【第07话：都什么年代了，不要再说创建线程只有三种方式了】

创建线程的几种方式（月薪1万的回答示范）

创建线程的方式有三种。第一种是继承Thread类，第二种是实现Runnable接口，第三种是实现Callable接口。

创建线程的几种方式（月薪1.5万的回答示范）

继承Thread类代码示例

```
public class Demo {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        myThread.start();
    }
}
class MyThread extends Thread{
    @Override
    public void run() {
        System.out.println("子线程");
    }
}
```

实现Runnable接口

```
public class Demo {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread thread = new Thread(myRunnable);
        thread.start();
    }
}
class MyRunnable implements Runnable{
    @Override
    public void run() {
        System.out.println("子线程");
    }
}
```

实现Callable接口

```
public class Demo {
    public static void main(String[] args) {
        MyCallable myCallable = new MyCallable();
        FutureTask<Integer> ft = new FutureTask<>(myCallable);
        Thread thread = new Thread(ft);
        thread.start();
        try {
            // 可以获取结果
            System.out.println(ft.get());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
class MyCallable implements Callable<Integer> {
    @Override
    public Integer call() throws Exception {
        Random random = new Random();
        int i = random.nextInt( bound: 10);
        return i;
    }
}
```

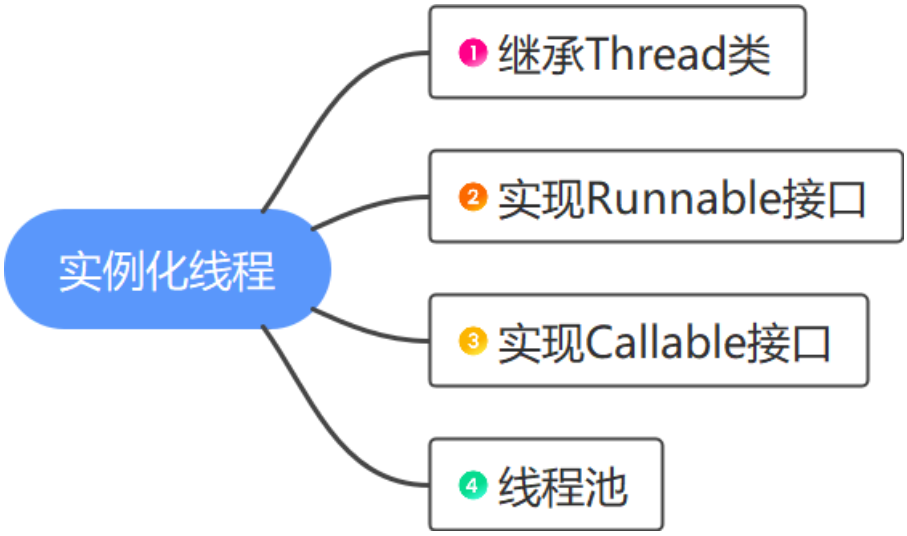
创建线程的方式有三种。第一种是继承Thread类，第二种是实现Runnable接口，第三种是实现Callable接口。

继承Thread类是需要创建一个类，作为Thread的子类。重写run方法。在run方法中添加线程功能。然后实例化这个子类就可以了。如果希望启动线程，可以使用start方法。

实现Runnable接口方式需要定义一个类，让类实现Runnable接口，并重写Runnable中抽象方法run方法。在run方法中添加线程功能。当需要创建线程时，把这个类示例作为Thread的构造方法参数传入。如果希望启动线程，需要通过Thread对象调用start()方法。

实现Callable接口方式需要定义一个类，实现Callable接口，实现时必须定义Callable泛型类型，表示可以通过线程执行最终返回结果类型。然后重写call方法，里面完成线程功能。最终返回一个结果。当需要实例化线程时时，需要把Callable实现类的对象作为FutureTask的构造参数，FutureTask是Runnable和Future接口的实现类。实例化FutureTask后，作为Thread构造参数传入。当线程执行结束后，可以通过FutureTask获得Callable实现类中call方法的返回结果。

