String类的常用方法

构造方法：

public String (byte[] bytes)

public String (char[] value)

普通方法：

public **char charAt(int index)**--->返回第index个字符

public **boolean contacins(CharSequence s)** --->当且仅当包含指定char返回true

public **int length()**--->返回字符串长度

public **int indexOf(String str)**--->返回出现str的第一个位置

public **boolean equals(Object obj)** --->将此字符串与指定对象比较

public **boolean equalsIgnoreCase(String str)** --->与str比较是否相等（忽略大小写）

public **String replace(char oldChar,char newChar)** --->用newChar替换oldChar

public **boolean startsWith(String str)** --->是否以str开头

public **boolean endsWith(String str)** --->是否以str结尾

public **String toUpperCase()**--->返回大写形式

public **String toLowerCase()**--->返回小写形式

public **String subString(int i)** --->返回从i开始到结尾的子字符串

public **String substring(int i ,int j)** --->返回从i到j的子字符串

public **String trim()**--->去掉开头和结尾空格

public **char[] toCharArray**--->将字符串转换为一个新的字符数组

## reflect反射

1. 获取类的class对象
2. Class.forName(“包名.类名”);--->常用
3. 类.class;
4. 对象.getClass();
5. 获得构造器

getDeclaredConstrustors()--->返回本类中所有的构造方法

getDecleardConstructor(类型.class...) --->返回本类中指定的构造方法

getConstructors()--->返回所有公开的构造方法

getConstructor(类型.class...)--->返回指定的构造方法

1. 实例化对象

newInstance()

注意：newInstance()是调用空构造，如果空构造不存在，会出现异常。由此可知，使用其他构造器创建对象比较麻烦，使用空构造特别简单。确保空构造存在。

私有构造器实例化对象时必须打开权限

1. 接口与父类

getInterfaces()--->确定此对象所表示的类或接口实现的接口

getSuperClass()--->返回此class所表示的实体（类、接口、基本类型或void）的超类class

1. 修饰符

getModifiers()--->返回修饰符

1. 属性

getDeclaredFields()--->返回本类中所有属性的一个Field数组

getFields()--->返回本类中所有公开属性的一个Field数组

1. 方法

getDeclaredMethod(方法名，参数)---> 获取某个方法，可以是私有

getDeclaredMethods()--->获取所有方法（本类中，包括私有）

getMethod(方法名，参数) --->获取某个方法，只能是公开的

getMethods()--->获取所有公共方法（本类，包括父类的）

1. 数组

Array

1. 创建数组

Object obj = Array.newInstance(int.class,5)

1. 获取大小

If(obj.getClass().isArray()){ 3) 判断是否为数组

System.out.println(Array.getLength(obj));

4) 设置值

Array.set(obj,0,100);

5) 获取值

System.out.println(Array.get(obj,0));

}

1. 类加载器

Bootstrap ClassLoader 此加载器采用c++编写，一般开发中很少见。

Extension ClassLoader 用来进行扩展类的加载，一般对应的是 jre\lib\ext目录中的类AppClassLoader 加载 classpath指定的类，是最常用的加载器。同时也是java中默认的加载器。 了解即可

1. 反射的相关操作
2. 操作属性

//1、获取Class对象

Class<?> clz=Class.forName("com.shsxt.ref.simple.User");

//2、获取对象

User u=(User)clz.newInstance();

//3、设置属性

Field field=clz.getDeclaredField("uname");

field.setAccessible(true);//打开权限

field.set(u, "0523");

//4、获取此属性

System.out.println(field.get(u));

1. 调用方法
2. 方法调用

//1、获取Class对象

Class<?> clz=Class.forName("com.shsxt.ref.simple.User");

//2、获取对象

User u=(User)clz.newInstance();

//3、获取方法

Method m =clz.getMethod("coding", String.class,String.class);

//4、成员方法的调用

m.invoke(u, "反射","多个参数");

//若是静态方法，传递null即可 因为静态方法属性类，不属于对象 m=clz.getMethod("testStatic",int.class);

m.invoke(null, 100);//与对象无关

1. 操作setter与getter 访问器

"set" + name.substring(0, 1).toUpperCase() + name.substring(1);

