# Java基础知识总结

## 1.1.1 文档注释

文档注释是一种功能级注释，用来说明一个类、一个方法或一个常量的作用。它可以根据javadoc命令生成说明文档，类似于Java JDK中的文档。

package api;

/\*\*

\* 文档注释是功能级注释，用来说明一个类，一个方法或一个常量的，因此只在上述三个地方使用。

\* 文档注释可以使用java自带的命令javadoc来对这个类生成手册。

\*

\* 在类上使用时用来说明当前类的整体功能。

\* @author FAN

\*/

public class dd.Demo {

/\*\*

\* sayHello中使用的问候语

\*/

public static final String INFO = "Hello!";

/\*\*

\* 为给定的用户添加一个问候语

\* @param name 指定的用户的名字

\* @return 返回了含有问候语的字符串

\*/

public String sayHello(String name){

return "Hello!"+ name;

}

}

## 1.1.2 Object类

Object类是类层次结构的根，每个类都有Object作为超类。它包含了一些常用的方法，如equals()、hashCode()、wait()、notify()、notifyAll()、toString()等。

## 1.1.3 String类

String表示一个字符串，直接继承于java.lang.Object类。它内部维护了一个可变的char数组，通常使用字面量方式创建对象。

## 1.1.4 StringBuilder和StringBuffer

String的修改性能很差，因此产生了StringBuilder和StringBuffer类来解决这个问题。它们提供了一些方法用于字符串的修改，如append()、replace()、insert()、reverse()、delete()等。

## 1.1.5 正则表达式

正则表达式是用来进行逻辑判断的工具。它具有一定的语法规则，包括谓词和量词等。

## 1.1.6 包装类

包装类用于将基本类型转化为面向对象编程的过程中。Java提供了一些包装类，如Byte、Short、Integer、Long、Float、Double、Character、Boolean等。它们提供了一些常用方法用于数据转换和操作。

## 1.1.7 集合API

**java.util.Collection<>**为所有集合类的顶级接口，提供了对集合进行操作的方法。它的子类包括**List<>**、**Set<>**、**Map<>**等。

* **Collections类**：提供了对集合的操作方法，如排序、查找等。它是一个包含静态方法的工具类，用于操作集合。
* **Collection接口**：在集合层次结构中是根接口，代表一组对象。它定义了一些通用的操作方法，如添加、删除、包含等。

常用方法：

* add
* remove
* addAll
* containsAll
* removeAll

List<String> list = new ArrayList<>();

Iterator<String> it = list.iterator();

while (it.hasNext()) {

String str = it.next();

System.out.println(str);

}

集合的遍历可以使用迭代器（iterator）遍历，或者使用foreach循环遍历。需要注意的是，迭代器和foreach循环无法通过集合的增删操作改变集合元素。

**List<>**、**Set<>**、**Map<>**等是常见的集合类型，分别表示列表、集合和映射。它们都有不同的实现类，如**LinkedList<>**、**ArrayList<>**、**HashSet<>**等，根据具体的需求选择合适的实现类。

集合和数组可以相互转换，使用**Arrays.asList()**方法可以将数组转换为集合，使用**集合.toArray()**方法可以将集合转换为数组。需要注意的是，转换后的集合和数组共享同一片内存空间。

集合的排序可以使用**Collections.sort()**方法，默认从小到大排序。也可以传入自定义的比较器（Comparator）来排序，或者实现Comparable接口并重写compareTo()方法来实现排序。

## 1.1.8 泛型

泛型是Java语言的一个重要特性，可以在使用一个类时指定其属性、方法参数或返回值的类型。它使程序变得更加灵活和丰富。

JDK5推出了泛型功能，使得程序员可以创建泛型类、泛型接口和泛型方法。泛型类和泛型接口在使用时可以指定具体的类型，如果不指定则默认为Object类型。

## 1.1.9 Lambda表达式

Lambda表达式是Java 8引入的新特性，可以简化匿名内部类的创建。它提供了一种更加简洁优雅的语法来实现函数式编程。

Lambda表达式的语法形式为：（）->{}.

