CharSequence

类型：接口

* 方法

int length()

返回字符序列的长度，即字符序列中字符的个数

char charAt(int index)

返回字符序列中指定索引出的字符；如果index小于零或者index不比字符序列的长度小则抛出IndexOutOfBoundsException异常

CharSequence subSequence(int start, int end)

返回给定字符序列的子序列，从start开始，到end结束；包含start但是不不包含end处的char；会抛出索引越界异常

String toString()

返回字符序列的字符串；字符序列就是字符串

？？？？？？？？？？？？？？？？？？？？？

public default IntStream chars()

jdk 1.8新增

public default IntStream chars() {  
 class CharIterator implements PrimitiveIterator.OfInt {  
 int cur = 0;  
  
 public boolean hasNext() {  
 return cur < length();  
 }  
  
 public int nextInt() {  
 if (hasNext()) {  
 return charAt(cur++);  
 } else {  
 throw new NoSuchElementException();  
 }  
 }  
  
 @Override  
 public void forEachRemaining(IntConsumer block) {  
 for (; cur < length(); cur++) {  
 block.accept(charAt(cur));  
 }  
 }  
 }  
  
 return StreamSupport.intStream(() ->  
 Spliterators.spliterator(  
 new CharIterator(),  
 length(),  
 Spliterator.ORDERED),  
 Spliterator.SUBSIZED | Spliterator.SIZED | Spliterator.ORDERED,  
 false);  
}

？？？？？？？？？？？？？？？？？？

public default IntStream codePoints()

jdk 1.8新增

public default IntStream codePoints() {  
 class CodePointIterator implements PrimitiveIterator.OfInt {  
 int cur = 0;  
  
 @Override  
 public void forEachRemaining(IntConsumer block) {  
 final int length = length();  
 int i = cur;  
 try {  
 while (i < length) {  
 char c1 = charAt(i++);  
 if (!Character.isHighSurrogate(c1) || i >= length) {  
 block.accept(c1);  
 } else {  
 char c2 = charAt(i);  
 if (Character.isLowSurrogate(c2)) {  
 i++;  
 block.accept(Character.toCodePoint(c1, c2));  
 } else {  
 block.accept(c1);  
 }  
 }  
 }  
 } finally {  
 cur = i;  
 }  
 }  
  
 public boolean hasNext() {  
 return cur < length();  
 }  
  
 public int nextInt() {  
 final int length = length();  
  
 if (cur >= length) {  
 throw new NoSuchElementException();  
 }  
 char c1 = charAt(cur++);  
 if (Character.isHighSurrogate(c1) && cur < length) {  
 char c2 = charAt(cur);  
 if (Character.isLowSurrogate(c2)) {  
 cur++;  
 return Character.toCodePoint(c1, c2);  
 }  
 }  
 return c1;  
 }  
 }  
  
 return StreamSupport.intStream(() ->  
 Spliterators.spliteratorUnknownSize(  
 new CodePointIterator(),  
 Spliterator.ORDERED),  
 Spliterator.ORDERED,  
 false);  
}