**Conllection 接口:**

**Set(HashSet,TreeSet)** 接口 =>无序，不重复的

**List(LinkedlList,ArrayList)** 接口,：

注意set(int index,value) 和add(index,value)不同，后者加入一个元素，将原来元素向后挤出一位。

**Map(HashMap,TreeMap)** 接口

含有一个返回iterator()方法。

Object put(object key,Object value) =>注意返回值有object，因为如果我们加入的key 已经存在于Map，则返回原来的value。

**Iterator 接口:**

HasNext(),next(),remove()

**Java.util.Collections 类:**

关于List容器的一些基本算法。注意与Collection接口不一样。

**Java.lang.Comparable接口：**

只有public int compareTo(Object obj)方法;

0=>this==obj,

正数=> this>obj,

负数=> this<obj,

因此要使用sort()方法必须实现Comparable()接口。

**两个对象相同：**

需要重写class 的equals 方法，也必须重写对象的hashCode()方法。两个对象一样，则其hashCode()返回值一定一样。

**简单的for循环：**

int[] a={1,2,3,4}

for(**int i : a**){}

类似的：

Collection c=new ArrayList();

c.add(‘a’);

c.add(‘b’);

for(**Object o : c**){}

这种方法适用于简单处理每个元素，但是不能访问特定的下标。

**选择合适的数据结构：**

Array读快改慢

Linked 改快读慢

Hash 介于两者之间。

**泛型：**

几个例子：

list: 遍历的时候可以用索引来遍历。

List<String> c = new ArrayList<String>;

c.add(“a”);

c.add(“b”);

c.add(“c”);

for (int i=0;i<c.size();i++){

String s =c.get(i);

System.out.println(s);

}

set:遍历的时候要用iterator 来遍历

Collection<String> c2 = new HashSet<String>{

c2.add(“a”);

c2.add(“b”);

c2.add(“c”);

for(Iterator<String> it=c2.iterator();it.hasNext();){

String s =it.get(i);

System.out.println(s);

}

Comparble<E> 接口可以实现泛型

Map<String,Integer> 指定两个泛型

**关于异常：**

异常的根类：Throwable, 包括Error,Exceptionl类

Error: 系统内部的错误

Exception: 其他的exception，和RuntimeException(可以不逮，因为出现的频率较高，如ArithmaticException)

逮异常时先逮小的异常，再逮大的异常。

自定义异常，继承Exception 或者RuntimeException.

注意：重写方法时需要抛出与愿方法抛出异常类型一样或者不抛出异常。

注意throws 和throw 的 区别：

当在定义方法时不想处理异常，可以在定义方法时throws相应的Exception,将其扔给调用该方法的调用者处理。 而throw 则是在方法内部throw 方法。

**关于常用类：**

String : 不可变的字符序列。

StringBuffer:可变的字符序列。

基础数据类型包装类 java.lang：

Integer,Double, Float,Character

Enum: java.lang.Enum

使用enum 就可以定义一个Enum对象,如：

Public enum MyColor {red,green,blue}

枚举类型对于已经定义好变量区域的范围最好，例如定义了四个季节，星期几。

**关于JAVA 中的包：**

package com.test1

编译的时候**javac –d . Hello.java**(-d 表示生成目录，. 表示当前目录)

运行的时候要写出所有的名称 **java com.test1.hello**

**注意要倒入外来包的class 时必须要声明为public类型。**具体的访问类型：

public:不同的包可以访问

protected:本包可以访问，不同包的子类可以访问

default:只能在本包中访问

private:只能再本类中访问。

如果要倒入的包中的类的方法都是静态方法，则可以使用静态倒入：

import static com.test1.Hello.\* (.\*代表倒入所有静态方法)

常用的几个包。Java.lang(常用包，如String)，java.lang.reflect(反射机制包，可以完成大量底层操作)，java.util(工具包)，java.net,java.io….

