DICOMの文字集合をRubyで扱う話
ー ISO/IEC 2022複数文字集合を読むぞ

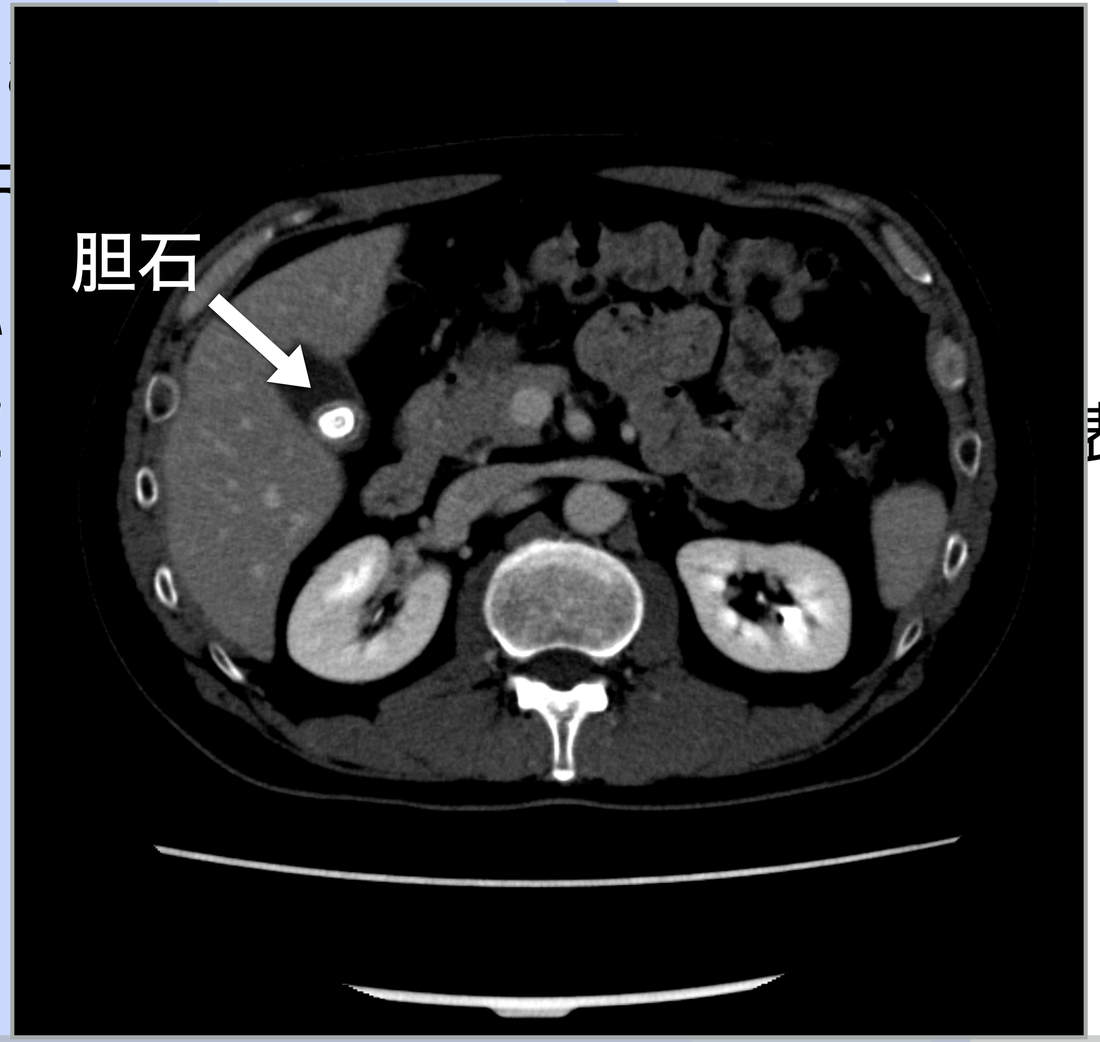
seki@ruby-lang.org

わたしについて

- Masatoshi Seki
- dRuby/Rinda/ERBを書きました
- ふだんはtoRubyというコミュニティにいます
- ポケモンカードWCS2010栃木県代表です

わたしについて

- M
- dF
- ふ
- ポ



にいます

表です

きれいに写ってる

○●● 今日のお話

- **DICOM**という医用画像のための規格がある
- いろんな文字集合を扱う必要があり**ISO/IEC 2022**を利用している

- 今日はDICOMとISO/IEC 2022を紹介して複数文字集合をRubyでどう読んだか説明します

身近にあるちょっとした問題を楽しむお話だよ

DICOM編

DICOMってなに？

- 医用画像を交換するための規格
- 病院でCTやMRを撮った時、お願いすると💿くれる
  - このときの画像のフォーマット
- 検査の予約や会計システムなどの内側でも使われてるよ

- 相互運用性（メーカー間・システム間・**世代**間）重要

プログラマ視点のDICOM(背景)

- 40年くらいの歴史がある
  - 1983年に策定がはじまり、1985年に最初の規格(DICOMの前身のACR-NEMA)が生まれる
  - JUNETとUsenetがつながったころ
    - HTMLはまだないし、もちろんunicodeもないぞ

- 相互運用性（メーカー間・システム間・**世代**間）重要

プログラマ視点のDICOM(技術)

- バイナリにエンコードされた属性リスト
  - 文字列ではないJSONを想像してみて
  - 属性の追加は容易 → 生き残れた理由かも
- いろいろな時代の制約や最適化
  - 属性の順序
  - 2バイトのアライメント，エンディアン…
  - **文字集合も！！**

parseは難しくない。150行くらいで書けた

属性の例

- (0008,0005)：文字集合
- (0010,0010)：患者名
- (0020,0032)：画像の場所（人体座標）
- (0028,0010)：画素数（Rows）
- (0028,0011)：画素数（Cols）

Ruby風に書いた属性リスト

```
dicom = [
  ...
  [[0x0008, 0x0005], "\\ISO 2022 IR 87\\ISO 2022 IR 13"], # charset
  ...
  [[0x0020, 0x0032], "-96.7773\\-36.77734\\-676.00"], # Image Position
 ...
  [[0x0028, 0x0010], 512], # Rows
  [[0x0028, 0x0011], 512], # Cols
  ...
]
```


