* 学习目标

* 能够理解并发和并行的概念

  * 并发:同一段时间间隔，交替执行线程

  * 并行：同一时间，在多处理器里面，执行多个线程，

  * 并发编程：CPU执行性能

* 能够理解多线程的概述

  * 程序，进程，线程（单一顺序控制流，每个线程有独立栈空间）

  * Java内置支持多线程机制

* 能够掌握线程创建和调度

  * new Thread(){ run方法}.start();// 阿里巴巴提示：使用线程池

* 能够掌握Java线程生命周期

  * New-Runnable-Running--dead

  * Running--block(sleep,join,wait,IO,同步代码块)-Runnbale的状态

* 能够理解Thread与Runnable的区别

  * Thread与Runnable的核心一点

    * 从Java单继承

    * Thread是线程，但是Runnbale不是线程

* 能够掌握Thread常见的方法

    * setName,getName,setPriority,getPriority(5:1-10)

    * Thread.sleep

    * setDeamon(true):守护线程和用户线程，（后台进程和前台线程）

    * join

    * Thread.yeild( )

    * Object：wait,notify

    * interrupt()

* 能够掌握线程的启动和停止

    * 理解start 和run的区别

* 停止正在运行的线程：共享变量

* 停止正在阻塞的线程:interrupt，共享变量

---

* 回顾

* 自定义注解

* 语法：public @interface MyAnnotation --->默认在：Class

* 注解的本质是接口，它可以定义属性的方法

int age();

* 定义属性方法类型的声明

* 基本数据类型，String，Class，注解的类型，枚举，以上一维数组

* 自定义注解，某个类使用

* 元注解

* @Target(ElementType.Type|Method|Contructor|Field|Parameter)

* @Retention(RetentionPolicy.SOURCE|CLASS|RUNTIME)

* @Document,@Inherit

* @Repeatable

* 注解的反射

* AnnotatedElement

* Package,Class,Contructor,Method,Field,Parameter

* isAnntotationPresent

* getAnnotation(),getAnnotations()

* getDeclaredAnnotation(),getDeclaredAnnotations()

* getAnnotationsByType(),getDeclaredAnnotationsByType()

* 注解替换配置文件（javaweb，框架）

* 自定义@Value，@Test，@Table，@Colunm

* 能够理解并发和并行的概念

* 并发（concurrency）：并发是指两个或多个事件在同一时间间隔发生

* 并发(concurrency)：指在同一时刻只能有一条指令执行，但多个进程指令被快速的轮换执行，使得在宏观上具有多个进程同时执行的效果，但在微观上并不是同时执行的，只是把时间分成若干段，使多个进程快速交替的执行。

* 当有多个线程在操作时，如果系统只有一个 CPU，则它根本不可能真正同时进行一个以上的线程，它只能把 CPU 运行时间划分成若干个时间段，再将时间段分配给各个线程执行，在一个时间段的线程代码运行时,其它线程处于挂起状态.这种方式我们称之为并发（Concurrent）

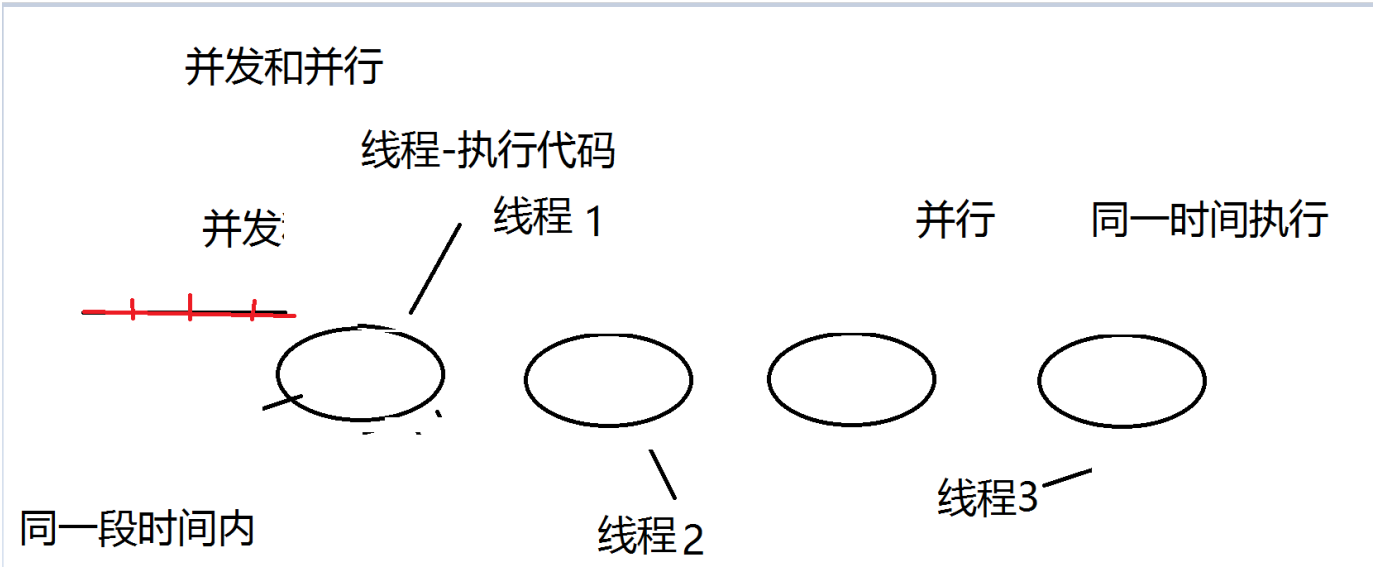
* 温馨提醒：并发编程的目标是充分的利用处理器的每一个核，以达到最高的处理性能

* 并行（parallel）：并行是指两个或者多个事件在同一时刻发生

* 并行 (parallel）：指在同一时刻，有多条指令在多个处理器上同时执行。所以无论从微观还是从宏观来看，二者都是一起执行的。

* 当系统有一个以上 CPU 时，则线程的操作有可能非并发。当一个 CPU 执行一个线程时，另一个 CPU 可以执行另一个线程，两个线程互不抢占 CPU 资源，可以同时进行，这种方式我们称之为并行（Parallel）。