**关于线程：**

java.lang.Thread

sleep: Thread 的静态方法。

终止一个线程，一般run方法结束了，线程就结束了，最好不要使用interrupt,stop 方法，一般这样(定义一个flag)：

class Runner implements Runnable

{

private Boolean flag=true;

public void run()

{

while(flag==true)

{

do something in this thread;

}

}

public void shutdown()

{

flag = false;

}

}

**线程的同步：**访问同一资源时候

1:synchronized(锁定对象)｛｝

2:给方法加锁即调用该方法的对象上锁。

//锁定方法1

public void add (String name)

{

synchronized(this)

{

num++;

try{Thread.sleep(1);}

catch(InterruptedException e){}

System.out.println();

}

}

//锁定方法2：锁定当前调用该方法的对象

public synchronized void add(String name)

{

num++;

try{Thread.sleep(1);}

catch(InterruptedException e){}

System.out.println();

}

}

注意：加了锁的方法，还能访问其他没有加锁的方法，加锁影响效率，但是相对安全，实际中要考虑是否加锁。

**join()和yield():**

join() 将线程合并到当前这个对象线程，如果使用了t1.join(),则需要等待t1运行结束合并到当前线程（需要catch InterruptedException）类似于调用了t1线程的run()方法。

Yield()是让出cpu,但注意并不是一直让出，而是下次线程。

**线程的优先级：**

默认的优先级Thread.NORM\_PRIORITY 是5,范围是0～10；

可以用t1.setPriority()设置优先级。

**wait()和notify():生产者和消费者多线程例子**

注意只有在synchronized 的方法才能使用wait(),notify(). 等待和叫醒一个线程，一般都是成双存在的。如果只有wait()没有notify(),则也容易产生死锁。

Wait() 时就不拥有锁了，sleep()时候仍拥有该锁。**Wait() 和notify()来自object**类，**sleep()来自Thread类**。

**关于Object 类：**

getclass(), hashcode(),wait(),notify(),notifyAll(),toString()…

toString():返回对象的字符串表示。如果要打印某个对象，需重写类的toString（）方法。

equals():比较两个对象的内容是否相同，**注意和==的区别**：一般a==b比较的是两个引用是否指向同一个对象，如果是new出来的对象，肯定不相同. String,Date 类都重写了equals()方法。［类似与python中的is(\_\_eq\_\_特殊方法)］

**关于网络类：**

**java.net**

**TCP/IP 参考模型：**

应用层（我们编写的程序，与tcp打交道），传输层（tcp层），网络层（IP层），物理层。

TCP(transmission control protocol):字节流,可靠的连接

UDP(user data protocal):不可靠的连接。

**socket:**

class Socket:客户端socket

class ServerSocket: 服务端的socket,其中accept()方法接受一个客户端。

端口号：一般两个字节（最多65536个端口号），自己编写占用端口最好是1024以上的。端口号也分tcp 端口和udp端口，每一个都最多65536端口。

注意：运行的时候先启动server端，再启动client端。别忘了关闭socket.

**UDP：**

服务器端：创建一个新的插口再某个窗口上：

DatagramSocket ds=new DatagramSocket(8888);

ds.receive(inputsteam);

客户端：DatagramPacket(字节数组，字节大小，插口地址（包括ip和端口）)

DatagramPacket dp=new DatagramPacket(buffer,buffer.length,new InetSocketAddress(“127.0.0.1”,8888));

DatagramSocket ds=new DatagramSocket(9999);//客户端端口

ds.send(dp);

ds.close;

**关于io流：**

**java.io**

**分类：**

数据流的方向：输入输出流

处理数据单位：字节流和字符流

功能不同：节点流(直接一根管道)和处理流（中间有过略的其他管道）

四个**抽象类**：

|  |  |  |
| --- | --- | --- |
|  | **字节流** | **字符流** |
| **输入流** | InputStream | Reader |
| **输出流** | OutputStream | Writer |