タグごとに値のエンコード方法が違うよ

属性のエンコード step1

```
dicom = [
   ...
   [[0x0008, 0x0005], "\\ISO 2022 IR 87\\ISO 2022 IR 13"], # charset
   ...
   [[0x0020, 0x0032], "-96.7773\\-36.77734\\-676.00"], # Image Position
  ...
   [[0x0028, 0x0010], 512], # Rows
   [[0x0028, 0x0011], 512], # Cols
   ...
]
```

タグ：2つの16bit整数

VR：値の表現方法を示す2文字 2byte

データ長：16bit/32bitの整数

値：VRに従って表現されたる。偶数バイト

属性のエンコード step2

転送構文 - transfer syntax

0008	0005	'CS'	001e	"\\ISO 2022 IR 87\\ISO 2022 IR 13"

```
00000140  08 00 05 00 43 53 1e 00  5c 49 53 4f 20 32 30 32  |....CS..\ISO 202|
00000150  32 20 49 52 20 38 37 5c  49 53 4f 20 32 30 32 32  |2 IR 87\ISO 2022|
00000160  20 49 52 20 31 33 08 00  08 00 43 53 16 00 4f 52  | IR 13....CS..OR|
```

- VRを明示するLittle Endian（Explicit Little）
- 他にImplicit Little, Explicit Bigがある
  - Implicit Bigはないみたい
  - メタ情報ブロックにどんな転送構文なのか書いてある
  - メタ情報はExplicit Little...

各社／各世代でいろんなエンコード方法があった名残なのかなー

DICM - DICOMファイルだよマーク

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  44 49 43 4d 02 00 00 00  55 4c 04 00 b0 00 00 00  |DICM....UL......|
00000090  02 00 01 00 4f 42 00 00  02 00 00 00 00 01 02 00  |....OB..........|
000000a0  02 00 55 49 1a 00 31 2e  32 2e 38 34 30 2e 31 30  |..UI..1.2.840.10|
000000b0  30 30 38 2e 35 2e 31 2e  34 2e 31 2e 31 2e 32 00  |008.5.1.4.1.1.2.|
000000c0  02 00 03 00 55 49 3e 00  31 2e 32 2e 33 39 32 2e  |....UI>.1.2.392.|
000000d0  32 30 30 30 33 36 2e 39  31 34 32 2e 31 30 30 30  |200036.9142.1000|
000000e0  33 33 30 32 2e 31 30 32  30 34 33 38 30 30 31 2e  |3302.1020438001.|
000000f0  33 2e 32 30 32 34 30 37  30 34 31 33 30 30 35 34  |3.20240704130054|
00000100  2e 38 30 30 31 33 02 00  10 00 55 49 14 00 31 2e  |.80013....UI..1.|
00000110  32 2e 38 34 30 2e 31 30  30 30 38 2e 31 2e 32 2e  |2.840.10008.1.2.|
00000120  31 00 02 00 12 00 55 49  16 00 31 2e 32 2e 33 39  |1.....UI..1.2.39|
00000130  32 2e 32 30 30 30 33 36  2e 39 31 34 32 2e 31 00  |2.200036.9142.1.|
00000140  08 00 05 00 43 53 1e 00  5c 49 53 4f 20 32 30 32  |....CS..\ISO 202|
00000150  32 20 49 52 20 38 37 5c  49 53 4f 20 32 30 32 32  |2 IR 87\ISO 2022|
00000160  20 49 52 20 31 33 08 00  08 00 43 53 16 00 4f 52  | IR 13....CS..OR|
```

- (0002,0000)はメタ情報全体の長さ
  - VRは'UL',データの長さは4byte,値は0xb0

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  44 49 43 4d 02 00 00 00  55 4c 04 00 b0 00 00 00  |DICM....UL......|
00000090  02 00 01 00 4f 42 00 00  02 00 00 00 00 01 02 00  |....OB..........|
000000a0  02 00 55 49 1a 00 31 2e  32 2e 38 34 30 2e 31 30  |..UI..1.2.840.10|
000000b0  30 30 38 2e 35 2e 31 2e  34 2e 31 2e 31 2e 32 00  |008.5.1.4.1.1.2.|
000000c0  02 00 03 00 55 49 3e 00  31 2e 32 2e 33 39 32 2e  |....UI>.1.2.392.|
000000d0  32 30 30 30 33 36 2e 39  31 34 32 2e 31 30 30 30  |200036.9142.1000|
000000e0  33 33 30 32 2e 31 30 32  30 34 33 38 30 30 31 2e  |3302.1020438001.|
000000f0  33 2e 32 30 32 34 30 37  30 34 31 33 30 30 35 34  |3.20240704130054|
00000100  2e 38 30 30 31 33 02 00  10 00 55 49 14 00 31 2e  |.80013....UI..1.|
00000110  32 2e 38 34 30 2e 31 30  30 30 38 2e 31 2e 32 2e  |2.840.10008.1.2.|
00000120  31 00 02 00 12 00 55 49  16 00 31 2e 32 2e 33 39  |1.....UI..1.2.39|
00000130  32 2e 32 30 30 30 33 36  2e 39 31 34 32 2e 31 00  |2.200036.9142.1.|
00000140  08 00 05 00 43 53 1e 00  5c 49 53 4f 20 32 30 32  |....CS..\ISO 202|
00000150  32 20 49 52 20 38 37 5c  49 53 4f 20 32 30 32 32  |2 IR 87\ISO 2022|
00000160  20 49 52 20 31 33 08 00  08 00 43 53 16 00 4f 52  | IR 13....CS..OR|
```

