# Java.lang.Math

1. 绝对值函数：abs，对各种数据类型求绝对值
2. 三角函数和反三角函数：cos sin tan acos asin atan atan2
3. 开立方cbrt ，开平方sqrt，四舍五入round，

# Java类型兼容

## 基本类型

基本强转如果目标类型无法容纳强转类型则无法强转（会丢失精度），如int转short，double转int，但反过来short可以安全的转为int。

## 引用类型

向上转型：子类对象装换为父类（或父类的父类）类型，称为向上转型。

向下转型：将父类转换为导出类的过程，称为向下转型。

# 数组的length是方法还是属性

<http://blog.csdn.net/i10mg/article/details/8741195>

**数组的length既不是方法也不是属性**，数组的length()方法被java编译器编译为arraylength指令（java对数组的操作是指令级的）。

# Set元素怎么做到不重复的

Java集合中判断两对象是否相同的原则为：

**先判断两对象的hashCode**，如果不等，则两对象不相等，判断结束，**如果想等，继续调用equals方法**，如果equals返回true，则两对象想对，否则不相等。

HashSet元素存入顺序由其hashcode决定。

# NIO和IO异同

NIO：<http://www.importnew.com/19816.html>

<http://wiki.jikexueyuan.com/project/java-nio-zh/java-nio-vs-io.html>

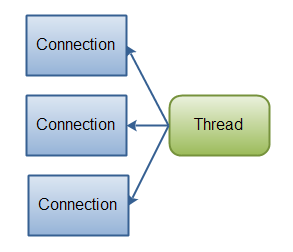
IO是**面向流**的，NIO是**面向缓冲区**的。

IO会**阻塞**当前线程，NIO**非阻塞**。

IO的Stream（InputStream或OutPutStream）是单向的，而Channel（FileChannel，SocketChannel）是双向的。

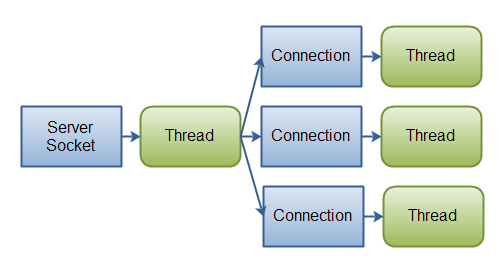
NIO允许我们只用一条线程来管理多个通道（网络连接或文件），随之而来的代价是解析数据相对于阻塞流来说可能会变得更加的复杂。

如果你需要同时管理成千上万的链接，这些链接只发送少量数据，例如聊天服务器，用NIO来实现这个服务器是有优势的。类似的，如果你需要维持大量的链接，例如P2P网络，用单线程来管理这些 链接也是有优势的。这种单线程多连接的设计可以用下图描述：



**Java NIO: A single thread managing multiple connections**

如果链接数不是很多，但是每个链接的占用较大带宽，每次都要发送大量数据，那么使用传统的IO设计服务器可能是最好的选择。



**Java IO: A classic IO server design - one connection handled by one thread.**

# Statement和PreparedStatement异同

<http://www.cnblogs.com/raymond19840709/archive/2008/10/13/1309657.html>

1. PreparedStatement是**预编译**的,对于**批量处理可以大大提高效率**. 也叫JDBC存储过程，（当你需要执行Statement对象多次的时候，PreparedStatement对象将会大大降低运行时间，当然也加快了访问数据库的速度。）
2. 使用 Statement 对象。在对数据库只**执行一次性存取**的时侯，用 Statement 对象进

处理。

1. PreparedStatement 对象的**开销**比Statement**大**，对于一次性操作并不会带来额外的好处。
2. statement每次执行sql语句，相关数据库都要执行sql语句的编译，preparedstatement是预编译得, preparedstatement支持批处理
3. 安全性：传递给PreparedStatement对象的参数可以被强制进行类型转换，使开发人员可以确保在插入或查询数据时与底层的数据库格式匹配，（在公共Web站点上,在没有首先通过PreparedStatement对象处理的情况下，所有的用户输入都不应该传递给SQL语句。）

选择PreparedStatement对象与否，在于相同句法的SQL语句是否执行了多次，而且两次之间的差别仅仅是变量的不同。如果仅仅执行了一次的话，它应该和普通的对象毫无差异，体现不出它预编译的优越性。

