[键入公司名称]

2014

Java文档

java

Me

[键入公司地址]

[一、 java编译与查看运行时内部结构 3](#_Toc458507807)

[二、 Java基本数据 4](#_Toc458507808)

[三、 java遍历 5](#_Toc458507809)

[(一) java遍历：for、do while、while、foreach、iterator的用法和区别 5](#_Toc458507810)

[(二) java遍历集合 6](#_Toc458507811)

[四、 java泛型 8](#_Toc458507812)

[(一) 泛型类： 8](#_Toc458507813)

[(二) 泛型方法： 9](#_Toc458507814)

[五、 java类型之间的关系判断（instanceof、isAssignableFrom） 11](#_Toc458507815)

[1) instanceof 11](#_Toc458507816)

[2) isAssignableFrom 11](#_Toc458507817)

[六、 Java知识外延 12](#_Toc458507818)

[(一) java.lang.Class类 12](#_Toc458507819)

[1) getName(),getSimpleName(),getCanonicalName ()三者之间的区别 12](#_Toc458507820)

[(二) 数据库访问部分 12](#_Toc458507821)

[1) javax.persistence.EntityManager; 12](#_Toc458507822)

[2) javax.persistence.Query 13](#_Toc458507823)

[3) 关于sql语句的说明 13](#_Toc458507824)

[4) 注意事项 13](#_Toc458507825)

# java编译与查看运行时内部结构

注：TestLambda.java 文件编写TestLambda类，并且其包名为package com.test;

编译：

1. 直接编译

javac com/test/TestLambda.java

1. 指定编码

javac -encoding UTF-8 com/test/TestLambda.java

查看运行时内部结构：

1. 直接反编译查看结构

javap -p com/test/TestLambda.class

1. 输出运行时生成的内部类和方法

java -Djdk.internal.lambda.dumpProxyClasses com/test/TestLambda

# Java基本数据

# java初始化顺序

java类初始化顺序：

父类--静态变量、静态初始化块：按照定义的顺序初始化

子类--静态变量、静态初始化块：按照定义的顺序初始化

子类main方法

父类--变量、初始化块：按照定义的顺序初始化

父类--构造器

子类--变量、初始化块：按照定义的顺序初始化

子类--构造器

# java序列化与反序列化

serialVersionUID 生成规则：只有静态属性以及构造方法变化时不改变其值，其余皆改变

当使用writeObject 与 readObject时：请注意写入属性的顺序和读取属性的顺序应该一致

反序列化时：通过二进制流直接生成对象，并不调用构造方法；

生成类的方式：new、Class.forName("xxx")、反射、反序列化；

其中new、Class.forName("xxx")、反射都是调用构造器，而反序列化不是；

反序列化：不调用构造器，自身相当于构造器的另一种写法；

# java遍历

## java遍历：for、do while、while、foreach、iterator的用法和区别

相同点： 五个都可以用来遍历数组和集合

不同点：

1. 形式差别

do while的形式是

do{ xxxx }while(yyyy);

while的形式是

while(yyyy){ xxxxxx }

for的形式是

for（int i=0;i<arr.size();i++）{...}

foreach的形式是

for（int　i：arr）{...}

iterator的形式是

Iterator it = arr.iterator();

while(it.hasNext()){ object o =it.next(); ...}

1. 条件差别

for需要知道集合或数组的大小，而且需要是有序的，不然无法遍历；

foreach和iterator都不需要知道集合或数组的大小，他们都是得到集合内的每个元素然后进行处理；

1. 多态差别

for和foreach都需要先知道集合的类型，甚至是集合内元素的类型，即需要访问内部的成员，不能实现态；

iterator是一个接口类型，他不关心集合或者数组的类型，而且他还能随时修改和删除集合的元素，举个例子：

public void display（Iterator<object> it）{

while(it.hasNext()){

system.out.print(it.next()+"");

}

}

当我们需要遍历不同的集合时，我们只需要传递集合的iterator（如arr.iterator()）看懂了吧，这就 是iterator的好处，他不包含任何有关他所遍历的序列的类型信息，能够将遍历序列的操作与序列底层的结构分离。迭代器统一了对容器的访问方式。这也是接口的解耦的最好体现。

1. 用法差别

for循环一般用来处理比较简单的有序的，可预知大小的集合或数组

foreach可用于遍历任何集合或数组，而且操作简单易懂，他唯一的不好就是需要了解集合内部类型

iterator是最强大的，他可以随时修改或者删除集合内部的元素，并且是在不需要知道元素和集合的类型的情况下进行的（原因可参考第三点：多态差别），当你需要对不同的容器实现同样的遍历方式时，迭代器是最好的选择！

