# 基本概念

序列化和反序列化是java总进行数据存储和数据传输的一种方式。

1. 对象的序列化：对象转化为字节
2. 对象的反序列化：字节转化为对象

说明：有时也将对象转化为字符串的过程也称为序列化，例如将对象转化为json字符串。

# Java中如何实现序列化和反序列化

1. 对象之间或间接实现Serializable接口
2. 添加序列化id(为反序列化提供保障)
3. 借助对象流对象实现小老虎和反序列化

class Message implements Serializable {  
 */\*此序列化ID保证了以后如果此类发生结构变化（例如加个属性）  
 \* 已经序列化的内容被反序列化时不会出错\*/* private static final long *serialVersionUID* = 2668788440691836642L;  
 private int id = 0;  
 private String context = "TestSerializable";  
 private String name = "Clark";  
  
 @Override  
 public String toString() {  
 return "Message{" +  
 "id=" + id +  
 ", context='" + context + '\'' +  
 ", name='" + name + '\'' +  
 '}';  
 }  
}  
  
public class Test01 {  
 static void serialiable(Object o,File file) throws IOException{  
 var fos = new FileOutputStream(file);  
 ObjectOutputStream oos = new ObjectOutputStream(fos);  
 oos.writeObject(o);  
 System.*out*.println("Serializable sucessful");  
 oos.close();  
 }  
 static Object deseSerializable(Object o,File file) throws IOException, ClassNotFoundException {  
 FileInputStream fis = new FileInputStream(file);  
 ObjectInputStream ois = new ObjectInputStream(fis);  
 Object o2 = ois.readObject();  
 System.*out*.println("Deseserialiable sucessful");  
 ois.close();  
 return o2;  
 }  
 public static void main(String[] args) throws Exception {  
 File file = new File("D:\\Java\\a.txt");  
 Message msg = new Message();  
 *//序列化  
// serialiable(msg,file);  
 //反序列化* Object obj = *deseSerializable*(msg,file);  
 *//输出反序列化后的对象* System.*out*.println(obj);  
 }  
}

# 设计一个SerializableCache（参考Mybatis）

## 设计SerializableCache

public class SerializableCache implements Cache{  
 private Cache cache;  
 public SerializableCache(Cache cache){  
 this.cache = cache;  
 }  
 *//将对象序列化存到字节数组中* private byte[] serializable(Object o) throws IOException {  
 *//1.构建字节数组输出流对象* ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 *//2.构建对象输出流* ObjectOutputStream oos = new ObjectOutputStream(baos);  
 *//3.执行对象序列化* oos.writeObject(o);  
 *//4.释放资源* oos.close();  
 baos.close();  
 *//5.返回字节数组* return baos.toByteArray();  
 }  
 *//将字节数组中的字节反序列化为对象* private Object deserializable(byte[] arr)throws Exception{  
 *//1.构建字节数组输入流对象* ByteArrayInputStream bais = new ByteArrayInputStream(arr);  
 *//2.构建对象输入流对象* ObjectInputStream ois = new ObjectInputStream(bais);  
 *//3.将字节数组内容读取为对象* Object obj = ois.readObject();  
 *//4.释放资源* ois.close();  
 bais.close();  
 *//5.返回对象* return obj;  
 }  
 @Override  
 public int size() {  
 return cache.size();  
 }  
 @Override  
 public void putObject(Object key, Object val) {  
 try {  
 *//1.将对象序列化(字节数组)* byte[] arr = serializable(val);  
 *//2.将对象存储到cache* cache.putObject(key, arr);  
 }catch (IOException e){  
 e.printStackTrace();  
 *//抛出异常* throw new RuntimeException(e);  
 }  
 }  
 @Override  
 public Object getObject(Object key) {  
 *//1.从cache获取对象(字节数组)* byte[] val = (byte[])cache.getObject(key);  
 try {  
 *//2.将字节数组中的内容反序列化* Object obj = deserializable(val);  
 return obj;  
 }catch (Exception e){  
 e.printStackTrace();  
 *//抛出异常* throw new RuntimeException(e);  
 }  
 }  
 @Override  
 public Object removeObject(Object key) {  
 return cache.removeObject(key);  
 }  
 @Override  
 public void clear() {  
 cache.clear();  
 }  
  
