期中考试的原题有可能出现在期末考试中

考试整学期的内容，侧重点在后几章

1 单选 2 20

2 填空 2 20

3 简答 每题10分 30

异常处理机制及相关关键字

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner in = new Scanner(System.in);

String input1 = in.next();

System.out.println(input1);

in = new Scanner(System.in);

String input2 = in.nextLine();

System.out.println(input2);

}

集合类常用的类型及其区别，常用的方法

List 更加关注索引顺序

ArrayList 快速遍历、迭代 少量的插入删除操作

LinkedList 迭代慢、快速的插入删除

方法：get set add remove size isEmpty clear

Set（集） 关注数据的唯一性

HashSet 无序的

LinkedHashSet 插入顺序

TreeSet 自然顺序 (升序)

方法：add remove size isEmpty clear

Queue

方法：offer poll remove peek put take

Map 键值对 key是唯一的 value不唯一

HashMap LinkedHashMap TreeMap

Put clear isEmpty remove size

Keyset values

Keyset().iterator

Values().iterator

Iterator it = map.entrySet().iterator();

While(it.hasNext()) {

Entry entry = (Entry)it.next();

entry.getKey();

entry.getValue();

}

**package** org.edu2act.zko1021.\_59;

// 期中考试

**public** **class** Demo {

// final类不能被继承

**static** **int** *i*; //默认为0

**public** **static** **void** main(String[] args) {

String s = "0123456";

System.***out***.println(*i*);

System.***out***.println(s.substring(2,5)); //从2开始，并不打印5

**new** Hi\_B();

}

}

**class** Hi\_A{

**public** Hi\_A(){ //3

System.***out***.println("Hi\_A");

}

{ //2 属于类的，先于构造函数执行

System.***out***.println("AAA");

}

**static** { //1

System.***out***.println("STATIC AAA");

}

}

**class** Hi\_B **extends** Hi\_A{

**public** Hi\_B(){ //3

System.***out***.println("Hi\_B");

}

{ //2

System.***out***.println("BBB");

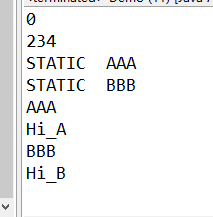
}

**static** { //1 类装载时，static方法就要调用，所以这两个最先调用

System.***out***.println("STATIC BBB");