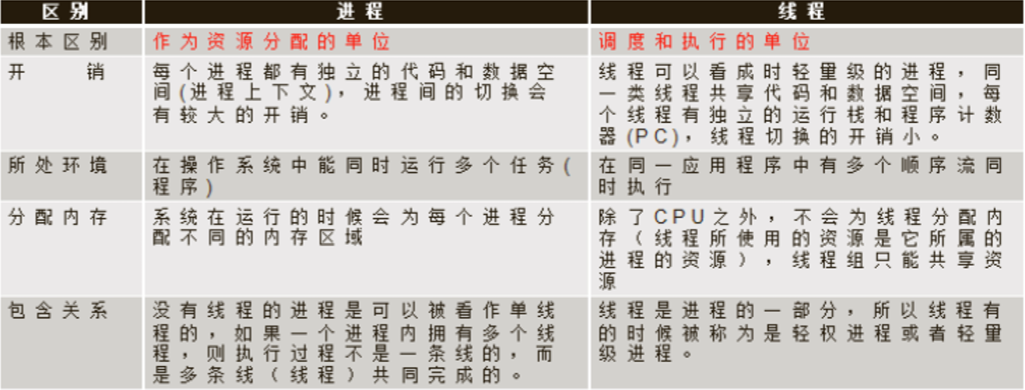
## IO流

1. 步骤：
2. 建立联系File file = new File(“ ”);
3. 选择流
4. 操作：写出 读取
5. 释放资源
6. 抽象父类InputStream OutputStream Reader Writer
7. 输入字节流：**FileInputStream, BufferedInputStream**, ~~ObjectInputStream, DataInputStream~~
8. 输出字节**FileOutputStream,BufferedOutputStream**~~ObjectOutputStream,DataOutputStream~~
9. 输入字符流：**FileReader, BufferedReader**
10. 输出字符流：**FileWriter, BufferedWriter**

## 多线程

1. 程序、进程、多线程

程序是指指令的集合，代码的集合；而进程是动态的概念，当程序执行时，系统分配进程；多线程是在同一进程下，充分利用资源，多条执行路径，共享资源。



1. 创建线程
2. **继承Thread类**

创建: 继承 Thread +重写 run

启动: 创建子类对象 对象.start()

注意：创建 Thread 子类的一个实例并重写 run 方法，run 方法会在调用 start()方法之后被执行

1. **实现Runnable接口**：面向接口编程，避免单继承局限

创建: 实现 Runnable +重写 run

启动:静态代理 Thread

a)、创建真实角色 实现类对象

b)、创建代理角色 Thread 对象+真实角色的引用

c)、代理角色.start()

缺点: 根据重写规则，run 方法不能对外声明异常，不能有返回值

1. **线程的5种状态**

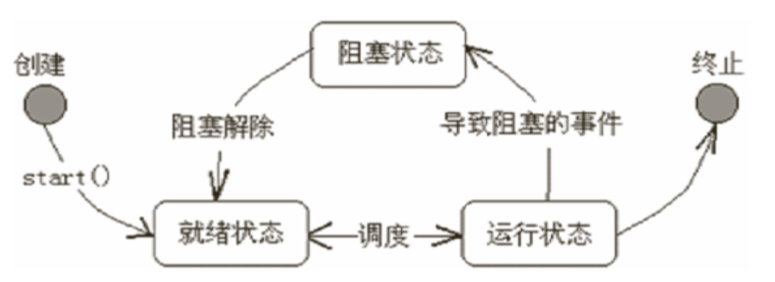
1)、新生状态: new

2)、就绪状态: runnable

3)、运行状态: running

4)、阻塞状态: blocked

5)、执行完毕: dead



1. 阻塞：sleep
2. 终止：不要调用 stop destory 方法

1、正常执行完毕，循环 次数已经到达

2、外部干涉

1)、线程中加入标识 -->属性

2)、线程体中 使用改标识 -->死循环

3)、对外提供改变改标识的方法 setXxx() terminate() a()...

4)、外部根据适当的时机调用该方法

1. 当前线程：Thread.currentThread.getName()
2. 优先级：只代表概率，不代表绝对先后顺序

MIN\_PRIORITY ： 1

NORM\_PRIORITY :5 默认优先级

MAX\_PRIORITY ：10

getPriority() setPriority()

1. 同步：synchronized
2. 同步块

synchronized +块:同步块

synchronized (引用类型|对象|类.class) { }

1. 同步方法

修饰符 synchronized 返回类型|void 方法签名{ }

1. 死锁
2. 几个方法

Sleep:静态方法 模拟网络延时,每个对象都一个排它锁，不会释放锁

join：合并线程

~~stop~~: 停止线程（不用）

1、线程类中加入标识

2、对外提供改变标识的方法

3、线程类中循环+循环条件使用标识

4、外部使用线程时，根据时机调用改变标识的方法

yield:相互礼让，防止一个线程过多占用cpu时间

1. 线程通信: : 生产者消费者模式（ 了解 ）

线程通信的目标是使线程间能够互相发送信号。另一方面，线程通信使线程能够等待其他线程的信号。

Java 有一个内建的等待机制来允许线程在等待信号的时候变为非运行状态。java.lang.Object 类定义了三个方法，wait()、notify()和 notifyAll()来实现这个等待机制。一个线程一旦调用了任意对象的 wait()方法，就会变为非运行状态，直到另一个线程调用了同一个对象的 notify()方法。为了调用 wait()或者 notify()，线程必须先获得那个对象的锁。也就是说，线程必须在同步块里调用wait()或者notify()。