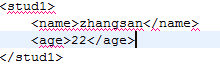
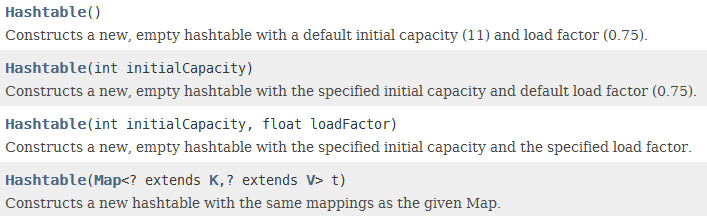
Java中的Properties集合

1. **HashTable 集合（哈希表）**：属于Map接口的一个子类，存在于**java.util包**中，继承于**java.Dictionary集合**中，有两个直接子类：Properties和UIDefaults。
2. **配置文件：**

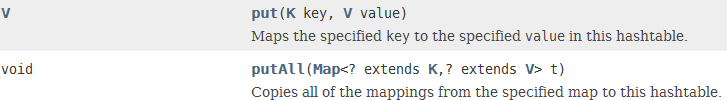
**简单的信息利用Properties存储，一个键一个值，即键值对；**

**复杂的信息利用XML** 存储 ：xml利用**标签**的形式进行存储信息，如。

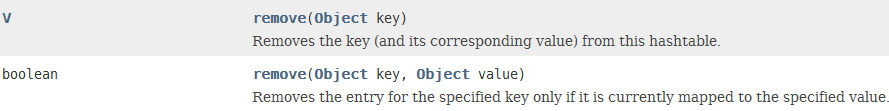
1. **HashTable集合：**
2. **构造方法：**



1. **重要方法：**
2. **添加：put和putAll**



1. **删除remove和清空clear**





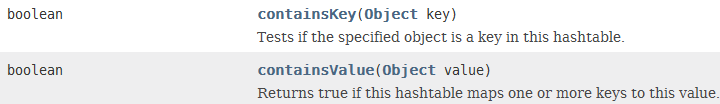
1. **获取：keySet、values、entrySet**







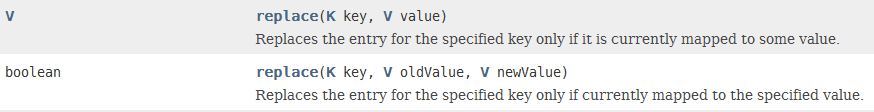
1. contains判断：



1. size方法：



1. replace方法：



1. **Properties集合** ：是**HashTable集合**的一个直接子类，可以与**流**进行关联。通常该集合用于操作以**键值对形式**存在的**配置文件（manifest）**。配置文件：就是存储配置信息的文件，如XML文件、数据库等。
2. **Properties集合**的特点：
3. 该集合的键和值都是字符串类型的，所以就没有必要指定**泛型**了；
4. 集合中的数据可以保存到**流**中，或者**从流中**获取；
5. Properties集合：
6. 构造方法：
7. **Properties（）**：



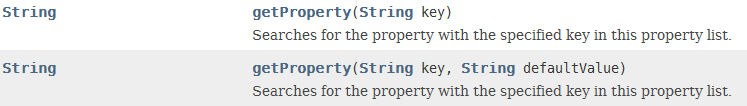
1. **Properties（Properties defaults）：**



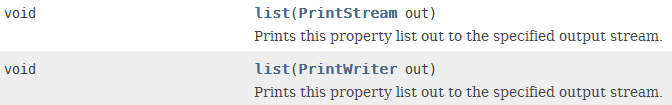
1. 特有方法：
2. **setProperties方法**：添加键值对，等同于HashTable的put方法。



1. **getProperties()：获取值。**

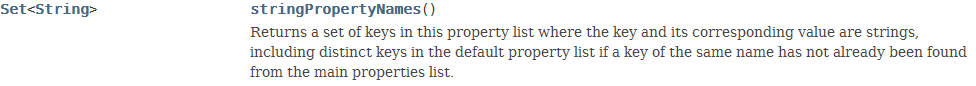


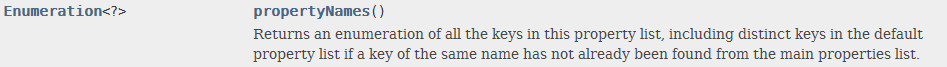
1. **list方法**：把集合中**所有的键值对**列举到**打印输出流**中，主要用于调试所用。