}

}



**for**(n=1;++n<4;);

System.***out***.println(n); 结果为4

**for**(n=1; n++<4;);

System.***out***.println(n); 结果为5

Buffered → 读的时候用的是readLine方法。

## 类的访问权限，封装，继承，多态

## 泛型 set map list 如何区分

进程 程序 线程的功能区别 线程的生命周期等

4 程序题 15 30

网络编程， Socket 编程，TCP协议，UDP协议 数据库编程

客户端和服务器端之间的通讯 重点在于核心代码

区分哪些是类 对象

类的声明和定义 一个java文件中只能有一个public类

对象的实例化

类成员的访问

## 成员方法的修饰符

## final修饰方法 不能被重修 修饰类 不能被继承

synchronized 同步

## 三种成员属性的权限以及缺省权限 包权限

## 成员方法 返回值？

## 方法重载的匹配问题 精确匹配 类型提升

方法重载 参数类型或个数不同 方法名相同

注意 返回值不同 是否静态 不作为重载依据

不同重载方法可以互相调用

## 包装器类型 匹配 类型提升

## 构造方法 不加返回值

方法传参 基本数据类型

引用数据类型的传递

类定义的对象一般都是引用传递

## 垃圾回收机制 其中基本的理论 回收情形 空指针 隔离引用

垃圾回收机制 JC

JAVA垃圾收集器受JVM的控制，JVM决定什么时候运行垃圾收集器。

当JVM感到内存不足时会运行垃圾收集器

在JAVA程序中可以请求JVM运行垃圾收集器，但无法保证JVM会答应请求

当没有现成能够访问对象时，该对象就是适合进行垃圾收集的。

空指针 隔离引用 i2 i3 i4 i2.next = i3; i3.n =i4; i4.n =i2; 都是null

## 包的使用 what 不同的包之间可以有同名的文件 三种包

包 package 命名空间 避免命名冲突

必须放在源程序的除去注释外的第一行 java.lang.String String是包名

使用方法： 系统内置的包 自定义包 第三方提供的包

3. import 导包 or 在每一个类名之前添加完整的包名

4. import指定类 import整个包 import java.util.array import java.util.\*

5. import 唯一价值 减少键入

## 导包的方法 通过全名来访问 or \*

## 几个重要的包 包含什么类 net上面有个不用的

重要包

java.lang String Math Ineger

java.awt 抽象

java.applet

java.net

java.util 包含为任务设置的使用程序累 随机数发生 日期日历相关

java.io

## 访问权限符 4种 理解

pirvate（类内） public protected（包内+子类） 缺省（包权限）

## 继承的基本概念 格式 一个类只能有一个父类 和后面的几个例子

## 静态成员的调用时间

静态代码块：当JVM加载所在类的时候调用一次，也就是第一次初始化这个类的对象的时候调用，而且整个程序中只调用一次，一般用来初始化类的静态成员  
普通代码块：每次实例化类的对象的时候都调用，且在构造方法之前调用，一般工程中只有特殊用途会用到  
构造方法：一般初始化类中的属性值

## super this 关键字

## 方法重写 规则

重写的规则

返回类型一致 相同的方法名 参数列表必须相同 不能重写为final的方法 不能缩小访问权限 如果一个方法不能被继承 则不能重写 例如 父类的private方法

## 抽象类 实现

接口 interface 为实现这个接口的类制订了一个标准

接口的继承 一个类可以实现多个接口 一个接口可以被多个无关的类实现 若不实现 → 接口 or 抽象

接口的继承也是多继承

Object类是所有类的父类

## hasCOde equals tostring 可能会考

public int hashCode(){

return name.hashCode()\*11;

}

## 多态 向上转型 多个类实现相同的接口 也会存在多态的问题 多态考点不多

向上转型 父类引用指向子类对象

向上转型既可以使用子类强大的功能，又可以抽取父类的特性 实现动态多态

不可以调用子类自己的。 会出错。

多态是如何实现的？ 重写

## 包装器类 应该会考

• Java的八种基本数据类型对应的包装器类分别为：Byte，

Short，Character，Integer，Long，Float，Double，

Boolean

有时也需要将包装器类型对象转换为基本数据类型

方法 返回类型

byteValue() byte

shortValue() short

intValue() int

longValue() long

floatValue() float

doubleValue() double

• 选择重载时匹配哪个方法时遵循以下原则：

– 是否有直接匹配的

– 是否加宽后直接能够匹配的

– 是否装箱后能够匹配的

– 是否有不定长参数能够匹配的

– 先装箱后加宽后能否匹配

• 可以先装箱，后加宽（int可以通过Integer编程Object）

内置类型 值传递 包装器类 引用传递

自动装箱 的过程 jdk支持

## 基本数据类型的加宽带来的问题 中的题目

## 异常会考 主要是异常的处理机制 捕获异常 抛出异常 区分关键字 Exception一定要写在最下面的catch

异常处理机制

• 对于异常Java中提供了两种常见的方式：

– 捕获异常

– 抛出异常

捕获异常

异常流程的代码和正常流程的代码分离，提高了程序的可

读性，简化了程序的结构

• 使用try和catch捕获异常

– try {

– //接受监视的程序块,在此区域内发生的异常,

– //由catch中指定的程序处理;

– }catch(要处理的异常种类和标识符) {

– //处理异常;

– }

try、catch处理流程

• try语句块中没有抛出任何异常

– 程序会跳过catch子句

• try语句块中抛出了catch子句中说明的异常

– 程序跳过try语句块中的其余代码

– 程序执行catch子句中的异常处理代码

finally语句定义一个总是被执行的代码块，而不考虑是否

出现异常

一个方法不处理它产生的异常,而是沿着调用层次向上传递

，由调用它的方法来处理这些异常，叫抛出异常

抛出异常: 不是出错产生，而是人为地抛出

– 语法：throw (异常对象);

