More Than You Ever Wanted To Know About Strings

Charles Oliver Nutter

Me

- Charles Oliver Nutter
- Principal Software Eng. at Red Hat
  - “Research and prototyping” group
- “JRuby Guy” since 2006
  - Thanks to Sun Microsystems, Engine Yard,and Red Hat!
- headius@headius.com
- @headius

JRuby

Many Challenges

- Dynamic language
- Native code integration
- Process management
- Multi-encoding string support

Agenda

- What is a string
- World of encodings
- JCodings
- JOni
- Wrap-up

Strings and Encodings

What is a String?

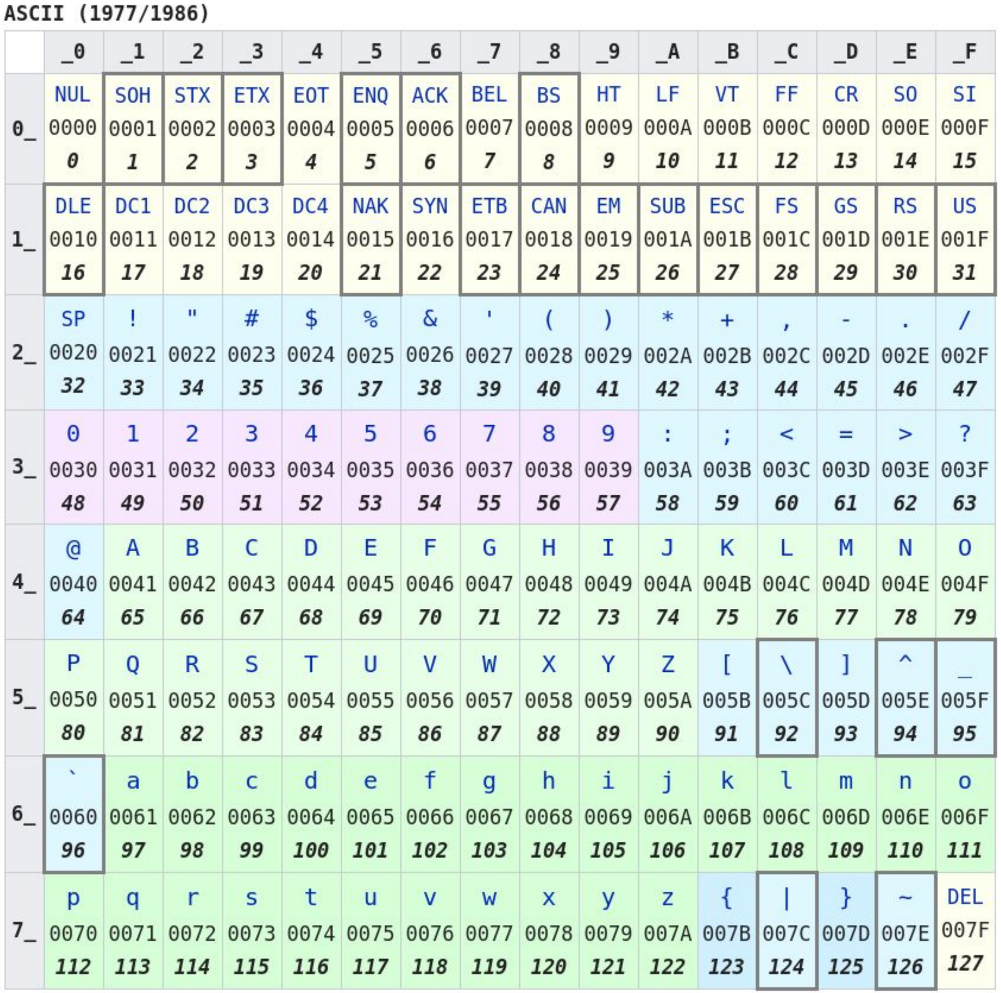
- A finite sequence of characters
  - A contiguous array
  - A tree of arrays, as in ropes
  - A list, perhaps immutable as in Erlang
- Mutable or immutable
- Constant or O(n) access time

What is a Character?