* 假设CPU只有一个核或者多个核，画图解释他们的区别



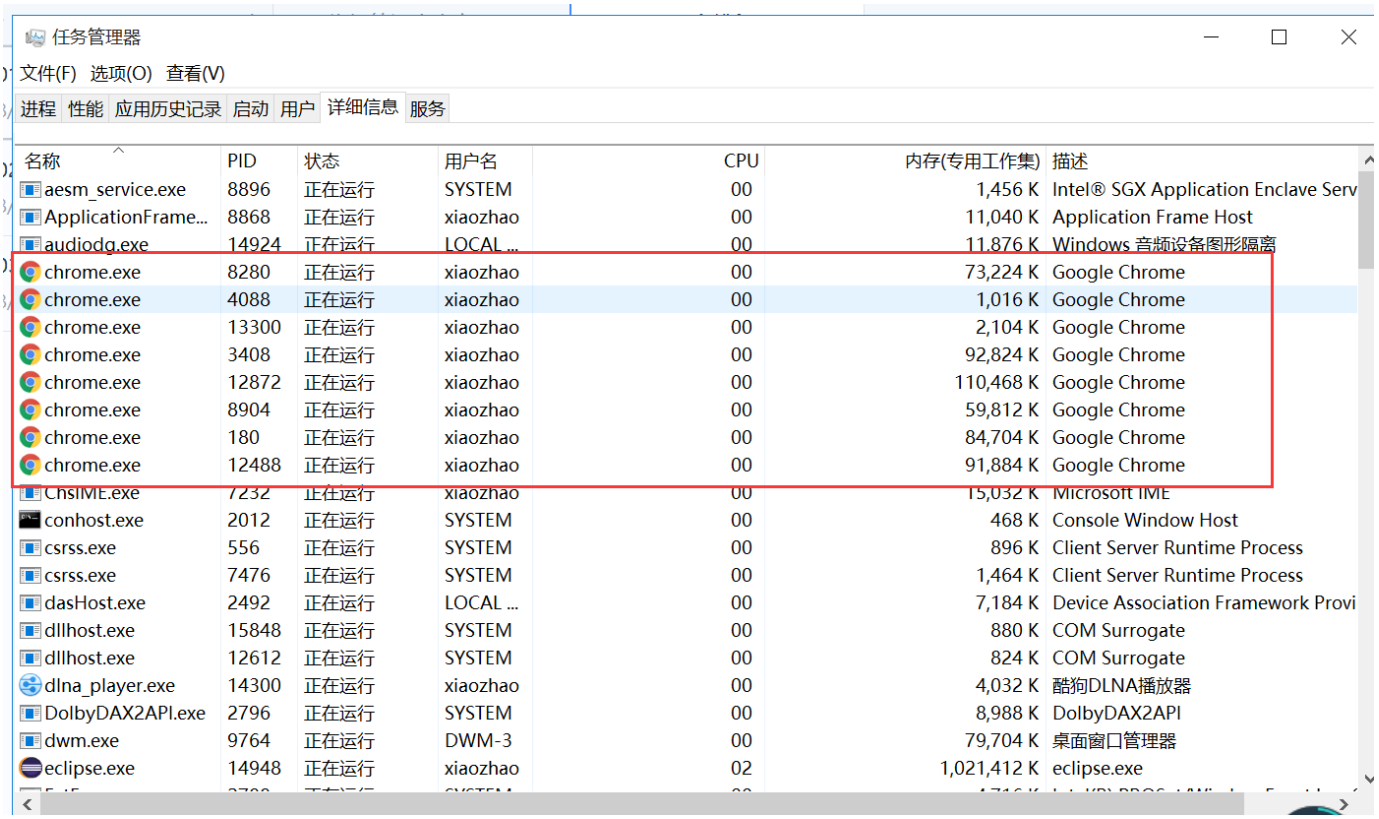
* 能够理解多线程的概述

* 程序、进程、线程的基本概念和关系

* 程序：是计算机指令的集合.程序是一组静态的指令集，不占用系统运行资源,不能被系统调度，也不能作为独立运行的单位,它以文件的形式存储在磁盘上

* 进程： 是一个程序在其自身的地址空间中的一次执行活动。比如，打开一个记事本，就是调用了一个进程。进程是资源申请、调度和独立运行的单位，因此，它使用系统中的运

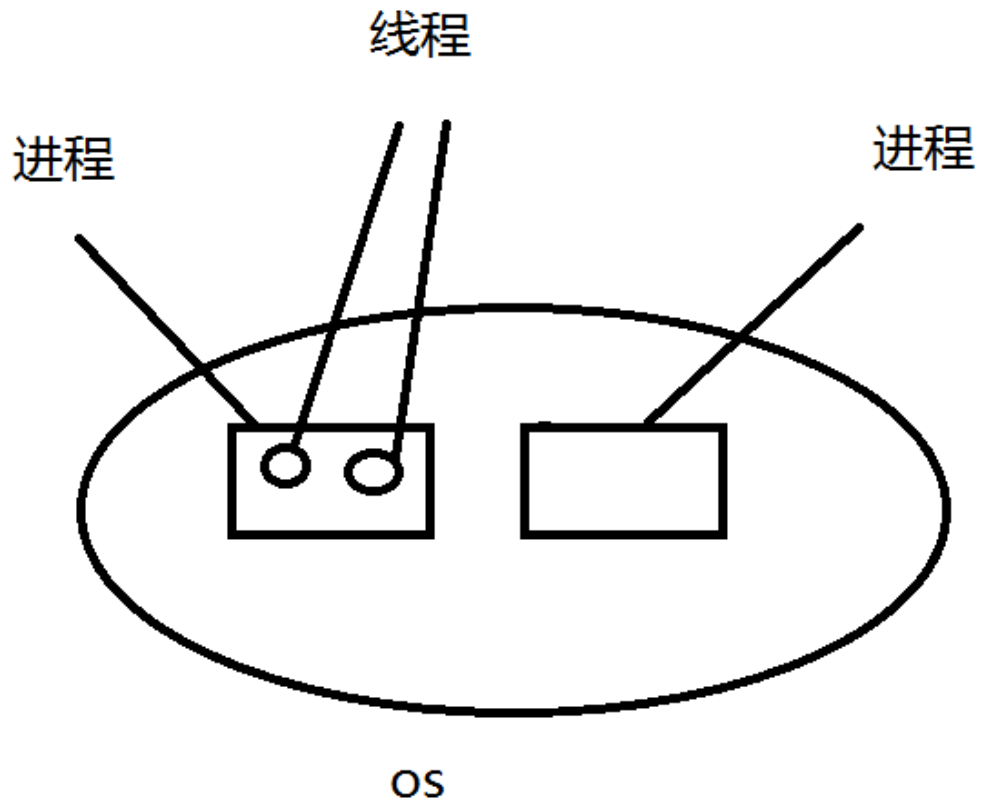
行资源；而程序不能申请系统资源，一个程序可以对应多个进程，例如著名的浏览器



名称	PID	状态	用户名	CPU	内存(专用工作集)	描述
aesm_service.exe	8896	正在运行	SYSTEM	00	1,456 K	Intel® SGX Application Enclave Serv
ApplicationFrame...	8868	正在运行	xiaozhao	00	11,040 K	Application Frame Host
audiodg.exe	14924	正在运行	LOCAL ...	00	11,876 K	Windows 音频设备图形隔离
chrome.exe	8280	正在运行	xiaozhao	00	73,224 K	Google Chrome
chrome.exe	4088	正在运行	xiaozhao	00	1,016 K	Google Chrome
chrome.exe	13300	正在运行	xiaozhao	00	2,104 K	Google Chrome
chrome.exe	3408	正在运行	xiaozhao	00	92,824 K	Google Chrome
chrome.exe	12872	正在运行	xiaozhao	00	110,468 K	Google Chrome
chrome.exe	8904	正在运行	xiaozhao	00	59,812 K	Google Chrome
chrome.exe	180	正在运行	xiaozhao	00	84,704 K	Google Chrome
chrome.exe	12488	正在运行	xiaozhao	00	91,884 K	Google Chrome
