1. Какие “строковые” классы вы знаете?  
   2. Какие основные свойства “строковых” классов (их особенности)?  
   3. Можно ли наследовать строковый тип, почему?  
   4. Дайте определение понятию конкатенация строк.  
   5. Как преобразовать строку в число?  
   6. Как сравнить значение двух строк?  
   7. Как перевернуть строку?  
   8. Как работает сравнение двух строк?  
   9. Как обрезать пробелы в конце строки?  
   10. Как заменить символ в строке?  
   11. Как получить часть строки?  
   12. Дайте определение понятию “пул строк”.  
   13. Какой метод позволяет выделить подстроку в строке?  
   14. Как разбить строку на подстроки по заданному разделителю?  
   15. Какой метод вызывается для преобразования переменной в строку?  
   16. Как узнать значение конкретного символа строки, зная его порядковый номер в строке?  
   17. Как найти необходимый символ в строке?  
   18. Можно ли синхронизировать доступ к строке?  
   19. Что делает метод intern()?  
   20. Чем отличаются и что общего у классов String, StringBuffer и StringBuilder?  
   21. Как правильно сравнить значения строк двух различных объектов типа String и StringBuffer?  
   22. Почему строка неизменная и финализированная в Java?  
   23. Почему массив символов предпочтительнее строки для хранения пароля?  
   24. Почему строка является популярным ключом в HashMap в Java?  
   25. Напишите метод удаления данного символа из строки.

Strings. Ответы

**1. Какие “строковые” классы вы знаете?**

* public final class **String**implements java.io.Serializable, Comparable<String>, CharSequence
* public final class **StringBuffer**extends AbstractStringBuilder implements java.io.Serializable, CharSequence
* public final class **StringBuilder**extends AbstractStringBuilder implements java.io.Serializable, CharSequence

**2. Какие основные свойства “строковых” классов (их особенности)?**

Все строковые классы — final (следовательно от них нельзя унаследоваться).

**String**.  
Строка — объект, что представляет последовательность символов(массив символов). Для создания и манипулирования строками Java платформа предоставляет общедоступный финальный (не может иметь подклассов) класс **java.lang.String**. Данный класс является неизменяемым (*immutable*) — созданный объект класса **String** не может быть изменен.

**StringBuffer**  
Строки являются неизменными, поэтому частая их модификация приводит к созданию новых объектов, что в свою очередь расходует драгоценную память. Для решения этой проблемы был создан класс **java.lang.StringBuffer**, который позволяет более эффективно работать над модификацией строки. Класс является *mutable*, то есть изменяемым — используйте его, если Вы хотите изменять содержимое строки. **StringBuffer** может быть использован в многопоточных средах, так как все необходимые методы являются синхронизированными(медленный).

**StringBuffer** благодаря synchronized он защищён от конфликта потоков: пока поток захвативший монитор объекта не закончит свою работу, все остальные потоки будут находиться в режиме ожидания(wait). **StringBuffer** переводит полученный объект(наприм. строку) в свой буфер, т.е. массив символов. Этот массив становится изменяемым и после завершения работы буфера можно снова перевести в строку(toString()).

**StringBuilder**  
**StringBuilder** — класс, что представляет изменяемую последовательность символов. Класс был введен в Java 5 и имеет полностью идентичный API с **StringBuffer**. Единственное отличие — **StringBuilder** не синхронизирован. Это означает, что его использование в многопоточных средах нежелательно. Следовательно, если вы работаете с многопоточностью, Вам идеально подходит **StringBuffer**, иначе используйте **StringBuilder**, который работает намного быстрее в большинстве реализаций.

Обработка строк в Java. Часть I: String, StringBuffer, StringBuilder: http://habrahabr.ru/post/260767/

**3. Можно ли наследовать строковый тип, почему?**

Классы объявлены final, поэтому наследоваться не получится.