– 如：throw new ArithmeticException();

如何抛出异常？

– 确定要抛出的异常类

• 系统提供的异常类

• 自定义的异常类

– 创建这个类的对象

– 将该对象抛出

断言不怎么考

## throw 和 throws 的区别

throws语句用在方法声明后面，表示该方法会抛出哪些异常，使

它的调用者知道要捕获这些异常。

– throw语句用在方法体内，表示抛出异常，是具体向外抛异常的

动作，它抛出一个异常实例。

– throws表示出现异常的一种可能性，并不一定会发生这些异常；

throw则是抛出了异常，执行throw则一定抛出了某种异常。

## 异常的类型

异常分为2大类

– Error：描述了Java运行系统中的内部错误以及资源耗尽错误

• 唯一的解决方法：尽力使程序安全地终止

– Exception：程序中需要关注的

• 运行时错误（RuntimeException）：在 Java 虚拟机正常运行期间

抛出的异常，由程序错误导致。 Java编译器允许程序中不对这类异

常做出处理。

– 错误的类型转换

– 数组下标越界

– 访问空指针

• 其他错误（如：IO异常、SQL异常）：一般是外部错误，Java编译器

要求在程序中必须处理这类异常

java的collection中的那幅图 重点关注实现类

队列比较少 set list考得应该多

## map非线性的存储结构 区别

## 每个的第一页 了解他们的特点

列表（ List ）：关心的是索引

– 对象按索引存储

– 可以存储重复元素

– 具有与索引相关的一套方法

• 主要实现类

– ArrayList：线性表

• 快速迭代，少量插入删除

– LinkedList：链表

• 迭代速度慢，快速插入删除

Queue接口

• java.util.Queue

• 队列是一种特殊的线性表，只允许在表的前端（front，

队头）进行删除操作，而在表的后端（rear，队尾）进行

插入操作

• 继承了Collection接口

• LinkedList实现了Queue接 口

映射（Map）

– 对象以键－值对（key-value）存储

– key不允许有重复，value允许有重复

• Map中元素，可以将key序列、value序列单独抽取出来

– 使用keySet()抽取key序列，将map中的所有keys生成一个Set。

– 使用values()抽取value序列，将map中的所有values生成一个

Collection。

List 索引 ArrayList和LinkedList的对比 后者插入删除快 etc

Set相当于字典 没有顺序 但是唯一

## map key - value 遍历

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

Set<String> keys = map.keySet();

for ( Iterator iterator = keys.iterator(); iterator.hasNext();)

System.out.println(map.get(iterator.next()); // zhuyi!!!

}

## 三个set的比较 优缺点 如何选择

HashSet

– 不能保证元素的排列顺序，顺序有可能发生变化

– 不是同步的，集合元素可以是null,但只能放入一个null

– 哈希表是通过使用称为散列法的机制来存储信息的，元素并没有

以某种特定顺序来存放；

• LinkedHashSet

– 以元素插入的顺序来维护集合的链接表，允许以插入的顺序在集

合中迭代；

– 遍历性能比HashSet好，但是插入时性能稍微逊色于HashSet

• TreeSet

– 提供一个使用树结构存储Set接口的实现，对象以升序顺序存储，

访问和遍历的时间很快；

## 迭代器 了解 注重算法而非类型

这种方式适用性更强

for ( Iterator iterator = Arrlist.iterator() ; iterator.hasNext();) // 最开始的iterator指向第一个元素前面的位置

Sring string = (String ) iterator.next();

}

## 泛型的定义和使用

泛型（Generics）

– 所谓泛型，即通过参数化类型来实现在同一份代码上操作多种数

据类型

– 泛型编程是一种编程范式，它利用“参数化类型”将类型抽象化

，从而实现更为灵活的复用

– 泛型赋予了代码更强的类型安全，更好的复用，更高的效率，更

清晰的约束

class Info<T> {

private T var;

public T getVar() {

return var;