当需要实现一个接口中的抽象方法时，如果接口只有一个抽象方法，则可以使用Lambda表达式来代替匿名内部类，使代码更加简洁。

在Java中，可以使用Lambda表达式的接口上都有一个注解**@FunctionalInterface**。

## 1.2.1 File类

**java.io.File**类是用于操作文件和目录的类，可以获取文件的属性信息、创建和删除文件、访问目录中的子项等操作。

使用File类可以做到：

* 访问文件或目录的属性信息，如名称、大小、修改时间等。
* 创建和删除文件或目录。
* 访问一个目录中的子项。

需要注意的是，File类不能访问文件的数据内容，只能获取文件的属性信息。

常用方法：

* **new File(String url)**
* **getName()**
* **length()**
* **canRead()**
* **canWrite()**
* **exists()**
* **createNewFile()**
* **delete()**
* **mkdir()**
* **mkdirs()**
* **list[]**
* **File[] listFiles(FileFilter)**

## 1.2.2 java.io 输入输出流

流类似于河流，流过去就不会再回来了，就好像人生不可回头，只能朝前走。流可以按功能分为低级流和高级流。低级流负责直接对数据进行操作，类似于原材料的流动，而高级流负责对数据进行加工处理，类似于工厂中的加工机器。高级流通常是建立在低级流之上的，实现数据加工的过程。

另一方面，流的读取方式可分为字节流和字符流。字符流一般被认为是高级流，因为数据最终存储为二进制字节数组。这种存储方式是由于计算机硬件设计的特性所决定的，计算机处理数据的基本单元是二进制的，这是由电子设备的工作原理决定的。CPU、内存和硬盘等设备都是以二进制数据为基础进行操作和存储的。硬盘利用磁盘来模拟二进制数据的存储，而内存则作为临时存储区域，用于加快数据的访问速度。为了提高数据的读取速度，计算机采用了缓存机制，即在内存中设置一个缓冲队列，用于存储需要频繁访问的数据。同时，多进程和多线程的技术被用于充分调度CPU的资源，以实现对数据的快速处理和操作。这些技术的应用使得现代电脑和手机能够更加高效地运行和处理数据。

### 字节流

* **java.io.InputStream** 和 **java.io.OutputStream**：字节流的顶级接口类，用于读取和写入字节流。常用方法包括**read()**、**read(byte[] bytes)**、**close()**等。
* **java.io.FileInputStream** 和 **java.io.FileOutputStream**：用于读取和写入文件的字节流。无新增方法。
* **java.io.BufferedInputStream** 和 **java.io.BufferedOutputStream**：提供了缓冲功能，内部默认采用1024长的字节数组进行数据处理，对数据进行缓冲读写操作。常用方法包括**flush()**、**write(byte[] bytes)**等。

### 对象流

* **java.io.ObjectInputStream** 和 **java.io.ObjectOutputStream**：用于读取和写入对象的流。提供了专门的方法**readObject()**和**writeObject()**用于读写对象。

### 字符流

* **超类**：**java.io.Reader** 和 **java.io.Writer** 是字符流的抽象类。
* **转换流**：**InputStreamReader** 和 **OutputStreamWriter**，用于将字节流转换为字符流的枢纽类。
* **缓冲字符流**：**BufferedReader** 和 **BufferedWriter**，提供了缓冲功能，用于提高字符读写的效率。
* **打印流**：**PrintWriter**，提供了打印功能，方便输出文本信息。

## 1.2.3 异常

**Throwable** 是所有异常的父类，分为 **Error** 和 **Exception**。**Error** 是系统级错误，程序员无需处理，而 **Exception** 分为受检查的异常和不受检查的异常。受检查的异常在编译时会被检测到，必须处理；不受检查的异常代表运行时异常，程序员可以通过改进代码来避免。异常对象可以自定义或使用 Java 提供的异常类型来满足业务需求。常见异常包括 **NullPointerException**、**IOException**、**IndexOutOfBoundsException** 等。异常处理方式包括 **try-catch**、**try-finally**、**try-catch-finally** 等。

检查异常在编译阶段会被检测到，必须处理，可以捕获或继续抛出；非检查异常在编译阶段无法检测到，程序员对此类异常可以选择性处理，典型代表为 **RuntimeException** 或其子类。**finally** 常用于资源的释放，但若在 **try** 块中使用了 **System.exit(0)**，则 **finally** 块中的代码不会执行。另外，可以利用自动关闭特性，在 **try** 中直接创建对象，无需手动关闭。

