Redis-like 的内存数据库

背景知识

1. 认识缓存

Cache 和 Buffer 都是缓存，主要区别是什么？

2. 认识 redis

Redis缓存你必须了解的！

全面对比 Redis 和 Memcached 的 6 点区别

Redis 和 Memcached 各有什么优缺点，主要的应用场景是什么样的？

聊聊 Redis 使用场景

Redis数据类型

Introduction to Redis data types

Redis commands

3. 亲自安装并使用 redis

进入到 linux 环境

安装

```
yum install redis
```

```
systemctl start redis
```

```
systemctl enable redis
```

执行

```
redis-cli
```

利用 Jedis 连接 redis

```
mvn archetype:generate
```

在 pom.xml 中添加依赖

```
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>3.1.0</version>
</dependency>
```

在 src/main/java/com/peixinchen/App.java 中实现示例代码

```
package com.peixinchen;

import java.time.LocalDateTime;
import redis.clients.jedis.Jedis;

public class App {
    public static void main(String[] args) {
        Jedis jedis = new Jedis("localhost");
        System.out.println(jedis.get("hello"));
        jedis.set("hello", LocalDateTime.now().toString());
        System.out.println(jedis.get("hello"));
    }
}
```

进行项目构建

```
mvn clean package
```

运行

```
mvn exec:java -Dexec.mainClass=com.peixinchen.App
```

项目前置知识介绍

1. Socket 编程对应的网络原理

```
import java.net.ServerSocket;
import java.net.Socket;

try (ServerSocket serverSocket = new ServerSocket(9989)) {
    try (Socket socket = serverSocket.accept()) {
        socket.getInputStream();    // InputStream 输入字节流
        socket.getOutputStream();   // OutputStream 输出字节流
    }
}
```

2. 多线程编程中的固定线程池

```
import java.util.concurrent.Executors;

ExecutorService pool = Executors.newFixedThreadPool(6);
service.execute(new Runnable() {
    @Override
    public void run() {
    }
});
pool.shutdown();
```