ChsIME.exe	7232	正在运行	xiaozhao	00	15,032 K	Microsoft IME
conhost.exe	2012	正在运行	SYSTEM	00	468 K	Console Window Host
csrss.exe	556	正在运行	SYSTEM	00	896 K	Client Server Runtime Process
csrss.exe	7476	正在运行	SYSTEM	00	1,464 K	Client Server Runtime Process
dasHost.exe	2492	正在运行	LOCAL ...	00	7,184 K	Device Association Framework Provi
dllhost.exe	15848	正在运行	SYSTEM	00	880 K	COM Surrogate
dllhost.exe	12612	正在运行	SYSTEM	00	824 K	COM Surrogate
dlna_player.exe	14300	正在运行	xiaozhao	00	4,032 K	酷狗DLNA播放器
DolbyDAX2API.exe	2796	正在运行	SYSTEM	00	8,988 K	DolbyDAX2API
dwm.exe	9764	正在运行	DWM-3	00	79,704 K	桌面窗口管理器
eclipse.exe	14948	正在运行	xiaozhao	02	1,021,412 K	eclipse.exe

* 线程： 一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务

* 多线程执行时，在栈内存中，其实每一个执行线程都有一片自己所属的栈内存空间。进行方法的压栈和弹栈



* 使用多线程编程的好处

* 可以更好的实现并行

* 恰当地使用线程时，可以降低开发和维护的开销，并且能够提高复杂应用的性能。

* CPU在线程之间开关时的开销远比进程要少得多。因开关线程都在同一地址空间内，只需要修改线程控制表或队列，不涉及地址空间和其他工作。

* 创建和撤销线程的开销较之进程要少。

* Java在多线程应用中的优势

* Java 给多线程编程提供了内置的支持

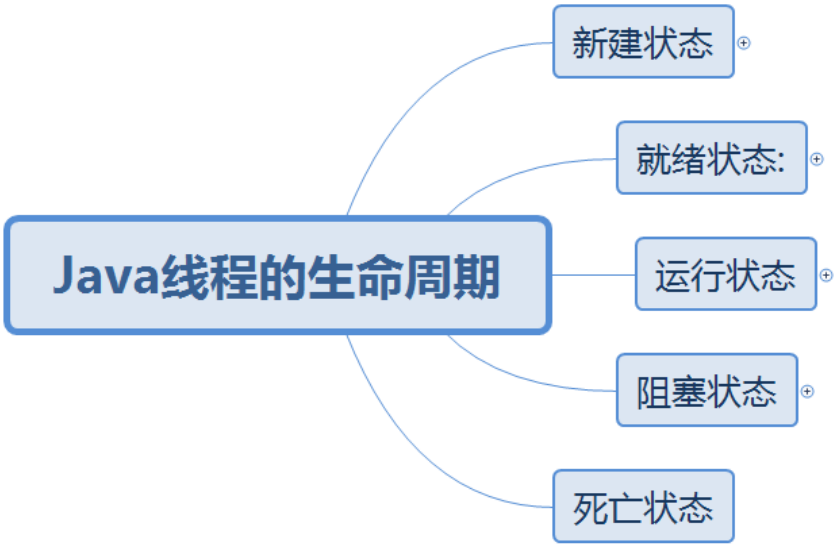
* 多线程操作会增加程序的执行效率。各线程之间切换执行，时间比较短，看似是多线程同时运行，但对于执行者CPU来说，某一个时刻只有一个线程在运行