- Glyph: what you see on the screen
- Grapheme: smallest indivisible piece
- Character: one or more graphemes in combination
    - Not tied to a specific glyph
- Early representations limited in scope
- Frequently single-locale

ASCII

- American Standard Code for Information Interchange
- First published in 1963
- 7-bit encoding, now typically lower half of 8-bit
  - Large contributor to bytes having 8 bits
- Largely unchanged since 1977
- Many modern encodings are compatible

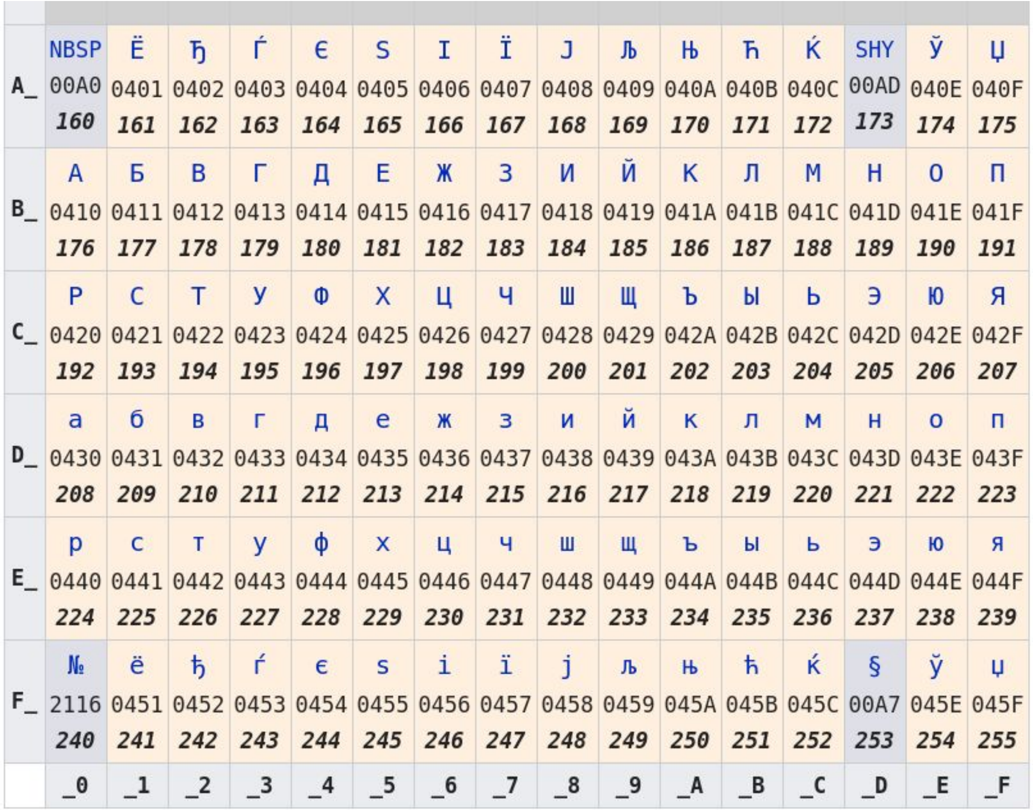


ASCII (1977/1986)