3. redis 协议详细介绍

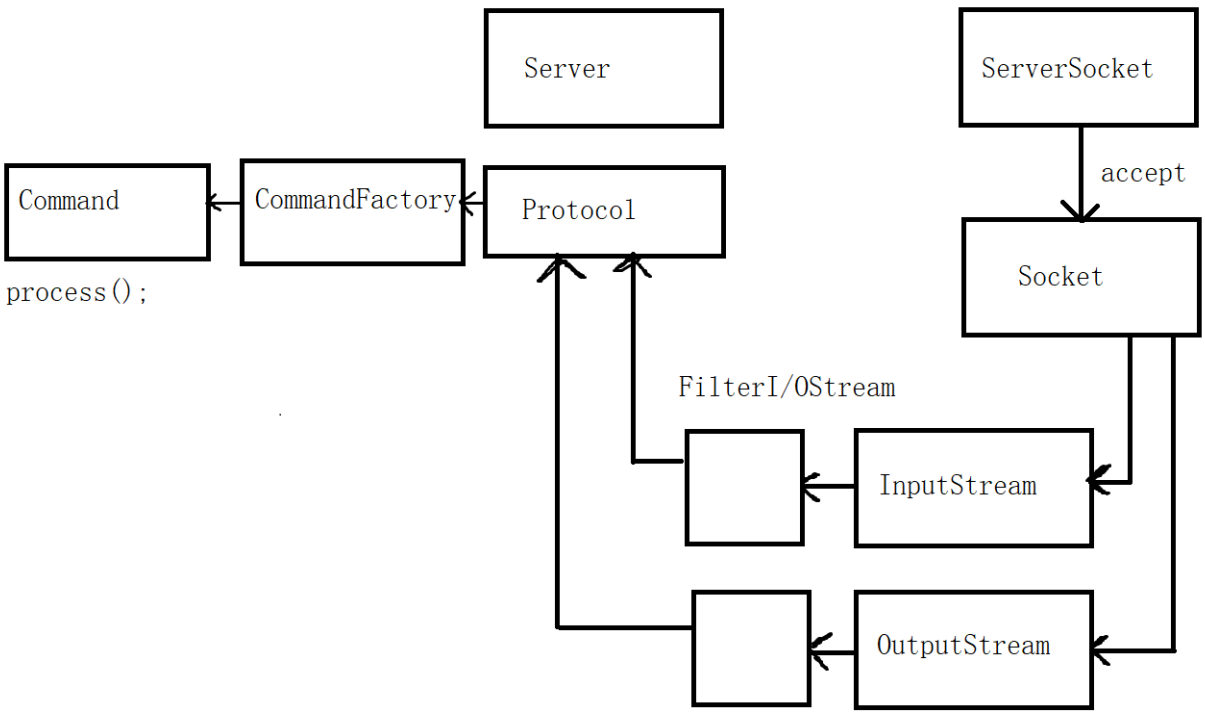
Redis Protocol specification

协议支持的数据类型

Redis 类型	作用	Java 中的类型	标记字节	格式说明
Simple String	返回 OK	String	'+'	'+OK\r\n'
Error	通知错误	Exception	'-'	'-WRONGTYPE Operation against a key holding the wrong kind of value'
Integer	整数	long	':'	':-1\r\n' 或者 ':1000\r\n'
Bulk String	字节流	byte[]	'$'	'$6\r\nfoobar\r\n' 或者 '$-1\r\n'
Array	数组	List	'*'	*'0\r\n' 或者* '2\r\n$3\r\nfoo\r\n$3\r\nbar\r\n' 或者 '*-1\r\n'

开发

类设计图



参考代码

异常

BedisException.java

```
public class BedisException extends Exception {
    public BedisException() {
    }

    public BedisException(String message) {
        super(message);
    }

    public BedisException(String message, Throwable cause) {
        super(message, cause);
    }

    public BedisException(Throwable cause) {
        super(cause);
    }

    public BedisException(String message, Throwable cause, boolean enableSuppression,
boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

RemoteException.java

```
public class RemoteException extends BedisException {
    public RemoteException() {
    }

    public RemoteException(String message) {
        super(message);
    }

    public RemoteException(String message, Throwable cause) {
        super(message, cause);
    }

    public RemoteException(Throwable cause) {
        super(cause);
    }

    public RemoteException(String message, Throwable cause, boolean enableSuppression,
boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

协议处理

Protocol.java

```
public class Protocol {
    public static Charset charset() {
        return Charset.forName("UTF-8");
    }

    public static Command readCommand(ProtocolInputStream is) throws BedisException {
        Object object = read(is);
        if (!(object instanceof List)) {
            throw new RemoteException("对方发送的命令不是 array 形式");
        }

        return CommandFactory.build((List<Object>)object);
    }

    public static Object read(ProtocolInputStream is) throws BedisException {
        return process(is);
    }

    private static Object process(ProtocolInputStream is) throws BedisException {
        int b;
        try {
            b = is.read();
        } catch (IOException e) {
            throw new BedisException(e);
        }
        switch (b) {
            case '+':
                return processSimpleString(is);
            case '-':
                throw new RemoteException(processError(is));
            case ':':
                return processInteger(is);
            case '$':
                return processBulkString(is);
            case '*':
                return processArray(is);
            default:
                throw new BedisException("读到了错误的类型");
        }
    }

    private static String processSimpleString(ProtocolInputStream is) throws
BedisException {
        return is.readLine();
    }

    private static String processError(ProtocolInputStream is) throws BedisException {
        return is.readLine();
    }
```

```
    private static long processInteger(ProtocolInputStream is) throws BedisException {
        return is.readIntegerCRLF();
    }

    private static byte[] processBulkString(ProtocolInputStream is) throws
BedisException {
        int len = (int)is.readIntegerCRLF();
        System.out.println(len);
        if (len == -1) {
            return null;
        }

        byte[] buf = new byte[len];
        // TODO: 对方恶意发送一个错误的长度的情况，没有处理
        int nRead = 0;
        while (nRead < len) {
            int size;
            try {
                size = is.read(buf, nRead, len - nRead);
            } catch (IOException e) {
                throw new BedisException(e);
            }
            if (size == -1) {
                throw new BedisException("对方中途关闭连接了");
            }

            nRead += size;
        }

        is.readCRLF();

        return buf;
    }

    private static List<Object> processArray(ProtocolInputStream is) throws
BedisException {
        int len = (int)is.readIntegerCRLF();
        if (len == -1) {
            return null;
        }

        List<Object> list = new ArrayList<>(len);
        for (int i = 0; i < len; i++) {
            try {
                list.add(process(is));
            } catch (RemoteException e) {
                list.add(e);
            }
        }

        return list;
    }
```

```
    public static void writeSimpleString(ProtocolOutputStream os, String string) throws
BedisException {
        try {
            os.write('+');
            os.write(string.getBytes(charset()));
            os.writeCRLF();
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    public static void writeError(ProtocolOutputStream os, String message) throws
BedisException {
        try {
            os.write('-');
            os.write(message.getBytes(charset()));
            os.writeCRLF();
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    public static void writeInteger(ProtocolOutputStream os, long v) throws
BedisException {
        try {
            os.write(':');
            os.writeInteger(v);
            os.writeCRLF();
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    public static void writeBulkString(ProtocolOutputStream os, String string) throws
BedisException {
        writeBulkString(os, string.getBytes(charset()));
    }

    public static void writeBulkString(ProtocolOutputStream os, byte[] bytes) throws
BedisException {
        try {
            os.write('$');
            os.writeInteger(bytes.length);
            os.writeCRLF();
            os.write(bytes);
            os.writeCRLF();
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    public static void writeNull(ProtocolOutputStream os) throws BedisException {
```

```
        try {
            os.write('$');
            os.writeInteger(-1);
            os.writeCRLF();
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    public static void writeArray(ProtocolOutputStream os, List<?> list) throws
BedisException {
        try {
            os.write('*');
            os.writeInteger(list.size());
            os.writeCRLF();

            for (Object o : list) {
                writeObject(os, o);
            }
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    @SuppressWarnings("unchecked")
    private static void writeObject(ProtocolOutputStream os, Object o) throws
BedisException {
        if (o instanceof String) {
            writeSimpleString(os, (String)o);
        } else if (o instanceof RemoteException) {
            writeError(os, ((RemoteException)o).getMessage());
        } else if (o instanceof Integer) {
            writeInteger(os, ((Integer)o).longValue());
        } else if (o instanceof Long) {
            writeInteger(os, (Long)o);
        } else if (o instanceof byte[]) {
            writeBulkString(os, (byte[])o);
        } else if (o == null) {
            writeNull(os);
        } else if (o instanceof List) {
            writeArray(os, (List<Object>)o);
        } else {
            throw new BedisException("写入了不认识的数据类型");
        }
    }
}
```