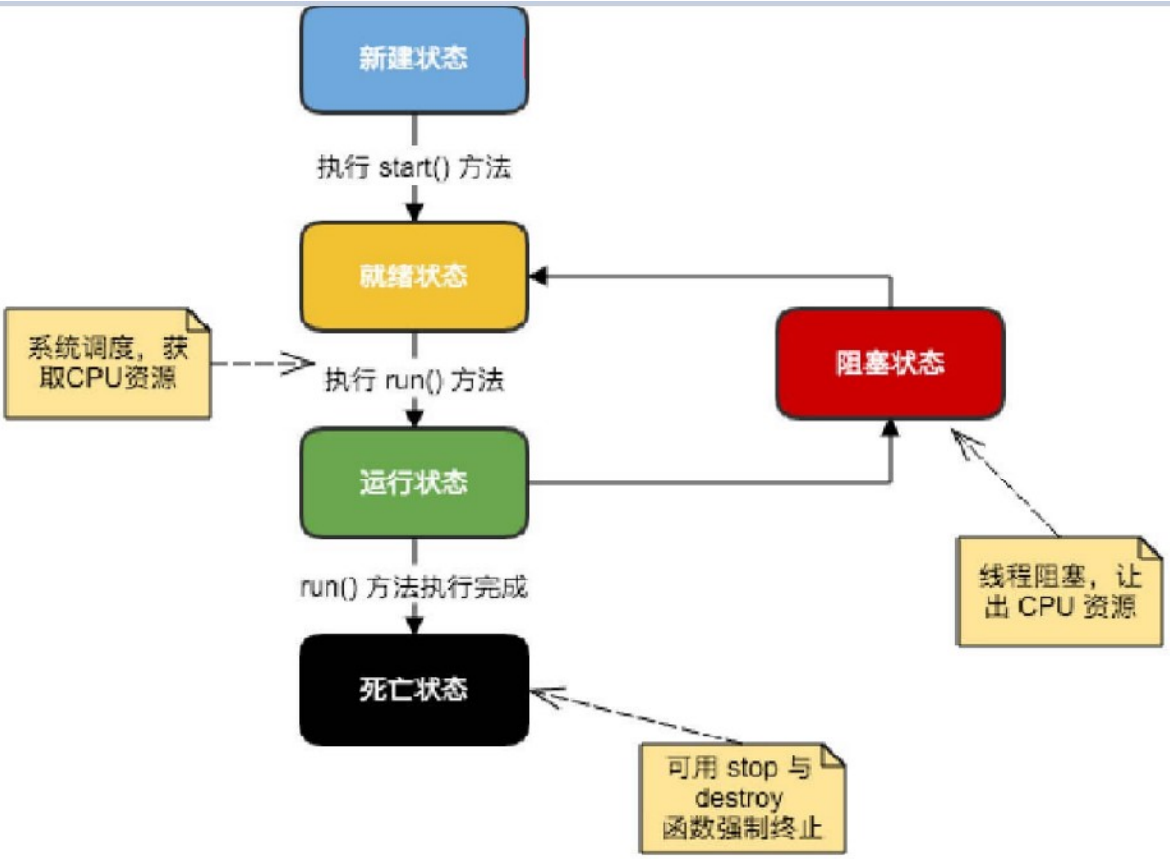
* 能够掌握线程创建和调度

```
* 打印main线程的名字，main方法也是一个线程在执行
public class Main {
    public static void main(String[] args) {
        System.out.println("main 线程名字: "+Thread.currentThread().getName());
    }
}
* 创建线程
public class MThread extends Thread {
    @Override
    public void run() {
        System.out.println("为被执行线程的名字:"+Thread.currentThread().getName()
    }
}
* 调度线程（启动线程）
public static void main(String[] args) {
        System.out.println("main 线程名字: "+Thread.currentThread().getName());
        MThread thread=new MThread();
        // 设置线程的名字
        thread.setName("mfirstThread");
```

```
        // 启动线程
        thread.start();
    }
结果:
main 线程名字: main
为被执行线程的名字:mfirstThread

* 匿名内部类的写法
 new Thread(){
            @Override
            public void run() {
                super.run();
                System.out.println("为被执行线程的名字:"+Thread.currentThread().g
            }
        }.start();
```

* 能够掌握Java线程生命周期





* 新建状态

新建状态

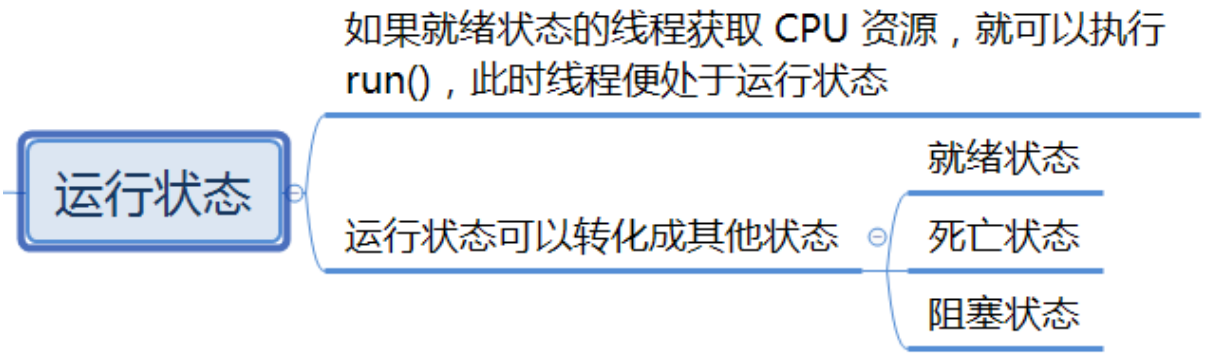
使用 new 关键字创建Thread 类或其子类建立一个线程对象后，该线程对象就处于新建状态。它保持这个状态直到程序 start() 这个线程

* 就绪状态:

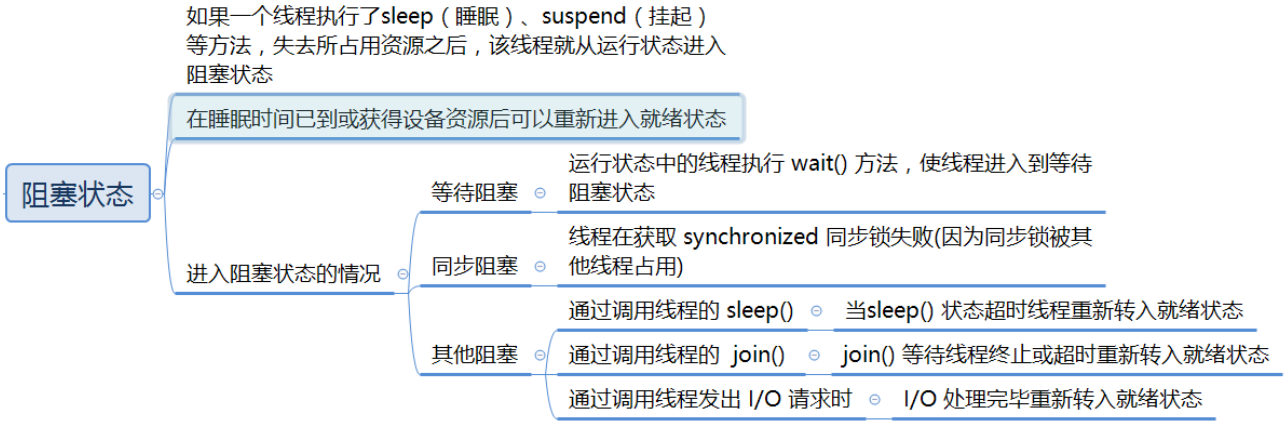
就绪状态:

当线程对象调用了start()方法之后，该线程就进入就绪状态。就绪状态的线程处于就绪队列中，要等待JVM里线程调度器的调度

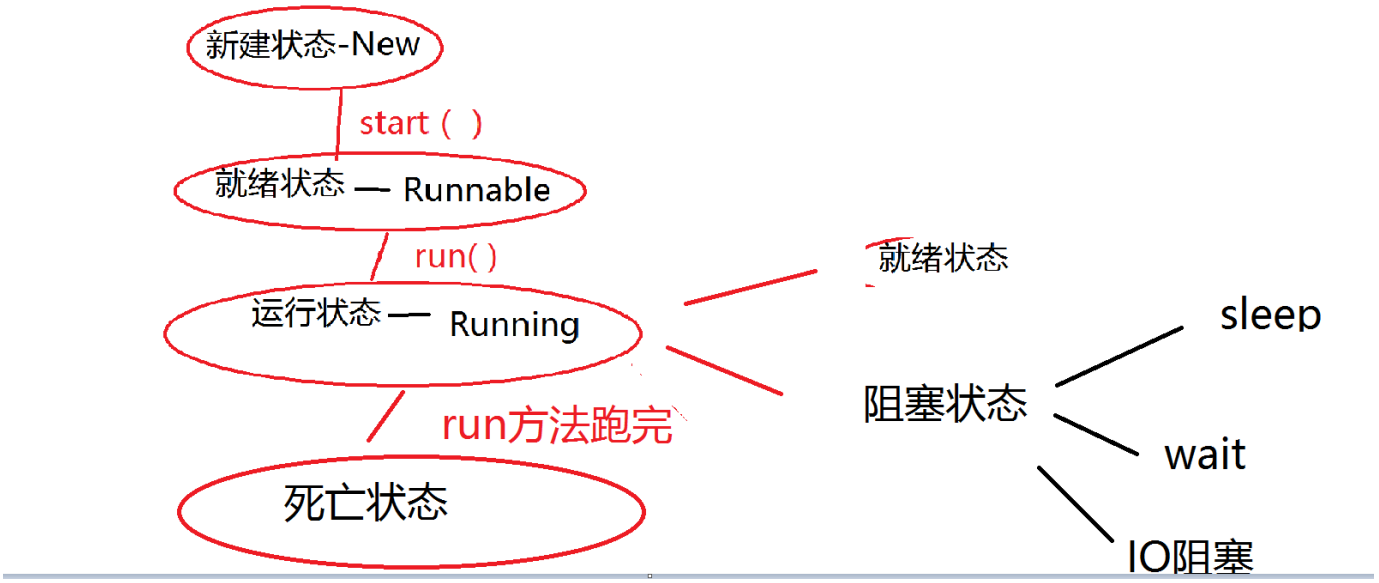
* 运行状态



* 阻塞状态

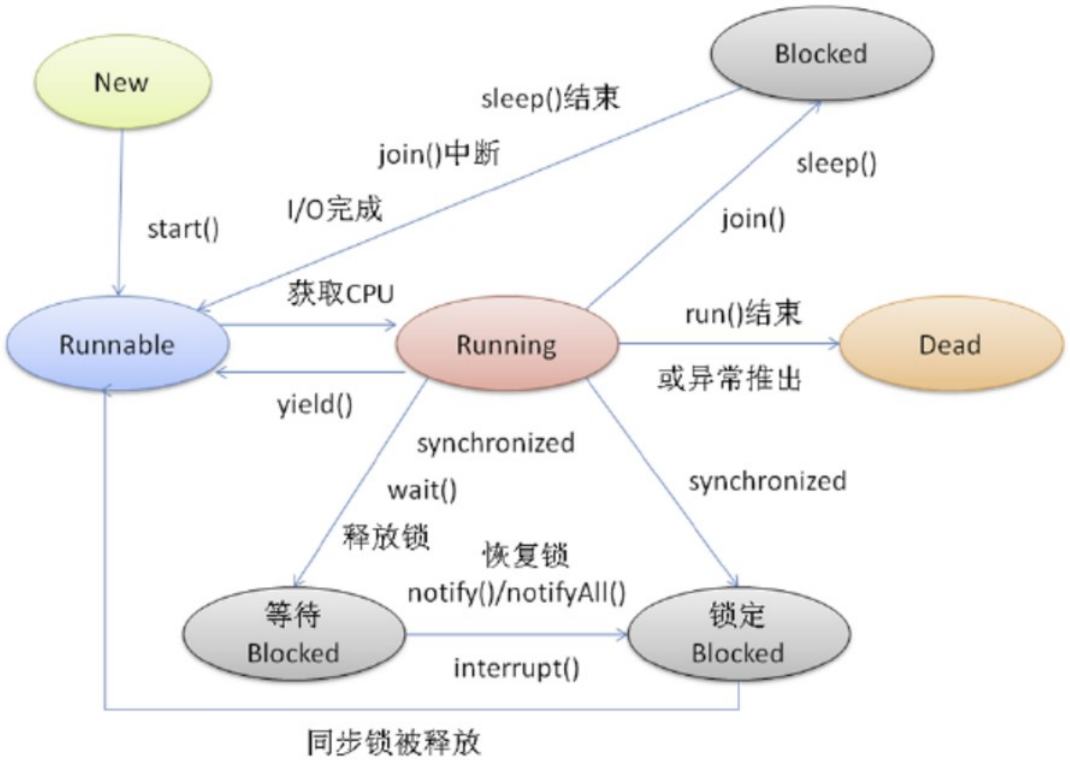


* 死亡状态



* 线程声明周期详细图（状态转换详细）

线程状态



线程状态转换

* 能够理解Thread与Runnable的区别

* 直接继承Thread类实现线程的方法存在局限性：由于Java是典型的单亲继承体系，因此一旦类继承Thread之后就不能再继承其他父类，对于一些必须通过继承关系来传播的特性这种方式显然会造成困扰 ，可以通过实现Runnable接口的方式来实现线程