**InputStream**：向程序中输入数据，数据类型是字节（8bit）.FileInputStream, ByteArrayInputStream, StringBufferInputStream.

常用方法：

int read() 一个字节地读

int read(byte[] buffer) 读取字节并且存储在一个数组buffer里面

**OutputStream:**

常用方法：

void write() 一个字节地写

void flush() 再close之前flush一下，

**Reader:** 单位是两个字节（16bits）

常用方法：

int read()throws IOException 读取一个字符

int read(char[] cbuf) throws IOException: 读取一系列字符并存储到数组buffer

**Writer:**

**节点流的类型：**

|  |  |  |
| --- | --- | --- |
| **类型** | 字符 | 字节 |
| File | FileReader  FileWriter | FileInputStream  FileOutputStream |
| Memory Array | CharArrayReader  ChaeArrayWritter | ByteArrayInputStream  ByteArrayOutputStream |
| Memory String | StringReader  StringWriter | ----- |
| Pipe | PipedReader  PipedWriter | PipedInputStream  PipedOutputStream |

**处理流的类型：**

|  |  |  |
| --- | --- | --- |
| 类型 | 字符 | 字节 |
| Buffering | BufferedReader  BufferedWriter | BufferedInputStream  BufferedOutputStream |
| Filtering | FilterReader  FilterWriter | FilterInputStream  FilterOutputStream |
| Converting between bytes and character | InputStreamReader  OutputStreamWriter |  |
| Object serialization |  | ObjectInputStream  ObjectOutputStream |
| Data conversion |  | DataInputStream  DataOutputStream |
| Counting | LineNumberReader | LineNumberInputStrem |
| Peeking ahead | pushbackReader | PushbackInputStream |
| printing | PrintWriter | PrintStream |

**缓冲流(buffer)：**对读写数据提供了缓冲的功能，减少读写的次数，提高了效率

BufferedReader(Reader in, int size) //自定义缓冲区大小

BufferedWriter(Writer out ,int size)

BufferedInputStream(InputStream in, int size)

BufferedOutputStream(OutputStream out,int size)

注意：BufferedReader 有一个好用的方法**readline()**,返回这一行的string.

**转换流：把字节转换成字符**

InputStreamReader需要和InputStream套接

OutputStreamWriter需要和OutputStream套接

转换流再构造时可以制定其编码集合。

注意：OutputStreamWriter 的对象可以直接写入一个字符串，很好调用。

注意：没有把字符转换成字节的。

**数据流：直接写基础的数据类型，不需要转换成字节数组。**

比如将long 类型的数据存储到文件上，可以将其先转换成字符数组，然后再写入到文件上，但是占用的内存会比较大。可以使用数据流。

DataInputStream 和 DataOutputStream 提供了可以存取个机器无关的原始数据类型（如int,double等）的方法。

DataInputStream(InputSream in)

DataOutputStream(OutputStream out)

注意：readUTF()适用于网络上的编程，读取unicode字符。

**print流：**

PrintWriter 和 PrintSream 都属于输出流，分别针对字符和字节，输出操作不会抛出异常，用户通过检查错误状态获取错误信息，且其具有自动flush功能。

**Object流：序列化－ObjectOutputStream 和 ObjectInputStream**

**Transient 关键字：**序列化的时候是透明的，不与考虑。如 transient int a=0;

则输出的一直是0.

**serializable接口** ： 没有包含任何方法，有很多子接口，序列化。

**externalizable接口:**  serializable的子接口，自己控制序列化过程，serializable是jdk帮我们控制序列化过程。

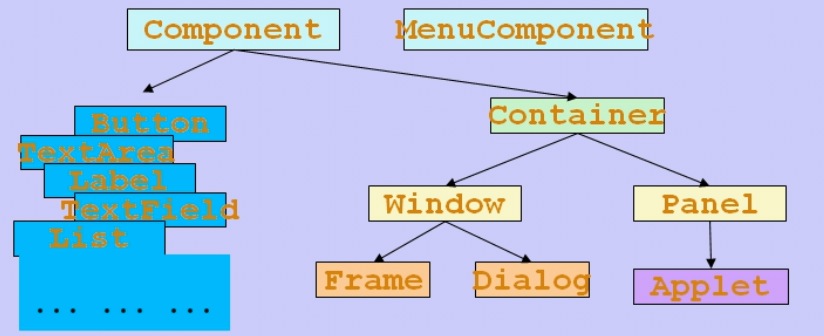
**关于GUI：**

**java.awt(abstract window toolkit)**

**javax.swing(java extension )**

**Component:** 所有可以显示的图形元素，如button,canvas, 一般要在container里面才能显示。

**Container:** 容器，其本身也是一种component, container是可以容纳其他component . container 分两种－window 和 panel，前者作为一个独立的窗口显示出来，分frame和diolog; 后者必须在 一个window 里面，但后者可以容纳一个其他内容。

****

**Frame: Window容器的子类**

setSize(),setBackground(),setVisible()….