- この辺りがメタ情報

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  44 49 43 4d 02 00 00 00  55 4c 04 00 b0 00 00 00  |DICM....UL......|
00000090  02 00 01 00 4f 42 00 00  02 00 00 00 00 01 02 00  |....OB..........|
000000a0  02 00 55 49 1a 00 31 2e  32 2e 38 34 30 2e 31 30  |..UI..1.2.840.10|
000000b0  30 30 38 2e 35 2e 31 2e  34 2e 31 2e 31 2e 32 00  |008.5.1.4.1.1.2.|
000000c0  02 00 03 00 55 49 3e 00  31 2e 32 2e 33 39 32 2e  |....UI>.1.2.392.|
000000d0  32 30 30 30 33 36 2e 39  31 34 32 2e 31 30 30 30  |200036.9142.1000|
000000e0  33 33 30 32 2e 31 30 32  30 34 33 38 30 30 31 2e  |3302.1020438001.|
000000f0  33 2e 32 30 32 34 30 37  30 34 31 33 30 30 35 34  |3.20240704130054|
00000100  2e 38 30 30 31 33 02 00  10 00 55 49 14 00 31 2e  |.80013....UI..1.|
00000110  32 2e 38 34 30 2e 31 30  30 30 38 2e 31 2e 32 2e  |2.840.10008.1.2.|
00000120  31 00 02 00 12 00 55 49  16 00 31 2e 32 2e 33 39  |1.....UI..1.2.39|
00000130  32 2e 32 30 30 30 33 36  2e 39 31 34 32 2e 31 00  |2.200036.9142.1.|
00000140  08 00 05 00 43 53 1e 00  5c 49 53 4f 20 32 30 32  |....CS..\ISO 202|
00000150  32 20 49 52 20 38 37 5c  49 53 4f 20 32 30 32 32  |2 IR 87\ISO 2022|
00000160  20 49 52 20 31 33 08 00  08 00 43 53 16 00 4f 52  | IR 13....CS..OR|
```

- (0002,0010)が転送構文
  - 1.2.840.10008.1.2.1はExplicitLittle

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  44 49 43 4d 02 00 00 00  55 4c 04 00 b0 00 00 00  |DICM....UL......|
00000090  02 00 01 00 4f 42 00 00  02 00 00 00 00 01 02 00  |....OB..........|
000000a0  02 00 55 49 1a 00 31 2e  32 2e 38 34 30 2e 31 30  |..UI..1.2.840.10|
000000b0  30 30 38 2e 35 2e 31 2e  34 2e 31 2e 31 2e 32 00  |008.5.1.4.1.1.2.|
000000c0  02 00 03 00 55 49 3e 00  31 2e 32 2e 33 39 32 2e  |....UI>.1.2.392.|
000000d0  32 30 30 30 33 36 2e 39  31 34 32 2e 31 30 30 30  |200036.9142.1000|
000000e0  33 33 30 32 2e 31 30 32  30 34 33 38 30 30 31 2e  |3302.1020438001.|
000000f0  33 2e 32 30 32 34 30 37  30 34 31 33 30 30 35 34  |3.20240704130054|
00000100  2e 38 30 30 31 33 02 00  10 00 55 49 14 00 31 2e  |.80013....UI..1.|
00000110  32 2e 38 34 30 2e 31 30  30 30 38 2e 31 2e 32 2e  |2.840.10008.1.2.|
00000120  31 00 02 00 12 00 55 49  16 00 31 2e 32 2e 33 39  |1.....UI..1.2.39|
00000130  32 2e 32 30 30 30 33 36  2e 39 31 34 32 2e 31 00  |2.200036.9142.1.|
00000140  08 00 05 00 43 53 1e 00  5c 49 53 4f 20 32 30 32  |....CS..\ISO 202|
00000150  32 20 49 52 20 38 37 5c  49 53 4f 20 32 30 32 32  |2 IR 87\ISO 2022|
00000160  20 49 52 20 31 33 08 00  08 00 43 53 16 00 4f 52  | IR 13....CS..OR|
```

