

线程控制的方法

- sleep()

它是Thread的一个静态方法，可以让程序暂停，参数是毫秒

- join()

线程排队，执行join方法的线程优先执行

- setDaemon(true)

调用该方法后线程变成守护线程（后台线程），在其它线程结束后它才结束

- yield()

主动放弃资源，回到就绪状态

- setPriority ()

设置线程的优先级。参数是1-10， MAX_PRIORITY 最高优先级， MIN_PRIORITY最低优先级

- wait() & notify() notifyAll()

Object对象的放，wait被调用线程进入阻塞状态，直到notify被调用才重新回到就绪

网络编程

三次握手，四次挥手

TCP/IP：长连接，可靠通信

OSI七层/四层

IP和端口：

xxx.xxx.xxx.xxx 四段组成，每段之间用.分开，每一段可以是0~255， 如： 192.168.3.255。IP一般对应电脑，相当于一栋大楼

端口：相当于一栋楼的房间号，对应一个应用程序。16位正整数，1024以下端口操作保留，自己写程序最好用3000以上

网段：LAN局域网， WAN广域网

A段： 10.0.0.0~10.255.255.255

B段： 172.16.0.0~172.31.255.255

C段： 192.168.0.0~192.168.255.255

InetAddress

用于描述IP或域名

```
InetAddress address = InetAddress.getByName("www.baidu.com");
System.out.println(address.getHostAddress()); //返回域名对应IP
System.out.println(address.getHostName()); //返回域名
System.out.println(address.isReachable(50)); //测试该地址再指定的时间能是
```

否能连通

URLEncoder, URLDecoder

对url进行编码或解码

```
String encode = URLEncoder.encode("中文", "UTF-8");
System.out.println(encode);
String decode = URLDecoder.decode(encode, "UTF-8");
System.out.println(decode);
```

URL,URLConnection

```
package com.hqyj.javaadvanceday07;

import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class URLDemo {

    public static void main(String[] args) {
        InputStream inputStream = null;
        try {
            URL url = new URL("http://www.baidu.com");//描述一个网页地址
            URLConnection connection = url.openConnection();//创建一个连接对象
            connection.connect(); //连通网页
            Map<String, List<String>> map = connection.getHeaderFields();//获取
            HTTP协议的头部信息
            Set<String> keySet = map.keySet();
            for(String s : keySet) {
                System.out.println(s + ": " + map.get(s));
            }

            inputStream = connection.getInputStream();//获取网页内容
            byte[] b = new byte[4096];
            while(inputStream.read(b)>0) {
                System.out.println(new String(b));
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            //
        }
    }
}
```

```

        try {
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}

}

```

Socket（套接字）通信

服务器端

```

package com.hqyj.javaadvanceday07;

import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerDemo {

    public static void main(String[] args) {
        Socket socket = null;
        OutputStream stream = null;
        PrintStream ps = null;
        ServerSocket ss = null;
        try {
            ss = new ServerSocket(3000);
            socket = ss.accept(); //接受客户端的请求。这个方法会阻塞程序运行，一直等到有请求之后才会继续往后运行
            stream = socket.getOutputStream();
            ps = new PrintStream(stream);
            ps.println("你好，我是谁谁谁"); //给客户端发送一条信息
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                ps.close();
                stream.close();
                socket.close();
                ss.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}

```

客户端

```
package com.hqyj.javaadvanceday07;

import java.io.BufferedReader;
import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;

public class ClientDemo {

    public static void main(String[] args) {
        BufferedReader reader = null;
        InputStream stream = null;
        Socket socket = null;
        try {
            socket = new Socket("127.0.0.1", 3000); //localhost和127.0.0.1都是指本
            //机地址
            stream = socket.getInputStream(); //从socket获取输入流（读服务器端返回的
            //信息）
            reader = new BufferedReader(new InputStreamReader(stream));
            String line = reader.readLine();
            System.out.println(line);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                reader.close();
                stream.close();
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

多线程Socket通信，实现多人群聊功能

服务器端线程

```
package com.hqyj.javaadvanceday07;

import java.io.BufferedReader;
```

```

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;

public class ServerThread extends Thread {
    private Socket socket;

    public ServerThread(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            InputStream inputStream = socket.getInputStream(); //接收客户端消息
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
            String line = null;
            while((line = bufferedReader.readLine())!=null) {
                System.out.println("接收到消息: " + line);
                for(Socket s : ServerThreadDemo.sockets) {
                    OutputStream outputStream = s.getOutputStream(); //发送消息给
所有客户端

                    PrintStream printStream = new PrintStream(outputStream);
                    printStream.println(line);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            ServerThreadDemo.sockets.remove(socket);
        }
    }
}

```

服务器端主程序

```

package com.hqyj.javaadvanceday07;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class ServerThreadDemo {
    public static List<Socket> sockets = new ArrayList<>();

    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(3000);
            while(true) {

```

```

        System.out.println("等待客户端连接...");
        Socket socket = serverSocket.accept();
        System.out.println("客户端连接成功");
        sockets.add(socket); //把socket连接添加列表
        new ServerThread(socket).start();
    }
} catch (IOException e) {
    e.printStackTrace();
}
}

