本来是想总结一下改bug的一些技巧的，但才疏学浅，未敢妄言过多。但这些天来接触的java常见异常与错误，还是该好好总结一下。

1、java.lang.nullpointerexception.空指针异常，解释就是程序遇上了空指针，简单来说就是调用了未经初始化的现象或者不存在的对象。我犯的几次错误都在于只定义了而没有用new进行初始化。

2、java.lang.arrayindexoutofboundsexception.数组越界错误。这是自己在调用字符串与数组之间相互转换时犯过的错误，常常会发生越界错误。其中还有length和size（）的区别，length一般是数组的长度属性，而size（）一般是用于求集合内元素的个数。

3、java.lang.illegalargumentexception.传入参数异常，一般来说就是传入参数非法，其实这种在使用泛型的时候算是很常见的，有些编译的时候并不会检查其类型，运行的时候才会。

4、java.lang.ClassFormatError.类格式错误。当Java虚拟机试图从一个文件中读取Java类，而检测到该文件的内容不符合类的有效格式时抛出。

5、java.lang.NoClassDefFoundError.未找到类定义错误。当java虚拟机或类加载器试图实例化某个类，而找不到类的定义时便会抛出此异常。

接下来自己該是总结一下一些常见的运行时错误了。

错误：指的是一个合理的应用程序不能截获的严重问题。大多数都是反常的情况。错误是VM的一个故障。错误很难处理，比如内存溢出。

关于linux的一些基本知识

先是关于一些文件操作的基本知识。

递归复制，windows下默认的就是递归复制，假如不是递归复制，则复制的文件还有子目录的话，其内容将不会被复制

tar相当于windows下的安装包，rpm相当于exe可执行文件

rpm -ql 应用程序名，查看文件目录

rpm -qa | grep 应用程序，查看是否有应用程序

touch，创建新文件

mkdir，创建新目录

这两个命令都可以带 -f 参数，force，强制的意思，就算原来有相同名字的文件或者目录，也是直接覆盖。

chmod，数字，目录或者文件，修改用户权限

通过修改配置文件来修改环境变量时，要使用source命令

pwd，当前目录

利用cd-,可以在最近使用的两个目录之中进行快速切换

还有在linux服务器下的一个问题，就是其跟客户端的文件传输问题。

我们

Linux下的守护进程，相当于windows的开机自启动。

还有linux中的变量说明：

$$,shell本身的PID

$0,shell本身的文件名

$1~n,添加到shell的各参数值,$1是第一参数，$n是第n参数

$#,添加到shell的参数个数

并且在shell中，我们常用的比较符号现在是使用为文件重定向，所以我们需要一些新的方式来表示那些比较符号，比如

-eq 等于

-gt 大于

-ge 大于等于

-lt 小于

-le 小于等于

还有使用MYIBATIS框架时，其中的xml配置文件中，也是使用与shell中相同的比较符号。

Shell脚本中没有大括号，只能用if和fi判断其开始于结束

CUR\_DIR=$(cd "$(dirname "$0")"; pwd)，可以获取当前目录

从校园到工作的路（五）

Valueof，对象的类型转换

还有Integer的parseInt方法

其实Interger类的valueOf方法也是调用了parseInt方法，但parseInt方法返回的是int值，是基本类型，而valueOf方法返回的是Integer，是包装类。

还有一种类型转换就是多态的那种方式了。父类与子类。比如

Map<Integer, Integer> map = new HashMap<Integer,Integer>();

其中父类调用子类是可以的，因为最终实现的时候都是子类的实现，父类中有的子类都会有。但反之不行，因为子类有的，父类中不一定有。

也就是说父类可以转换成子类，但子类不能转换为父类。

对象的赋值是地址标示的传递，即两个对象共同使用同一段内存地址。

Instanceof 操作符可以判断一个类是否实现了某个接口，也可以判断一个实例对象是否属于某个类。

向上转型可以放心使用，但如果是向下转型，一定要用instanceof先判断，后使用。

这一部分还有很多自己没有完全理解的地方。

先挖个坑，以后来填。

———————————————————————————————

从校园到工作的路（六）——关于阅读源码的方法

作为一名程序猿，阅读别人的代码是我们日常工作中必须的一部分。而代码的水准是不定的。

那么我们如何阅读别人的源码，并从中受益呢？

下面是JAVA类库中的String类的部分代码

/\*\*

\* Allocates a new {@code String} that contains characters from a subarray

\* of the character array argument. The {@code offset} argument is the

\* index of the first character of the subarray and the {@code count}

\* argument specifies the length of the subarray. The contents of the

\* subarray are copied; subsequent modification of the character array does

\* not affect the newly created string.

\*

\* **@param** value

\* Array that is the source of characters

\*

\* **@param** offset

\* The initial offset

\*

\* **@param** count

\* The length

\*

\* **@throws** IndexOutOfBoundsException

\* If the {@code offset} and {@code count} arguments index

\* characters outside the bounds of the {@code value} array

\*/

**public** String(**char** value[], **int** offset, **int** count) {

**if** (offset < 0) {

**throw** **new** StringIndexOutOfBoundsException(offset);

}

**if** (count < 0) {

**throw** **new** StringIndexOutOfBoundsException(count);

}

// Note: offset or count might be near -1>>>1.

