## 线程和线程池

C3PO数据库连接池

<http://blog.csdn.net/suzunshou/article/details/47014545>

Dbutils基础知识介绍

<https://www.cnblogs.com/CQY1183344265/p/5854418.html>

## 数据库连接池原理

连接池基本的思想是在系统初始化的时候，将数据库连接作为对象存储在内存中，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接对象。

使用完毕后，用户也并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。而连接的建立、断开都由连接池自身来管理。

同时，还可以通过设置连接池的参数来控制连接池中的初始连接数、连接的上下限数以及每个连接的最大使用次数、最大空闲时间等等。也可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

线程的三种实现方式：

1. Thread类，可extends
2. Runable接口，需要实现run方法，通过new Thread类调用
3. callable接口，需要实现call方法，通过 FutureTask 调用，并获取返回值，可结合线程池使用。

## 四类线程池

1. newCachedThreadPool

创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

这种类型的线程池特点是：

工作线程的创建数量几乎没有限制(其实也有限制的,数目为Interger. MAX\_VALUE), 这样可灵活的往线程池中添加线程。

如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为1分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。

在使用CachedThreadPool时，一定要注意控制任务的数量，否则，由于大量线程同时运行，很有会造成系统瘫痪。

2、newFixedThreadPool

创建一个指定工作线程数量的线程池。每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。

FixedThreadPool是一个典型且优秀的线程池，它具有线程池提高程序效率和节省创建线程时所耗的开销的优点。但是，在线程池空闲时，即线程池中没有可运行任务时，它不会释放工作线程，还会占用一定的系统资源。

3、newSingleThreadExecutor

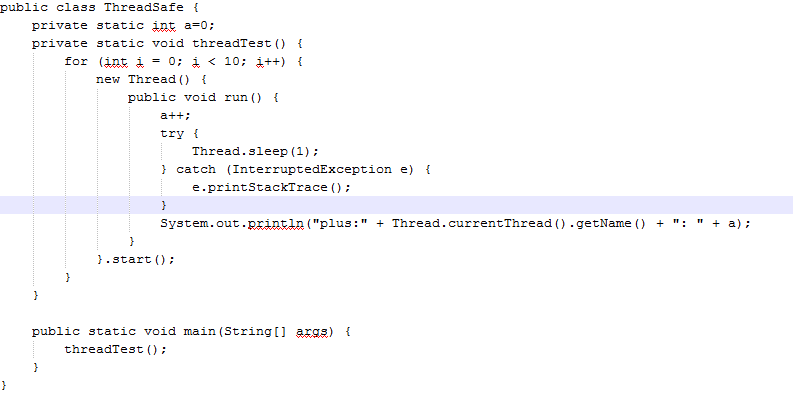
创建一个单线程化的Executor，即只创建唯一的工作者线程来执行任务，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。如果这个线程异常结束，会有另一个取代它，保证顺序执行。单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的。