FilterInputStream 和 FilterOutputStream

ProtocolInputStream.java

```
public class ProtocolInputStream extends FilterInputStream {
    /**
```

```
     * Creates a <code>FilterInputStream</code>
     * by assigning the  argument <code>in</code>
     * to the field <code>this.in</code> so as
     * to remember it for later use.
     *
     * @param in the underlying input stream, or <code>null</code> if
     *           this instance is to be created without an underlying stream.
     */
    public ProtocolInputStream(InputStream in) {
        super(in);
    }

    public String readLine() throws BedisException {
        return new String(readLineBytes(), Protocol.charset());
    }

    public byte[] readLineBytes() throws BedisException {
        List<Byte> bytes = new ArrayList<>();
        try {
            while (true) {
            int b = read();
            if (b == '\r') {
                int c = read();
                if (c == '\n') {
                    break;
                }

                bytes.add((byte)b);
                bytes.add((byte)c);
            } else {
                bytes.add((byte)b);
            }
        }
        } catch (IOException e) {
            throw new BedisException(e);
        }

        if (bytes.size() == 0) {
            return new byte[0];
        }

        byte[] ret = new byte[bytes.size()];
        for (int i = 0; i < bytes.size(); i++) {
            ret[i] = bytes.get(i);
        }

        return ret;
    }

    public long readIntegerCRLF() throws BedisException {
        try {
            int b = read();
            long value = 0;
```

```
            boolean isNegative = (b == '-');
            if (!isNegative) {
                value = b - '0';
            }

            while (true) {
                b = read();
                if (b == '\r') {
                    if (read() != '\n') {
                        throw new BedisException("没有 CRLF 结尾");
                    }

                    break;
                } else {
                    value = value * 10 + (b - '0');
                }
            }

            return isNegative ? -value : value;
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }

    public void readCRLF() throws BedisException {
        try {
            int b = read();
            if (b != '\r') {
                throw new BedisException("不是 CR");
            }

            b = read();
            if (b != '\n') {
                throw new BedisException("不是 LF");
            }
        } catch (IOException e) {
            throw new BedisException(e);
        }
    }
}
```

ProtocolOutputStream.java

```
public class ProtocolOutputStream extends FilterOutputStream {
    /**
     * Creates an output stream filter built on top of the specified
     * underlying output stream.
     *
     * @param out the underlying output stream to be assigned to
     *            the field <tt>this.out</tt> for later use, or
     *            <code>null</code> if this instance is to be
     *            created without an underlying stream.
     */
```

```
    public ProtocolOutputStream(OutputStream out) {
        super(out);
    }

    public void writeCRLF() throws IOException {
        write('\r');
        write('\n');
    }

    public void writeInteger(long v) throws IOException {
        if (v < 0) {
            write('-');
            v = -v;
        }

        byte[] valueBytes = Long.toString(v).getBytes(Protocol.charset());
        write(valueBytes);
    }
}
```