**4. Дайте определение понятию конкатенация строк.**

Конкатенация — операция объединения строк, что возвращает новую строку, что является результатом объединения второй строки с окончанием первой. Операции конкатенации могут быть выполнены так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | StringBufferstringBuffer = new StringBuffer();  StringBuilderstringBuilder = new StringBuilder();    String str = "ABC";  str += "DEF";    String str2 = "one".concat("two").concat("three");    stringBuffer.append("DDD").append("EEE");  stringBuilder.append("FFF").append("GGG");  System.out.println(str + " " +str2 + " " + stringBuffer.toString() + " " + stringBuilder.toString());//ABCDEF onetwothree DDDEEE FFFGGG |

Сравнение производительности конкатенации строк:  
Оператор ‘+=’ >**92.243 с**;  
String.concat() > **1.254 с**;  
StringBuffer> **1.208 с**;  
StringBuilder> **1.121 с**.

Конкатенация и настройки JVM: http://microfork.com/string-concatenation-java/

**5. Как преобразовать строку в число?**

У каждой обертки для примитивов есть свой метод valueOf(String s), который возвращает преобразованное численное значение из строки. При этом форматы строки и принимаемого типа должны совпадать(целочисленные с целочисленными…).

В большинстве случаев эффективнее использовать parse-методы классов-оболочек(parseInt(),parseDouble()), т.к. данные методы возвращают полученные значения в виде примитивов, которые при использовании будут попадать в Пул примитивов, что избавляет от создания лишних копий.

valueOf в свою очередь каждый раз будет создавать новую копию(дубликат):

public static Double valueOf(String s) throws NumberFormatException {  
 return new Double(*parseDouble*(s));  
}

т.о. valueOf внутри всё равно использует parse…

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | String x = "523.5";  Double xd = Double.valueOf(x);  /\*  Integer xy = Integer.valueOf(x); //java.lang.NumberFormatException: For input string: "523.5"  \*/  System.out.println(xd); //523.5 |

**6. Как сравнить значение двух строк?**

Оператор == работает с ссылками объекта String. Если две переменные String указывают на один и тот же объект в памяти, сравнение вернет результат true. В противном случае результат будет false, несмотря на то что текст может содержать в точности такие же символы. Оператор == не сравнивает сами данные типа char. Для сравнения посимвольно необходимо использовать метод equals();

*[подробнее о работе «equals()» и «==», смотри документ: \_Core\_part\_3\_\_javastudy\_ru]*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | String s1 = new String("ABC");          String s2 = new String("ABC");          String s3 = "ABC";          String s4 = "ABC";          System.out.println(s1==s2); //false          System.out.println(s3==s4); //true. Т.к. один набор литералов будет указывать на одну область памяти          System.out.println(s1.equals(s2));//true, !!!! т.к. в реализации equals() сначала идёт проверка на «==»!!!!!            s1=s2;          System.out.println(s1==s2); //true          if("someString" == "someString") { //true              System.out.println("true");          }          System.out.println(s1.compareTo(s2)); //0          System.out.println("C".compareTo("A")); //2          System.out.println("A".compareTo("C")); //-2 |

public boolean equals(Object anObject) {  
 if (this == anObject) {  
 return true;  
 }  
 return (anObject instanceof String aString)  
 && (!*COMPACT\_STRINGS* || this.coder == aString.coder)  
 && StringLatin1.*equals*(value, aString.value);  
}

**7. Как перевернуть строку?**

|  |  |
| --- | --- |
| 1  2  3  4 | String s = "ABCDEFG";  StringBuilderstringBuilder = new StringBuilder(s);  stringBuilder.reverse();  System.out.println(stringBuilder.toString()); //GFEDCBA |

Можно и алгоритмом переставляя каждый char, но это на ваше усмотрение:).

**8. Как работает сравнение двух строк?**

Строка в Java — это отдельный объект, который может не совпадать с другим объектом, хотя на экране результат выводимой строки может выглядеть одинаково. Просто Java в случае с логическим оператором == (а также !=) сравнивает ссылки на объекты.