- (0008,0005)は使用する文字集合
  - ISO 2022 IR 87, IR 13 と IR 6を使う宣言

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  44 49 43 4d 02 00 00 00  55 4c 04 00 b0 00 00 00  |DICM....UL......|
00000090  02 00 01 00 4f 42 00 00  02 00 00 00 00 01 02 00  |....OB..........|
000000a0  02 00 55 49 1a 00 31 2e  32 2e 38 34 30 2e 31 30  |..UI..1.2.840.10|
000000b0  30 30 38 2e 35 2e 31 2e  34 2e 31 2e 31 2e 32 00  |008.5.1.4.1.1.2.|
000000c0  02 00 03 00 55 49 3e 00  31 2e 32 2e 33 39 32 2e  |....UI>.1.2.392.|
000000d0  32 30 30 30 33 36 2e 39  31 34 32 2e 31 30 30 30  |200036.9142.1000|
000000e0  33 33 30 32 2e 31 30 32  30 34 33 38 30 30 31 2e  |3302.1020438001.|
000000f0  33 2e 32 30 32 34 30 37  30 34 31 33 30 30 35 34  |3.20240704130054|
00000100  2e 38 30 30 31 33 02 00  10 00 55 49 14 00 31 2e  |.80013....UI..1.|
00000110  32 2e 38 34 30 2e 31 30  30 30 38 2e 31 2e 32 2e  |2.840.10008.1.2.|
00000120  31 00 02 00 12 00 55 49  16 00 31 2e 32 2e 33 39  |1.....UI..1.2.39|
00000130  32 2e 32 30 30 30 33 36  2e 39 31 34 32 2e 31 00  |2.200036.9142.1.|
00000140  08 00 05 00 43 53 1e 00  5c 49 53 4f 20 32 30 32  |....CS..\ISO 202|
00000150  32 20 49 52 20 38 37 5c  49 53 4f 20 32 30 32 32  |2 IR 87\ISO 2022|
00000160  20 49 52 20 31 33 08 00  08 00 43 53 16 00 4f 52  | IR 13....CS..OR|
```

DICOMの文字集合

- DICOM文書でいろんな文字集合を扱う仕組みがある

- extensionなし - 文書の中に一種類の文字集合だけ
  - ASCII, Latin-1, utf-8, GB18030...
- extensionあり - 文書の中に複数の文字集合がある
  - ISO/IEC 2022の技術をもとにしてる