注意：f.pack()可以自动包裹各种组建，可以不用手动设置大小(setSize)

**Panel:**

也有setSize(),setBackground(),setBound()…. 将panel 对象加入到frame 对象里面。

**布局管理器：panel向frame里面加入的方式，或者 frame 也有布局管理器。可以利用setLayout（FlowLayout 的一个对象）设置相应的布局管理器. Panel 默认布局管理器是flow, frame 默认的是border.**

FlowLayout: panel的默认布局管理方式，默认是居中。

GridLayout：布局管理器分割成规则的**矩形**  小格，从左到右，从上到下。

BorderLayout: Frame类的默认布局管理器，东西南北中 五个区域，默认加到中间。f.add(Button b1, BorderLayout.North)，加入到北边。

CardLayout,

GridBagLayout.

**事件监听：java.awt.event, 有ActionEvent,MouseEvent,KeyEvent…**

事件源对象：例如button b;

事件对象：传给监听器

监听器：实现了某种接口的对象,例如Monitor m

注册：监听器将监听对象注册，b.addActionListener(m)，

**事件源 addActionListener(实现了对应接口的监听器的对象)**

**注意：**一种事件对应与一种接口，ActionEvent –ActionListener

**注意：**一个监听器对象可以监听多个事件源对象（例如多个button按钮），但是如果要区分不同的事件，则需要将事件源setActionCommand.

**注意：**在一个监听器类里面要访问事件对象，可以通过e.getSource()返回Object对象，然后再强制转换成所需要的对象。

**注意：**文本输入框，TextField, 若要输入的对象在文本框中不可见，则可用其方法tf.setEchoChar(“\*”),像密码输入框那样。

**Graphics类：java.awt**

重写public void paint(Graphics g)｛｝方法，该方法是来自**Container .**这种方法比较特别，**每次需要重画的时候**会自动调用了该方法。对象.repaint()方法将会自动调用paint()方法重画。(实际是repaint()-update()-paint());

**MouseEvent: java.awt.event**

监控器要实现**MouseListene**r接口，但是实现该接口要覆写五个函数，如果不需要复写这么多方法，**MouseAdapter** 类实现了该接口，可以继承该类。

**WindowEvent: java.awt.event**

WindowListener,与上述类似，要实现该接口，要实现7个方法，但是继承相应的WindowAdapter 则更方便。

**KeyEvent: java.awt.event**

与上述类似。

**注意:** 在不同的类之间，如果我们要访问不同类的成员变量或者方法，如果定义两两之间访问的方法，那将会非常麻烦，但是如果定义一个公共的接口，使得每次访问其他类的时候都通过这个接口，那么将会大大减少代码复杂度。这就是门面设计模式。

类似的在一个类(A)中想要访问另一个类(B)的成员变量，则可以在A中定义一个B 对象的成员变量，然后在构造函数中传入B 对象的引用，这样就可以访问B 类中的各种方法和变量了。

简单的例子：

public B{}

public A{

B b =null;

A(B b){

this.b=b;

}

}

**内部类：上述问题也可以用内部类解决，直接访问外面包装类的成员变量，类似于python的闭包。注意是类的成员变量，并不是类中其他方法的局部变量。因此要需要将要访问的变量在类中定义未类的成员变量**

**匿名类（局部类）：在方法内写的类，没有名字，当我们的类调用和修改次数很少时，不建议使用匿名类。使用内部类就足以了**

**关于反射机制：**

**java.lang.Class;java.lang.ClassLoader； java.lang.reflect**

ClassLoader:类加载机制，运行期间动态加载，不是一次全部加载完所有class.static 语句块只生成一次。

**三种类加载器：**AppClassLoader-> ExtClassLoader-> BootStrapClassLoader

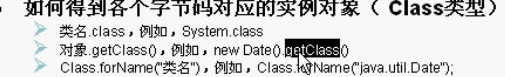
AppClassLoader: 加载classpath下指定的所有的jar包或目录