注：

for，while是一样的，形式上的不同

foreach，Iterator也是一样的，形式上的不同

性能肯定是用迭代器的好一些。迭代器能直接访问集合内部元素。而且有的集合没有索引，没法用for来遍历，只有用迭代器。

## java遍历集合

1. 对set的遍历

迭代遍历

Set<String> set = new HashSet<String>();

Iterator<String> it = set.iterator();

while (it.hasNext()) {

String str = it.next();

System.out.println(str);

}

for循环遍历：

for (String str : set) {

System.out.println(str);

}

优点还体现在泛型 假如 set中存放的是Object

for循环遍历：

Set<Object> set = new HashSet<Object>();

for (Object obj: set) {

if(obj instanceof Integer){

int aa= (Integer)obj;

}else if(obj instanceof String){

String aa = (String)obj

}

........

}

1. 对map的遍历

迭代遍历

Set set = map.keySet();

Iterator it = set.iterator();

while(it.hasNext()){

String str = it.next().toString();

}

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

while (it.hasNext())

{

Map.Entry pairs = (Map.Entry)it.next();

System.out.println(pairs.getKey() + " = " + pairs.getValue());

}

for循环遍历：

Map<String,String> paraMap=new HashMap<>();

for(String dataKey : paraMap.keySet())

{

System.out.println(dataKey );

}

这里要注意的是,paraMap是怎么样定义的,如果是简单的Map paraMap = new HashMap();那前面的String就只能换成Object了.

对整Map的key和value都进行循环,如下:

Java代码

复制代码 代码如下:

for(Map.Entry<String, Object> entry : paraMap.entrySet())

{

System.out.println(entry.getKey()+": "+entry.getValue());

}

例子：

Map<String,String> map = new HashMap<String,String>(){};

map.put("a", "A");

map.put("b", "B");

map.put("c", "C");

//第一种

Set set = map.keySet();

Iterator it = set.iterator();

while(it.hasNext()){

    String str = it.next().toString();

}

//第二种

Set<Entry<String,String>> entryset = map.entrySet();

Iterator iter = entryset.iterator();

while(iter.hasNext()){

    Entry<String,String> entry = (Entry<String,String>)iter.next();

}

//第三种

for(Entry<String,String> entry : map.entrySet()){

    String strkey = entry.getKey();

    String strval = entry.getValue();

}

首先，使用entryset比使用keyset的效率要高。所以建议用后2种方法。

   第二和第三种的区别其实就是使用迭代器iteratior和使用一般的for语句的区别，和map本身倒没什么关系。最简单的判断方法，就是对要迭代的内容（本例是个map,如果是一个list同理）,在循环过程中，你需要进行add或者remove操作不。for语句的循环是不能做这样的操作的，只能使用迭代器。

   不过，个人认为，for语句虽然看上去代码少一点，但敲起来没iterator有感觉

# java泛型

## 泛型类：

**class** classType<T **extends** Number,V **extends** Object>

{

T t;

V v;

}

**class** classType2<T ,V **extends** Object>

{

T t;

V v;

}

**class** classType3<T **extends** Number,V >

{

T t;

V v;

}

## 泛型方法：

❶：

**public**<T> T getObject(Class<T>tc) **throws** InstantiationException,IllegalAccessException

{

T t= tc.newInstance();

// try {

// T t2= tc.newInstance();

// } catch (InstantiationException | IllegalAccessException e) {

// // **TODO** Auto-generated catch block

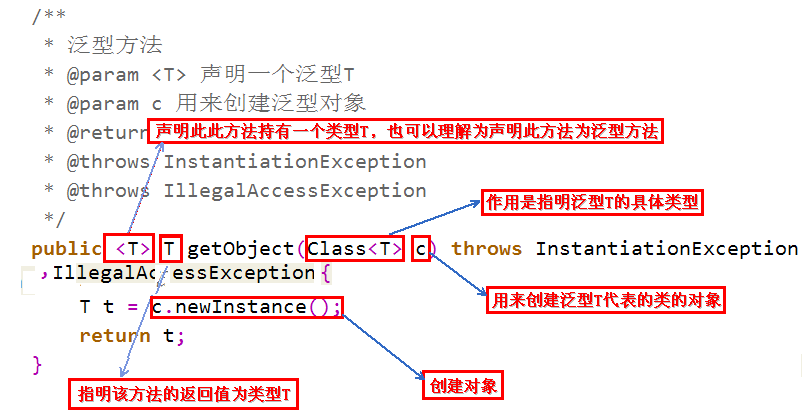
// e.printStackTrace();

// }

**return**t;

}

* 1. <T>：声明此方法有一个类型T，也可以理解为声明此方法为泛型方法
  2. Class<T>：作用是指明泛型T的具体类型，传值方法：getObject(xxxx.class);
  3. tc：用来创建泛型T代表的类的对象
  4. tc.newInstance()：创建泛型对象