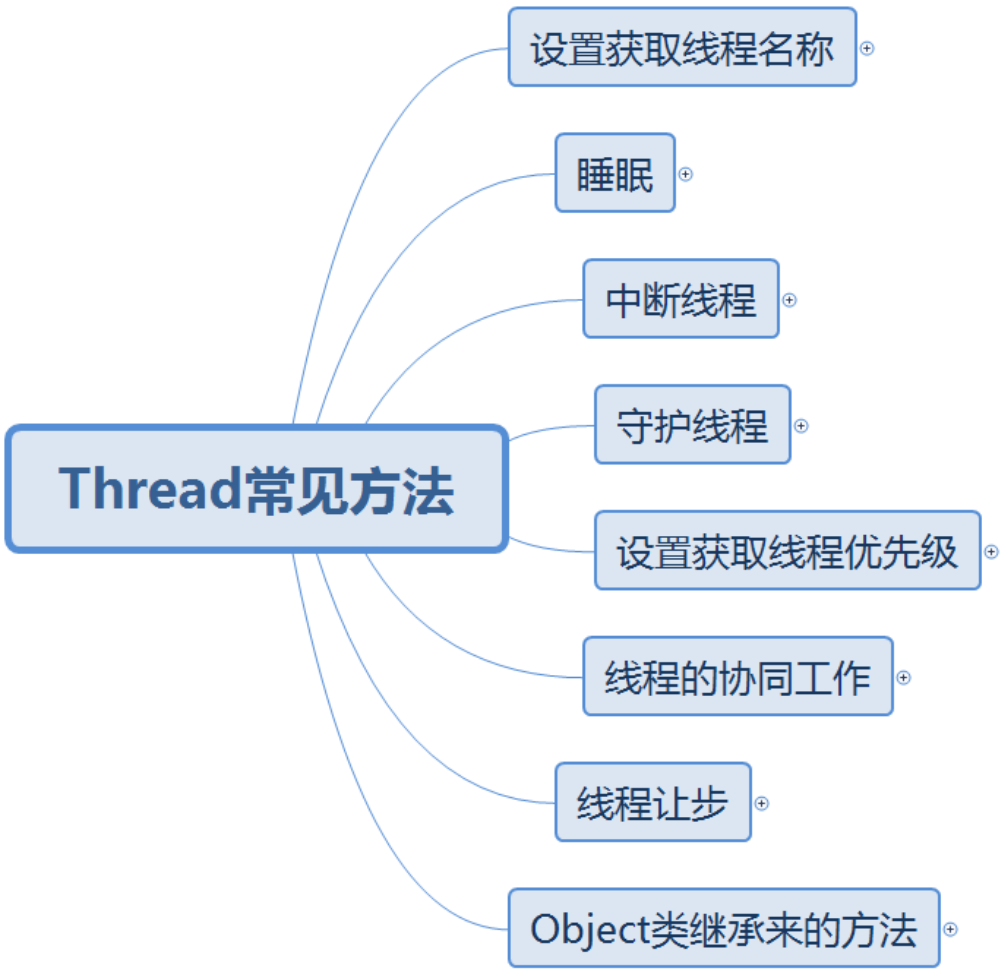
```
public class B extends A implements Runnable{
    // B 也是线程
    @Override
    public void run() {
        // 线程的代码
        System.out.println("线程的代码");
    }

}
```

1 单继承不能再继承Thread
2 可以实现Runnable接口，但Runnable不是线程

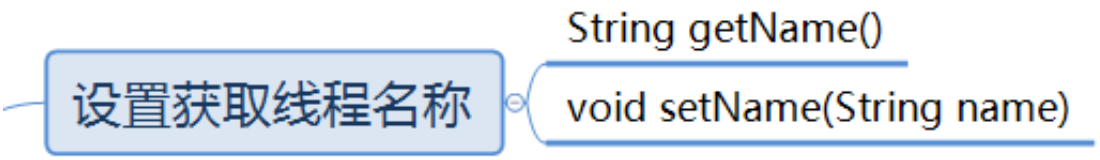
```
new Thread(new B()).start();
```

```
* 常见的写法
 public static void main(String[] args) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("为被执行线程的名字:"+Thread.currentThread().g
            }
        }).start();
    }
* 结果
 为被执行线程的名字:Thread-0
```

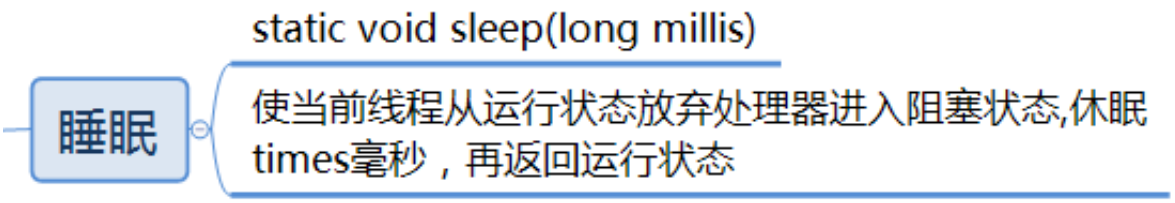
* 能够掌握Thread常见的方法



* 设置获取线程名称

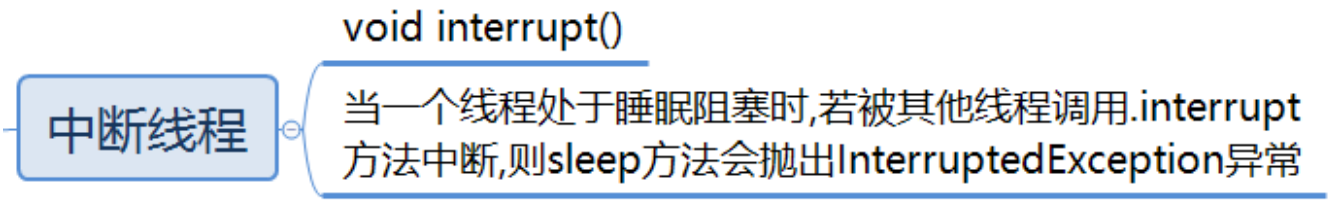


* 睡眠




```
案例一：每隔一秒显示一次，可能会出现跳秒
public static void main(String[] args) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss"
                while (true){
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println(sdf.format(new Date()));
                }
            }
        }).start();
    }