**throws** 用于直接抛出异常不处理，**throw** 用于抛出异常，由对应的异常对象捕获。

## 1.2.4 Socket

**Socket** 在 **java.net** 包下，用于通信的套接字，直接继承自 **Object** 类。常用的类包括 **Socket** 和 **ServerSocket**，分别用于客户端和服务端。**ServerSocket** 绑定端口，调用 **accept()** 方法接收 **Socket** 的连接请求；**Socket** 调用 **connect()** 方法传入 IP 地址和端口号连接 **ServerSocket**。通信过程中使用 IO 流进行数据的读写，**Socket.getInputStream()** 和 **Socket.getOutputStream()** 用于输入和输出。

socket.connect(new InetSocketAddress("127.0.0.1", 9999)); // 异常深刻的编程体会！！！

## 1.2.5 Thread

**Thread** 是进程下对时间的进一步分配，线程之间的关系是平等的。Java 的线程采用抢占机制，优先级高的线程先执行。线程常用方法包括 **setPriority()**、**setName()**、**getName()**、**getPriority()**、**sleep(1000)**（休眠1秒）、**wait()**、**notify()**、**notifyAll()**、**join()**、**interrupt()** 等。

多线程共同运行可能产生线程交互和安全问题，可以通过 **synchronized** 方法解决线程安全问题。在多线程同步时，需要选择合适的锁对象。常用的锁对象包括 **类.class**、**this**、共享资源等。当一个线程获得特定锁时，其他线程会进入阻塞状态，只有等待该线程释放锁后才能进入就绪状态。

1.1.1文档注释

是一种功能级注释，可根据javadoc命令生成说明文档(类似Java的jdk说明文档)，用来说明一个类，一个方法或者常量，方便提高代码的可读性。

package api;

/\*\*

\* 文档注释是功能级注释，用来说明一个类，一个方法或一个常量的，因此只在上述三个地方使用。

\* 文档注释可以使用java自带的命令javadoc来对这个类生成手册。

\*

\* 在类上使用时用来说明当前类的整体功能。

\* @author FAN

\*/

public class dd.Demo {

/\*\*

\* sayHello中使用的问候语

\*/

public static final String INFO = "Hello!";

/\*\*

\* 为给定的用户添加一个问候语

\* @param name 指定的用户的名字

\* @return 返回了含有问候语的字符串

\*/

public String sayHello(String name){

return "Hello!"+ name;

}

}

1.1.2Object类

Object类是类层次结构的根。每个类都有 Object作为超类。所有对象，包括数组，实现这个类的方法，其中包含了equals(),hashCode(),wait(),notify(),notifyAll(),toString()方法。

1.1.3Srting类

String表示一个字符串，直接继承于java.lang.Object类,内部维护了一个可变的长的char数组，自身被final过了，通常赋值方式为字面量创建，二是new String(),第一种方式声明的方式会现在堆内存中的字符串常量池中寻找是否存在该对象，如果存在就会返回该对象的地址给变量，不存在就会在字符串常量池中创建该对象，以此达到String类对象的重用的目的，直接new String()的String对象不会出现在字符串常量池中，因此推荐字面量赋值创建对象。对于引用类型==比较的是地址，equals()采用的是默认比较方法，系统的类采用的都是已重写好的equals()方法，比较的内容，而自建对象需要重写equals()才能实现想要的效果，否则默认比较地址。 提供的方法：length(),indexOf(char),charAt(int idx),equals(Obj),compareTo(Obj),maches(),subString(),endsWith(),startsWith(),valueOf()等其他方法。

1.1.4StringBulider和StringBuffer

因为String的修改性能很差，于是就产生了StringBuffer和StringBuilder类来解决这个问题，特点：修改效率高。两者差别，Buffer线程不安全，Builder线程安全。

提供的方法：append(),replace(),insert(),reverse(),delete()等方法。

1.1.5正则表达式

用来进行逻辑判断的好东西。

语法规则：

谓词 [abc]-匹配abc中任意字符,(abc)匹配abc字符，某字符直接出现则直接匹配它，.任意字符。\w 匹配0-9\_a-zA-Z任意字符

量词，?0-1 \*0-多,+1-多 {a,}a次以上,{a,b}a以上b以下。

1.1.6包装类

产生：想让基本类型参与到面向对象编程的过程。

byte Byte char Character short Short int Integer long Long float Float double Double

boolean Boolean。同一方法：包装类名.valueof(a),将a类型转化为该包装类。.MAX\_VALUE,.MIN\_VALUE,获取其最大值和最小值，包装实例.类型名Value(),将该包装类的值以某种类型返回。Integer采用类似String 的常量池，在一定范围(默认为-128 - 127)重用对象,可进行手动设置(根据业务逻辑设置)。

其中数字类型的包装类都继承自java.lang.Number,而char和boolean的包装类直接继承自Object

Number是一个抽象类,定义了一些方法,目的是让包装类可以将其表示的基本类型转换为其他数字类型.