4、newScheduleThreadPool

创建一个定长的线程池，而且支持定时的以及周期性的任务执行，支持定时及周期性任务执行。

## 线程安全，预防脏读

线程脏读：



输出：

plus:Thread-5: 4

plus:Thread-3: 4

plus:Thread-0: 4

plus:Thread-4: 5

plus:Thread-8: 5

plus:Thread-9: 6

plus:Thread-7: 9

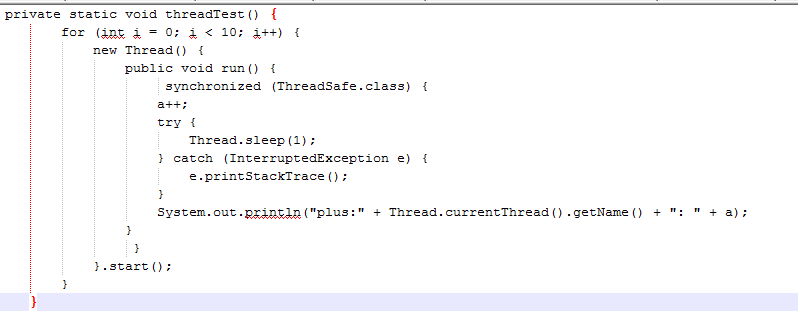
plus:Thread-2: 10

plus:Thread-6: 10

plus:Thread-1: 10

为了防止线程并发时的脏读情况，可采用以下方法避免

1. 对代码块使用synchronized



输出：

plus:Thread-3: 1

plus:Thread-7: 2

plus:Thread-0: 3

plus:Thread-6: 4

plus:Thread-9: 5

plus:Thread-8: 6

plus:Thread-2: 7

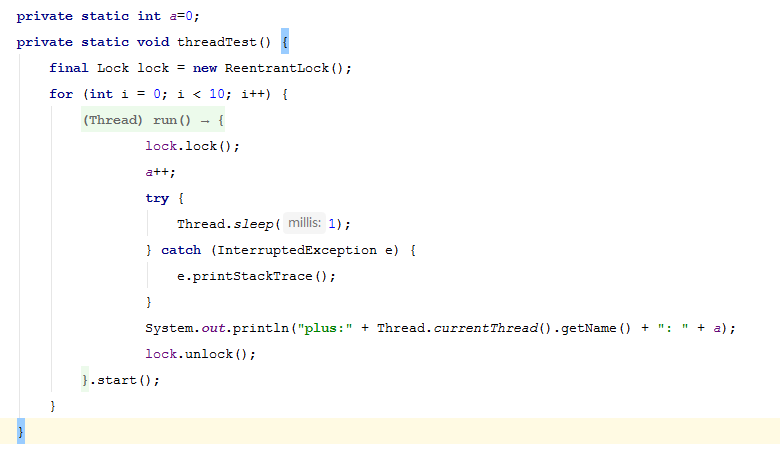
plus:Thread-5: 8

plus:Thread-4: 9

plus:Thread-1: 10

1. 使用ReentrantLock类

Lock()--Unlock()



tryLock()—unlock()



输出：

plus:Thread-0: 5

plus:Thread-6: 8

plus:Thread-1: 13

plus:Thread-2: 14

plus:Thread-4: 15

plus:Thread-8: 16

plus:Thread-5: 17

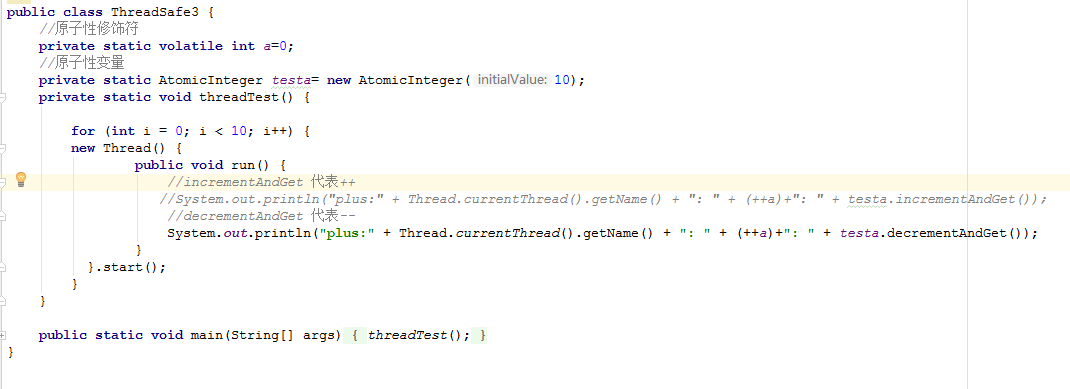
plus:Thread-9: 18

plus:Thread-3: 19

plus:Thread-7: 20

lockInterruptibly()

1. 原子变量内部锁，AtomicInteger只能同步一个值，不能同步代码块



输出：

plus:Thread-1: 1: 9

plus:Thread-0: 2: 8

plus:Thread-3: 4: 6

plus:Thread-5: 3: 7

plus:Thread-2: 5: 5

plus:Thread-8: 6: 4

plus:Thread-9: 7: 3

plus:Thread-7: 8: 2

plus:Thread-4: 9: 1

plus:Thread-6: 10: 0

## 加密算法

1. 直接使用写好的加密算法

public final static String MD5(String pwd) {

//用于加密的字符

char md5String[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9','A', 'B', 'C', 'D', 'E', 'F' };

try {

//使用平台的默认字符集将此 String 编码为 byte序列，并将结果存储到一个新的 byte数组中

byte[] btInput = pwd.getBytes();