ExtClassLoader:加载jre/lib/ext下所有的extension 的jar包

BootStrapClassLoader: 加载jre/lib/rt 下系统的内置jar包,

ClassLoader类里面loadClass()方法就是先在父类加载器加载，要写自己的classloader类，要继承ClassLoader类，复写findClass()方法

Class 是所有java类的总类，也就是说所有java类都是Class的实例，都有相对应的**字节码**，如何得到字节码：



9个基本类型，不包括String，Array.

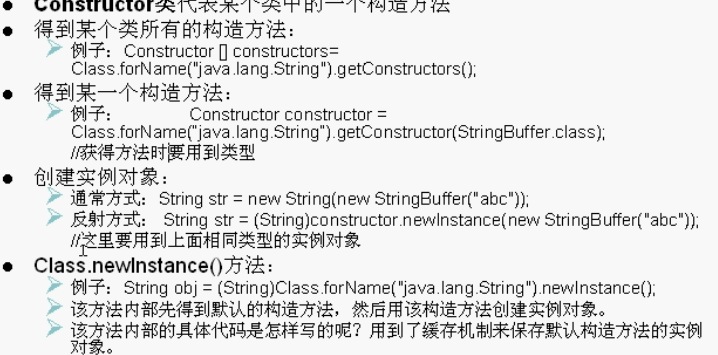
反射就是将java 中各个成分映射到java类上， 如Field, Method, Constructor,Package….

**Constructor 类**:构造方法

Constructor 是有相应类的字节码的getConstructor()方法所得。

Constructor constructor1 = String.class.**getConstructor**(StringBuffer.class);

String str2 = (String)constructor1.**newInstance**(new StringBuffer(“abc”));



**Field类**:成员变量

对于公有的属性变量，可以通过将实例的(即我们定义的类的代码段)的get

Fields()方法（在Class类中，如类ReflectedPoints 有两个变量，私有的x和共有的y：

**ReflectedPoints rp1 = new ReflectedPoints(2, 3);**

**Field field1 = rp1.getClass().getField("y");**

**System.out.println(field1.get(rp1));**

对于私有的属性x我们不能直接通过getFields()方法得到，但是可以通过getDeclaredFields()得到，要访问，我们也需要将setAccessible(true):

**Field field2 = rp1.getClass().getDeclaredField("x");**

**field2.setAccessible(true);**

**System.out.println(field2.get(rp1));**

**Method类**: Java类中的方法的反射

类似的Method实例对象要使用Class().getMethod()方法来创建，其本身要使用invoke()来调用方法。

**String s2 = "abc";**

**Method methodCharAt = String.class.getMethod("charAt", int.class);**

**System.out.println(methodCharAt.invoke(s2, 1));**

如果**invoke(obj,parameter)**中**第一个参数是null**,那么便是不要对象实例来调用方法，那么这个方法肯定**是static的**。例如调用其他包的main方法。**因此调用方法一定要有一个对象，除非是静态的。**

**java.lang.reflect.Array：**Java类中数组的反射

**注意**，和Arrays类不一样，Arrays是操作数组的一些方法的类。

Array包的方法里面可以通过反射机制调用原数组，有getLength()[获取数组的长度],get(object,int)[获得原数组在index上的值]……

**关于设计模式：**

**iterator模式:**

自己写一个ArrayList: 1:用数组实现；当数组要越界时，自动开辟一块更大的空间 2:用链表实现，用head 和tail来记录链表信息

设计模式之iterator .

System.arraycopy()复制array.