extensionなしはRubyのencodingに似てるよ

ここまでの重要なこと

- DICOMには長い歴史があり、いまも生き残ってる
- 世代を越えて互換性を重視するので、いろんな流行りを取り込んできた（と思う）
- 世界中の国の都合に合わせた文字の表現に対応してる

ISO/IEC 2022編

ISO/IEC 2022

- 複数文字集合を扱う符号化方式
- 文字集合は扱える文字の種類
- 複数の文字集合を一つの文字列の中に収めるぞ！
- UNICODEではないよ

様々な文字コードを整理しようとした、のかな？

○●● いろんな文字コード表

- おさらいします

ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p								
1	SOH	DC1	!	1	A	Q	a	q								
2	STX	DC2	"	2	B	R	b	r								
3	ETX	DC3	#	3	C	S	c	s								
4	EOT	DC4	$	4	D	T	d	t								
5	ENQ	NAK	%	5	E	U	e	u								
6	ACK	SYN	&	6	F	V	f	v								
7	BEL	ETB	'	7	G	W	g	w								
8	BS	CAN	(	8	H	X	h	x								
9	HT	EM	)	9	I	Y	i	y								
A	LF	SUB	*	:	J	Z	j	z								
B	VT	ESC	+	;	K	[	k	{								
C	FF	FS	,	<	L	\	l	|								
D	CR	GS	-	=	M	]	m	}								
E	SO	RS	.	>	N	^	n	~								
F	SI	US	/	?	O	_	o	DEL								

なぜこの並び？

JIS X 0201

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p				ー	タ	ミ		
1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
2	STX	DC2	"	2	B	R	b	r			「	イ	ツ	メ		
3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
4	EOT	DC4	$	4	D	T	d	t			、	エ	ト	ヤ		
5	ENQ	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BEL	ETB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	GS	-	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	RS	.	>	N	^	n	‾			ョ	セ	ホ	゛		
F	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ	゜		