❷：

**public**<T> Set<?> getT(Tt)

{

**returnnull**;

}

有通配符修饰的变量，可以遍历，不能修改，比如List<?> list,该list变量就只能够用于遍历；

# java类型之间的关系判断（instanceof、isAssignableFrom）

### instanceof

用法：

实例 instanceof 类名/接口名

作用：

判断该实例是否是（类名/接口名）或则其子类的实例；

另一个写法：

类名/接口名.class.isInstance(实例)

例：

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

System.out.println(list.getClass().isInstance(new ArrayList<String>()));

System.out.println(list instanceof ArrayList);

System.out.println(ArrayList.class.isInstance(list));

System.out.println(list.getClass().isAssignableFrom(ArrayList.class));

System.out.println(ArrayList.class.isAssignableFrom(list.getClass()));

### isAssignableFrom

用法：

(类名1/接口名1).class.isAssignableFrom(类名2/接口名2)

作用：

判断（类名1/接口名1）是否是（类名2/接口名2）的父类或其自身

注：

instanceof：子-->父

isAssignableFrom：父-->子

# Java知识外延

## java.lang.Class类

### getName(),getSimpleName(),getCanonicalName ()三者之间的区别

|  |  |  |
| --- | --- | --- |
| 函数 | 非数组 | 数组 |
| getName() | 包.类名称 | [L包.类名称 |
| getSimpleName() | 类名称 | 类名称[] |
| getCanonicalName() | 包.类名称 | 包.类名称[] |

## 数据库访问部分

### javax.persistence.EntityManager;

String sql="select \* from picc\_gift\_info r where r.registerId=?registerId ";

javax.persistence.Query query = new javax.persistence.EntityManager().createNativeQuery(sql, GiftReceivedVo.class);

query.setParameter("registerId", registerId);

query.setFirstResult(start);

query.setMaxResults(limit);

List<GiftReceivedVo> list = (List<GiftReceivedVo>) query.getResultList();

1. 持久化实体persist()：

new EntityManager().persist(new GiftReceivedVo().setId(“100”))

1. 按主键查询find()：

GiftReceivedVo temp=new EntityManager().find(GiftReceivedVo.class,”100”);

1. 删除实体remove()：

new EntityManager().remove(new EntityManager().find(GiftReceivedVo.class,”100”));

1. 更新实体merge()：更新至数据库(处不处于管理模式都会更新！)

GiftReceivedVo temp=new EntityManager().find(GiftReceivedVo.class,”100”);

temp.setName(“大王哟”);

new EntityManager().merge(temp);

1. 更新实体flush()：更新至数据库（只更新处于管理模式的bean）

GiftReceivedVo temp=new EntityManager().find(GiftReceivedVo.class,”100”);

temp.setName(“大王哟”);

new EntityManager().flush (temp);

1. 从数据库中更新refresh()：与数据库同步

GiftReceivedVo temp=new EntityManager().find(GiftReceivedVo.class,”100”);

new EntityManager().refresh (temp);

1. 检测实体是否被管理contains()：好比flush()与merge()

GiftReceivedVo temp=new EntityManager().find(GiftReceivedVo.class,”100”);

Boolean b=new EntityManager().contains (temp);

b==true:处于管理模式；

b==false:未处于管理模式；

1. 分离管理的实体clear():

当处理了大量的实体后，这些实体都会处于实体管理器中，将会消耗大量的内存，使得程序运行变慢；如果要减少消耗，则可以使用clear()方法，将正在被管理的的实体从持久化内容中分离出来

new EntityManager().clear();

如果调用clear()方法，则之前所有对实体所做的任何改变都会被丢失，所以在调用clear()方法之前先调用flush()方法保存更改！

### javax.persistence.Query

关于占位符的问题：

* 如果是Query qry = new EntityManager().createNativeQuery(sql,XXXXX.class)则占位符一律是”?”;
* 如果是Query qry = new EntityManager().createQuery(sql,XXXXX.class)则占位符如果是数字，则为 “?1”；如果是字符串，则为”:XXXXXX”;

关于更新的问题：

* 如果是insert into 或者 update 则为：Query q = this.em.createNativeQuery(sql )；q.executeUpdate();

### 关于sql语句的说明

如果使用原生sql，则sql拼接语句中尽量不要出现\n等java类的转义字符，可能会出错！！！

### 注意事项

关于模糊查询：

Sql:like ?gift\_Name

query.setParameter("gift\_Name", "%"+map\_action.get("giftNameSearch").toString().trim()+"%");

关于in查询：

Sql:in (t1,t2……)

Java:使用循环赋值;