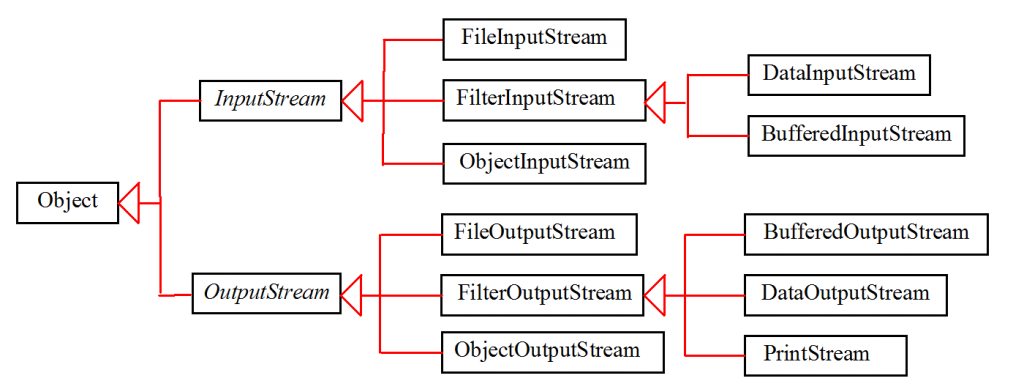
# 分析一个IO的架构

<http://www.cnblogs.com/lilixin/p/5721694.html>

java.io包中类对应两类流：

1. 结点流：直接从指定位置文件或内存区域读取；
2. 过滤流：这类流往往以其他输入流作为它的输入源

字节流：

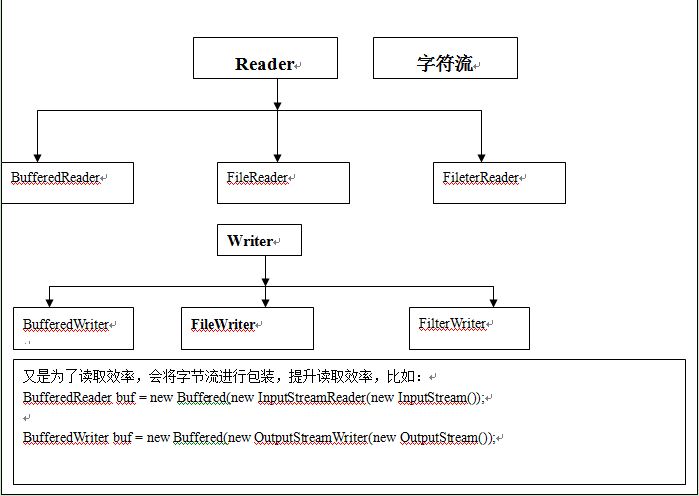


FileInputStream和FileOutputStream属于结点流，两者源端都是磁盘文件；

BufferedInputStream和BufferedOutputStream属于过滤器流；

DataInputStream和DataOutputStream属于过滤器流，适合于网络上的数据传输；

字符流：



字符流主要是用来处理字符的。Java采用16位的Unicode来表示字符串和字符，对应的字符流按输入和输出分别称为readers和writers。

InputStreamReader和OutputStreamWriter

在构造这两个类对应的流时，它们会自动进行转换，将平台缺省的编码集编码的字节转换为Unicode字符。对英语环境，其缺省的编码集一般为ISO8859-1。

BufferedReader和BufferedWriter

这两个类对应的流使用了**缓冲**，能大大提高输入输出的效率。这两个也是**过滤器流**，常用来对InputStreamReader和OutputStreamWriter进行处理

# 为什么要序列化

<http://blog.csdn.net/dreamtdp/article/details/15378329>

序列化就是将内存中的类或者对象（你写的类都是存储在内存中的）变成可以存储到存储媒介中的流，你将类序列化成流之后可以通过互联网传输给别人，你也可以反序列化将别人的序列化流转换成内存中的对象。

序列化时只保存对象的状态（成员变量）；

当一个父类实现序列化，子类自动实现序列化，不需要显式实现Serializable接口；

当一个对象的实例变量引用其他对象，序列化该对象时也把引用对象进行序列化；

并非所有的对象都可以序列化（都适合被设计为可序列化的）。

# 产生对象的方法有哪些

# ConcurrentMap和HashMap区别