}

public void setVar(T var) {

this.var = var;

}

}

public boolean equals(Object b ) {

if ( o instanceof Cat && (Cat)o.getName().equals(this.getName())) {

return true;

} else {

return false;

}

}

## 通配符了解一下即可 所以我觉得应该不考吧

10 void addAnimal(List<? extends Animal> animals)

Animal可以是类或接口

接受所声明变量类型的任何子类型

父类

void addAnimal(List<? super Dog> animals)

11 List<?> 与List< ? extends Object> 完全相同

List<Object> 与它们完全不同

12 泛型通配符只能用于引用的声明中 不可以在创建对象时使用

不可以使用采用了泛型通配符的引用调用使用了泛型参数的方法 P61

## 流与文件 创建读写获取 随机存取（no）

File dir = new File("C://1//");

boolean r = false;

if(!dir.exists()){

r = dir.mkdir();

if (r) {

System.out.println('目录创建成功');

}

}

// dir.mkdirs(); 创建目录时也会创建父目录 创建级联目录

File f = new File("C://1//a.txt");

if(!f.existd()){

r= false;

try {

r= f.createNewFile();

} catch (IOException e) { // 路径出错可能会出异常

e.printStackTrace();

}

if (r) {

System.out.println('文件创建成功');

}

}

5

try {

Writer w = new FileWriter(f);

w.write("hello world");

w.close();

} catch (IOException) {

...

}

6

String[] fs = dir.list();

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

System.out.println(fs[i]);

}

7 f.delete();

if (r) {

System.out.println('delete成功');

}

## 流的分类 字节流和字符流的区别

3 根据流的方向分类 输入流 输出流

4 根据操纵对象的类型分类

字符流 读取单位为字符 本质基于字节流 文本 不一定是char 也可以 utf-8各种

Reader 读取单位为1字符，2字节（16位）

字节流 读取单位为字节 一般用于文件传输 MP3 电影 外部设备中文件和外部设备中的文件显示

只要处理纯文本数据，优先考虑字符流，其他的形式采用字节流。

Reader r = new FileReader(new File("C://test.txt"); FILENOTFOUNDEXCEPTION

char cbuf = new char[256];

int len = r.read(cbuf); IOEXCEPTION

System.out.println(new String(cbuf,0,len));

r.close();

Writer w = new FileReader(new File("C://test.txt");

w.write('123');

w.close();

5 readObject 流 --反序列化--> 对象

wirteObject 对象 --序列化--> 流

6 // 将对象序列化后存入文件person.out

ObjectOutputStream oout = new ObjectOutputStream(new FileOutputStream(file)); //文件 传到这个地方。 要有try catch

oout.writeObject(p); //对象p try catch

oout.close();

//读取文件，反序列化后得到对象

ObjectInputStream oin = new ObjectInputStream(new FileInputStream(file));

Object newPerson = oin.readObject();

//如果不是Object 要强转 (Cat) oin.readObject();

oin.close();

System.out.println(newPerson);

7 BufferedReader ... 直接读\写一行字符 有时间再看看代码

## 程序 进程 线程 区别 5种声明周期很重要 非常清楚地说清楚

程序、进程、线程

– 程序是一段静态的代码，是应用软件执行的蓝本;

– 进程是程序的一次动态执行过程，它对应了从代码加载、执行至

执行完毕的一个完整过程，这个过程也是进程本身从产生、发展

至消亡的过程

– 线程是比进程更小的执行单位。进程在其执行过程中，可以产生

多个线程，形成多条执行线索，每条线索，即每个线程也有它自

身的产生、存在和消亡的过程，也是一个动态的概念.

进程

一个应用程序一个进程

独立功能的程序

通过多个线程占据系统资源

进程之间数据状态完全独立

线程

一个进程可以有多个线程

执行程序的最小单元

占用CPU的基本单位

线程间共享一块内存空间

Java中的线程

– 每个Java程序都有一个默认的主线程

– 当JVM加载代码发现main方法之后，就会立即启动一个线程，这

个线程称为主线程

• 主线程的特点：

– 是产生其他子线程的线程

– 不一定是最后完成执行的线程

线程完整的生命周期包括五个状态：新建、就绪、运行、阻塞和死亡.