自动拆装箱特性：

该特性是编译器认可的.当编译器编译源代码时发现有基本类型和引用类型相互赋值使用时会自动补充代码来完成他们的转换工作,这个过程称为自动拆装箱。JDK5推出了自动拆装箱功能，当基本类型采用赋值方式转化为对应包装类,会触发装箱，当需要计算表达式时就会拆箱为基本数据类型参与运算。

1.1.7集合API

java.util.Collection<>为所有集合类的顶级接口，Collection下面有很多实现类。

Collections.sort(),Collections直接继承自java.lang.Object类。这个类只包含操作或返回集合的静态方法。它包含多态算法操作的集合，“包装”，它返回一个按指定的集合支持的新的集合，和其他一些零碎的东西。

Collection类 在集层次结构根接口。一个集合代表一组对象，被称为元素。一些集合允许重复的元素，而其他的则不允许。有些是有序和无序的人。JDK没有提供这个接口的任何直接的实现：它提供更具体的子接口像 Set和 List实现。此接口通常用于通过收集和操纵他们最大的共性是理想。常用方法: add,remove,addAll,containsAll,removeAll。集合的遍历采用是iterator(迭代器)遍历 和.foreach()实现accept接口来遍历元素，迭代器和foreach不能通过集合的增删方式操作元素。

JDK5 推出增加版for(数据类型 i : 集合名或数组)，若是集合编译器会将其迭代器模式，数组就会平常的遍历，JVM并不认。

while(it.hasNext()){

String str = (String)it.next();

System.out.println(str);

}

List<> linkedList<>,ArrayList<>,区别 单链表实现和类数组实现，可重复集合。

Set<> 不可重复集合，HashSet<>哈希存储

Map<> HashMap<> TreeMap<>，键值对，哈希存储方式和树存储方式。

集合和数组的转换：

List<数组> Arrays.asList(数组名), Object[] 集合名.toArray(),其中数组转化为集合后，对集合的操作会影响数组，对数组的操作也会影响集合(原因：数组存储元素地址，集合和数组指向同一片地址空间)。而将集合转化为数组后，两者互不影响(原因：)。

集合的排序：

Collections.sort(),默认采用从小到大排序，Collections.sort(comparator cp) 传入比较器来排序，还有一种方法是重写Comparable 接口，为了保证低侵入性，就应该使用创建比较器的比较。

1.1.8泛型

Jdk5推出泛型的功能，为了解决想要表达一个它既是这也是那的概念，给出一个抽象的概念，使得程序变得更加丰富。<?> 中间放得是代表，直接将它将当成数据类型来用就行，根据传入得参数来决定它到底是什么类型。泛型也称为参数化类型,允许我们在使用一个类时指定它当中属性,方法参数或返回值的类型.

泛型在集合中被广泛使用,用来指定集合中的元素类型.有泛型支持的类在使用时若不指定泛型的具体类型则默认为原型Object。

1.1.9Lambda

Lambda表达式-JDK8之后推出的新特性

语法:

(参数列表)->{

方法体

}

当使用匿名内部类创建时，如果实现的接口只有一个抽象方法，则可以使用lambda表达

式代替，使代码更简洁优雅。

在java中可以使用lambda表达式代替匿名内部类创建所需要实现的接口时，该接口上都有一个注解:@FunctionalInterface。

1.2.1File类

文件类，负责获取文件的属性信息，增删文件，不能获取文件存储的信息(得利用java.io)。File类的每一个实例可以表示硬盘(文件系统)中的一个文件或目录(实际上表示的是一个抽象路径)

使用File可以做到:

1:访问其表示的文件或目录的属性信息,例如:名字,大小,修改时间等等

2:创建和删除文件或目录

3:访问一个目录中的子项

但是File不能访问文件数据.