```

* 中断线程

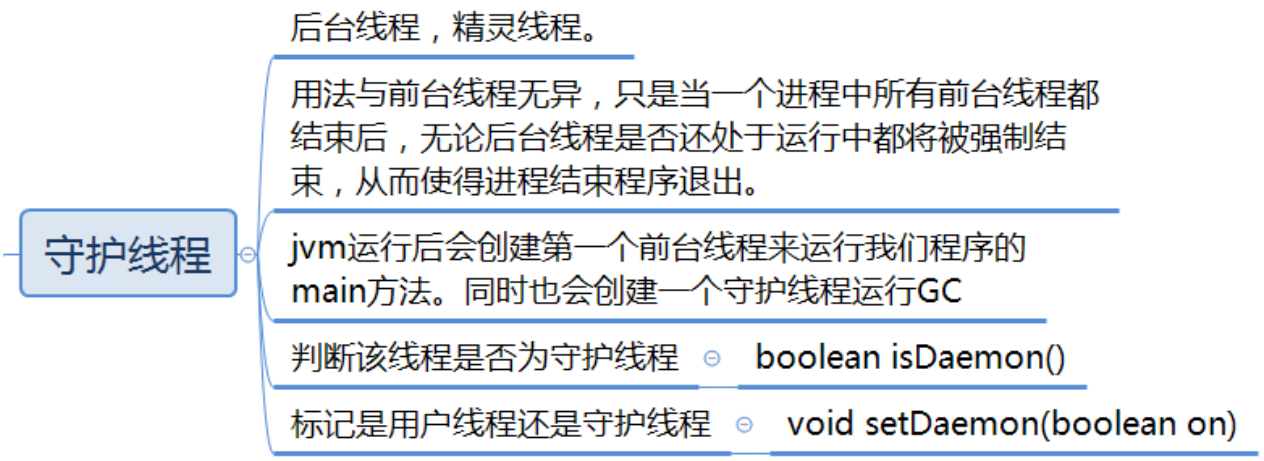


案例：

```
public class Main4 {
    public static void main(String[] args) {
        //处于睡眠阻塞的线程
        Thread t1=new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("xiaohei:我开完了黑，睡觉了");
                try {
                    Thread.sleep(1000*60*60*5);
                } catch (InterruptedException e) {
                    System.out.println("xiaohei:我操，你干嘛?");
                }
            }
        });
        t1.start();
        // 中断睡眠阻塞的线程
        new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("xiaobai:开始砸门!");
                for (int i = 0; i <10 ; i++) {
                    System.out.println("xiaobai:我砸我砸我砸...");
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("xiaobai:门砸开了");
                System.out.println("xiaobai:搞定");
                t1.interrupt();
```

```
            }
        }).start();
    }
}
结果:
xiaohei:我开完了黑，睡觉了
xiaobai:开始砸门!
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:我砸我砸我砸...
xiaobai:门砸开了
xiaobai:搞定
xiaohei:我操，你干嘛?
```

* 守护线程

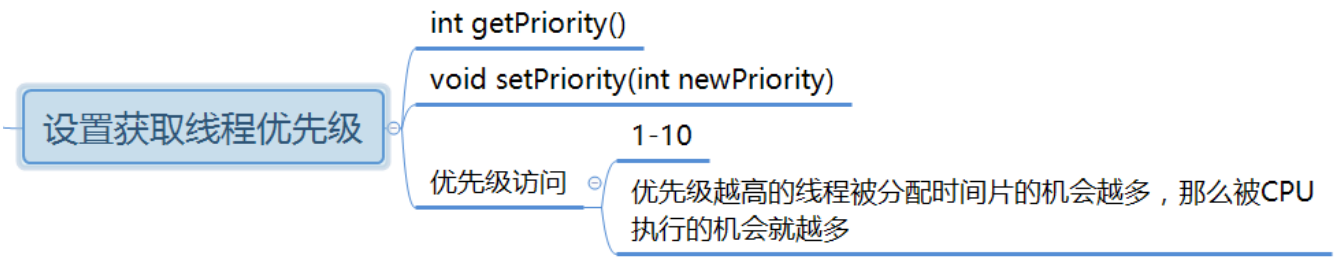



```
案例一:
public class Main5 {
    public static void main(String[] args) {
        // 前台线程
        new Thread(new Runnable() {
            @Override
```

```
            public void run() {
                for (int i = 0; i <5 ; i++) {
                    System.out.println("xiaohei:我还活着...");
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }).start();

        Thread d=new Thread(new Runnable() {
            @Override
            public void run() {
                while (true){
                    System.out.println("xiaobai:你假如去世了，我也要离开了...");
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        d.setDaemon(true);
        d.start();
    }
}
结果:
xiaohei:我还活着...
xiaobai:你假如去世了，我也要离开了...
xiaohei:我还活着...
xiaobai:你假如去世了，我也要离开了...
xiaohei:我还活着...
xiaobai:你假如去世了，我也要离开了...
xiaohei:我还活着...
xiaobai:你假如去世了，我也要离开了...
xiaohei:我还活着...
xiaobai:你假如去世了，我也要离开了...
```

```
47
48 Process finished with exit code 0
```

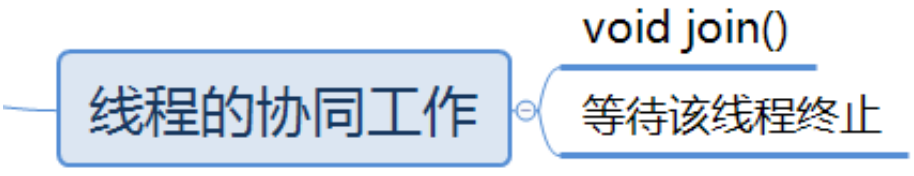
* 设置线程的优先级




```
1  * 案例
2  public class Main6 {
3      public static void main(String[] args) {
4          Thread min=new Thread(new Runnable() {
5              @Override
6              public void run() {
7                  for (int i = 0; i <10; i++) {
8                      System.out.println("min...");
9                      try {
10                         Thread.sleep(100);
11                     } catch (InterruptedException e) {
12                         e.printStackTrace();
13                     }
14                 }
15             }
16         });
17
18         Thread norm=new Thread(new Runnable() {
19             @Override
20             public void run() {
21                 for (int i = 0; i <10; i++) {
22                     System.out.println("norm...");
23                     try {
24                         Thread.sleep(100);
25                     } catch (InterruptedException e) {
26                         e.printStackTrace();
```

```
                    }
                }
            }
        });

        Thread max=new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i <10; i++) {
                    System.out.println("max...");
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        min.setPriority(Thread.MIN_PRIORITY);
        // 默认可以不写
        norm.setPriority(Thread.NORM_PRIORITY);
        max.setPriority(Thread.MAX_PRIORITY);
        max.start();
        norm.start();
        min.start();
    }
}
```

* 线程协同工作

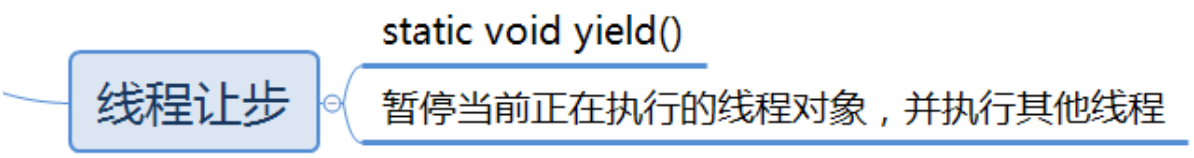



```
* 案例一
package com.lg.test4;