ISO-8859-1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p			NBSP	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{			«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l	|		¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}			-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL			¯	¿	Ï	ß	ï	ÿ

ISO-8859-2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p			NBSP	°	Ŕ	Đ	ŕ	đ
1	SOH	DC1	!	1	A	Q	a	q			Ą	ą	Á	Ń	á	ń
2	STX	DC2	"	2	B	R	b	r			˘	˛	Â	Ň	â	ň
3	ETX	DC3	#	3	C	S	c	s			Ł	ł	Ă	Ó	ă	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			Ľ	ľ	Ĺ	Ő	Í	ő
6	ACK	SYN	&	6	F	V	f	v			Ś	ś	Ć	Ö	ć	ö
7	BEL	ETB	'	7	G	W	g	w			§	ˇ	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	Č	Ř	č	ř
9	HT	EM	)	9	I	Y	i	y			Š	š	É	Ů	é	ů
A	LF	SUB	*	:	J	Z	j	z			Ş	ş	Ę	Ú	ę	ú
B	VT	ESC	+	;	K	[	k	{			Ť	ť	Ë	Ű	ë	ű
C	FF	FS	,	<	L	\	l	|		Ź	ź	Ě	Ü	ě	ü
D	CR	GS	-	=	M	]	m	}			-	˝	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			Ž	ž	Î	Ţ	î	ţ
F	SI	US	/	?	O	_	o	DEL			Ż	ż	Ď	ß	ď	˙

いろんな文字コード表

- ひとつの表では ﾎﾟｹﾓﾝ Pokémon ポケモン を書けない
- 複数の表をまとめる良い方法は…

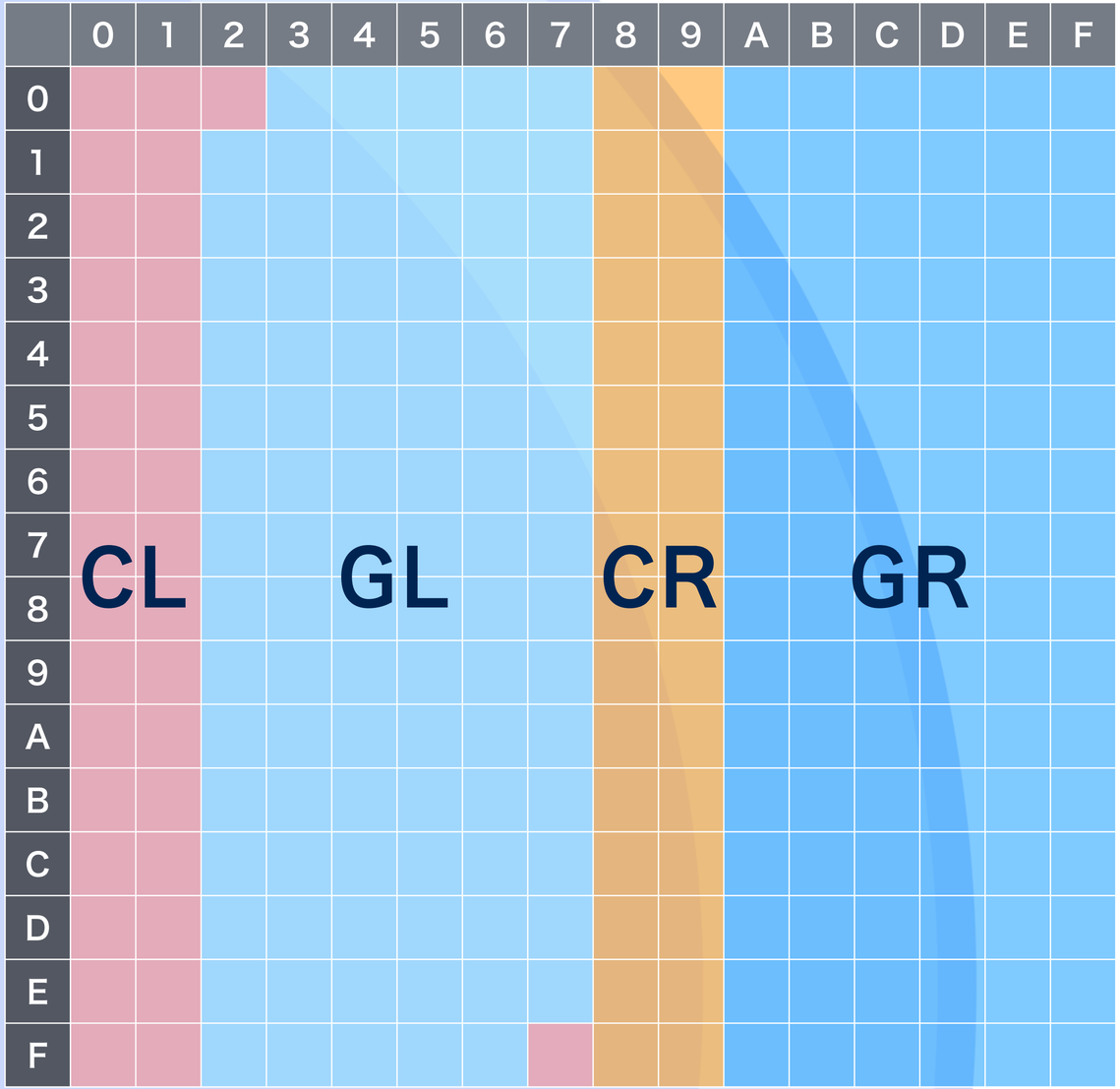
4つの領域にわけて

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p			NBSP	°	Ŕ	Đ	ŕ	đ
1	SOH	DC1	!	1	A	Q	a	q			Ą	ą	Á	Ń	á	ń
2	STX	DC2	"	2	B	R	b	r			˘	˛	Â	Ň	â	ň
3	ETX	DC3	#	3	C	S	c	s			Ł	ł	Ă	Ó	ă	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			Ľ	ľ	Ĺ	Ő	Í	ő
6	ACK	SYN	&	6	F	V	f	v			Ś	ś	Ć	Ö	ć	ö
7	BEL	ETB	'	7	G	W	g	w			§	ˇ	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	Č	Ř	č	ř
9	HT	EM	)	9	I	Y	i	y			Š	š	É	Ů	é	ů
A	LF	SUB	*	:	J	Z	j	z			Ş	ş	Ę	Ú	ę	ú
B	VT	ESC	+	;	K	[	k	{			Ť	ť	Ë	Ű	ë	ű
C	FF	FS	,	<	L	\	l	|		Ź	ź	Ě	Ü	ě	ü