Метод equals сравнивает посимвольно на эквивалентность. (см. ответ 6)

**9. Как обрезать пробелы в конце строки?**

|  |  |
| --- | --- |
| 1  2  3 | String s = "a    ";  System.out.println(s.trim() + "b");//ab  System.out.println(s + "b");//a    b |

trim() – не статический, поэтому требуется проверка на NULL.

**10. Как заменить символ в строке?**

Можно использовать метод replace(Char Sequencetarget, Char Sequencereplacement), который меняет символы в строке(заменяет все совпадения в строке до её конца!!!!). Можно преобразовать в массив символов и заменить символ там. Можно использовать StringBuilder и метод setCharAt(int index, char ch)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | String sb = "AABAA";  String s = "ABCDEF".replace("C", "\*\*");    String sb2 = sb.replace(sb, "##");  System.out.println(s + " " + sb2); //AB\*\*DEF and ##    String fs = "123456789";  char[] charSequence = fs.toCharArray();  charSequence[3] = ' |

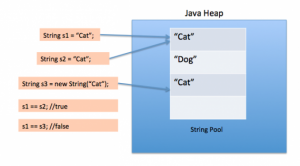
**11. Как получить часть строки?**

Методы substring(int beginIndex, int lastIndex) и substring(int beginIndex) — возвращает часть строки по указанным индексам.

|  |  |
| --- | --- |
| 1  2  3  4 | String fs = "123456789";  String sub = fs.subSequence(3,6).toString();  String sub2 = fs.substring(3,6);  System.out.println(sub2); //456 |

**12. Дайте определение понятию “пул строк”.**

**Пул строк** – это набор строк, который хранится в памяти Javaheap. Мы знаем, что String это специальный класс в Java, и мы можем создавать объекты этого класса, используя оператор new точно так же, как и создавать объекты, предоставляя значение строки в двойных кавычках.  
Диаграмма ниже объясняет, как пул строк размещается в памяти Javaheap и что происходит, когда мы используем различные способы создания строк.

[](https://javastudy.ru/wp-content/uploads/2016/01/poolStrings.png)

Пул строк возможен исключительно благодаря неизменяемости строк в Java и реализации идеи интернирования строк. Пул строк также является примером паттерна Приспособленец (Flyweight).  
Пул строк помогает экономить большой объем памяти, но с другой стороны создание строки занимает больше времени.  
Когда мы используем двойные кавычки для создания строки, сначала ищется строка в пуле с таким же значением, если находится, то просто возвращается ссылка, иначе создается новая строка в пуле, а затем возвращается ссылка.  
Тем не менее, когда мы используем оператор new, мы принуждаем класс String создать новый объект строки, а затем мы можем использовать метод intern() для того, чтобы поместить строку в пул, или получить из пула ссылку на другой объект String с таким же значением.

Ниже приведен пример, показывающий работу пула строк.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | String s1 = "Cat";  String s2 = "Cat";  String s3 = new String("Cat");    System.out.println("s1 == s2 :"+(s1==s2)); //s1 == s2 :true  System.out.println("s1 == s3 :"+(s1==s3)); //s1 == s3 :false |

**13. Какой метод позволяет выделить подстроку в строке?**

В дополнении к «как получить часть строки» можно использовать метод string.indexOf(char c), который вернет индекс первого вхождения символа в строке. Таким образом потом можно использовать этот номер для выделения подстроки с помощью substring();

indexOf() перегружен и способен принимать подстроку, которую нужно найти в данной строке. Если подстрока найдена, то возвращается индекс первого элемента подстроки.