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL 0000 *0*	SOH 0001 *1*	STX 0002 *2*	ETX 0003 *3*	EOT 0004 *4*	ENQ 0005 *5*	ACK 0006 *6*	BEL 0007 *7*	BS 0008 *8*	HT 0009 *9*	LF 000A *10*	VT 000B *11*	FF 000C *12*	CR 000D *13*	SO 000E *14*	SI 000F *15*
1_	DLE 0010 *16*	DC1 0011 *17*	DC2 0012 *18*	DC3 0013 *19*	DC4 0014 *20*	NAK 0015 *21*	SYN 0016 *22*	ETB 0017 *23*	CAN 0018 *24*	EM 0019 *25*	SUB 001A *26*	ESC 001B *27*	FS 001C *28*	GS 001D *29*	RS 001E *30*	US 001F *31*
2_	SP 0020 *32*	! 0021 *33*	" 0022 *34*	# 0023 *35*	$ 0024 *36*	% 0025 *37*	& 0026 *38*	' 0027 *39*	(0028 *40*	) 0029 *41*	* 002A *42*	+ 002B *43*	, 002C *44*	- 002D *45*	. 002E *46*	/ 002F *47*
3_	0 0030 *48*	1 0031 *49*	2 0032 *50*	3 0033 *51*	4 0034 *52*	5 0035 *53*	6 0036 *54*	7 0037 *55*	8 0038 *56*	9 0039 *57*	: 003A *58*	; 003B *59*	< 003C *60*	= 003D *61*	> 003E *62*	? 003F *63*
4_	@ 0040 *64*	A 0041 *65*	B 0042 *66*	C 0043 *67*	D 0044 *68*	E 0045 *69*	F 0046 *70*	G 0047 *71*	H 0048 *72*	I 0049 *73*	J 004A *74*	K 004B *75*	L 004C *76*	M 004D *77*	N 004E *78*	O 004F *79*
5_	P 0050 *80*	Q 0051 *81*	R 0052 *82*	S 0053 *83*	T 0054 *84*	U 0055 *85*	V 0056 *86*	W 0057 *87*	X 0058 *88*	Y 0059 *89*	Z 005A *90*	[005B *91*	\\ 005C *92*	] 005D *93*	^ 005E *94*	_ 005F *95*
6_	` 0060 *96*	a 0061 *97*	b 0062 *98*	c 0063 *99*	d 0064 *100*	e 0065 *101*	f 0066 *102*	g 0067 *103*	h 0068 *104*	i 0069 *105*	j 006A *106*	k 006B *107*	l 006C *108*	m 006D *109*	n 006E *110*	o 006F *111*
7_	p 0070 *112*	q 0071 *113*	r 0072 *114*	s 0073 *115*	t 0074 *116*	u 0075 *117*	v 0076 *118*	w 0077 *119*	x 0078 *120*	y 0079 *121*	z 007A *122*	{ 007B *123*	| 007C *124*	} 007D *125*	~ 007E *126*	DEL 007F *127*

ISO-8859

- Utilize high 128 values
- Latin variants and non-latin
- Thai
- Not enough room for most Asian languages
- ISO-8859-5 superceded by KOI-R, Windows-1251