D	CR	GS	-	=	M	]	m	}			-	˝	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			Ž	ž	Î	Ţ	î	ţ
F	SI	US	/	?	O	_	o	DEL			Ż	ż	Ď	ß	ď	˙

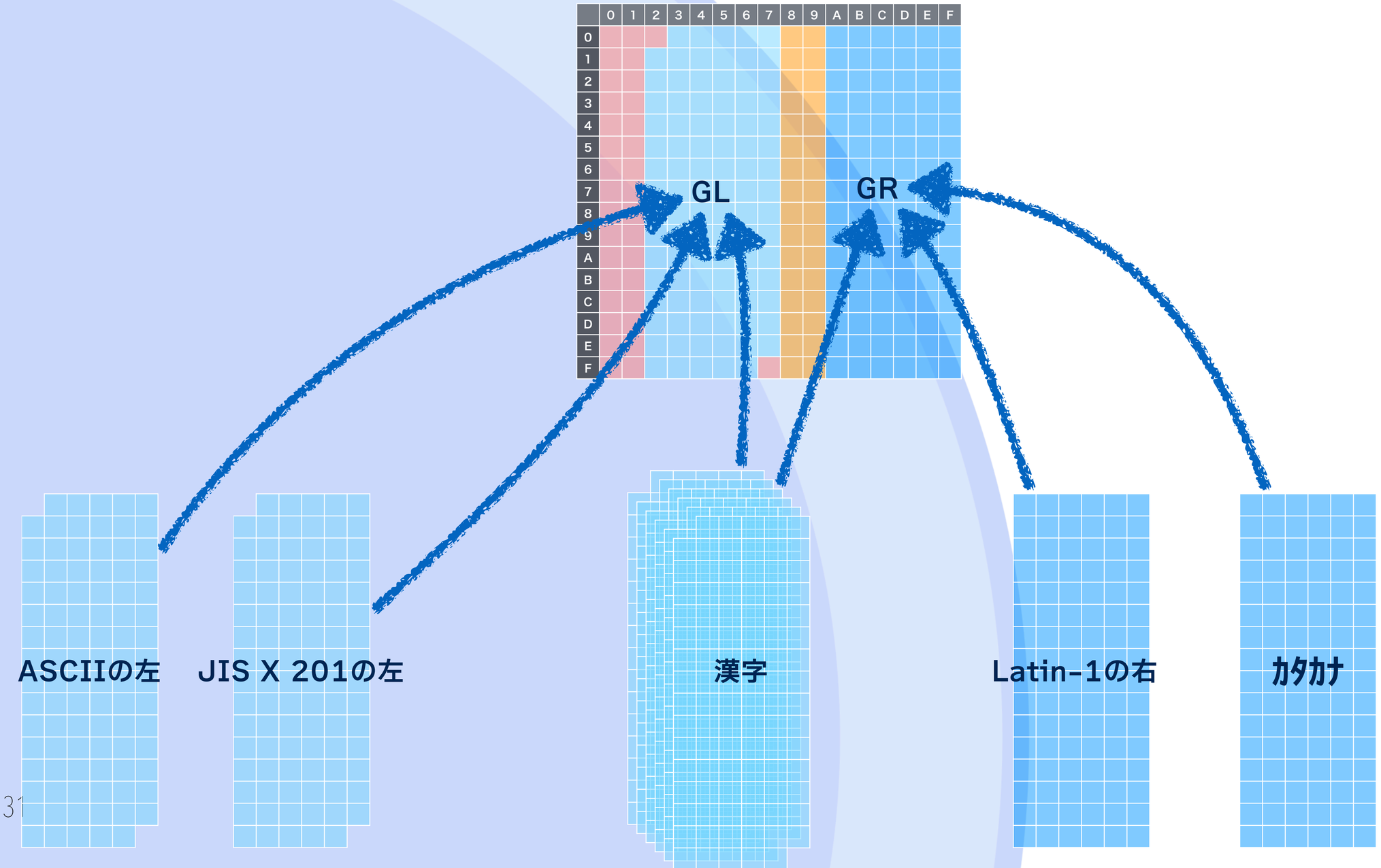
4つの領域にわけて

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p			NBSP	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{			«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l	|		¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}				½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL			¯	¿	Ï	ß	ï	ÿ

4つの領域にわけて

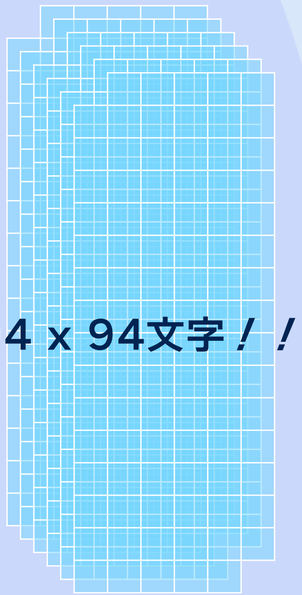


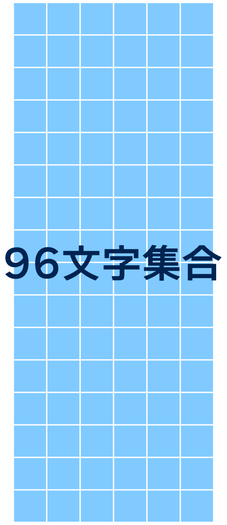
GLとGRの文字を交換しよう



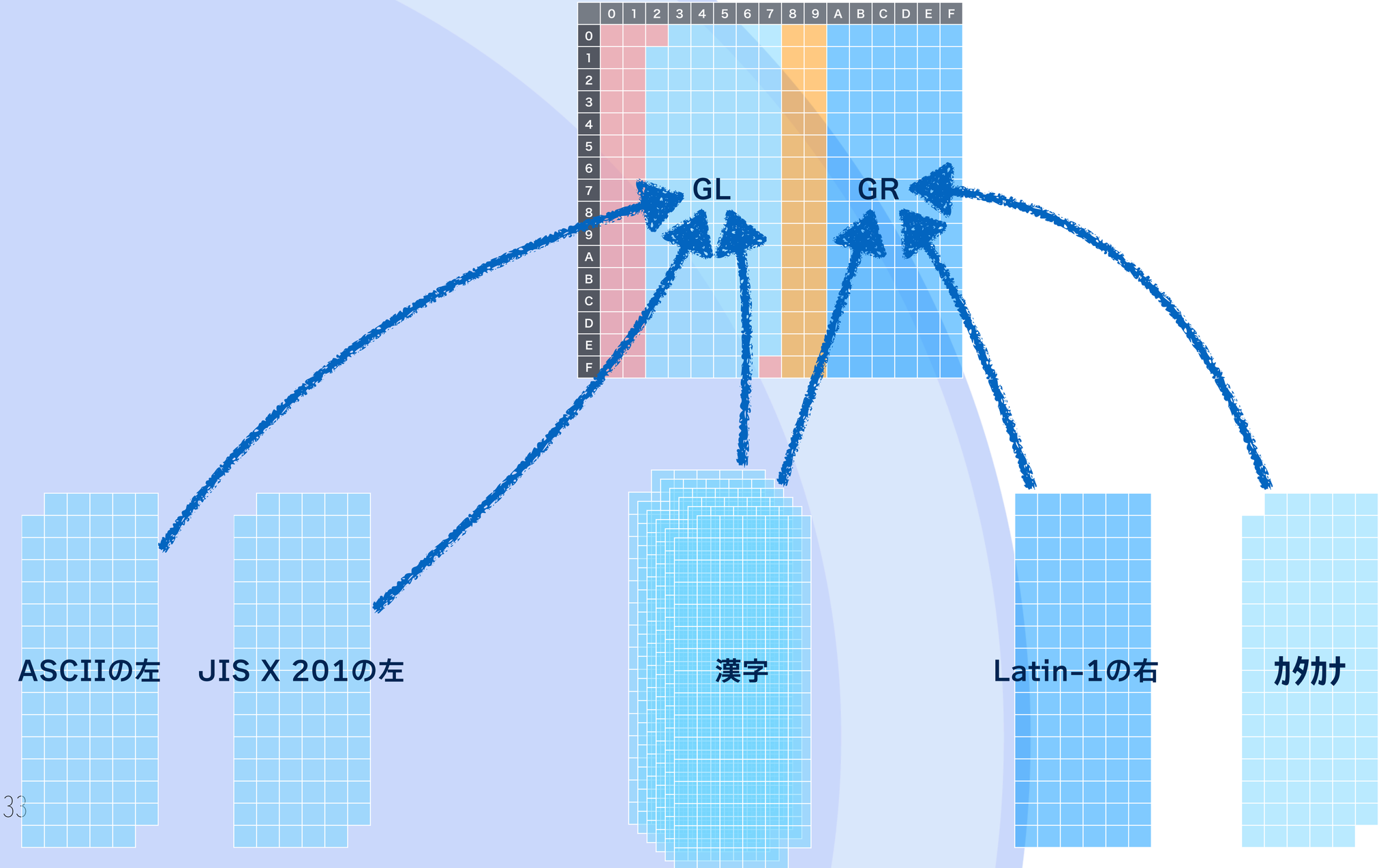
94文字集合と96文字集合

- 左にあったやつが94文字集合
- 右にあったやつが96文字集合
- 94はGRにもGLにも配置できるが96はGRだけ
- 漢字は94x94文字集合→「区点コード」

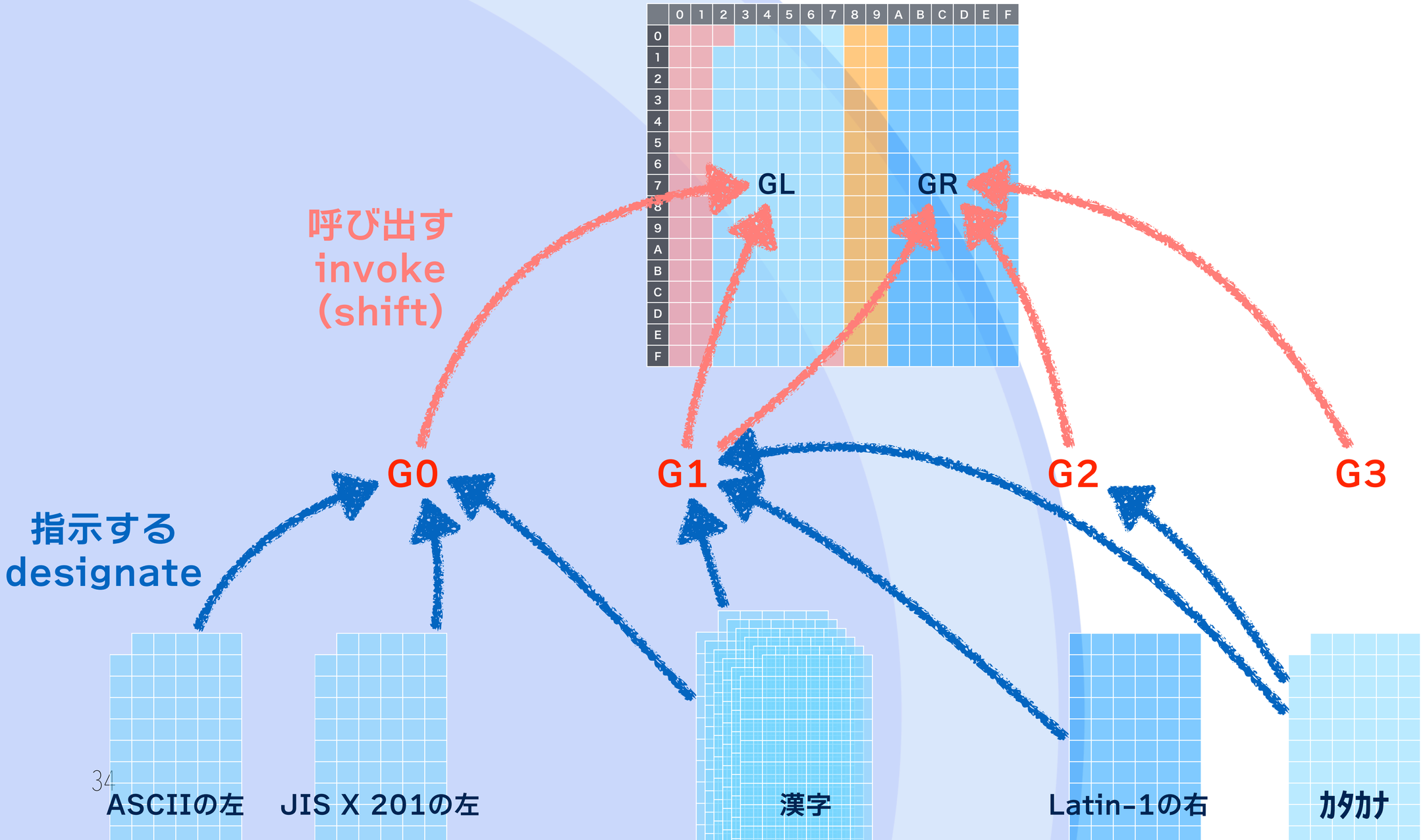




GLとGRの文字を交換しよう



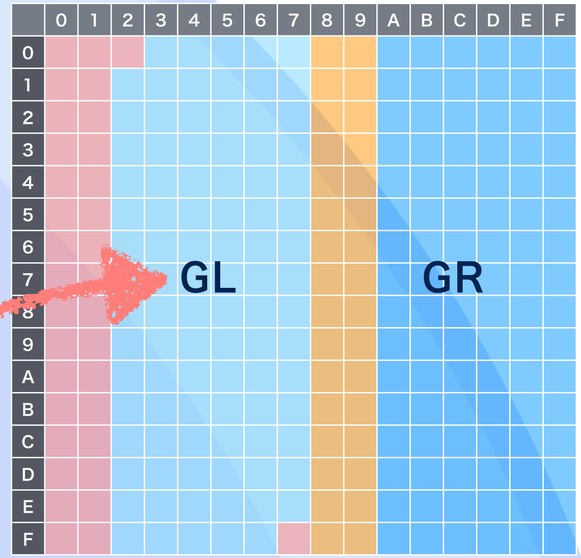
2段階で操作する


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F
GL
GR
呼び出す
invoke
(shift)
G0
G1
G2
G3
指示する
designate
ASCIIの左
JIS X 201の左
漢字
Latin-1の右
カタカナ

○●● 操作？

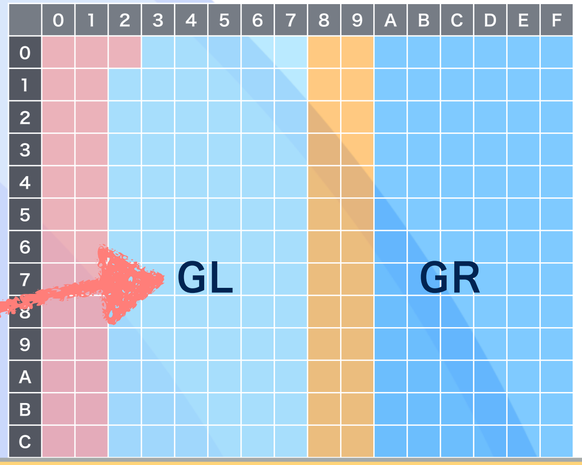
- 文字コードに決まった初期状態にする
- 文字列にCL/CRの制御文字を使って命令を埋め込む

iso-2022-jp



G0　G1　G2　G3

iso-2022-jp



初期状態
G0にASCIIを指示
GLにG0を呼び出す

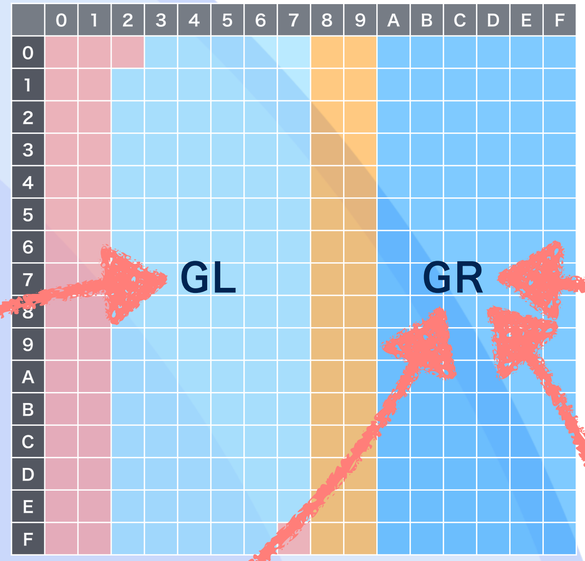
```
"分散Ruby".encode('iso-2022-jp')
1b 24 42 4a 2c 3b 36 1b 28 42 52 75 62 79
```

G0　G1　G2　G3

G0に漢字を指示
GLにG0を呼び出す

G0にASCIIを指示
GLにG0を呼び出す

euc-jp



G0

G1

G2

G3

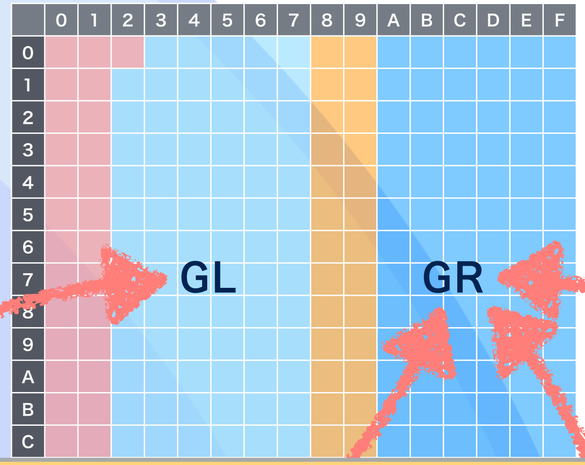
ASCIIの左

漢字
JIS X 0208

カタカナ

漢字
JIS X 0212

euc-jp