方法：

NewFile(Stringurl),getName(),length(),canRead(),canWriter(),existes(),createNewFile(),delete,mkdir,mkdirs,list[],File[] listFiles(FileFilter)。

1.2.2java.io输入输出流

流类似于河流，流过去就不会再回来了，就好像人生不可回头，只能朝前走。"流可以按功能分为低级流和高级流。低级流负责直接对数据进行操作，类似于原材料的流动，而高级流负责对数据进行加工处理，类似于工厂中的加工机器。高级流通常是建立在低级流之上的，实现数据加工的过程。

另一方面，流的读取方式可分为字节流和字符流。字符流一般被认为是高级流，因为数据最终存储为二进制字节数组。这种存储方式是由于计算机硬件设计的特性所决定的，计算机处理数据的基本单元是二进制的，这是由电子设备的工作原理决定的。CPU、内存和硬盘等设备都是以二进制数据为基础进行操作和存储的。硬盘利用磁盘来模拟二进制数据的存储，而内存则作为临时存储区域，用于加快数据的访问速度。为了提高数据的读取速度，计算机采用了缓存机制，即在内存中设置一个缓冲队列，用于存储需要频繁访问的数据。同时，多进程和多线程的技术被用于充分调度CPU的资源，以实现对数据的快速处理和操作。这些技术的应用使得现代电脑和手机能够更加高效地运行和处理数据。"

字节流

java.io.InputStream java.io.OutputStream

字节流的顶级接口类(继承java.lang.Object类)，int read() 返回低八位值-1到255(原因,以字节存储方式)，int read(byte[] bytes)读取一段字节数组，返回读取个数，可采用循环一直读，再根据设置或信息主动或被动输入或输出结束，-1都表示读取结束。 close()关闭。OutputStream有flush(),write(),write(byte[] bytes)字节数组。

java.io.FileInputStream java.io.FileOutputStream 输入输出文件流。无新增方法，

java.io.BufferedInputStream java.io.BufferedOutputStream

内部默认采用1024长的byte[]，将数据进行处理，读数据就直接将1024个数据全部读过来，等到程序从该文件流中读取过了1024长的字节，就会再读一次，以此类推。对于输出，则会等到数据装填满才会提交过去。因为产生了数据丢失或数据同步更新问题，采用flush(),可直接将数据输出过去，建议所有流都加上StandartCharset.UTF-8的限制，保证读写的编码和解码的正确性，调用所有的关闭方法都会自动flush(),flush()具有传递作用。

对象流

java.io.ObjectInputStream java.io.ObjectOutputStream

专门用来读取对象的流，对象流，readUTF()，以UTF-8形式读取字符串。writeUTF()，以UTF-8形式写出字符串。实现InputStream,Objectinput接口，写入和写出的对象必须进行序列化和反序列化的操作,readObject(),writeObject()。