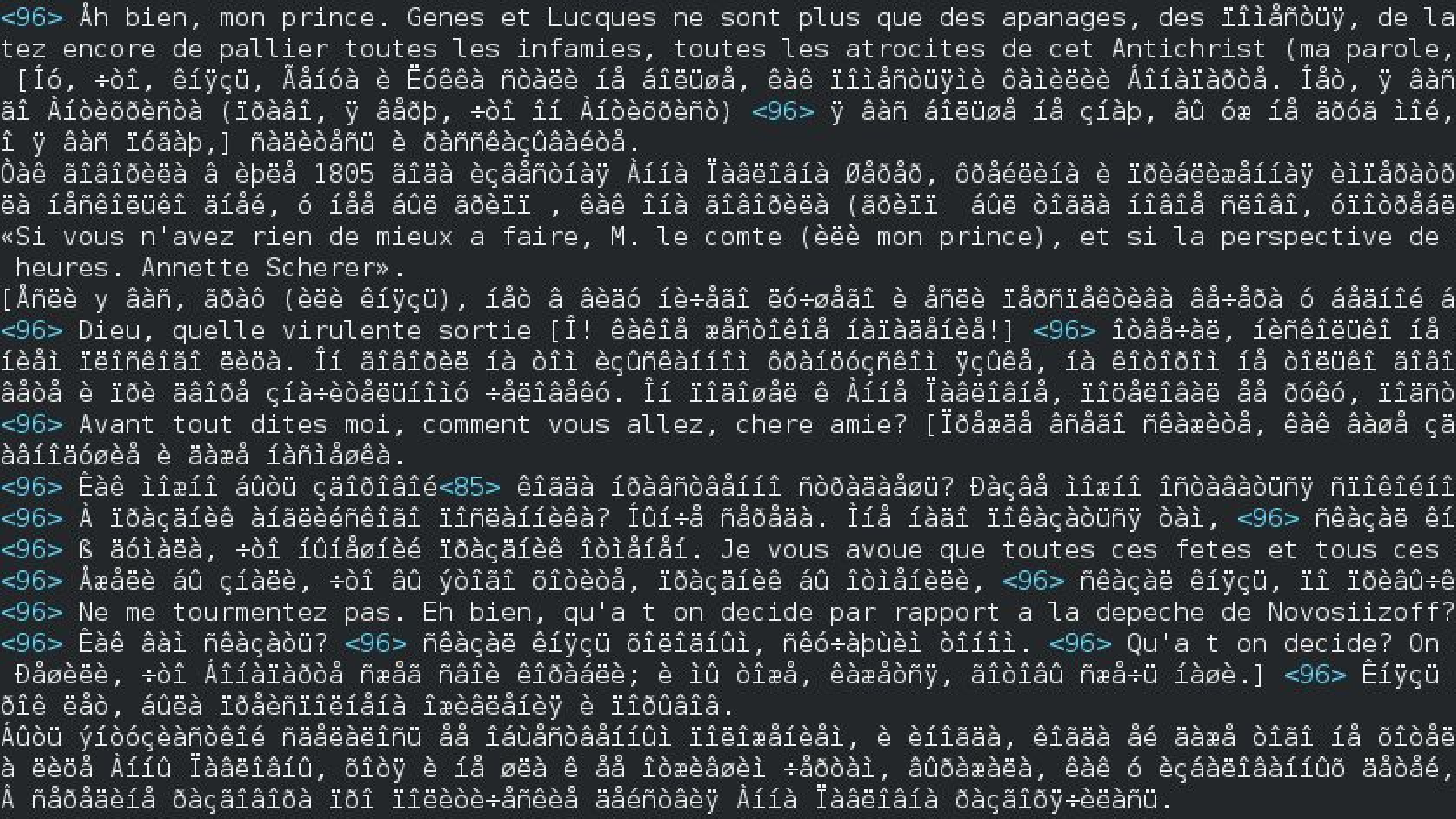


A_	NBSP 00A0 160	Ё 0401 161	Ђ 0402 162	Ѓ 0403 163	Є 0404 164	Ѕ 0405 165	І 0406 166	Ї 0407 167	Ј 0408 168	Љ 0409 169	Њ 040A 170	Ћ 040B 171	Ќ 040C 172	SHY 00AD 173	Ў 040E 174	Џ 040F 175
B_	А 0410 176	Б 0411 177	В 0412 178	Г 0413 179	Д 0414 180	Е 0415 181	Ж 0416 182	З 0417 183	И 0418 184	Й 0419 185	К 041A 186	Л 041B 187	М 041C 188	Н 041D 189	О 041E 190	П 041F 191
C_	Р 0420 192	С 0421 193	Т 0422 194	У 0423 195	Ф 0424 196	Х 0425 197	Ц 0426 198	Ч 0427 199	Ш 0428 200	Щ 0429 201	Ъ 042A 202	Ы 042B 203	Ь 042C 204	Э 042D 205	Ю 042E 206	Я 042F 207
D_	а 0430 208	б 0431 209	в 0432 210	г 0433 211	д 0434 212	е 0435 213	ж 0436 214	з 0437 215	и 0438 216	й 0439 217	к 043A 218	л 043B 219	м 043C 220	н 043D 221	о 043E 222	п 043F 223
E_	р 0440 224	с 0441 225	т 0442 226	у 0443 227	ф 0444 228	х 0445 229	ц 0446 230	ч 0447 231	ш 0448 232	щ 0449 233	ъ 044A 234	ы 044B 235	ь 044C 236	э 044D 237	ю 044E 238	я 044F 239
F_	№ 2116 240	ё 0451 241	ђ 0452 242	ѓ 0453 243	є 0454 244	ѕ 0455 245	і 0456 246	ї 0457 247	ј 0458 248	љ 0459 249	њ 045A 250	ћ 045B 251	ќ 045C 252	§ 00A7 253	ў 045E 254	џ 045F 255
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F