String source = "ВасяПупкин";

String sub = "ася";

int index = source.indexOf(sub);//вернёт индекс буквы «а»

* Если необходимо получить все символы после «я»(не включая этот символ), то:
  + String result = source.substring(index + sub.length() + 1);
    - // если включая «я», то: source.substring(index + sub.length()) ;
* С НАЧАЛА строки ДО «я»:
  + String result = source.substring(0, index + sub.length()); //
    - *Params:*
      * *beginIndex – the beginning index, inclusive.*
      * *endIndex – the ending index, exclusive.*
  + String result = source.substring(0, index + sub.length() - 1); // включая «я»

indexOf() может вернуть «-1», в случае, если символ либо подстрока не найдены. => во избежание исключения перед вызовом substring с использованием результата работы метода indexOf(), необходимо проверить результат на «-1».

Варианты выбрасывания исключения:

*Throws:*

*IndexOutOfBoundsException – if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex.*

**14. Как разбить строку на подстроки по заданному разделителю?**

Мы можем использовать метод split(String regex) для разделения строки на массив символов, используя в качестве разделителя регулярное выражение. Метод split(String regex, int numOfStrings) является перегруженным методом для разделения строки на заданное количество строк. Мы можем использовать обратную черту для использования специальных символов регулярных выражений в качестве обычных символов.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | String line = "I am a java developer";  String[] words = line.split(" ");  String[] twoWords = line.split(" ", 2);// 2 – лимит слов    System.out.println("String split with delimiter: "+Arrays.toString(words));//String split with delimiter: [I, am, a, java, developer]  System.out.println("String split into two: "+Arrays.toString(twoWords));//String split into two: [I, am a java developer]    //split string delimited with special characters    String wordsBySpecialSymbols = "I|am|a|java|developer";  String[] words = wordsBySpecialSymbols.split("\\|");    System.out.println("String split with special character: "+ Arrays.toString(words));//String split with special character: [I, am, a, java, developer] |

Альтернативные варианты:

String source = "I|am|a|java|developer";

* String[] result = Pattern.compile("\\|").split(source);//вернёт массив строк
  + String resultAsString = Arrays.toString(result);//преобразование массива строк в строку по умолчанию

Варианты форматированного вывода:

1. source.replaceAll(“\\|”, “ ”); ⬄ source.replace(“|”, “ ”);
2. Arrays.stream(result).collect(Collectors.joining(“ ”)).//joining() – создаёт Collector, который конкатенирует стрим строк(т.е. стрим уже должен содержать строки) в одну строку без разделителя!(в нашем случае с разделителем: joining(“ ”))
3. Pattern.compile("\\|").splitAsStream(source).collect(Collectors.joining(“ ”));

String result = “I am a java developer”;

**15. Какой метод вызывается для преобразования переменной в строку?**

|  |  |
| --- | --- |
| 1  2  3 | public static String valueOf(Object obj) {          return (obj == null) ? "null" : obj.toString();  } |

**16. Как узнать значение конкретного символа строки, зная его порядковый номер в строке?**

str.charAt(int i) вернет символ по индексу.

**17. Как найти необходимый символ в строке?**

str.indexOf(charch) или lastIndexOf(char c) — вернет индекс первого и последнего вхождения символа.

|  |  |
| --- | --- |
| 1  2  3 | String fs = "12345678904";  int a = fs.indexOf("456"); //3  int b = fs.lastIndexOf("4"); //10 |

**18. Можно ли синхронизировать доступ к строке?**

String сам по себе потокобезопасный класс. Если мы работаем с изменяемыми строками, то нужно использовать StringBuffer.

**19. Что делает метод intern()?**

Когда метод intern() вызван, если пул строк уже содержит строку, эквивалентную к нашему объекту, что подтверждается методом equals(Object), тогда возвращается ссылка на строку из пула. В противном случае объект строки добавляется в пул и ссылка на этот объект возвращается.  
Этот метод всегда возвращает строку, которая имеет то же значение, что и текущая строка, но гарантирует что это будет строка из пула уникальных строк.  
Ниже приведен пример работы метода intern()

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | String a = "string a";  String b = new String("string a");  String c = b.intern();    System.out.println(a == b); //false  System.out.println(b == c); //false  System.out.println(a == c); //true |

**20. Чем отличаются и `что общего у классов String, StringBuffer и StringBuilder?**

В дополнение к ответу вначале приведу сравнение производительности классов.

