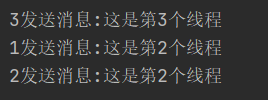
1. ThreadLocal类
   1. 我们先使用一个例子来说明一下ThreadLocal类

|  |
| --- |
| package com.company.多线程;  public class ThreadLocalDeamo {  public static void main(String[] args) {  Channel channel = new Channel();  new Thread(()->{  MessageDemo messageDemo = new MessageDemo();  messageDemo.setMes("这是第"+1+"个线程");  channel.setMessageDemo(messageDemo);  channel.send();  },"1").start();  new Thread(()->{  MessageDemo messageDemo = new MessageDemo();  messageDemo.setMes("这是第"+2+"个线程");  channel.setMessageDemo(messageDemo);  channel.send();  },"2").start();  new Thread(()->{  MessageDemo messageDemo = new MessageDemo();  messageDemo.setMes("这是第"+3+"个线程");  channel.setMessageDemo(messageDemo);  channel.send();  },"3").start();  }  }  class Channel {  private static final ThreadLocal<MessageDemo> THREAD\_LOCAL = new ThreadLocal();  private static MessageDemo messageDemo = null ;  public void setMessageDemo(MessageDemo mes){  messageDemo = mes;  }  public void send(){  //THREAD\_LOCAL.get().getMes();  System.out.println(messageDemo.getMes());  }  }  class MessageDemo{  private String mes ;  public void setMes(String mes){  this.mes = mes;  }  public String getMes(){  return (Thread.currentThread().getName()+"发送消息:"+this.mes);  }  } |

在该程序中的结果为: 

也就是说在线程中我们new MessageDemo 时进行覆盖，也就是说三个线程操作的时同一个对象，此时3线程先执行完，而后1线程还未执行完2线程就将messageDemo的mes属性给覆盖了。导致1线程的数据错误。此时为了解决这样的问题JDK1.2提供了ThreadLocal类给线程提供一个抓门。

正确范例:

|  |
| --- |
| package com.company.多线程;  public class ThreadLocalDeamo {  public static void main(String[] args) {  Channel channel = new Channel();  new Thread(()->{  MessageDemo messageDemo = new MessageDemo();  messageDemo.setMes("这是第"+1+"个线程");  channel.setMessageDemo(messageDemo);  channel.send();  },"1").start();  new Thread(()->{  MessageDemo messageDemo = new MessageDemo();  messageDemo.setMes("这是第"+2+"个线程");  channel.setMessageDemo(messageDemo);  channel.send();  },"2").start();  new Thread(()->{  MessageDemo messageDemo = new MessageDemo();  messageDemo.setMes("这是第"+3+"个线程");  channel.setMessageDemo(messageDemo);  channel.send();  },"3").start();  }  }  class Channel {  private static final ThreadLocal<MessageDemo> THREAD\_LOCAL = new ThreadLocal();  private static MessageDemo messageDemo = null ;  public void setMessageDemo(MessageDemo mes){  //messageDemo = mes;  THREAD\_LOCAL.set(mes);  }  public void send(){  //System.out.println(messageDemo.getMes());  System.out.println(THREAD\_LOCAL.get().getMes());  }  }  class MessageDemo{  private String mes ;  public void setMes(String mes){  this.mes = mes;  }  public String getMes(){  return (Thread.currentThread().getName()+"发送消息:"+this.mes);  }  } |

二、所以ThreadLocal是什么呢？

从名字我们就可以看到ThreadLocal叫做线程变量，意思是ThreadLocal中填充的变量属于当前线程，该变量对其他线程而言是隔离的。ThreadLocal为变量在每个线程中都创建了一个副本，那么每个线程可以访问自己内部的副本变量。

三、 应用场景

1、线程间数据隔离。

2、进行事务操作，用于存储线程事务信息。

3、数据库连接，Session会话管理。

我们使用数据库的时候首先就是建立数据库连接，然后用完了之后关闭就好了，这样做有一个很严重的问题，如果有1个客户端频繁的使用数据库，那么就需要建立多次链接和关闭，我们的服务器可能会吃不消，怎么办呢？如果有一万个客户端，那么服务器压力更大。

这时候最好ThreadLocal，因为ThreadLocal在每个线程中对连接会创建一个副本，且在线程内部任何地方都可以使用，线程之间互不影响，这样一来就不存在线程安全问题，也不会严重影响程序执行性能。是不是很好用。

注意:每一个线程通过ThreadLocal只能保存一个数据。