异步调用实现对比（java与C#）

异步调用主要用于当前程序的执行不用等待调用方法执行结束就可以继续执行。用一个最简单的例子来说，当前的方法要调用一个发送短信的方法，但是发送短信的方法调用了外部的接口，这样就导致短信发送方法耗费的时间很长，或者网络的故障等原因使得发送短信的方法影响了当前的程序。

在这种情况下就需要使用异步调用机制，一个正常的程序流程如下：

1、进入当前方法

2、调用远程方法

3、离开当前方法

在上面的3个步骤中，传统的方法执行步骤是按照1.2.3的顺来来执行，这样第2执行很长时间就会导致第3布一致等待执行。对于上面的步骤来说，异步调用就可执行完1之后立即执行2同时也执行3，即步骤3不用等待步骤2的执行结束。

异步调用的实现的原理就是多线程，在当前的程序上单独使用一个线程来调用方法，而不用阻塞当前的线程。这样就实现了异步调用。异步调用在java中使用多线程实现，在C#使用委托机制实现。

下面是一个最简单的异步调用实现，不用等待调用的结果，但是他表现了两种语言对异步调用的处理，如果需要更复杂的功能，再这种思想上改进即可。

**异步调用JAVA**

使用多线程匿名内部类实现异步调用。

/\*\*

\* 发送短信的线程

\* 发送短信使用多线程处理，作为一个单独的事务

\*/

**package** com.sinosoft.lis.pubfun.sms;

**import** com.sinosoft.lis.pubfun.GlobalInput;

**import** com.sinosoft.utility.TransferData;

**import** com.sinosoft.utility.VData;

/\*\*

\* **@author** ChengNing

\* **@date** 2012-9-24

\*/

**public** **class** **SmsSendAsync**{

**private** **final** **GlobalInput** cGlobalInput;

**private** **String** prtNo;

/\*\*

\* 构造子，发送短息必须的数据

\* **@param** prtNo

\* **@param** cGlobalInput

\*/

**public** SmsSendAsync(**String** prtNo) {

**this**.prtNo = prtNo;

}

/\*\*

\* 发送短信

\*/

**public** **void** send() {

//新定义线程来处理发送

**new** **Thread**(){

/\* 调用短信接口发送短信

\* 异步发送短信

\* 发送短信方法，组织数据，调用发送接口。

\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

**public** **void** run(){

**//这里可以写一个等待方法，让当前线程等待10秒钟现象更明显**

//控制台打印日志

**System**.*out*.println(Content);

}

}.start();

}

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(**String**[] args) {

**String** prtNo = "1001200912310155555";

//调用异步发送短信

**SmsSendAsync** sendAsync = **new** **SmsSendAsync**(prtNo);

sendAsync.send();

**System**.*out*.println("已经签单");

}

}

**异步调用C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace ConsoleApp

{

/// <summary>

/// 异步调用

/// </summary>

class AsyncCall

{

/// <summary>

/// 定义委托，用于异步调用

/// </summary>

private delegate void AsyncMethod();

/// <summary>

/// 实际执行的核心方法,调用发送短信

/// </summary>

private void SendSMS()

{

Console.WriteLine("SendSMS Method Start....");

//wait 10sec,相当于发送短信耗费了10秒，包括连接接口和短信发送平台

Thread.Sleep(10000);

Console.WriteLine("SendSMS Method End......");

}

/// <summary>

/// 使用异步方法调用

/// </summary>

private void AsyncCallMethod()

{

Console.WriteLine("Main Method Start....");

//将需要异步调用的方法委托给创建的AsyncMethod，不用等待SendSMS执行结束，当前线程继续往下执行

AsyncMethod async = new AsyncMethod(this.SendSMS);

async.BeginInvoke(null,null);

Console.WriteLine("Main Method End......");

}

/// <summary>

/// 使用同步方法调用

/// </summary>

private void SyncCallMethod()

{

Console.WriteLine("Main Method Start....");

//当前线程一致等待SendSMS方法执行

SendSMS();

Console.WriteLine("Main Method End......");

}

public static void Test()

{

AsyncCall test = new AsyncCall();

Console.WriteLine("测试同步调用");

//测试同步调用

test.SyncCallMethod();

Console.WriteLine("测试异步调用");

test.AsyncCallMethod();

}

/\*\*

\* 测试结果

\*

测试同步调用

Main Method Start....

Async Method Start....

Async Method End......

Main Method End......

测试异步调用

Main Method Start....

Main Method End......

Async Method Start....

Async Method End......

\*

\*

\*

\*

\*

\*

\*

\*/

}

}