创建线程的几种方式（月薪2万+的回答示范）



继承Thread类

```
public class Demo {
    public static void main(String[] args) {
        Thread thread = new Thread() {
            @Override
            public void run() {
                System.out.println("子线程");
            }
        };
        thread.start();
    }
}
```

Runnable接口

```
@FunctionalInterface
public interface Runnable {
    When an object implementing interface Runnable is used to create a thread, starting the thread
    causes the object's run method to be called in that separately executing thread.
    The general contract of the method run is that it may take any action whatsoever.
    See Also: Thread.run()
    public abstract void run();
}
```

基于实现Runnable接口创建线程

```
public class Demo {
    public static void main(String[] args) {
        Thread thread = new Thread(() -> System.out.println("子线程"));
        thread.start();
    }
}
```

Callable接口

```
@FunctionalInterface
public interface Callable<V> {
    Computes a result, or throws an exception if unable to do so.
    Returns: computed result
    Throws: Exception – if unable to compute a result
    V call() throws Exception;
}
```

实现Callable接口

```
public class Demo {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        FutureTask<Integer> ft = new FutureTask<>(() -> {
            Random random = new Random();
            return random.nextInt( bound: 10);
        });
        Thread thread = new Thread(ft);
        thread.start();
        System.out.println(ft.get());
    }
}
```

使用线程池

```
public class Demo {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 5);
        executorService.submit(()-> System.out.println("功能"));
        executorService.shutdown();
    }
}
```

在Java中创建线程一共有四种：分别是继承Thread类、实现Runnable接口、实现Callable接口和使用线程池。

继承Thread类方式只需要创建一个Thread类的子类并重写run方法就可以了。如果只使用一次，可简化为匿名内部类的形式。相比于定义一个Thread类的子类方式更加简单。这种方式在简单点的线程使用时可以使用。

实现Runnable接口方式。由于Runnable本身是一个函数式接口。里面只定义了一个名称为run的抽象方法。所以当只用一次Runnable这个实现类时，可以基于Lambda表达式简化写法。

创建线程时基于Thread类中构造方法参数为Runnable的构造方法，进行实现。简化为() -> {}的方式进行实现。简单方便。如果实现只有一行代码还可以胜利大括号。

这种方式是基于接口实现的，相比直接继承Thread类更加符合面向对象编程思想。

实现Callable接口方式。Callable接口是从Java 5开始出现的，里面只有一个无参抽象方法call，返回值类型与接口泛型相同，到了Java 8版本后定义为函数式接口。相比于继承Thread类和实现Runnable接口。实现Callable接口方式功能更加强大。能够获取线程执行结果，即call的返回值。这也是Java自带的线程之间数据传输的一种方式。也是Spring Cloud框架中Hystrix的请求合并中使用的一种线程实现方式。

既然Callable是函数式接口，使用Lambda表达式更加方便。直接使用() -> {}就可以实现，并重写call方法。

Thread类构造方法并没提供Callable类型参数。需要借助FutureTask，FutureTask是Runnable接口和Future接口的实现类。Future接口中get方法作用就是获取线程执行的最终结果。最后把FutureTask实例作为Thread构造方法参数就可以创建线程对象了。

如果线程执行完成想要获取线程执行结果，可以通过FutureTask的get方法获取。

在阿里巴巴开发规范中明确规定。当使用线程时，应该使用线程池。虽然线程池可能默认占用更多的内存资源。但是在并发场景或多线程下，线程池可以很好的管理线程直接调度，加快获取线程对象的时间。

Java 5版本的JUC中自带线程池Executor。可以使用ThreadPoolExecutor或ForkJoinPool线程池类型。设定好核心线程数后，会一次性创建多个线程对象。创建好线程池后通过submit把线程需要执行的任务提交给线程池。这就是创建线程的第四种方式。