Java

知

识

点

2019-12-06

学习计划：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 技术点 | 截至 | 完成情况 |
| 1 | Jojo的SpringCloud微服务安全，学完后用大白话总结在本word | 12月15号 | 15% |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |

**目录**

# 一 javaSE

## 1.1面向对象

面向对象与内存解析

方法重载与内存解析

this关键字

super关键字

继承

重写

继承中的构造方法

Object的toString方法

Object的equals方法

hashCode解释

static关键字

对象转型

多态

抽象类

final关键字

Interface

## 1.2 异常

## 1.3数组

数组的创建和使用

## 1.4常用类

String

StringBuffer

Integer缓冲

枚举类

## 1.5容器

容器API\_Collection

迭代器Iterator

List

Set

Compareable

Map

自动装箱&拆箱

泛型

## 1.6 IO

FileInputStream&FileOutputStream

FIleReader &FileWriter

缓冲流

转换流

DataIO&ByteArrayIO

打印流

对象流

对象序列化&反序列化

## 1.8 网络编程

网络基础

IP

TCP

UDP

TCP\_SOCKET

DUP

专题：线程池等-------@see多线程与高并发专题

# 二 Spring

# 三 SpringMVC

# 四Mybatis

# 五 互联网架构基础必备

## 多线程与高并发

## java内存模型JMM

## JVM调优案例实战

## 经常被问的操作系统问题

## 网络原理解读

## 设计模式&设计原则

## ~~UML~~

# 六 分布式必备基础技能

## 互联网架构演变

## 高负载高并发架构设计指导思想

## LVS

## Keepalived

## 流量介入层高性能服务

## Zookeeper

## 消息中间件

## ELK

## 分布式事务

## 单点登录

## 分布式任务调度

## 缓存

# 七 Redis & Netty

# 八 微服务

## RPC构建分布式

## SpringBoot

## SpringCloud

## Docker

## ~~Kebernetes（暂可不学）~~

## ~~Service Mesh（暂可不学）~~

## ~~微服务的学与思（面试就要喷这些玩意！）~~

# 九 分布式存储

## Mysql优化

## FastDFS

## OpenResty

# 十 工具

## Maven

## Git

## Jenkins

# 十一 源码解析

## JDK源码解析

## Spring源码解析

## MyBatis源码解析

## Dubbo源码解析

## SpringMVC源码解析（阅读SpringMVC源码解析书籍）

## ~~Netty源码解析~~

。

# 第五章 多线程与高并发

## 5.1，线程基础概念

### 5.1.1基本概念

#### **线程**

一个程序里边不同的执行路径

#### 线程和进程的区别

每个进程都有独立的代码和数据空间（进程上下文），进程间切换会有较大的开销。

线程可以看成是轻量级的进程，同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器，线程切换开销小。

多进程：在操作系统中能同时运行多个任务（程序）

多线程：在同一应用程序中有多个顺序流同时执行

我们的windows同时运行着好多个线程，所以说机器是支持多线程的。当然也支持多进程。windows、linux、unix都是多进程、多线程的，dos是只支持单进程的，同一个时间点只能执行一个进程执行。

|  |
| --- |
| 进程：进程是一个静态的概念，机器上的一个class文件，一个exe文件。  程序的执行过程，要把程序的代码放到内存里，放到代码区里，一个进程准备开始，进程已经产生了，但还没有开始执行，这叫进程。是一个静态的概念。平时说的进程的执行说的是进程里面主线程开始执行了。在机器里面运行的实际上都是线程。 |
|  |
| CPU计算速度快，把自己的时间分成一个个时间片，这个时间片执行你一会，下一个时间片执行它一会，轮着来。虽然有几十个线程，轮着执行由于速度很快，在我们看来就像是多个线程同时执行的。但实际上，一个时间点上，CUP只有一个线程在运行。（如果机器是多核，有多个CPU，才是真正意义上的多线程） |