ISO-8859-5 Cyrillic characters

Single Byte is Not Enough

- ASCII is just Latin
- ISO encodings are Latin + X
  - What if you need two different X?
- Languages with >255 (or 128) characters
  - Chinese, Japanese, Korean
- Archaic languages, symbols, pictographs
- Bad behavior if you pick wrong encoding



```
<96> Åh bien, mon prince. Genes et Lucques ne sont plus que des apanages, des ïîìåñòüÿ, de la
tez encore de pallier toutes les infamies, toutes les atrocites de cet Antichrist (ma parole,
 [Íó, ÷òî, êíÿçü, Ãåíóà è Ëóêêà ñòàëè íå áîëüøå, êàê ïîìåñòüÿìè ôàìèëèè Áîíàïàðòå. Íåò, ÿ âàñ
ãî Àíòèõðèñòà (ïðàâî, ÿ âåðþ, ÷òî îí Àíòèõðèñò) <96> ÿ âàñ áîëüøå íå çíàþ, âû óæ íå äðóã ìîé,
î ÿ âàñ ïóãàþ,] ñàäèòåñü è ðàññêàçûâàéòå.
Òàê ãîâîðèëà â èþëå 1805 ãîäà èçâåñòíàÿ Àííà Ïàâëîâíà Øåðåð, ôðåéëèíà è ïðèáëèæåííàÿ èìïåðàòð
ëà íåñêîëüêî äíåé, ó íåå áûë ãðèïï , êàê îíà ãîâîðèëà (ãðèïï  áûë òîãäà íîâîå ñëîâî, óïîòðåáë
«Si vous n'avez rien de mieux a faire, M. le comte (èëè mon prince), et si la perspective de
 heures. Annette Scherer».
[Åñëè y âàñ, ãðàô (èëè êíÿçü), íåò â âèäó íè÷åãî ëó÷øåãî è åñëè ïåðñïåêòèâà âå÷åðà ó áåäíîé á
<96> Dieu, quelle virulente sortie [Î! êàêîå æåñòîêîå íàïàäåíèå!] <96> îòâå÷àë, íèñêîëüêî íå
íèåì ïëîñêîãî ëèöà. Îí ãîâîðèë íà òîì èçûñêàííîì ôðàíöóçñêîì ÿçûêå, íà êîòîðîì íå òîëüêî ãîâî
âåòå è ïðè äâîðå çíà÷èòåëüíîìó ÷åëîâåêó. Îí ïîäîøåë ê Àííå Ïàâëîâíå, ïîöåëîâàë åå ðóêó, ïîäñò
<96> Avant tout dites moi, comment vous allez, chere amie? [Ïðåæäå âñåãî ñêàæèòå, êàê âàøå çä
àâíîäóøèå è äàæå íàñìåøêà.
<96> Êàê ìîæíî áûòü çäîðîâîé<85> êîãäà íðàâñòâåííî ñòðàäàåøü? Ðàçâå ìîæíî îñòàâàòüñÿ ñïîêîéíî
<96> À ïðàçäíèê àíãëèéñêîãî ïîñëàííèêà? Íûí÷å ñðåäà. Ìíå íàäî ïîêàçàòüñÿ òàì, <96> ñêàçàë êí
<96> ß äóìàëà, ÷òî íûíåøíèé ïðàçäíèê îòìåíåí. Je vous avoue que toutes ces fetes et tous ces
<96> Åæåëè áû çíàëè, ÷òî âû ýòîãî õîòèòå, ïðàçäíèê áû îòìåíèëè, <96> ñêàçàë êíÿçü, ïî ïðèâû÷ê
<96> Ne me tourmentez pas. Eh bien, qu'a t on decide par rapport a la depeche de Novosiizoff?
<96> Êàê âàì ñêàçàòü? <96> ñêàçàë êíÿçü õîëîäíûì, ñêó÷àþùèì òîíîì. <96> Qu'a t on decide? On
 Ðåøèëè, ÷òî Áîíàïàðòå ñæåã ñâîè êîðàáëè; è ìû òîæå, êàæåòñÿ, ãîòîâû ñæå÷ü íàøè.] <96> Êíÿçü
ðîê ëåò, áûëà ïðåèñïîëíåíà îæèâëåíèÿ è ïîðûâîâ.
Áûòü ýíòóçèàñòêîé ñäåëàëîñü åå îáùåñòâåííûì ïîëîæåíèåì, è èíîãäà, êîãäà åé äàæå òîãî íå õîòåë
à ëèöå Àííû Ïàâëîâíû, õîòÿ è íå øëà ê åå îòæèâøèì ÷åðòàì, âûðàæàëà, êàê ó èçáàëîâàííûõ äåòåé,
Â ñðåäèíå ðàçãîâîðà ïðî ïîëèòè÷åñêèå äåéñòâèÿ Àííà Ïàâëîâíà ðàçãîðÿ÷èëàñü.
```

Multi-byte encodings

- Multi-byte encodings to the rescue
  - And oh the pain
  - Incompatible representations
  - Obvious ASCII, ISO-8859 issues
- If only byte had been 16 bits!

