.1.2 查看键值对

在 Web 页面中输入要查看键值对的名称。

执行后，可在客户端中查看到已成功查看到刚刚创建好的键值对。

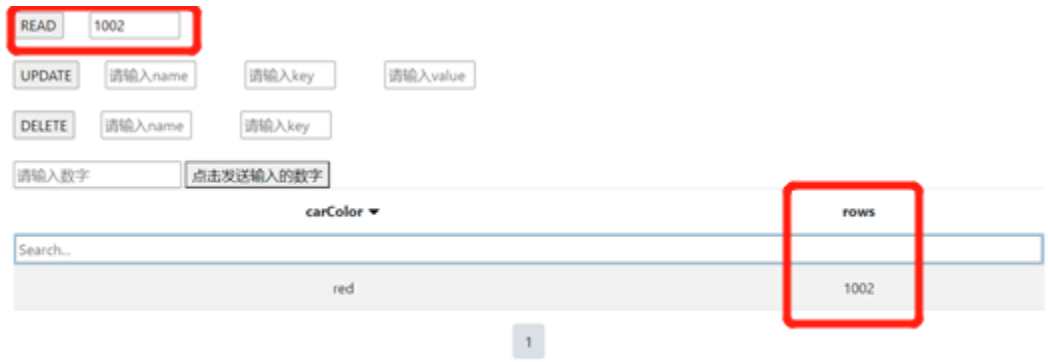

图 11　查看键值对示意图

8.1.3 更新键值对

在 Web 页面中输入要更新键值对的 name 和它的键与值。

执行后，可在客户端中查看更新操作已成功。

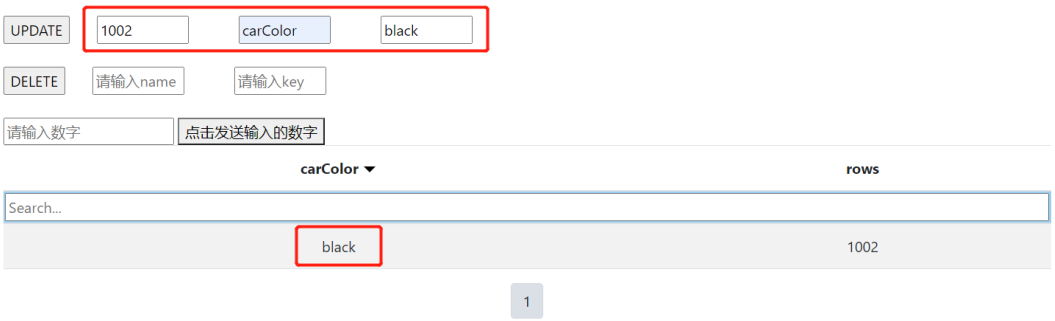

图 12　更新键值对示意图

8.1.4 删除键值对

在 Web 页面中输入要删除键值对的 name 和 key。

执行后，可在客户端中查看到已删除成功。

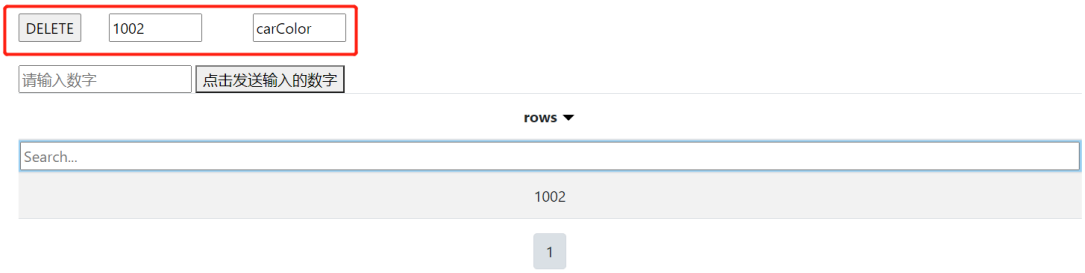

图 13　删除键值对示意图

8.2 额外功能

8.2.1 基数查询

在 Web 页面中输入 Redis 基数查询的总元素个数。

执行后，可看到查询成功的时间对比结果：

图 14　基数查询示意图

9. 总结

本次实验中，我按照实验的要求，在安装了 CentOS8 的阿里云服务器中部署好了 Gunicorn 服务器、Flask 微框架以及 Redis，并在此基础上编写了服务端代码，在后端实现了增删查改等功能，同时还实现了基数查询等特色操作。此外，我还借助 JavaScript、HTML、CSS、jQuery 等工具开发了数据库客户端，实现了数据库的展示、增删查改操作和对基数查询结果的展示。

本次实验使我获益匪浅，让我对 Linux 环境下的工具部署、后端开发、Web 开发有了更好的理解。在实验的过程中，我遇到了不少的问题，比如 Redis 各种命令的使用、Web 客户端的数据接收与展示等问题，但都通过询问同学与查阅资料解决了，这极大地锻炼了我解决问题的能力。