新建状态：线程对象已经创建，还没有在其上调用start()

方法。

• 可运行状态：当线程调用start方法，但调度程序还没有把

它选定为运行线程时线程所处的状态。

• 运行状态：线程调度程序从可运行池中选择一个线程作为

当前线程时线程所处的状态。这也是线程进入运行状态的

唯一方式。

等待/阻塞/睡眠状态：其共同点是：线程仍旧是活的，但

是当前没有条件运行。它是可运行的，当某件事件出现，

他可能返回到可运行状态。

• 死亡状态：当线程的run()方法完成时就认为它死去。线

程一旦死亡，就不能复生。 一个死去的线程上调用start()

方法，会抛出java.lang.IllegalThreadStateException异

常。

## 线程的定义 出大题 run start 等方法

线程离开运行状态的方法：

– 调用Thread.sleep()：使当前线程睡眠至少多少毫秒（尽管它可能

在指定的时间之前被中断）。

– 调用Thread.yield()：不能保障太多事情，尽管通常它会让当前运

行线程回到可运行性状态，使得有相同优先级的线程有机会执行

– 调用join()方法：保证当前线程停止执行，直到调用join方法的线

程完成为止。然而，如果调用join的线程没有存活，则当前线程

不需要停止

计数器，从1-100之间计数，每隔1秒计数一次。

public class MyThread extends Thread {

public void run() {

for (int i = 0; i < 100; i++) {

System.out.print(i);

try {

Thread.sleep(1000);

System.out.print(" 线程睡眠1000毫秒！\n");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

new MyThread().start();

}

}

• 线程睡眠是帮助其他线程获得运行机会的最好方法。

• 线程睡眠到期自动苏醒，并返回到可运行状态，不是运行

状态。

• sleep()中指定的时间是线程不会运行的最短时间。因此，

sleep()方法不能保证该线程睡眠到期后就开始执行。

• sleep()是静态方法，只能控制当前正在运行的线程。

线程中的常用方法

• interrupt():“吵醒”休眠的线程，唤醒“自己”;

• yield():暂停正在执行的线程，让同等优先级的线程运行;

• join():当前线程等待调用该方法的线程结束后，再排队等

待CPU资源;

• stop():终止线程.

start():启动线程，让线程从新建状态进入就绪队列排队;

• run():线程对象被调度之后所执行的操作;

• sleep():暂停线程的执行，让当前线程休眠若干毫秒;

• currentThread():返回对当前正在执行的线程对象的引用;

• isAlive():测试线程的状态，新建、死亡状态的线程返回

false;

Java中两种创建线程的方式：

– 继承Thread类

• 重写run() 方法

• new一个线程对象

• 调用对象的 start() 启动线程

实现Runnable接口

– 实现run() 方法

– 创建一个Runnable类的对象r，new MyRunnable()

– 创建Thread类对象并将Runnable对象作为参数，new Thread(r)

– 调用Thread对象的start()启动线程

## 网络编程看老师写的代码 ppt从第八页开始一定要真正的理解

## URL相关

RL类：统一资源定位符，指向互联网“资源”的指针。

url = new URL("http://");

// 打开连接

URLConnection conn = url.openConnection();

// 得到输入流

InputStream is = conn.getInputStream();

// 关于IO流的用法和写法一定要熟悉

OutputStream os = new FileOutputStream("d:\\baidu.png");

byte[] buffer = new byte[2048];

int length = 0;

while (-1 != (length = is.read(buffer, 0, buffer.length))) {

os.write(buffer, 0, length);

}

is.close();

os.close();

URL网络编程实例：文件下载

• 下载服务器端文件，基本思路：

– 创建URL对象：URL url = new URL( 文件地址 );

– 获取服务器端输入流：InputStream is = url.openStream();

