<http://blog.csdn.net/Ro_bot/article/details/51620064>

版权声明：个人随笔，欢迎讨论

目录[(?)[+]](http://blog.csdn.net/Ro_bot/article/details/51620064)

## 心跳机制

心跳机制是定时发送一个自定义的结构体(心跳包)，让对方知道自己还活着，以确保连接的有效性的机制。

大部分CS的应用需要心跳机制。心跳机制一般在Server和Client都要实现，两者实现原理基本一样。Client不关心性能，怎么做都行。

如果应用是基于TCP的，可以简单地通过SO\_KEEPALIVE实现心跳。TCP在设置的KeepAlive定时器到达时向对端发一个检测TCP segment，如果没收到ACK或RST，尝试几次后，就认为对端已经不存在，最后通知应用程序。这里有个缺点是，Server主动发出检测包，对性能有点影响。

应用自己实现   
Client启动一个定时器，不断发心跳；   
Server收到心跳后，给个回应；   
Server启动一个定时器，判断Client是否存在，判断方法这里列两种：   
时间差和简单标志。

1. 时间差策略

收到一个心跳后，记录当前时间(记为recvedTime)。

判断定时器时间到达，计算多久没收到心跳的时间(T)=当前时间 - recvedTime(上面记录的时间)。如果T大于某个设定值，就可以认为Client超时了。

1. 简单标志

收到一个心跳后，设置连接标志为true；

判断定时器时间到达，查看所有的标志，false的，认为对端超时了；true的将其设成false。

上面这种方法比上面简单一些，但检测某个Client是否离线的误差有点大。

### Demo

此处我们实现一个发送对象，例子简陋，实际在工作中还需按需求修改。

#### 实体类

package com.lee.entity;

import java.io.Serializable;

public class Entity implements Serializable {

private static final long serialVersionUID = 1L;

private String name;

private String sex;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

@Override

public String toString() {

return "Entity [name=" + name + ", sex=" + sex + "]";

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33

**服务端**

#### ServerHeart.java

package com.lee.server;

import java.io.ObjectInput;

import java.io.ObjectInputStream;

import java.net.ServerSocket;

import java.net.Socket;

import com.lee.entity.Entity;

public class ServerHeart extends Thread {

private ServerSocket server = null;

Object obj = new Object();

@Override

public void run() {

try {

server = new ServerSocket(9090);

while (true) {

Socket client = server.accept();

synchronized (obj) {

new Thread(new Client(client)).start();

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 客户端线程

\*

\* @author USER

\*

\*/

class Client implements Runnable {

Socket client;

public Client(Socket client) {

this.client = client;

}

@Override

public void run() {

try {

while (true) {

ObjectInput in = new ObjectInputStream(client.getInputStream());

Entity entity = (Entity) in.readObject();

System.out.println(entity);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

/\*\*

\* 程序的入口main方法

\*

\* @param args

\*/

public static void main(String[] args) {

System.out.println("开始检测客户端是否在线...");

new ServerHeart().start();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69

**客户端**

#### ClientHeart.java

package com.lee.client;

public class ClientHeart extends Thread {

@Override

public void run() {

try {

while (true) {

ClientSender.getInstance().send();

synchronized (ClientHeart.class) {

// this.wait(5000);

Thread.sleep(2000);

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 程序的入口main方法

\*

\* @param args

\*/

public static void main(String[] args) {

ClientHeart client = new ClientHeart();

client.start();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33

#### ClientSender.java

package com.lee.client;

import java.io.ObjectOutputStream;

import java.net.InetAddress;

import java.net.Socket;

import com.lee.entity.Entity;

public class ClientSender {

private ClientSender() {

}

Socket sender = null;

private static ClientSender instance;

public static ClientSender getInstance() {

if (instance == null) {

synchronized (ClientHeart.class) {

instance = new ClientSender();

}

}

return instance;

}

public void send() {

try {

sender = new Socket(InetAddress.getLocalHost(), 9090);

while (true) {

ObjectOutputStream out = new ObjectOutputStream(sender.getOutputStream());

Entity obj = new Entity();

obj.setName("xiaoming");

obj.setSex("男");

out.writeObject(obj);

out.flush();

System.out.println("已发送...");

Thread.sleep(5000);

}

} catch (Exception e) {

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45