 @Override  
 public String toString() {  
 return cache.toString();  
 }  
}

## 设计一个类实现Serializable接口

class Problem implements Serializable {  
 private static final long *serialVersionUID* = 4217076308499133113L;  
 private int id;  
 private String title;  
  
 public void setId(int id) {  
 this.id = id;  
 }  
 public void setTitle(String title) {  
 this.title = title;  
 }  
 public int getId() {  
 return id;  
 }  
 public String getTitle() {  
 return title;  
 }  
 @Override  
 public String toString() {  
 return "Problem{" +  
 "id=" + id +  
 ", title='" + title + '\'' +  
 '}';  
 }  
}

## 设计测试类

public class Test02 {  
 public static void main(String[] args) {  
 SerializableCache cache = new SerializableCache(new PertetualCache());  
 Problem p1 = new Problem();  
 p1.setId(1);  
 p1.setTitle("A");  
 cache.putObject("p1",p1);  
 System.*out*.println(cache);  
 System.*out*.println(cache.getObject("p1"));  
 }  
}

# 设计一个类，在网络中传播

1.涉及自定义序列化和反序列化

2.客户端、服务端模拟

3.加密

4.线程

## /\*\*问题答复对象\*/

class Reply implements Serializable {  
 private static final long *serialVersionUID* = 6793300061729344487L;  
 private Integer id;  
 private String content;

### /\*自定义序列化过程(按照官方制定标准定义writeObject()方法)\*/

private void writeObject(ObjectOutputStream oos)  
 throws IOException {  
 *//1.获取加密对象(Base64)* var encoder = Base64.*getEncoder*();  
 *//2.对内容进行加密* byte[] res = encoder.encode(content.getBytes());  
 content = new String(res);  
 *//3.对内容进行序列化* oos.defaultWriteObject();  
 }

### /\*自定义反序列化过程\*/

private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {  
 *//1.反序列化（含操作：把数据直接赋值给对象的相应属性）* ois.defaultReadObject();  
 *//2.获取解密对象* var decoder = Base64.*getDecoder*();  
 *//3.进行解密* byte[] res = decoder.decode(content);  
 content = new String(res);  
 }  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getContent() {  
 return content;  
 }  
 public void setContent(String content) {  
 this.content = content;  
 }  
 @Override  
 public String toString() {  
 return "Reply{" +  
 "id=" + id +  
 ", content='" + content + '\'' +  
 '}';  
 }  
}

## //测试类

public class Test03 {

### //模拟服务端

static void doServer() throws IOException, ClassNotFoundException {  
 *//1.创建服务器* ServerSocket server = new ServerSocket(9999);  
 *//2.等待客户端连接（只连接一次）* Socket socket = server.accept();  
 System.*out*.println("客户端已连接");  
 *//3.获取流对象并读取网络数据* InputStream is = socket.getInputStream();  
 ObjectInputStream ois = new ObjectInputStream(is);  
 Object o = ois.readObject();  
 System.*out*.println(o);  
 *//4.释放资源* ois.close();  
 socket.close();  
 server.close();  
 }

### //模拟客户端

static void doClient(Object o) throws IOException {  
 *//1.创建客户端socket对象* Socket socket = new Socket("127.0.0.1",9999);  
 *//2.获取流对象进行数据写操作* OutputStream os = socket.getOutputStream();  
 ObjectOutputStream oos = new ObjectOutputStream(os);  
 oos.writeObject(o);  
 *//3.释放资源* oos.close();  
 socket.close();  
 }

### //测试

public static void main(String[] args){  
 *//1.启动服务(服务器启动后会进入阻塞状态，要想启动客户端，可以在不同线程中实现)* new Thread(){  
 @Override  
 public void run() {  
 try {  
 *doServer*();  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 }.start();  
 *//2.启动客户端，写数据到服务端* try {  
 Reply reply = new Reply();  
 reply.setId(100);  
 reply.setContent("hello");  
 *doClient*(reply);  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
}

# 序列化粒度怎么控制

1. 用Transient修饰不需要序列化的属性
2. 让序列化接口实现Externalizable接口，自己指定属性的序列化和反序列化过程，但是要序列化的对象对应的类必须使用public修饰

# 序列化性能问题及如何优化

其他序列化框架，kryo。。。等