命令相关

Command.java

```
public interface Command {
    void setArgs(List<?> args);

    void process(ProtocolOutputStream os);
}
```

AbstractCommand.java

```
public abstract class AbstractCommand implements Command {
    protected Database database = Database.getInstance();
    protected List<Object> args;

    @Override
    @SuppressWarnings("unchecked")
    public void setArgs(List<?> args) {
        this.args = (List<Object>)args;
    }
}
```

GETCommand.java

```
public class GETCommand extends AbstractCommand implements Command {
    @Override
    public void process(ProtocolOutputStream os) {
        try {
            if (args.size() != 1) {
                Protocol.writeError(os, "ERR wrong number of arguments for 'get'
command");
```

```
            }

            String key = new String((byte[])args.get(0), Protocol.charset());
            String value = database.stringGet(key);
            if (value == null) {
                Protocol.writeNull(os);
                return;
            }

            Protocol.writeBulkString(os, value);
        } catch (BedisException e) {
            e.printStackTrace();
        }
    }
}
```

SETCommand.java

```
public class SETCommand extends AbstractCommand implements Command {
    @Override
    public void process(ProtocolOutputStream os) {
        try {
            if (args.size() != 2) {
                Protocol.writeError(os, "ERR wrong number of arguments for 'set'
command");
            }

            String key = new String((byte[])args.get(0), Protocol.charset());
            String value = new String((byte[])args.get(1), Protocol.charset());
            database.stringSet(key, value);
            Protocol.writeSimpleString(os, "OK");
        } catch (BedisException e) {
            e.printStackTrace();
        }
    }
}
```

CommandFactory.java

```
public class CommandFactory {
    public static Command build(String commandName) throws BedisException {
        String fullClassName = "com.bittech.commands." + commandName.toUpperCase() +
"Command";
        try {
            Class<?> cls = Class.forName(fullClassName);
            if (!Command.class.isAssignableFrom(cls)) {
                throw new BedisException("不是合法的命令");
            }
            return (Command)cls.newInstance();
        } catch (ClassNotFoundException | IllegalAccessException |
InstantiationException e) {
            throw new BedisException("不是合法的命令", e);
```

```
        }
    }

    public static Command build(List<?> args) throws BedisException {
        if (args.size() == 0) {
            throw new BedisException("对方发送的命令 array 的长度为 0");
        }

        Object object = args.remove(0);
        if (!(object instanceof byte[])) {
            throw new BedisException("对方发送的命令不是 Bulk String");
        }

        String commandName = new String((byte[])object, Protocol.charset());
        Command command = build(commandName);
        command.setArgs(args);
        return command;
    }
}
```

数据存储

Database.java

```
public class Database {
    private Database() {
    }

    private static Database instance = new Database();
    public static Database getInstance() {
        return instance;
    }

    private Map<String, String> strings = new HashMap<>();
    private Map<String, List<String>> lists = new HashMap<>();
    //private Map<String, Map<String, String>> hashes = new HashMap<>();

    public synchronized void stringSet(String key, String value) {
        strings.put(key, value);
    }

    public synchronized String stringGet(String key) {
        return strings.get(key);
    }

    public synchronized void listPush(String key, String value) {
        List<String> list = lists.computeIfAbsent(key, k -> new ArrayList<>());
        list.add(value);
    }

    public synchronized String listPop(String key) {
        List<String> list = lists.get(key);
        if (list == null || list.size() == 0) {
```

```
            return null;
        }

        return list.remove(0);
    }

    public synchronized int listLen(String key) {
        List<String> list = lists.get(key);
        if (list == null) {
            return 0;
        }

        return list.size();
    }

    public synchronized List<String> listGet(String key) {
        List<String> list = lists.get(key);
        if (list == null) {
            return null;
        }

        return new ArrayList<>(list);
    }
}
```

测试

因为我们支持的协议和 redis 保持一致，所以可以直接拿到 Linux 上利用 redis-cli 进行测试。

未来

1. 支持更多的数据类型
2. 数据想办法持久化
3. 提升处理性能

写在简历上的话

技术关键词（tag） redis、字节流协议、内存数据库、多线程、网络、TCP等等