字符流

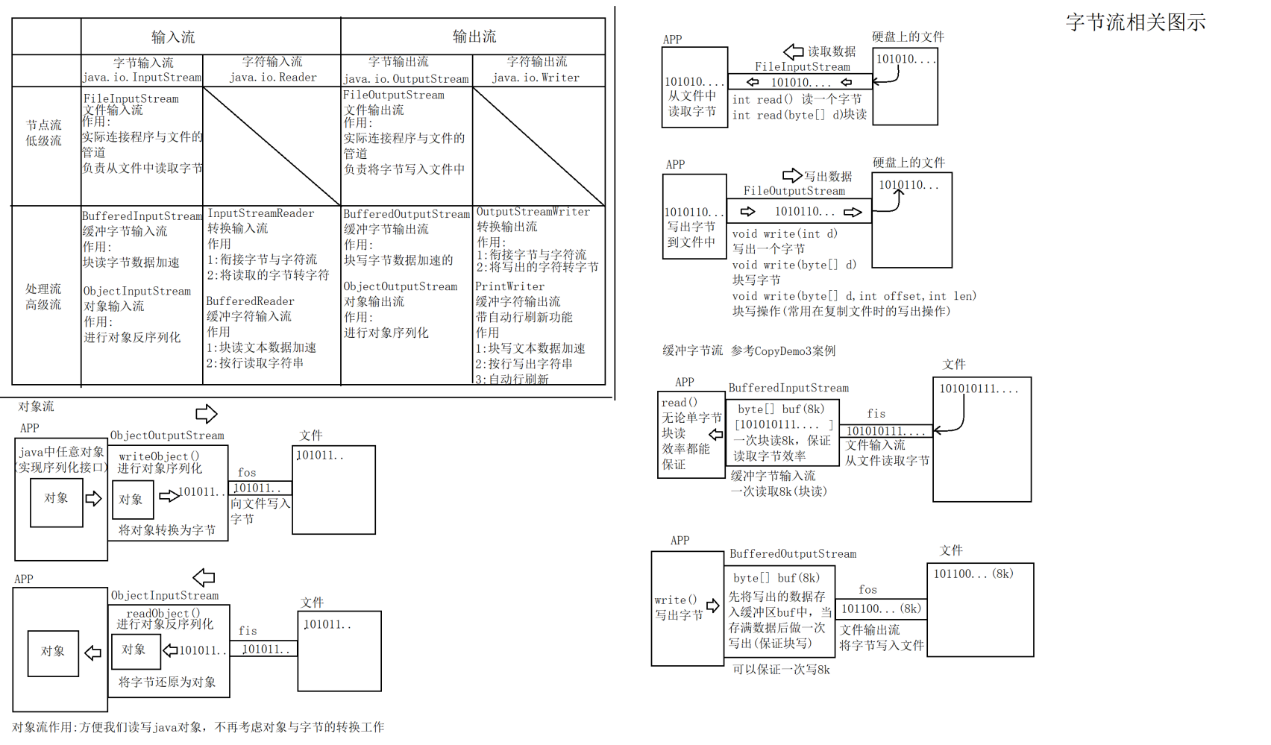
超类 java.io.Reader,java.io.Writer 抽象类

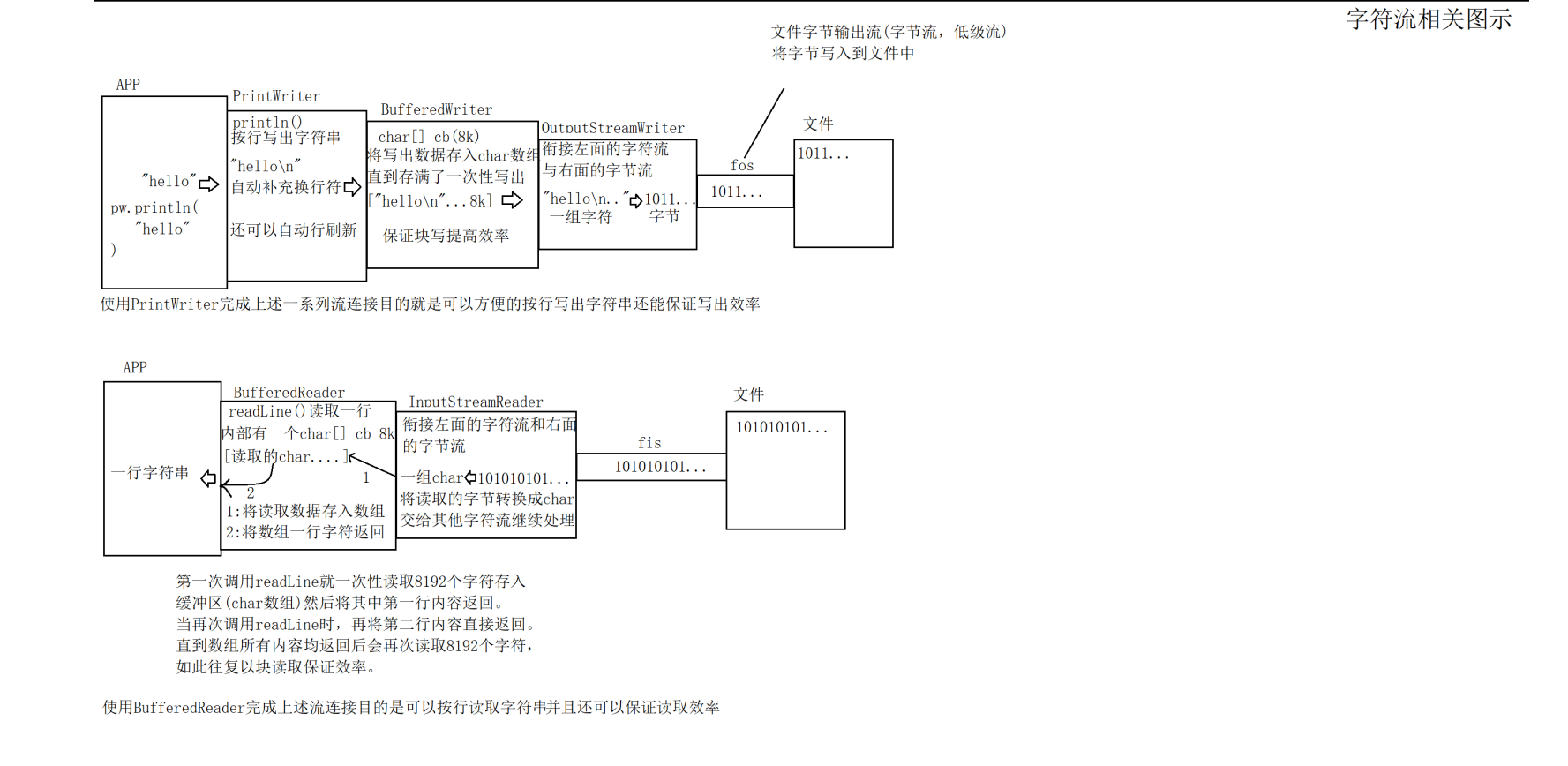
转换流 InputStreamReader, OutputStreamWriter,