**策略模式：两个接口：Comparable :java.lang Comparator :java.util**

实现对任何 object的排序，Comparable接口有compareTo方法来实现比较的规则。

根据不同的属性对物体实现排序，Comparator接口有compare方法，要求实现该接口的类必须实现怎样比较，例如我们有一个猫类，有weight 和height两个属性，首先我们要实现Comparable接口以确保我们能比较两个猫类对象，但是在重写Comparable接口的 compareTo ()方法时，我们使用不同的比较准则-根据什么比较大小，例如CatHeightComparator()，CatWeightComparator()对象。

使用泛型可以在实现上述两个接口时指定要比较的类型，这样就可以不用在强制转换成我们所需要的类型了。

Java.util.Arrays 里面有sort()方法，

**单例多例模式：**

不提供公共的创建实例方法，即构造方法时私有的，但有一个**getInstance()**共有方法，getInstance()方法内部可以自定义产生实例的条件，这里返回一个对象实例，这里getInstance()是**静态工厂模式。**如果有多个实例，就是多例模式。

**简单工厂：**

一个简单的例子，交通工具，Moveable接口 (有run()方法)，VehicleFactory接口(有creat()方法)，这里的**VehicleFactory**就是**简单工厂模式**，其内的creat()方法可以控制产生实例的方法。可以用getInstance ()方法产生。

**抽象工厂：**

一系列的产品管理，比如司机开车，吃苹果，拿着武器。。。。集合成一个DefaultFactory类,有createCar(),createApple(),createWeapon(),三个方法，但是如果我们改变了Factory，如MagicFactory类，相对应我们就需要重新写方法了。要统一DefaultFactor和MagicFactory类，我们抽象成一个AbstractFactory接口或者抽象类，内部有三个抽象方法，Vehicle createVehicle(),

Weapon createWeapon(), Food createFood(), 然后Vehicle，Weapon,Food是三个接口。这样MagicFctory 和 DefaultFactory就可以实现AbstracFactory接口，实现相应的create方法。

总的来说，**抽象工厂将不同工厂里面同类型的产品的统一成一个接口**，我们更加方便扩展我们的工厂。

**Bean工厂:**

内有getBean

**动态代理：**

聚合：将操作一个类中的方法抽象成一个类，这个类就是静态代理。聚合类比继承关系要更方便。代理类应该实现和被代理类实现相同的接口，这样就会有多态存在，我们更好调用。

**关于正则表达式：java.lang.String; java.util.regex.Patteran; java.util.regex.Matcher**

Pattern p = Pattern.[compile](../../../java/util/regex/Pattern.html#compile-java.lang.String-)("a\*b");

Matcher m = p.[matcher](../../../java/util/regex/Pattern.html#matcher-java.lang.CharSequence-)("aaaaab");

boolean b = m.[matches](../../../java/util/regex/Matcher.html#matches--)();

**<=>** boolean b = Pattern.matches("a\*b", "aaaaab");

上述三个式子更加有效率。

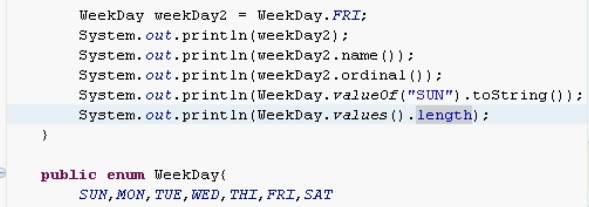
**关于静态倒入：导入类中的静态方法，**

import static java.lang.Math.\*;

**关于可变参数：默认将可变的参数放入到一个数组里**

public int add(int x,**int … args**) 这里args是一个数组。

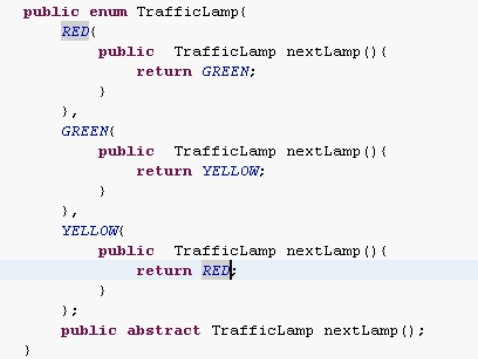
**关于enum 类：**



WeekDay 是enum类型的class，其中较重要的几个方法，WeekDay.values()返回所有枚举的一个数组，这样我们容易遍历。

枚举类型的构造方法必须是private,

下面是一个实现了抽象方法的enum类型，实现了nextLamp()方法。

****

单例的类可以用enum且只有一个实例的来创建，实际上，因为enum类型的构造方法是私有的，我们只能通过其公共的接口来调用并创建实例，因此当enum只有一个实例时就是单例模式。

