Кодирование символов

Для передачи информации люди используют знаковую систему. Знаковая система реализуется в виде character set (набор символов)

character set (набор символов) или repertoire (репертуар) включает в себя символы, которые нужны для конкретной цели - для поддержания западноевропейских языков, иероглифических языков, специальные символы (пиктограммы), [1].

С появлением компьютеров возникла потребность представить эту знаковую систему в компьютере, поскольку это самый простой и органичный интерфейс между человеком и машиной.

Поскольку компьютер непосредственно «понимает» только числа, значит нужно представить наборы символов в виде чисел, то есть закодировать их.

coded character set (закодированный набор символов) - набор символов в котором уникальный номер был присвоен каждому символу. Единицы закодированного набора символов известны как code point(*точки кода)*. Значение точки кода представляет позицию символа в закодированном наборе символов, [1].

В зависимости от количества символов, которые нужно представить, могут потребоваться числа «емкостью» один, два или более байта.

Закодированные наборы символов называют code pages (кодовые страницы), [1].

Например, набор символов английского я зыка можно представить с помощью однобайтных чисел. Исторически это было реализовано в кодовой странице [ASCII](http://www.robelle.com/library/smugbook/ascii.html):  


В настоящее время наиболее часто используется набор символов Unicode.

Unicode - универсальный набор символов, то есть стандарт, определяющий в одном месте все символы, необходимые для написания большинства существующих языков, используемых на компьютерах, [1].

Unicode стремится быть, и в значительной степени уже есть, расширением всех других наборов символов, которые были закодированы.

Текст на компьютере или в Сети состоит из символов. *Символы* представляют буквы алфавита, знаки препинания или иные символы.

В прошлом, различные организации собрали различные наборы символов и создали кодирования для них. Один набор может включать только основанные на латинице западноевропейские языки, другой может охватывать кириллицу, третий дальневосточные языки (такие как японский), другие могут быть одними из многих наборов разработанных специфическим способом для представления другого языка где-то в мире.

[Unicode консорциум](http://www.unicode.org/) обеспечивает большой, единый набор символов, который направлен для включения всех символов, необходимых для любой системы письма в мире, включая древние скрипты (такие как Клинопись, Готика и Египетские Иероглифы). В настоящее время он является основным для архитектуры сети и операционных систем, и поддерживается всеми основными веб-браузерами и приложениями. [Стандарт Unicode](http://www.unicode.org/standard/standard.html) также описывает свойства и алгоритмы для работы с символами.

Кодовые страницы предоставляют алфавит для машинного представления символов. Но кроме этого для кодирования нужен еще и алгоритм кодирования.

Простейшим вариантом является использование позиции символа в кодовой странице, но это не всегда возможно.

character encoding (кодирование символов) отражает способ (алгоритм) с помощью которого набор символов отображается в байтах для манипуляций на компьютере.

Один набор символов может быть представлен разными кодировками, [1].

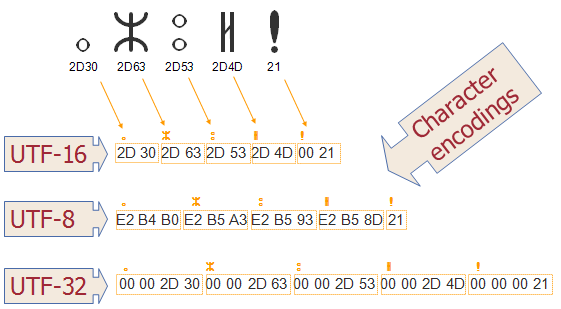
Например, Unicode использует разные способы кодирования. Они называются UTF-8, UTF-16, и UTF-32.

**UTF-8** использует 1 байт для представления символов в наборе ASCII, два байта для символов в нескольких алфавитных блоках, и три байта для остальных BMP. Дополнительные символы используют 4 байта.

**UTF-16** использует 2 байта для любого символа в BMP, и 4 байта для дополнительных символов.

**UTF-32** использует 4 байта для всех символов.

Здесь BMP (Base Multilanguage Plane) – базовая многоязыковая плоскость.