由字节流转换为字符流的枢纽类，该读写类就是一个转换后的字符流的类。

BufferedReader,BufferedWriter

PrintWriter. 类上，不再赘述。





1.2.3异常

Throwable是所有异常的父类，分为Error 和Exception, error为系统级错误，程序猿无需处理，Exception有受检查的异常和不受检查的异常(针对编译器)，受检查的异常是必须处理的，不受检查的异常代表有运行时异常，程序猿可以通过改进代码避免发生运行时异常。异常对象可以自建，也可以使用java提供的日常去满足业务的需求当系统提供的异常类型，不足以满足我们对异常的需求时，我们可以选择自定义异常，尤其是 和业务相关性比较的异常。 例如PhoneNotFoundException; 我们自定义异常一般都是继承RuntimeException或Exception类型，然后重写相关方法。，因情况而定，常见异常NullPointerException,IOException, IndexOutOfBoundsException等异常。try catch try finally try catch finally，让对应的异常对象捕获，来处理异常。

检查异常：编译阶段，编译器能够检测到异常，这种异常必须处理，处理的方式，要么捕获，要么继续抛出。

非检查异常：编译阶段无法检测到的异常，程序员对此类异常可以有选择性的处理。 典型代表为RuntimeException或它的子类。

细节，在主程序中中try执行完后return后还会执行finally里面的代码，finally常用于资源的释放,使用System.exit(0)就不会执行finally()中的方法。自关闭特性，我们创建的对象需要关闭时，一种方法是写在finally中，还有一种方式在try()接口中直接创建， 这样对象无需手动再关闭。前提时try内部的对象需要实现 AutoCloseable 接口。

throws 直接抛出异常不处理，throw 抛出异常，让对应的异常对象捕获。

1.2.4Socket

socket 在java.net包下，负责通信的套接字，直接继承Object类，常用的 的类为：Socket,SeverSocket,一个为服务端，一个为客户端，SeverSocket绑定端口，调用accept()方法接收Socket的连接请求，Socket调用connect()方法传入IP地址和端口号连接ServerSocket。其中采用IO流进行数据的读写，Socket.getInputStream,Socket.getOutputStream()来进行输入和输出。

socket.connect(new InetSocketAddress("127.0.0.1",9999));(异常深刻的编程体会！！！)