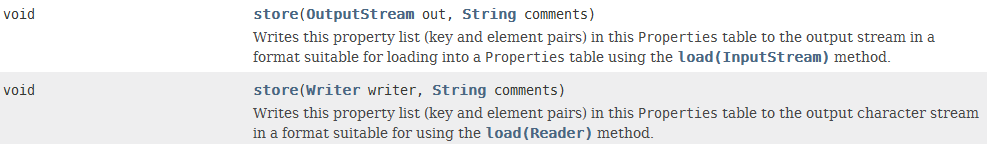


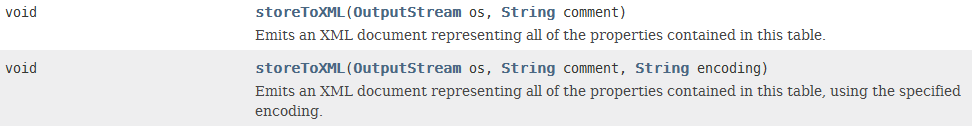
1. **stringPropertyNames（）：返回key的Set集合。**

**propertyNames方法：返回key的枚举，已经很少使用了。**

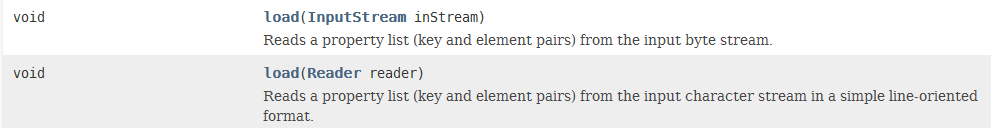


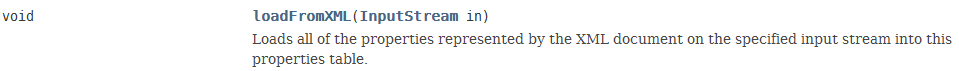


1. **store方法：（存储方法）**使Properties集合与字节输出流或者字符输出流关联，**把Properties集合中的键值对通过写入流写到文件中，从而实现持久化**。 



1. **load方法**：（加载方法）使Properties集合与**文件读取流**关联，把文件中的**键值对**信息加载到**Properties集合**中。与**store方法**相反。





1. 其他的方法都是继承于**父类的**。
2. **遍历Properties集合**：利用Properties集合中的**stringPropertyNames（）方法遍历Properties集合中的键值对。**

**例子： Set<String> set = prop.stringPropertyNames();//获取key的Set集合**

for(Iterator<String> ir = set.iterator();ir.hasNext();) {

String key = ir.next();

System.out.println( key+ " "+prop.getProperty(key));

}

1. System类中有个静态方法：getProperties（），返回一个Properties集合。此方法会把**系统的基本信息**列举到**Properties**中。



1. **list方法演示**：

**System.getProperties()** 返回系统的所有的配置信息，返回类型为Properties集合，在调用list方法就可打印到控制台（System.out）上。

Properties prop = **new** Properties();

prop = System.*getProperties*();

prop.list(System.***out***);

1. **store方法演示**：

例子：把系统的基本信息输出到e:\\systemInfo.properties文件中。

File file = new File("e:\\systemInfo.properties");

FileWriter fw = new FileWriter(file); **//如果文件不存在，写入流会自动创建**

Properties prop = new Properties();

prop = System.getProperties();

**prop.store(fw,"System Information : ");**

fw.close();

1. **load方法演示**：

File file = new File("e:\\systemInfo.properties");

FileReader fr = new FileReader(file);

Properties prop = new Properties();

prop.load(fr);

Set<String> set = prop.stringPropertyNames();

for(Iterator<String> ir = set.iterator();ir.hasNext();) {

String key = ir.next();

System.out.println( key+ " "+prop.getProperty(key));