**if** (offset > value.length - count) {

**throw** **new** StringIndexOutOfBoundsException(offset + count);

}

**this**.value = Arrays.*copyOfRange*(value, offset, offset+count);

}

其实对绝大多数语言来说，通过阅读其源码与说明都是深入了解一门语言的必经之路。

而单就在上面这一个简单的String的构造函数中，我也受益良多：

1. 对参数的检查
2. 对异常的抛出（PS：刚入门的时候还以为只能在try...throw...catch这种模式下抛出异常和处理异常的）
3. 考虑到了一些极端的情况，比如为什么使用offset > value.length-count,而却没有使用offset + count > value.length,注释中有详细解释

// Note: offset or count might be near -1>>>1.（Σ( ° △ °|||)︴，一开始居然不记得>>>是逻辑右移，也就是无符号右移，汇编经验喂了狗了。。。）

其实这就是说offset+count，两个正数相加，可能会产生溢出，-1>>>1可以得到最大的正整数，这也是溢出的表现。

1. 还有风格与命名。风格都是简单明了，便于阅读，而在命名上，基本上都能达到望文生义。

上面这些都是优秀的源代码不可欠缺的一部分。

现在主流的一些优秀的源代码的观点如下：

1、安全性

2、可测试性和稳健性

3、可维护性

4、简单

5、可重用性

自己编写代码的时候要时常回顾一下自己是否做到了这些。而在阅读别人代码的时候，要看到别人是否做到了这些，别人是怎么做到的，如果没有做到，那换做自己，該是怎么改。

代码写出来应该是优雅的，假如自己的实现不优雅，肯定是因为自己没有找到最好的方法。

从校园到工作的路（七）

那些数组与字符串之间的相互转换。

最常见也最简单的莫过于char[]数组与String类型之间的转换。

String类的内部就是用char数组实现的

/\*\* The value is used for character storage. \*/

**private** **final** **char** value[];

/\*\*

\* Initializes a newly created {@code String} object so that it represents

\* an empty character sequence. Note that use of this constructor is

\* unnecessary since Strings are immutable.

\*/

**public** String() {

**this**.value = **new** **char**[0];

}

**public** String(**char** value[]) {

**this**.value = Arrays.*copyOf*(value, value.length);

}

由上面这些可以看出char[]数组天然就可以转成String类型，那么如果string类型转为char数组呢？我们可以通过String中的charAt()方法来实现，其源码如下

**public** **char** charAt(**int** index) {

**if** ((index < 0) || (index >= value.length)) {

**throw** **new** StringIndexOutOfBoundsException(index);

}

**return** value[index];

}

可以通过这个方法获取字符串中的每一个值，然后将这些值转换到字符数组中。

接下来还有常用的String与int[]的转换，或者其它基本类型之间的转换。

如果只是正常的String比如”1234”想要转换为整数就是int型的1234，java中有如下方法

**public** **static** **int** parseInt(String s) **throws** NumberFormatException {

**return** *parseInt*(s,10);

}

默认是使用十进制

Java中最大有36进制，因为我们正常表示不同进制的时候是位数上是显示0~9A~Z的。

若是Int型转换成String型，有以下三种方式

分享一段在int型转换为String型的过程中的函数

**final** **static** **int** [] ***sizeTable*** = { 9, 99, 999, 9999, 99999, 999999, 9999999,

99999999, 999999999, Integer.***MAX\_VALUE*** };

// Requires positive x

**static** **int** stringSize(**int** x) {

**for** (**int** i=0; ; i++)

**if** (x <= ***sizeTable***[i])

**return** i+1;

}

这个函数是获取整数变成字符串所需要分配的空间。

一般来说，int型转换为String型主要有以下几种方法

String s = String.valueOf(i)；

String s = Integer.toString();

String s = “”+i;

还可以用StringBuffer类来实现，其append方法如下

@Override

**public** **synchronized** StringBuffer append(**int** i) {

toStringCache = **null**;

**super**.append(i);

**return** **this**;

}

最后可以使用其toString方法

@Override

**public** **synchronized** String toString() {

**if** (toStringCache == **null**) {

toStringCache = Arrays.*copyOfRange*(value, 0, count);

}

**return** **new** String(toStringCache, **true**);

}

可以将Stringbuffer转换为String。

从校园到工作的路（八）——关于编程中的编码问题

编码，我们下意识便会想到UTF-8，GKB这些。

关于在SecureCRT中的编码，还有Eclipse中的编码

ASCII编码只支持基础拉丁字符。其设计：用一个字节（8个位）来表示一个字符，并保证最高位的取值永远为’0’.

ISO 8859-1对应西欧语言，Latin-1是ISO 8859-1的别名。它表示整个西欧的字符集范围。

BOM：Byte Order Mark

UTF-8 BOM又叫UTF-8的签名，对UTF-8没有作用，是为了支持UTF-16，UTF-32.BOM签名是为了告诉编辑器当前文件采用何种编码，方便编辑器识别。虽然BOM在编辑器中不显示，但会产生输出，就像多了一个空行。



全世界统一一种编码。为了能够独立表现世界上的所有字符，Unicode采用四个字节表示一个字符。

为什么会有UTF-8的产生，两个原因：

1. Unicode表示汉字或者拉丁字符，其前面两个或三个字节都是0，这是浪费的。
2. 计算机不知道四个字节表示的是Unicode字符还是4个ASCII字符

无论是UTF-8，还是Unicode，其与GBK都是不兼容的，两者之间不能通过任何算法加以转换。如需转换，一般通过GBK查表方式进行。

常见问题及解答：

Windows中的NotePad的编码ANSI保存选项代表什么？

ANSI是windows默认的编码方式。对于英文文件是ASCII编码，对于中文是GB2312编码。所以将一个UTF-8的文件存为ANSI格式，中文部分就会产生乱码。

从校园到工作的路（九）——那些Java中的排序

三种sort

从校园到工作的路（十）——java中的那些正则表达式用法

从校园到工作的路（十一）——jetty与Tomcat的一些比较

从校园到工作的路（十二）——java多线程