### 5.1.2启动线程

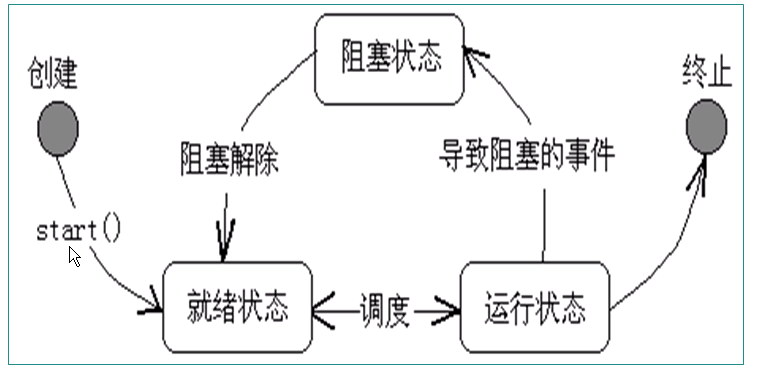
面试问，回答2种方式，他要是说3种，说线程池启动，线程池本质上也是2种中的某一种。

1. 定义线程类实现Runnable接口

Thread myThread = new Thread(target) //target为Runnable接口类型

1. 定义类继承Thread，重写run

### 5.1.3 线程的状态（初级版）



|  |
| --- |
| 一个线程类，new出来只是内存里的一个对象。调用start方法，不是立即执行，而是就绪状态，排队等候，因为CPU可能正在执行其他的线程。CPU高兴了，把对象调到CPU里才得以执行。执行的过程中，可能你的时间片到头了，还回到就绪状态去排队去，等候下次轮到你，一直到执行完了，就终止了。 |

### 5.1.4常用方法



#### 5.4.1 Thread.sleep()

线程睡眠指定的毫秒数

#### 5.4.2 Join

合并某个线程，等待某个线程执行结束，再恢复当前线程的运行。

例子程序：

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) {  MyThread2 t1 = **new** MyThread2(**"abcde"**);  t1.start();  **try** {  */\*\*  \* 把t1线程合并到主线程，即等待t1线程执行完，再执行主线程  \* 相当于方法调用  \*/* t1.join();  } **catch** (InterruptedException e) { }  *//等t1执行完了，才有机会执行这* **for**(**int** i=1;i<=10;i++){  System.***out***.println(**"i am main thread"**);  }  } }  **class** MyThread2 **extends** Thread {  MyThread2(String s){*//给线程起名字* **super**(s);  }   @Override  **public void** run(){  **for**(**int** i =1;i<=10;i++){  System.***out***.println(**"i am "**+getName());  **try** {  *sleep*(1000);  } **catch** (InterruptedException e) {  **return**;  }  }  } } |

执行结果：

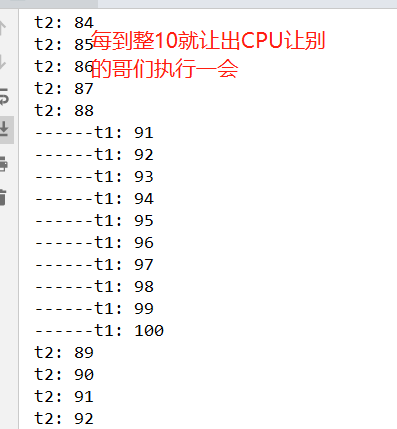




#### 5.4.3 Yield()

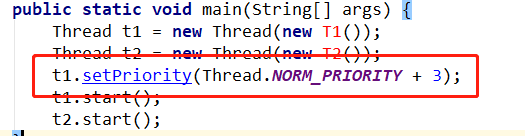
高风亮节，让出CPU，给其他线程执行的机会。

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) {  MyThread3 t1 = **new** MyThread3(**"------t1"**);  MyThread3 t2 = **new** MyThread3(**"t2"**);  t1.start();  t2.start();  } } **class** MyThread3 **extends** Thread {  MyThread3(String s){**super**(s);}  **public void** run(){  **for**(**int** i =1;i<=100;i++){  System.***out***.println(getName()+**": "**+i);  **if**(i%10==0){*//能被10整除，让出时间片，不明显  yield*();  }  }  } } |



#### 5.4.4 Priority

线程优先级



### 5.1.5 线程同步

### 5.1.6 Synchronized 锁升级

### 5.1.7 Synchronized 同步方法与非同步方法

### 5.1.8 Synchronized 锁重入

### 5.1.9 异常与锁

### 5.1.10 volatile 关键字

### 5.1.11 AtomicXXX 原子类

### 5.1.12 wait notify （面试高频）

线程同步

生产者消费者

## 5.2 JUC 同步工具

### 5.2.1 CAS自旋原理

### 5.2.2 ReentrantLock 可重入锁

### 5.2.3 Condition 条件等待与通知

### 5.2.4 LATCH门闩