Chinese, Japanese, Korean

- Hiragana and Katakana: 46 characters each
- Hanzi/Kanji: thousands of characters
  - Over 100k in Chinese
  - Typically 2000-7000 considered mainstream
- Multiple encodings still in use today
  - All have variable-width characters
  - Most are at least ASCII-compatible
  - Some mandated by government, industry standards

Unicode

Unicode 88

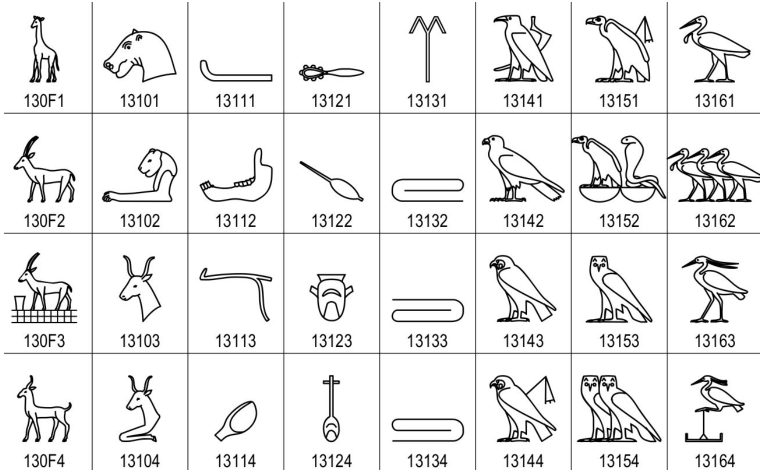
- 16-bit characters
- 14 bits usable => 16k characters
- Only contemporary, in-use languages considered
- Unicode 1.0 followed in 1991

Universal Character Set (UCS)

- ISO-10646
- Developed in tandem with Unicode since 1991
- Standardized character codepoints
- Excludes inter-character relationships
  - Ligatures
  - Script direction
  - Sorting
- UCS-2 chosen for Java 1.0

Unicode 2.0 and Beyond

- Multi-word “surrogates” to support >16k chars
- UCS-2 becomes UTF-16
- Unicode continues to evolve


130F1
13101
13111
13121
13131
13141
13151
13161
130F2
13102
13112
13122
13132
13142
13152
13162
130F3
13103
13113
13123
13133
13143
13153
13163
130F4
13104
13114
13124
13134
13144
13154
13164

We Have a Problem

- Single byte encodings are more efficient, but…
    - Only two alphabets at a time
    - Pick wrong encoding, garbled text
    - No indication what encoding to use
- Multibyte encodings are more complete, but...
    - Efficient byte width for English
    - Multibyte for everything else
    - Sometimes “over-unify” as with Han Unification
- We may never solve this

Java

Strings in Java

- char[] and length (plus some other bits)
- In the beginning, UCS-2
  - 16 bits per character should be enough for anyone!
  - Most non-asian languages waste one byte
- Later changed to UTF-16
  - char could not change width
  - Surrogates for characters >16k
- We are stuck with it

Problems with Java’s UTF-16

- Unavoidable encoding overhead
  - Encode/decode to/from byte[]
  - Frequently UTF-8 to UTF-16 and back
  - Large part of IO performance gap with C/++
- ASCII range wastes 9 bits per character
  - Improved in Java 9 with compact 7-bit strings
- No alternative representation
  - All strings must be representable as UTF-16 characters
- Worst of all worlds?

Ruby

Strings in Ruby

- byte[], length, and encoding
  - Every string can have its own encoding
  - Every string knows its own encoding
  - Methods for both byte and character operations
- Complex implementation, but it works
  - Decode/encode/transcode only when needed
  - String IO can be nearly free
  - Impossible to match with fixed-encoding strings

The Ruby Way

- Too many problems with String/char[]
- Character logic had to be duplicated
- String methods had to be encoding-aware
  - Encoding negotiation and transcoding
- New Regex engine
  - Multi-encoding support
- Interop with Java became harder
  - Frequent transcoding to UTF-16 and back
  - Default to UTF-16 for Java interop

It works!

And now you can use it in Java!