```

```java
/**
 * @author xiaozhao
 */
public class Main7 {
    public static boolean isFinish=false;
    public static void main(String[] args) {
        // 下载图片的线程
        Thread downThread=new Thread(new Runnable() {
            @Override
            public void run() {
                // 模拟开始下载
                System.out.println("download:开始下载图片...");
                for (int i = 1; i <= 100; i++) {
                    System.out.println("download:已完成" + i + "%");
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("download:图片下载完毕");
                isFinish = true;
            }
        });
        downThread.start();

        // 显示图片的线程
        Thread showImgThread=new Thread(new Runnable() {
            @Override
            public void run() {
                // 等待下载完
                try {
                    downThread.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                if(!isFinish){
                    throw new RuntimeException("show:图片还没有下载完");
                }
                System.out.println("show:图片显示完成！");
```

```
            }
        });
        showImgThread.start();

    }
}


* 结果
download:开始下载图片...
download:已完成1%
...
download:已完成100%
download:图片下载完毕
show:图片显示完成!
```

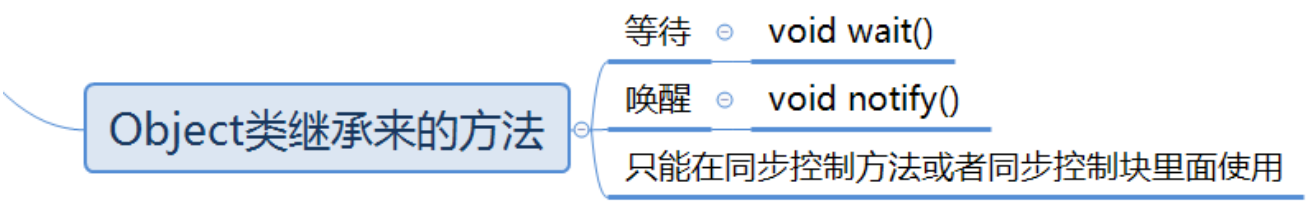
* 线程让步




```
* 案例一
public class Main8 {
    public static void main(String[] args) {
        Thread t2=new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i <10 ; i++) {
                    System.out.println("t2:"+i);
                }
            }
        });

        Thread t1=new Thread(new Runnable() {
            @Override
```

```
            public void run() {
                for (int i = 0; i <100 ; i++) {
                    System.out.println("t1:"+i);
                    if(i==10){
                        t2.start();
                        Thread.yield();// 暂停当前正在执行的线程对象，并执行其他线
                    }
                }
                long end=System.currentTimeMillis();
            }
        });
        t1.start();
    }
}
```

* Object类继承来的方法




```
* 案例
public class Main9 {
    public static boolean isFinish=false;
    public static Object lock=new Object();
    public static void main(String[] args) {
        // 下载图片的线程
        Thread downThread=new Thread(new Runnable() {
            @Override
            public void run() {
                // 模拟开始下载
                System.out.println("download:开始下载图片...");
                for (int i = 1; i <= 100; i++) {
                    System.out.println("download:已完成" + i + "%");
                    try {
```

```
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("download:图片下载完毕");
            isFinish = true;
            synchronized (lock){
                lock.notify();
            }
            System.out.println("download:开始下载视频...");
            for (int i = 1; i <= 100; i++) {
                System.out.println("download:已完成" + i + "%");
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    });
    downThread.start();

    // 显示图片的线程
    Thread showImgThread=new Thread(new Runnable() {
        @Override
        public void run() {
            // 等待图片下载完
            try {
                synchronized (lock){
                    // wait()阻塞会在以下两种情况被解除,
                    // 1:当download线程结束.
                    // 2:当调用了download的notify()
                    lock.wait();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if(!isFinish){
                throw new RuntimeException("show:图片还没有下载完");
```

```
55                  }
56                  System.err.println("show:图片显示完成！");
57              }
58          });
59          showImgThread.start();
60
61      }
62  }
```

* 能够掌握线程的启动和停止

 * 线程的启动

 * 可以使用start方法来启动（调用后并不是立即执行多线程代码，而是使该线程变为就绪状态，什么时候运行是由操作系统调度决定的）

 * 直接调用run方法，不是启动线程的方法，而是正常类型的对象调用跟线程无关。

```
// 新建状态
MThread mThread=new MThread();
// 就绪状态
//mThread.run();// 直接调用，当成普通类来使用
mThread.start();//等着CPU去调度进入运行状态
//运行状态
//死亡状态
```

* 线程停止

 * 线程的停止远比线程的启动情况复杂

 * 在Thread线程类中提供了stop()方法用于停止一个已经启动的线程，但是它已经被废弃，不建议使用，因为它是不安全的，容易引起死锁。

```
@Deprecated
public final void stop()
```

 * 停止一个正在运行的线程

 * 线程对象的run()方法所有代码执行完成后，线程会自然消亡，因此如果需要在运行过程提前停止线程，可以通过改变共享变量值的方法让run()方法执行结束

```
2
3  public class TestStopThread extends Thread{
4      private boolean flag=true;
5      @Override
6      public void run() {
7          while(flag) {
8              System.out.println("xxx");
9          }
10     }
11
12     public void stopThread() {
13         flag=false;
14     }
15
16     public static void main(String[] args) {
17         TestStopThread tt=new TestStopThread();
18         tt.start();
19         tt.stopThread();
20     }
21 }
22
```

```
1  * 案例
2  public class TestStopThread extends Thread{
3      private boolean flag=true;
4      @Override
5      public void run() {
6          while(flag) {
7              System.out.println("xxx");
8          }
9      }
10     public void stopThread() {
11         flag=false;
12     }
13     public static void main(String[] args) {
14         TestStopThread tt=new TestStopThread();
15         tt.start();
16         tt.stopThread();
17     }
18 }
19
```

* 停止一个阻塞线程

* 使用Thread提供的interrupt()方法，因为该方法虽然不会中断一个正在运行的线程，但是它可以使一个被阻塞的线程抛出一个中断异常，从而使线程提前结束阻塞状态，退出堵塞代码

```
    public static void main(String[] args) {
        StopSleepThread tt=new StopSleepThread();
        tt.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
//      tt.stopThread();
        tt.interrupt();
    }
```

```
    @Override
    public void run() {
        while(flag) {
            try {
                Thread.sleep(1000*60*60);
            } catch (InterruptedException e) {
                e.printStackTrace();
                flag=false;
            }
        }
    }
```

```
1  public class StopSleepThread extends Thread{
2      private boolean flag=true;
3      @Override
4      public void run() {
5          while(flag) {
6              try {
7                  Thread.sleep(1000*60*60);
8              } catch (InterruptedException e) {
9                  e.printStackTrace();
10                 flag=false;
11             }
12         }
13     }
14
15     public void stopThread() {
```

```
        flag=false;
    }

    public static void main(String[] args) {
        StopSleepThread tt=new StopSleepThread();
        tt.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        tt.interrupt();
    }
}
```