Сравнение производительности. Linux

|  |  |  |  |
| --- | --- | --- | --- |
| Класс | Open JDK 1.6.0\_18 | HotSpot 1.6.0\_20 | JRockit 4.0.1 |
| String | 27390ms | 26850ms | 26940ms |
| StringBuffer | 35.55ms | 34.87ms | 15.41ms |
| StringBuilder | 33.01ms | 31.78ms | 12.82ms |

**21. Как правильно сравнить значения строк двух различных объектов типа String и StringBuffer?**

Привести их к одному типу и сравнить.

**22. Почему строка неизменная и финализированная в Java?**

Есть несколько преимуществ в неизменности строк:

1. Строковый пул возможен только потому, что строка неизменна в Java, таким образом виртуальная машина сохраняет много места в памяти(heapspace), поскольку разные строковые переменные указывают на одну переменную в пуле. Если бы строка не была неизмененяемой, тогда бы интернирование строк не было бы возможным, потому что если какая-либо переменная изменит значение, это отразится также и на остальных переменных, ссылающихся на эту строку.
2. Если строка будет изменяемой, тогда это станет серьезной угрозой безопасности приложения. Например, имя пользователя базы данных и пароль передаются строкой для получения соединения с базой данных и в программировании сокетов реквизиты хоста и порта передаются строкой. Так как строка неизменяемая, её значение не может быть изменено, в противном случае любой хакер может изменить значение ссылки и вызвать проблемы в безопасности приложения.
3. Так как строка неизменная, она безопасна для многопоточности и один экземпляр строки может быть совместно использован различными потоками. Это позволяет избежать синхронизации для потокобезопасности, строки полностью потокобезопасны.
4. Строки используются в Javaclassloader и неизменность обеспечивает правильность загрузки класса при помощи Classloader. К примеру, задумайтесь об экземпляре класса, когда вы пытаетесь загрузить java.sql.Connection класс, но значение ссылки изменено на myhacked.Connection класс, который может осуществить нежелательные вещи с вашей базой данных.
5. Поскольку строка неизменная, её hashcode кэшируется в момент создания и нет необходимости рассчитывать его снова. Это делает строку отличным кандидатом для ключа в Map и его обработка будет быстрее, чем других ключей HashMap. Это причина, почему строка наиболее часто используемый объект в качестве ключа HashMap.

**23. Почему массив символов предпочтительнее строки для хранения пароля?**

Строка неизменяемая в Java и хранится в пуле строк. С тех пор, как она была создана, она остается в пуле, пока не будет удалена сборщиком мусора, поэтому, когда мы думаем, что закончили работу с паролем, он остается доступным в памяти некоторое время, и нет способа избежать этого. Это риск безопасности, поскольку кто-либо, имеющий доступ к дампу памяти сможет найти пароль в виде чистого текста.  
Если мы используем массив символов для хранения пароля, мы можем очистить его после того, как закончим с ним работать. Таким образом, мы можем контролировать, как долго он находится в памяти, что позволяет избежать риска безопасности, свойственного строке.

**24. Почему строка является популярным ключом в HashMap в Java?**

Поскольку строки неизменны, их хэшкод кэшируется в момент создания, и не требует повторного пересчета. Это делает строки отличным кандидатом для ключа в Map и они обрабатываются быстрее, чем другие объекты-ключи HashMap. Вот почему строки преимущественно используются в качестве ключей HashMap.

**25. Напишите метод удаления данного символа из строки.**

Мы можем использовать метод replaceAll для замены всех вхождений в строку другой строкой. Обратите внимание на то, что метод получает в качестве аргумента строку, поэтому мы используем класс Character для создания строки из символа, и используем её для замены всех символов на пустую строку.

Java

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static String removeChar(String str, char ch) {          if (str == null)              return null;          return str.replaceAll(Character.toString(ch), "");  } |