ByteList

ByteList

- byte[], begin, length, and encoding
- StringBuffer-like operations on byte[]
- “Unsafe” access to byte[] allowed
- Expanding to provide character logic
  - Multi-byte character (MBC) support
  - Optimizations for random access time
- Maven: org.jruby.extras:bytelist:1.0.15
- https://github.com/jruby/bytelist
- Help wanted!

JCodings

JCodings

- Decode/encode codepoint ⇔ byte[]
- Character-walking
- Validation
- Transcoding from one encoding to another
- Maven: org.jruby.extras:jcodings:1.0.19
- https://github.com/jruby/jcodings

Encoding Support

- UTF-8 through UTF-32, all endians
- ISO-8859-1 through 16
    - OpenJDK doesn’t even support all of these
    - I have patches for a few of them
- Shift-JIS, EUC_JP, GB, Big5
- IBM and Windows codepages
- More possible in the future

Basics

```java
byte[] utf8Bytes = "møøse".getBytes("UTF-8");

assertEquals(7, utf8Bytes.length);
assertEquals(5, UTF8Encoding.INSTANCE.strLength(utf8Bytes, 0, 7));
assertEquals(2, UTF8Encoding.INSTANCE.length(utf8Bytes[1]));
assertEquals('ø', UTF8Encoding.INSTANCE.mbcToCode(utf8Bytes, 1, 3));
```

Transcoding

- byte[] to byte[] with minimal overhead
- Pausable stateful transcoder
- Epic inner loop ported from C
  - Nested switches, loops, gotos
- Better perf than byte[] => char[] => byte[]
- Comparable perf to Charset

Transcoding

```
---

EConv econv = TranscoderDB.open("UTF-8", "UTF-16", 0);

byte[] src = "foo".getBytes("UTF-8");
byte[] dest = new byte["foo".getBytes("UTF-16").length];

econv.convert(src, new Ptr(0), 3, dest, new Ptr(0), dest.length, 0);

assertArrayEquals("foo".getBytes("UTF-16"), dest);
```

Bonus Features

- CR/LF negotiation
  - CR, LF, CRLF normalization
- XML entity replacement
  - &lt; &gt; &amp; &quot; &apos;
  - &#x12345 character references
- Multi-stage transcoding
  - When there’s no direct translation between two encodings
  - SJIS-SoftBank => UTF8-SoftBank => UTF-8 => CP51932 => CP50220
  - Entity replacement, CRLF translation

Universal Newline

```
EConv econv = TranscoderDB.open("", "", EConvFlags.UNIVERSAL_NEWLINE_DECORATOR);

byte[] src = "foo\r\nbar".getBytes();
byte[] dest = new byte[7];

econv.convert(src, new Ptr(0), 8, dest, new Ptr(0), dest.length, 0);

assertArrayEquals("foo\nbar".getBytes(), dest);
```

XML Attrs and Character Refs

```
EConv econv = TranscoderDB.open("utf-8".getBytes(), "euc-jp".getBytes(),
        EConvFlags.XML_ATTR_CONTENT_DECORATOR |
        EConvFlags.XML_ATTR_QUOTE_DECORATOR |
        EConvFlags.UNDEF_HEX_CHARREF);

byte[] src = "<\u2665>&\"\u2661\"".getBytes(UTF8);
...
econv.convert(src, new Ptr(0), src.length, dest, destP, dest.length, 0);
assertArrayEquals(
        "\"&lt;&#x2665;&gt;&amp;&quot;&#x2661;&quot;\"".getBytes(),
        Arrays.copyOf(dest, destP.p));
```

JCodings Performance

- Difficult to compare
  - Going through UTF-16 skews results
- Faster than two-stage
- Similar to decode or encode stages alone
- Has not been a bottleneck for JRuby

JCodings Users

- Facebook Presto
  - High-speed character IO without char[] decoding
- JRuby and TruffleRuby
- JetBrains RubyMine and other Ruby IDEs

JOni

JOni

- Port of Oniguruma from CRuby
  - Some divergence but we try to track them
- Match directly on byte[]
- Full JCodings encoding support
- Pluggable regex grammars (Java, Ruby, JS, …)
- Stackless bytecode machine
- Maven: org.jruby.joni:joni:2.1.11
- https://github.com/jruby/joni