**内省：Introspector**

JavaBean类：特殊的java类。

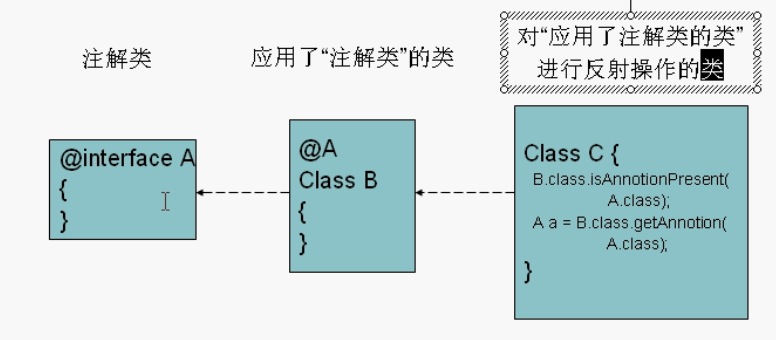
常用的是BeanUtils jar包，是apache的:

import org.apache.commons.beanutils.BeanUtils;

import org.apache.commons.beanutils.PropertyUtils;

用此包的好处是客户端处理属性都是通过字符串输入和输出的，这种包在web开发的时候运用的很多.

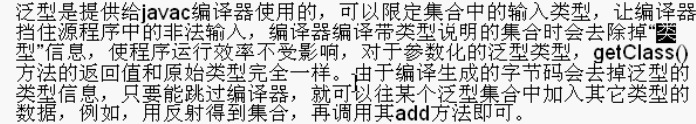
**注解：annotation**



annotation可以有不同的属性，类似于接口的抽象方法



**generic 泛型：**



简单地说，泛型是编译时javac使用，但是字节码编码时我们并没有用到泛型，因此我们可以通过反射(透过字节码class来调用相应地方法修改)，例如：

ArrayList<Integer> a1 = new ArrayList<Integer>();此时a1只能装Integer类型，但是当我们透过反射来调用ArrayList的add方法，我们同样可以加入其他类型的元素，比如string.

注意数组不能使用泛型来规定指定类型。

**ArrayList<Integer> a2[] = new ArrayList<Integer>()**

**问好通配符**：? ->当泛型的对象作为函数参数传入进来时

**private static void printCollections(Collection<?> a1)**

**<=>**

**private static <T> void printCollections(Collection<T> a1)**

拓展的?通配符：

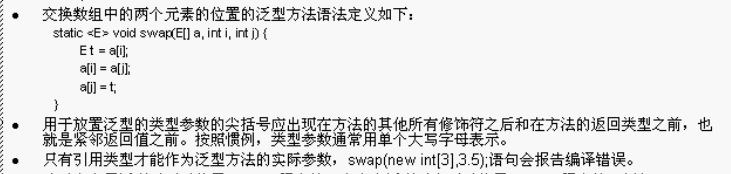
**private static void printCollections(Collection<? Extends Number> a1)**

**🡪泛型的类型必须时Number的子类包括Number 类**

**private static void printCollections(Collection<? super Number> a1)**

**🡪泛型的类型必须时Number的父类包括Number 类**

**注意：**自定义泛型借鉴于c++的template，但是没有其功能强大。

****

**在泛型中一个数组里必须是引用类型，不能是primitive数据类型，不然不能实现template效果。**

自定义泛型类:

**public class AutoGenericClass<T>**

{private T field 1;

public void save(T obj){}

public T getById(int id){}

}

可以通过反射机制(Method ,getGenericParameter(),getRawType())来获得泛型的实际参数类型。