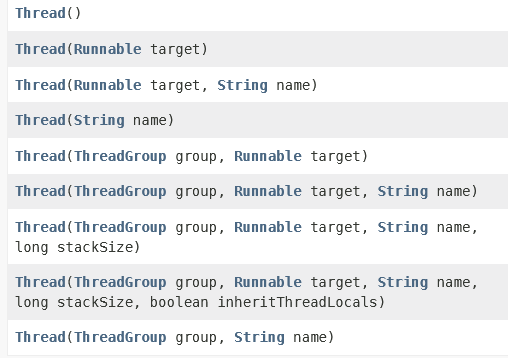
1. Callable
   * + 1. Callable是JDK1.5引入的，我们首先应该明确Runnbale与Callable的区别

|  |  |  |  |
| --- | --- | --- | --- |
|  | 是否会抛出异常 | 是否有返回值 | 方法名 |
| Runnable | 不会抛出异常 | 没有返回值 | run() |
| Callable | 会抛出异常 | 返回泛型指定的数据类型 | call() |

两个都是函数式接口。

* + - 1. 我们知道在Thread类里面有如下的构造方法:



其中Thread()构造方法中并没有Callable 类型的参数，因此我们无法直接new Thread（）并将Callable实现类直接传入。

现在我们来分析一下官方文档的继承/实现关系:

|  |  |
| --- | --- |
|  | 在构造方法中传入Callable参数。 |

从上面源码分析，我们可以看出futureTask类实现了 Runnable接口，而FutureTask类又可以传入Callable参数

因此我们可以知道，如何传入Callable进入Thread中。

范例:

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.concurrent.Callable;  import java.util.concurrent.ExecutionException;  import java.util.concurrent.FutureTask;  public class ThreadCallable {  public static void main(String[] args) throws Exception {  Phone phone = new Phone();  FutureTask futuretask = new FutureTask(phone);  new Thread(futuretask,"A线程").start();  String str = (String) futuretask.get();  System.out.println(str);  }  }  class Phone implements Callable<String> {  @Override  public String call() throws Exception {  System.out.println("welcome to Callable");  return "this is a phone";  }  } |

利用Java的多态性，new Thread()中也可以传入Runnable的子类，因此我们传入传入FutureTask到Thread中，利用FutureTask中的get方法可以获取到线程返回的值。

* + - 1. 通过上述的代码我们可以得到线程返回的值，但是同样我们也要清楚。如果我们过早的调用get()方法，而get方法所指定的线程又没有结束，那么主线程就会等待知道指定线程结束。

如:

|  |
| --- |
| public class ThreadCallable {  public static void main(String[] args) throws Exception {  Phone phone = new Phone();  FutureTask futuretask = new FutureTask(phone);  new Thread(futuretask,"A线程").start();    String str = (String) futuretask.get();    System.out.println("进行main线程中");    System.out.println(str);  }  } |

这里当我们调用get()方法时若A线程没有结束，那么主线程就会等待A线程，所以为了提升性能我们通常将get()方法放在最后面。