//信息摘要是安全的单向哈希函数，它接收任意大小的数据，并输出固定长度的哈希值。

MessageDigest mdInst = MessageDigest.getInstance("MD5");

//MessageDigest对象通过使用 update方法处理数据， 使用指定的byte数组更新摘要

mdInst.update(btInput);

// 摘要更新之后，通过调用digest（）执行哈希计算，获得密文

byte[] md = mdInst.digest();

// 把密文转换成十六进制的字符串形式

int j = md.length;

char str[] = new char[j \* 2];

int k = 0;

for (int i = 0; i < j; i++) { // i = 0

byte byte0 = md[i]; //95

str[k++] = md5String[byte0 >>> 4 & 0xf]; // 5

str[k++] = md5String[byte0 & 0xf]; // F

}

//返回经过加密后的字符串

return new String(str);

} catch (Exception e) {

return null;

}

}

在main方法中调用：

String str4 = "123456";

System.out.println(encodeByMd5(str4));

//输出：4QrcOUm6Wau+VuBX8g+IPg==

1. 使用commons-codec 包下的md5

//使用commons.codec包下DigestUtils的md5Hex方法，对abc进行加密

String str1 = DigestUtils.md5Hex(str);

System.out.println(str1.toUpperCase());

输出：abc转化成大写字母：900150983CD24FB0D6963F7D28E17F72

1. 使用比MD5更安全的 SHA-1加密

String str11 = DigestUtils.sha1Hex(str);

System.out.println(str11);

输出： a9993e364706816aba3e25717850c26c9cd0d89d

1. 使用Base64的decodeBase64、encodeBase64进行加解密

byte[] b = Base64.decodeBase64("YWJj".getBytes());

System.out.println(new String(b));//输出abc

byte[] code = Base64.encodeBase64("123456".getBytes());

System.out.println(new String(code)+"--");//输出：MTIzNDU2

1. 使用安全性更高的Md5Crypt，Md5Crypt基于算子的MD5，以$开头，安全性高，每次加密的结果都不一样。

String str2 = Md5Crypt.md5Crypt("abc".getBytes());

System.out.println(str2);//abc

## 类的属性反射

前提条件：

存在两个person类，有相同的属性

1. 第一种方法：

读取person类的各个属性，及值

public static void getFile(Object object) {

Class clz=object.getClass();

Field[] fields = clz.getDeclaredFields();

for (Field field : fields) {

try {

field.setAccessible(true);

System.out.println(field.getName()+ " value "+ field.get(object));

} catch (IllegalArgumentException | IllegalAccessException e) {

e.printStackTrace();

}

}

}

将反射类属性的方法封装成copy，使用时直接传参：

public static void copy(Object from, Object to) {

Class fromClass= from.getClass();

Class toClass= to.getClass();

Field[] fields = toClass.getDeclaredFields();

for (Field field : fields) {

try {

field.setAccessible(true);

System.out.println(field.getName());

Field fromfield = fromClass.getDeclaredField(field.getName());

fromfield.setAccessible(true);

field.set(to, fromfield.get(from));