– 文件读写：从输入流中读取字节写入到输出流（文件）中。

String sUrl = "http://dl.zhishi.sina.com.cn/upload/

90/92/43/1167909243.14728960.jpg";

URL url = new URL(sUrl); // 创建URL对象

InputStream in = url.openStream(); // 获得网络输入流

// 创建文件输出流

FileOutputStream out = new FileOutputStream("qq.jpg");

int b;

while ((b = in.read()) != -1) {

out.write(b); // 写入文件

}

// 关闭输入输出流

out.close(); in.close();

URL网络编程实例：获取响应信息

• 获取服务器HTTP响应消息（消息头和消息主体）。

– 访问网址：http://software.hebtu.edu.cn/

– 获取该网页的服务器字符编码、文档类型、服务器响应状态码、

网页主体等。

URL url = new URL(rootUrl); // 创建Url对象

// 得到URLConnection连接对象

URLConnection conn = url.openConnection();

HttpURLConnection hc = (HttpURLConnection) conn;

// 获得响应消息头

conn.getContentType();

conn.getContentLength();

conn.getContentEncoding();

// 获得HTTP消息状态码

hc.getResponseCode();

hc.getResponseMessage();

// 获得HTTP响应消息主体

hc.getContent();

## socket一定会考 不要忘了try catch PPT P26

## 实现通讯 实现即可 不用引入多线程

/\*\*

\* 任务：实现聊天程序

\* 姓名：王肖

\* 学号：2014011764

\* 班级：5班

\* 日期：2016/5/23

\* 文件：TCPServer.java

\*/

package com.zko1021.www1;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class TCPServer {

public static void main(String[] args) {

ServerSocket server;

BufferedReader bufr;

Socket client = null;

try {

server = new ServerSocket(8898);

while (true) {

client = server.accept();

OutputStream os = client.getOutputStream();

bufr = new BufferedReader(new InputStreamReader(client.getInputStream()));

byte[] input = new byte[1024];

System.in.read(input);

os.write(input);

os.flush();

String inputstr = new String(input);

if (inputstr.startsWith("end")) {

break;

}

os.close();

}

} catch (IOException e) {

System.out.println("出现错误 " + e.getMessage());

} finally {

try {

client.close();

System.exit(0);

} catch (IOException e) {

System.out.println("出现错误 " + e.getMessage());

}

}

}

}

/\*\*

\* 文件：TCPCilent.java

\*/

package com.zko1021.www;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.Socket;

import java.net.UnknownHostException;

public class TCPClient {

public static void main(String[] args) {

Socket client = null;

try {

client = new Socket("127.0.0.1", 8898);

BufferedReader bufr;

bufr = new BufferedReader(new InputStreamReader(client.getInputStream()));

OutputStream os = client.getOutputStream();

while (true) {

byte[] input = new byte[1024];

System.in.read(input);

os.write(input);

os.flush();

String inputstr = new String(input);

if (inputstr.startsWith("end")) {

break;

}

}

os.close();

} catch (UnknownHostException e) {

System.out.println("出现错误 " + e.getMessage());

} catch (IOException e) {

System.out.println("出现错误 " + e.getMessage());

} finally {

try {

client.close();

System.exit(0);

} catch (IOException e) {

System.out.println("出现错误 " + e.getMessage());

}

}

}

}

## P49 P51

Import java.io.BufferedInputStream;

Public class Client {

Public static void main(String[] args) {

String str = “”;

Try {

While(str.length() < 1) {

System.out.println(“input your name:”);

Str = new DataInputStream(new BuffereedInputStream(System.in)).radLine();

}

} catch (Exception e) {

}

Send(str);

}

Public static void send(String str) {

Try {

DatagramSocket ds = new DatagramSocket();

DatagramPacket dp = new DatagramPacket(str.getBytes(),str.length(),InetAddress.getByName(“127.0.0.1”),8888);

ds.send (dp);

byte [] buf = new byte[100];

DatagramPacket revedp = new DatagramPacket(buf,100);

ds.receive(recedp);

System.out.println(New String(buf,0,recedp.getLength());

ds.close();