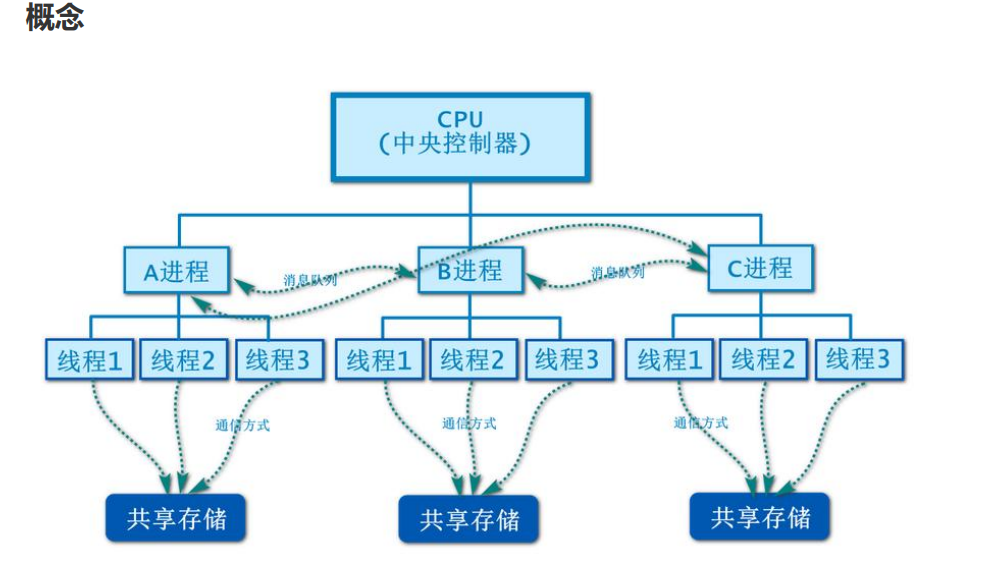
1.2.5Thread

Thread 是进程下对时间的进一步分配,线程之间的关系是平等的,java的线程是采用抢占机制的，谁抢到谁就执行，新创建出来的线程的优先级默认是5,可以setPriority,setName(),getName(),getPriority(),sleep(1000)(休眠1秒),wait(),notify(),notifyAll(),join(),interrupt(),线程可以用来同时做不同的事情，在近乎同一时间进行多件事情,实现一些好玩的事情~。多线程共同运行产生了线程的交互和安全问题，通过synchronized的方法解决线程产生的安全问题，多线程同步需要一把锁对象。

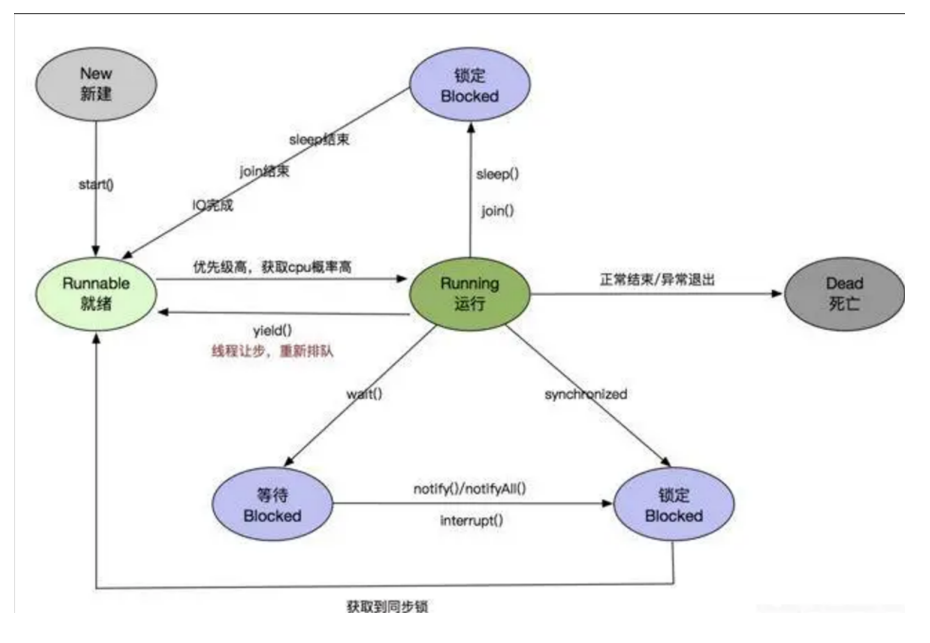
锁对象的选择：

1. 类.class –类的类(2进制类的形式)—原因—只被JVM加载一次，所有类都可见。
2. this 本来
3. 共享资源(多线程)。

当一个线程获得特定锁时，其他线程进入堵塞状态，只有等待该线程释放锁，且拿到锁才可以进入就绪状态。



线程的生命周期



1.2.6Reflect

reflect 是对JVM加载的.class再一次采用面向对象的思想，对2进制的多个类又抽象化，变成类的类,对Field,method,constructor抽象也得到对应得对应的类，学习这个反射常用于框架及设计创建类的实现，首先类名.class 获得类的类,再调用newInstance方法，然后选择对应的构造器,就可以产生一个对象，对于其他用法可以查资料。框架设计：获取配置文件的键的值，然后通过获取该值创建对应的现象，不同的值代表的对象在编程中都实现过同一个接口或者继承过某同一个类，实现只需要修改配置文件的信息，就可以使用不同的效果，但又不需要对原来的代码进行修改(这非常好)。

使用反射原因：由于无法直接调用某个类的方法或者属性，亦或者构造器，可以获取并调用，对于配置文件读取的信息想要产生对象，而字符串的类型无法直接创建，只能使用Class.forName(“完整的包名和路径”)得到该类的类，然后去newInstance(),实例对象.getClass();