На рисунке показано кодирование символов с помощью перечисленных кодировок:

В следующей таблице, в первой строке чисел представлена позиция символа в кодировке Unicode. Другие строки показывают значения байтов, используемых для представления этого символа в определенной кодировке символов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Latin A.** | **Hebrew alef.** | **Han ideograph AN.** | **Значение китайского иероглифа 'обрубок дерева'.** |
| **Место кода** | U+0041 | U+05D0 | U+597D | U+233B4 |
| **UTF-8** | 41 | D7 90 | E5 A5 BD | F0 A3 8E B4 |
| **UTF-16** | 00 41 | 05 D0 | 59 7D | D8 4C DF B4 |
| **UTF-32** | 00 00 00 41 | 00 00 05 D0 | 00 00 59 7D | 00 02 33 B4 |

Строки в Java

**Определение строк**

В язы­ках С и C++ отсутствует встроенная поддержка такого объекта, как строка. В них при необхо­димости передается адрес последовательности байтов, содержимое которых трактуется как символы до тех пор, пока не будет встречен нулевой байт, отмечающий конец строки.

Работа со строками в Java представлена тремя классами String, StringBuffer, StringBuilder. Такое разделение позволяет осуществлять оптимизацию производительности, но требует от пользователя знать в какой ситуации какой класс лучше применять:

* **String**. Этот класс не что иное, как объ­ектное представление неизменяемого символьного массива. В этом классе есть методы, которые позволяют сравнивать строки, осуществлять в них поиск и извлекать определенные символы и подстроки. Применяется для одиночных неизменяемых строк;
* **StringBuffer**.Представляет расширяемые и доступные для изменений последовательности символов, позволяя вставлять символы и подстроки в существующую строку и в любом месте. Класс синхронизирован, то есть его нужно использовать при работе в многопоточной среде;
* **StringBuilder**. Добавлен в JDK 5, он во всем идентичен классу StringBuffer, за исключением того, что он не синхронизирован. При однопоточном использовании StringBuilder практически всегда в 1.2-1.5 раза быстрее, чем StringBuffer. Но цена скорости - небезопасное поведение в многопоточной среде;

Для изменяемых строк рекомендуется использовать StringBuilder, если явно не требуется StringBuffer.

String, и StringBuffer объявлены final, что означает, что ни от одного из этих классов нельзя производить подклассы. [4].

**Конкатенация**

Конкатенацию строк нужно осуществлять с учетом производительности. Можно выделить такие способы конкатенации:

* **Способ #1: плюс**  
  String result = a + b;  
  Самый очевидный. На самом деле за этой конструкцией скрывается достаточно емкая операция. Тут создается новый StringBuilder и в него добавляются все полученные на вход строки. Далее вызывается toString()
* **Способ #2: String.concat(String)**String result = a.concat(b);  
  Использование «родных» методов String-а даст ощутимый прирост в производительности. Важно понимать, что метод **concat** не изменяет строку, а лишь создает новую как результат слияния текущей и переданной в качестве параметра.
* **Способ #3: StringBuffer/ StringBuilder**
* String result = new StringBuffer(a).append(b).toString();  
  Хороший способ для конкатенации многих строк, например, в цикле.

Autoboxing и Unboxing

* **Автоупаковка (autoboxing)** - это процесс автоматической инкапсуляции данных простого типа, такого как int или double, в эквивалентную ему объектную оболочку типа, как только понадобится объект этого типа. При этом нет необходимости в явном создании объекта нужного типа;
* **Автораспаковка (auto-unboxing)** - это процесс автоматического извлечения из упакованного объекта значения, когда оно потребуется. Вызовы методов, таких как intValue() и doubleValue(), становятся ненужными.

Благодаря автоупаковке исчезает необходимость в создании вручную объекта для инкапсуляции значения простого типа. Вам нужно только присвоить это значение указателю на объект типа-оболочки.

Помимо простых случаев присваивания, автоупаковка выполняется автоматически каждый раз, когда данные базового типа должны быть преобразованы в объект, а автораспаковка — при необходимости преобразования объекта в значение базового типа, [5]:

* Когда аргумент передается в метод или когда значение возвращается методом;
* В выражениях;
* При размещении в коллекциях;

Исходя из вышесказанного, следует ограничить использование оболочек типов только теми случаями, для которых требуется объектное представление данных простых типов.

1. Кодирование символов: Основные понятия, <http://www.w3.org/International/articles/definitions-characters>
2. Кодировка символов для начинающих, <http://www.w3.org/International/questions/qa-what-is-encoding>
3. The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!), <http://www.joelonsoftware.com/articles/Unicode.html>
4. Работа со строками, <http://www.helloworld.ru/texts/comp/lang/java/java/09.htm>
5. Автоупаковка и автораспаковка, <http://www.codenet.ru/webmast/java/autoboxing.php>