JOni Versus java.util.regex

- byte[] vs char[]
- j.u.r recurses, blows stack for large input
- Better performance for most forms
- Supports richer Ruby regex features
- Interruptible

Construction and Searching

```
public void regexExample(String pattern, byte[] str, int from, int to) {
   Regex reg = new Regex(pattern);
   reg = new Regex(pattern, Syntax.Java);

   Matcher m = reg.matcher(str);
   Region region;

   int r = m.search(from, to, Option.NONE);

   try {
       r = m.searchInterruptible(from, to, Option.NONE);
   } catch (InterruptedException ie) {
       // hooray for interruptible regex
   }
...
```

Capture Regions

```
...
   // extract regions
   region = m.getEagerRegion();

   int start = region.beg[1];
   int end = region.end[1];

   System.out.println(new String(str, start, end));
}
```

JOni Performance

- 2-3x faster than java.util.regex for most loads
- Avoids decoding step for bytes from IO
- Far fewer failure cases
- Interruptible for pathological cases

JOni Users

- Facebook Presto
  - Again, avoiding transcode overhead
- Nashorn
  - Modified port to use char[]
  - Grammar support intact
  - Potential replacement for j.u.regex?
- JetBrains RubyMine
- SourceClear Maven plugin

Real World Examples

Convert Database to ISO-8859-1

```
Connection conn =
        DriverManager.getConnection("jdbc:postgresql://localhost/headius");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from test_varchar");
int column = rs.findColumn("someString");
```

Convert Database to ISO-8859-1

```
---

// get UTF-8 character length of column
int utf8Size = rs.getMetaData().getColumnDisplaySize(column);

// reserve enough ISO-8859-1 buffer for longest chars
int isoSize = utf8Size * ISO8859_1Encoding.INSTANCE.maxLength();
byte[] iso = new byte[isoSize];
```

Convert Database to ISO-8859-1

```
---

while (rs.next()) {
    byte[] utf8 = rs.getBytes(column);

    // transcode into buffer
    Ptr in = new Ptr(0), out = new Ptr(0);
    EConv converter = TranscoderDB.open("UTF-8", "ISO-8859-1", 0);
    EConvResult result = converter.convert(
            utf8, in, utf8.length, iso, out, iso.length, 0);
```

Convert Database to ISO-8859-1

```
---

// check result of transcoding and print out bytes
System.out.println("result: " + result);
System.out.println("original string: " + rs.getString(column));
System.out.println("bytes in UTF-8: " + Arrays.toString(utf8));
System.out.println("bytes in ISO-8859-1: " +
                   Arrays.toString(Arrays.copyOf(iso, out.p)));
```

Search "Воина и мир"

```
File book1 = new File("book1.txt");
FileInputStream fis = new FileInputStream(book1);

byte[] bytes = new byte[(int)book1.length()];
fis.read(bytes);

byte[] nameBytes = "Пьер".getBytes("Windows-1251");

Regex regex = new Regex(nameBytes);
Matcher matcher = regex.matcher(bytes);
```

Search "Воина и мир"

```
---

int index = 0;
int count = 0;
while ((index = matcher.search(index, bytes.length, 0)) >= 0) {
    index += nameBytes.length;
    count++;
}

System.out.println("Found the string \"Пьер\" " + count + " times");
```

Wrapping Up

Conclusion

- Java’s string is still evolving
  - But we’re stuck with char[] for now
  - We want UTF-8 inside String!
- We live in a multi-encoding world
  - Use our libraries to avoid char[] overhead
  - Help us improve and integrate better with Java String
- java.util.regex needs an overhaul
  - Failure cases are catastrophic
  - Matching on byte[] could be added

Help Wanted!

- ByteList (https://github.com/jruby/bytelist)
  - Oldest library, most cruft
  - Deprecated unsafe methods
  - Missing or inaccurate docs in places
- JCodings (https://github.com/jruby/jcodings)
  - More documentation and examples
  - Performance analyses
  - Additional encodings
- JOni (https://github.com/jruby/joni)
  - Performance analysis
  - Code cleanup and documentation

Большое спасибо!

- Charles Oliver Nutter
  - headius@headius.com
  - @headius
- https://github.com/jruby/bytelist
- https://github.com/jruby/jcodings
- https://github.com/jruby/joni