}

1. java中的配置文件扩展名一般为**.properties** ；windows系统的配置文件一般的扩展名为**.ini** 。**配置文件的作用**：可以记录上次使用软件的一些基本设置信息，使下次启动程序时，仍可以按照上次设置好的运行。
2. **FileWriter fw1 = new FileWriter(file);** 是创建一个名字与file同名的空文件，也就是把file覆盖掉。**FileWriter fw1 = new FileWriter(file，true);** 是在原来file文件基础上进行添加，所以原来的file文件并没有改变，也没有创建新的文件。
3. Load方法的实现原理：

private static void myLoad(Properties prop) throws IOException {

File file = new File("e:\\manifest.properties");

BufferedReader bufr = new BufferedReader(new FileReader(file));

String line = null;

while((line = bufr.readLine())!= null) {

**if(line.startsWith("#"))//properties中的注释以#开头**

continue;

**String[] str2 = line.split("=");**

prop.setProperty(str2[0],str2[1]);

}

bufr.close();

}

1. **对已有的配置文件的信息进行修改：**

**思路：**读取这个配置文件，然后利用**load方法**将这个文件中的**键值对**信息加载到集合中，再通过Properties**集合**的setProperty方法对数据进行修改，最后通过**store方法**将**修改过的数据**存储到**配置文件**中。

**代码：**

**Properties prop = new Properties();**

**File file = new File("e:\\Info.properties");**

**if(!file.exists()) {**

**file.createNewFile();**

**}**

**FileReader fr = new FileReader(file);**

**prop.load(fr);**

**FileWriter fw = new FileWriter(file);//一定要放在load方法之后**

因为 FileWriter fw1 = new FileWriter(file);创建一个输出流，相当于创建了一个新的文件，又因为名字一样，所以这样把原来的文件覆盖了，使原有的数据都消失了，变成了一个新的空的文件。

再次调用load方法就会导致，集合Properties中也为空，后面的修改语句变成了添加语句了。new FileWriter(file，true);不会创建新的文件，但是只是添加了，会导致键值对的重复。

**prop.list(System.out);//修改前的键值对**

**prop.setProperty("bbc","UUUUxiugai");**

**prop.store(fw,"info");**

**fr.close();**

**fw.close();**

**prop.list(System.out);//调试所用，修改后的键值对**

1. 定义功能，获取一个应用程序的运行次数，如果**超过5次**，给出 “使用次数已到，请注册”的提示，并不要运行程序。

分析：

**思路**：① 应该有个计数器，每次程序运行一次需要计数一次，并且是在原有的基础上加一次。

② 计数器就是一个变量：程序启动时候，进行计数，计数器必须存在于内存并进行运算。可是运行一结束，计数器就消失了。再次启动该程序，计数器就会重新被初始化了。而我们需要的是多次启动同一个应用程序，使用的是同一个计数器，这就需要计数器的生命周期变长，从**内存**存储到硬盘文件中。

③如何使用这个计数器呢？

首先，程序启动时，应该先读取这个用于记录计数器信息的配置文件，获取上一次计数器的次数，并进行运行次数的判断；其次，计数器自增，并将自增后的次数重新存储到**配置文件**中。

④文件中的信息该如何进行存储并体现呢？

直接存储值可以，**但是含义不明确**，所以很有必要对每个值取个名字。这就有了键和值的对应，可以使用**键值对**。此映射关系可以有**Map集合**来搞定，但是同时又需要读取硬盘上的数据，此时**Map + IO = Properties集合。**

代码： Properties prop = new Properties();

File file = new File("e:\\mani.properties");

if(!file.exists())

file.createNewFile();

FileReader fr = new FileReader(file);

prop.load(fr);

int num = 0;

String count = prop.getProperty("count");

if(count == null) {

num = 0;

System.out.println("这是第1次使用此软件。");

}

else {

num = Integer.parseInt(count);

if(num >=5 ) {

throw new RuntimeException("已经使用过5次，请购买此软件。");

}

else

System.out.println("这是第"+(num+1)+"次使用此软件。");

}

count = Integer.toString(num+1);

prop.setProperty("count",count);

FileWriter fw = new FileWriter(file);

prop.store(fw,"info");

fr.close();fw.close();