} catch (Exception e) {

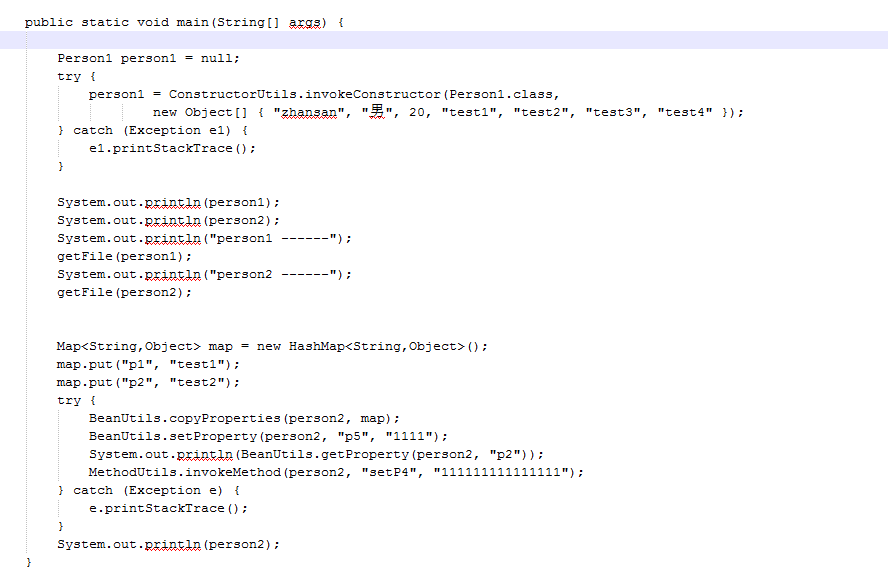
e.printStackTrace();

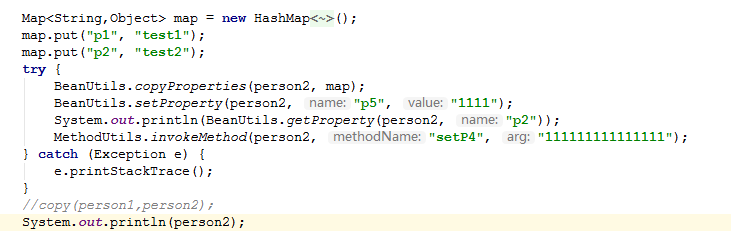
}

}

}

1. 使用commons-beanutils包下的BeanUtils工具类





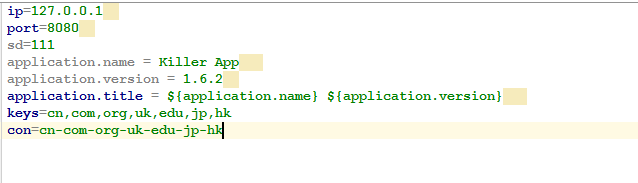
## 文件读取

1. 使用JDK自带的的类读取，Properties工具类



1. 使用commons.configuration包下的PropertiesConfiguration类读取properties文件

Properties文件内容：





*/\*\*   
 \*   
 \* 依次输出结果如下   
 \* 127.0.0.1   
 \* 8080   
 \* Killer App 1.6.2   
 \* defaultid   
 \* [cn, com, org, uk, edu, jp, hk]   
 \* [cn, com, org, uk, edu, jp, hk]   
 \* [cn, com, org, uk, edu, jp, hk]   
 \*/*

1. 使用commons.configuration包下的XMLConfiguration类读取xml文件



*/\*\* 依次输出结果如下：  
 boy: tom  
 student.name: [lily, lucy]  
 title: [abc, cbc, bbc, bbs]  
 ball[@size]: 20  
 student(1)[@id]: 2  
 student(0).name[@go]: common1  
 student.name(0)[@go]: common1  
 \*/*

1. 将对象转化成xml文件，并输出

先创建对象person, @XmlRootElement

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

public class Person1 {

private String name;

private String sex;

private int age;

private String p1;

private String p2;

private String p3;

private String p4;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getP1() {

return p1;

}

public void setP1(String p1) {

this.p1 = p1;

}

public String getP2() {

return p2;

}

public void setP2(String p2) {

this.p2 = p2;

}

public String getP3() {

return p3;

}

public void setP3(String p3) {

this.p3 = p3;

}

public String getP4() {

return p4;

}

public void setP4(String p4) {

this.p4 = p4;

}

public Person1() {

}

public Person1(String name, String sex, int age, String p1, String p2, String p3, String p4) {

super();

this.name = name;

this.sex = sex;

this.age = age;

this.p1 = p1;

this.p2 = p2;

this.p3 = p3;

this.p4 = p4;

}

@Override

public String toString() {

return "Person1 [name=" + name + ", sex=" + sex + ", age=" + age + ", p1=" + p1 + ", p2=" + p2 + ", p3=" + p3

+ ", p4=" + p4 + "]";

}

}



第5节：

1. HttpUtils 封装 httpget,httppost
2. FastJson，解析JSON