}
}

```

客户端线程

```

package com.hqyj.javaadvanceday07;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.Socket;

public class ClientThread extends Thread {
    private Socket socket;

    @Override
    public void run() {
        try {
            InputStream inputStream = socket.getInputStream();
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
            String line = null;
            while((line = bufferedReader.readLine())!=null) {
                System.out.println(line);
            }

        } catch (IOException e) {

            e.printStackTrace();
        }
    }

    public ClientThread(Socket socket) {
        this.socket = socket;
    }
}

```

```
package com.hqyj.javaadvanceday07;

import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class ClientThreadDemo {

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("127.0.0.1",3000);
            new ClientThread(socket).start();
            Scanner scanner = new Scanner(System.in);
            String line = null;
            OutputStream outputStream = socket.getOutputStream(); //把键盘输入的消息发送给服务器
            PrintStream printStream = new PrintStream(outputStream);
            while((line = scanner.nextLine())!=null) {
                printStream.println(line);
            }

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

UDP

不是长连接（发送/接受信息时才连接），消息不可靠。（相当于发短信）

常用的即时通讯软件一般采用UDP

类加载和反射

类加载和初始化

把类读入到内存中，初始化（静态属性，静态初始化代码块）的过程称为类加载。同一个虚拟机，类的加载和初始化只会执行一次

什么情况下类会被初始化：

1.创建实例对象， new XXX()

- 2.调用类的静态方法或属性
- 3.使用反射方式创建对象
- 4.创建子类实例，所有的父类全部会被初始化
- 5.java命令运行一个类，它也会被初始化

反射

- 可以获得类的Class对象，通过Class对象获得类的相关信息（构造方法，属性，方法）

获取Class对象有三种方式

- Student.class
- student.getClass()
- Class.forName("com.hqyj.javaadvanceday07.Student")

```
package com.hqyj.javaadvanceday07;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class ReflectDemo {

    //获取类的Class对象: Student.class, student.getClass(),
    //Class.forName("com.hqyj.javaadvanceday07.Student")
    //获取类信息（构造方法，属性，方法）
    public static void main(String[] args) throws ClassNotFoundException {
        //      Class<Student> clazz = Student.class;
        //      Student student = new Student();
        //      Class<? extends Student> clazz = student.getClass();
        Class<?> clazz =
        Class.forName("com.hqyj.javaadvanceday07.Student");//类的全限定名（带上包名）

        //获取类的方法
        Method[] methods = clazz.getMethods();
        for(Method m : methods) {
            System.out.println(m);
        }

        //获取属性（public）
        Field[] fields = clazz.getFields();
        for(Field f:fields) {
            System.out.println(f);
        }

        //获取构造方法
        Constructor<?>[] constructors = clazz.getConstructors();
        for(Constructor c : constructors) {
            System.out.println(c);
        }

    }

}
```


- 通过反射创建对象，调用方法

```
package com.hqyj.javaadvanceday07;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.List;

public class ReflectInvoke {

    //第一个参数类的全限定名，第二个参数是要调用的方法名
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();

        try {
            Class<?> clazz = Class.forName(args[0]); //创建class对象
            Object instance = clazz.newInstance(); //创建实例

            Method method = clazz.getMethod(args[1]); //获得方法
            Object result = method.invoke(instance); //调用方法，第一个参数是实例，后面的参数是该方法的实参
            System.out.println(result);

            Method method2 = clazz.getMethod("setName", String.class); //调用有参数的方法。第二个参数是setName方法的参数的Class类型
            method2.invoke(instance, "Mike"); //第二个参数就是调用setName方法传递的实参
            System.out.println(instance);

            /*
            int i = 0;
            while(true) {
                System.out.println(++i);
                list.add(new Student());
            }
            */
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

注解

创建注解

New Annotation Type

Annotation Type
Create a new annotation type.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected

☒ Add @Retention: ☐ Source ☐ Class ☒ Runtime 作用域 (有效时期)

☐ Add @Target:

☐ Type ☐ Field ☐ Method
☐ Parameter ☐ Constructor ☐ Local variable
☐ Annotation type ☐ Package ☐ Type parameter
☐ Type use

适用位置

☐ Add @Documented

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

```
package com.hqyj.javaadvanceday07;

import static java.lang.annotation.RetentionPolicy.RUNTIME;

import java.lang.annotation.Retention;

@Retention(RUNTIME) //运行时有效
public @interface Auth {
    public String role() default "admin"; //注解的属性，方法名是属性名字，default后面
    //是默认值
}
```

注解的使用

先再类或方法添加注解

```
@Auth(role = "root")
public String getName() {
    return name;
}
```

通过反射得到方法对象，再获取该方法上的注解

```
package com.hqyj.javaadvanceday07;

import java.lang.reflect.Method;

public class AnnotationDemo {

    public static void main(String[] args) throws NoSuchMethodException,
SecurityException {
        Class<?> clazz = Student.class;
        Method method = clazz.getMethod("getName");
        Auth annotation = method.getAnnotation(Auth.class); //获得注解对象
        System.out.println(annotation.role()); //返回注解的role属性
    }

}
```