} catch(Exception e) {

e.printStackTrace();

}

}

Import class Server {

Public static void main(String[] args) {

Try {

DatagramSocket ds = new DatagramSocket(8888);

Byte[] buf = new byte[100];

DatagramPacket dp = new DatagramPakcet(buf, 100);

ds.receive(dp);

System.out.println(new String(buf, 0 , dp.getLength());

String str = “success!”;

DatagramPaket senddp = new DatagramPacket(str.getBytes(), str.length(), dp.getAddress, dp.getPort());

ds.send(senddp);

ds.close();

} catch (Exception e) {

}

UDP网络编程实例

• 客户端程序：

– 实例程序：

// 创建DatagramSocket对象

DatagramSocket client = new DatagramSocket();

// 准备请求数据

byte[] buf = “客户端请求数据".getBytes();

InetAddress address = InetAddress.getLocalHost();

// 创建DatagramPacket对象

DatagramPacket request =

new DatagramPacket(buf, buf.length, address , 8888);

// 发送请求

client.send(request);

UDP网络编程实例

• 服务器程序：

– 实例程序：

// 创建DatagramSocket对象，监听特定端口

DatagramSocket server = new DatagramSocket(8888);

// 准备空缓冲区

byte[] buf = new byte[1024];

// 循环等待客户端请求

while (true) {

// 创建DatagramPacket对象

DatagramPacket request = new DatagramPacket(buf, buf.length);

// 接收客户端请求

server.receive(request);

// 准备服务器端响应数据包

byte[] resBuf = "from server: ".getBytes();

DatagramPacket response = new DatagramPacket(

resBuf, resBuf.length, request.getAddress(), request.getPort());

// 发送服务器响应

server.send(response);

