经历的过程：

（1）类加载： ClassLoader

（2）链接 保证Java字节代码结构上的正确性

（解析策略）：（1）链接是依次加载需要的其他类

（2）延迟解析：当正真用到的时候解析

验证:

准备:

解析: 类的引用——>触发类加载，内联---提高效率

（3）初始化：创建class对象

类的静态域，静态代码块（属于class对象）

他的直接父类也会被初始化（接口不会初始化）

怎样会触发初始化呢？

（1）New操作

（2）调用类的静态方法

（3）访问类的静态域（不是final的常值常量）

（4）进行反射操作

（4）创建对象的实例

（1）调用父类的构造函数（如果在代码中没有调用super（），则由编译器为你添加对应的代码（编译器很神奇））

（2）初始化类中的实例域

（3）执行构造中的其他方法

对象的克隆：

（1）实现标记接口：Cloneable接口

（2）复写clone()方法 supe.Clone() ------浅拷贝 （适用于值的情况，如果对象中引用多，就不行）

包含复制构造方法的clone方法：

什么是包含对象复制的构造方法

其中的构造如下：

**public** User(User user) {

**this**.name = user.getName();

**this**.email = user.getEmail();

}

序列化：

Transient 关键字 private transient String password;（密码重要，不能用默认的序列化）

private void writeObject(ObjectOutputStream stream)throws IOException {

stream.defaultWriteObject(); //先按默认方式序列化

stream.writeObject(change(password.getBytes()));--🡪>>>自定义序列化

stream.writeInt(count);

}

private byte[] change(byte[] buff){

for(int i=0;i<buff.length;i++){

int b=0;

for(int j=0;j<8;j++){

int bit=(buff[i]>>j & 1)==0 ? 1:0;

b+=(1<<j)\*bit;

}

buff[i]=(byte)b;

}

return buff;

}

自定义序列化过程 ------>.>通过Java再带的序列化框架

 通过传入的对象流对对象内部的状态进行过修改

（1）自定义一个方法叫writeobject（）；

（2）在上述方法内部先调用outputstream.defaultWriteObject()方法

（3）在定义自己的序列化过程

单例类的反序列化

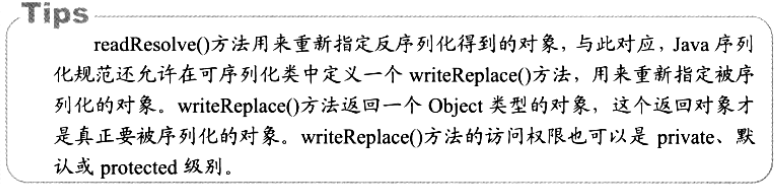
单例在整个程序运行过程中应该只有一个实例，而反序列化可以打破这个状态-------->>>在要反序列化的类中实现

private Object readResolve() throws ObjectStreamException{

return INSTANCE;

}

在反序列化的过程中，会调用这个方法。。。。所有实现单实例只要在该方法中返回原对象的实例即可



为了保证不同的序列化之间兼容或者不兼容，可以显示的制定序列化的serialVersionUID的值-------》在类中添加 private static final long serivalVersionUID=-1293819L

定制的序列化除了通过实现serizable接口外，

还可以实现Externalizable接口（序列化的过程完全由自己控制，序列化后流的格式啊，怎么读取数据的，怎么进行创建啊），通过接口定义的writeExternal()和readExternal()方法实现序列化，不过通过实现此接口的可序列化类必须实现无参的构造方法

简单例子：



对象的完整性验证：



主要代码：

input.registerValidation(new UserValidator(this), 0);---在进行反序列化的代码。

验证器的简单实现（如下）：

**private** **static** **class** UserValidator **implements** ObjectInputValidation {

**private** NewUser user;

**public** UserValidator(NewUser user) {

**this**.user = user;

}

**public** **void** validateObject() **throws** InvalidObjectException {

**if** (user.getAge() < 0) {

**throw** **new** InvalidObjectException("验证出错");

}

}

}