}

## 反射可能有简单的选择填空

Java有着一个非常突出的动态相关机制：Reflection

在运行状态中，对于任意一个类，都能够知道其所有属性和方法；对于任意一个对象，都能够调用其任意一个方法；

——这种动态获取信息以及动态调用对象的方法的功能称为java语言的反射机制

Java反射提供以下功能

1

• 在运行时判断任意一个对象所属的类

2

• 在运行时构造任意一个类的对象

3

• 在运行时判断任意一个类所具有的成员变量和方法

4

• 在运行时调用任意一个对象的方法

5

• 生成动态代理

Java反射机制的原理

• Java运行时的数据区

堆区 栈区 方法区

Java虚拟机装载某类型时，类装载 虚拟机装载某类型时，类装载

器会定位相应的 器会定位相应的class文件，然后将 文件，然后将

其读入到虚拟机中，并提取 其读入到虚拟机中，并提取class中 中

的类型信息，信息存储到方法区中。

Java反射机制中设计的类

• Class：类的实例表示正在运行的 Java 应用程序中的类和

接口

• Field：提供有关类或接口的属性的信息，以及对它的动态

访问权限

• Constructor：提供关于类的单个构造方法的信息以及对

它的访问权限

• Method：提供关于类或接口上单独某个方法的信息

Java反射机制的应用

• 操作数据库

– 动态创建SQL语句

• 解析XML

– 解析XML动态生成对象

• 动态代理

• 其它的框架中使用

– Struts框架、Spring框架、Hibernate框架

Java反射机制的缺点

• 主要的缺点是对性能有影响

– 使用反射基本上是一种解释操作，您可以告诉JVM您希望做什么

并且它满足您的要求。这类操作总是慢于直接执行相同的操作

## JDBC的概念和类型

JDBC是Java Database Connectivity的简称

是由一组Java语言编写的类和接口组成，是一种用于执行SQL语句的规范

JDBC的驱动包括四种类型：

– Jdbc-odbc bridge jabc-odbc桥驱动模式

– native-API-bridge java本地API模式

– java-middleware jdbc

– pure java driver jdbc直接数据库模式

## 驱动 JDBC中主要的类和接口 statement控制事务 P14 + try catch 运行一下

JDBC API 提供两类主要接口：

– 一是面向开发人员的java.sql程序包，使得Java程序员能够进行数

据库连接，执行SQL查询，并得到结果集合。

– 一是面向底层数据库厂商的JDBC Drivers

JDBC中主要的类和接口

• Driver接口

• DriverManager类

• Connection类

• Statement类

• PreparedStatement类

• ResultSet类

Driver接口：每个JDBC数据库驱动程序都会提供Driver接

口的具体实现

• 如果想连接数据库，必须先加载数据库厂商提供的数据库

驱动程序

• 不同类型的JDBC数据库驱动程序在编程时的加载方法也

不同。

• DriverManager是 JDBC 的管理层，管理一组JDBC驱动

程序的基本服务。

• DriverManager类的主要作用是跟踪可用的驱动程序，并

在数据库和相应驱动程序之间建立连接。

• 调用Class.forName()方法将显式地加载驱动程序类。

• 调用DriverManager.getConnection()方法将建立与数据

库的连接，得到与数据库连接的Connection对象。

• Connection类是JDBC规范中的最核心的类。

– Statement对象和ResultSet对象等都直接或者间接的来源于它

• Connection对象表示与特定数据库的连接(会话)。

– 得到Statement对象

• createStatement()

• prepareStatement(String sql)

• prepareCall(String sql)

– 为了保证数据库事务的原子性，Connection可以设置手动提交事

务

Statement是向数据库提交SQL语句并返回相应结果的工具。语

句可以是SQL查询、修改、插入或者删除

– PreparedStatement接口

• 防止SQL注入攻击(使用占位符“?”)

• 提高SQL的执行性能(在执行之前有预处理)

• 避免使用SQL方言

• 提高JDBC中有关SQL代码的可读性

– CallableStatement接口用于执行 SQL 存储过程的接口

PreparedStatement类：是Statement的子接口

• 用PreparedStatement类效率会更高。使用

PreparedStatement有很多优势，总结如下：

– 防止SQL注入攻击(使用占位符“?”)。

– 提高SQL的执行性能(在执行之前有预处理)。

– 避免使用SQL方言提高JDBC中有关SQL代码的可读性。

• PreparedStatement实例要通过Connection对象调用

prepareStatement(String sql)方法获得。

ResultSet

• ResultSet表示数据库结果集的数据表，通常通过执行查

询数据库的语句生成

• ResultSet 对象具有指向其当前数据行的光标

package com.zko1021.www;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class Demo {

public static void main(String[] args) {

try {

// 加载数据库的驱动程序

Class.forName("com.mmysq.jdbc.Driver");

// 建立数据库连接

String url = "jdbc:mysql://127.0.0.1:3306/em/useUnicode=true&characterEncoding=UTF-8";

Connection conn;

conn = DriverManager.getConnection(url, "root", "");

// 执行数据库操作SQL

Statement stmt;

stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("select \* from employees");

while (rs.next()) {

System.out.println(rs.getString("name"));

}

// 关闭数据库链接

conn.close();

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

## 考试不用封装

## DAO模

查询和插入应该会考 删除和改动也看一下

对同一个数据库表的增、删、改、查的封装——DAO模

式。

public class EmployeeDao {

ConnectionManager cm = new ConnectionManager();

public List<Employee> select() throws Exception{

List<Employee> list = new ArrayList<Employee>();

Connection conn = cm.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("select \* from employees");

while (rs.next()) {

Employee e = new Employee();

e.setName(rs.getString(“name”));//省略其他属性的封装

list.add(e);

}

conn.close();

return list;

}

//省略增（insert）、删（delete）、改（update）的数据库操作的封装方法

}

## 数据库连接池 了解

通过JDBC访问数据库，每次访问都要经历与数据库建立

连接，打开连接，访问数据库，关闭连接几个步骤，与数

据库建立并打开连接是一件既费力又费时的工作，频繁的

发生这种操作会严重耗费系统资源，导致系统性能下降，

严重的会导致系统崩溃。

• 数据库连接池就是为了解决上面的问题而出现的。数据库

连接池负责动态的分配、管理和释放数据库连接。