```
'分散ﾚﾋﾞｰ'.encode('euc-jp')
ca ac bb b6 8e d9 8e cb  8e de 8e b0
```

G0

G1

G2

G3

初期状態
G0にASCIIを指示
GLにG0を呼び出す
G1に漢字を指示
GRにG1を呼び出す

GRにG2を呼び出す
1文字で元に戻る
（シングルシフト）

ASCIIの左

漢字
JIS X 0208

ｶﾀｶﾅ

漢字
JIS X 0212

操作に見えてきた！

- 文字コードに決まった初期状態にする
- 文字列にCL/CRの制御文字を使って命令を埋め込む

初期状態
G0にASCIIを指示
GLにG0を呼び出す

```
"分散Ruby".encode('iso-2022-jp')
1b 24 42 4a 2c 3b 36 1b 28 42 52 75 62 79
```

G0に漢字を指示
GLにG0を呼び出す

G0にASCIIを指示
GLにG0を呼び出す

操作に見えてきた！

- 文字コードに決まった初期状態にする
- 文字列にCL/CRの制御文字を使って命令を埋め込む

```
"分散Ruby".encode('iso-2022-jp')
1b 24 42 4a 2c 3b 36 1b 28 42 52 75 62 79
```

- GL/GR, G0-G3を操作するプログラムぽい！

DICOMでのISO/IEC 2022

- DICOMではシングルシフトは使わない
- G0はGLにG1はGRに呼び出されている
- 属性(0008,0005)でその文書が使用する文字集合、デフォルトの文字集合を宣言する

つまりG2, G3は使わない。GLとGRに注目すれば良い

DICOMの文字列をRubyで編

- もうちょっとがんばるとRubyの話だよ
- DICOMの文字列(binary)を適切なencodingをもったRuby文字列に変換したい！
  - きょうのゴール

```
[["Pok", #<Encoding:US-ASCII>, "Pok"],
 ["\xE9", #<Encoding:ISO-8859-1>, "é"],
 ["mon ", #<Encoding:US-ASCII>, "mon "],
 ["\xCE\xDF\xB9\xD3\xDD", #<Encoding:CP50221 (dummy)>, "ﾎﾟｹﾓﾝ"]]
```

(0008,0005)

- 属性(0008,0005)でその文書が使用する文字集合、デフォルトの文字集合を宣言する
  - defined termと呼ばれる名前で記述
  - "ISO 2022 IR "で始まる名前は2022 Extension

- Extensionなしの場合はRubyのencodingで対応できる

Extensionなし

defined term	Ruby encoding
なし	ascii
ISO_IR 100	windows-1252
ISO_IR 101	iso-8859-2
ISO_IR 109	iso-8859-3
ISO_IR 110	iso-8859-4
ISO_IR 144	iso-8859-5
ISO_IR 127	iso-8859-6
ISO_IR 126	iso-8859-7
ISO_IR 138	iso-8859-8
ISO_IR 148	windows-1254
ISO_IR 203	iso-8859-15
ISO_IR 13	shift-jis (の一部)
ISO_IR 166	tis-620
ISO_IR 192	utf-8
GB18030	GB18030
GBK	gbk

IRはISOの登録番号らしい。なおISO_IR 13は0x0201カナなのでsjisではない

2022 Extension

- (0008,0005)に文字集合を複数指定する
- 列挙された文字集合が利用できる
- 先頭の文字集合が初期状態になる

```
["ISO 2022 IR 13", "ISO 2022 IR 87"]
```

デフォルトはASCII + ｶﾅ

漢字も使用するよ

- 先頭が省略されたときはIR 6(ASCII)を意味する

```
["", ISO 2022 IR 87", "ISO 2022 IR 13"]
```

デフォルトはIR 6(ASCII)

漢字とｶﾅも使用するよ

2022 Extension

- 文字集合ごとに使える操作が決められている
- 先頭に書いてある文字集合の操作が初期状態になる

defined term	ESC		
ISO 2022 IR 6	1b 28 42	G0/GL	ASCII
ISO 2022 IR 100	1b 2d 41	G1/GR	Latin-1の右
	1b 28 42	G0/GL	ASCII
ISO 2022 IR 13	1b 29 49	G1/GR	JIS X 0201 ｶﾀｶﾅ
	1b 28 4a	G0/GL	JIS X 0201 の左
ISO 2022 IR 87	1b 24 42	G0/GL	JIS X 0208 漢字
ISO 2022 IR 149	1b 24 29 43	G1/GR	韓国語 euc-kr

先頭がIR 13なら、G1にｶﾀｶﾅ、G0にJIS X 0201のローマ字が初期値になる

文字列中で使える操作

表は抜粋。defined termは16、操作は17ある

仕様わかった？

- GL/GRをそれぞれ独立して設定できるのがワナ

作戦

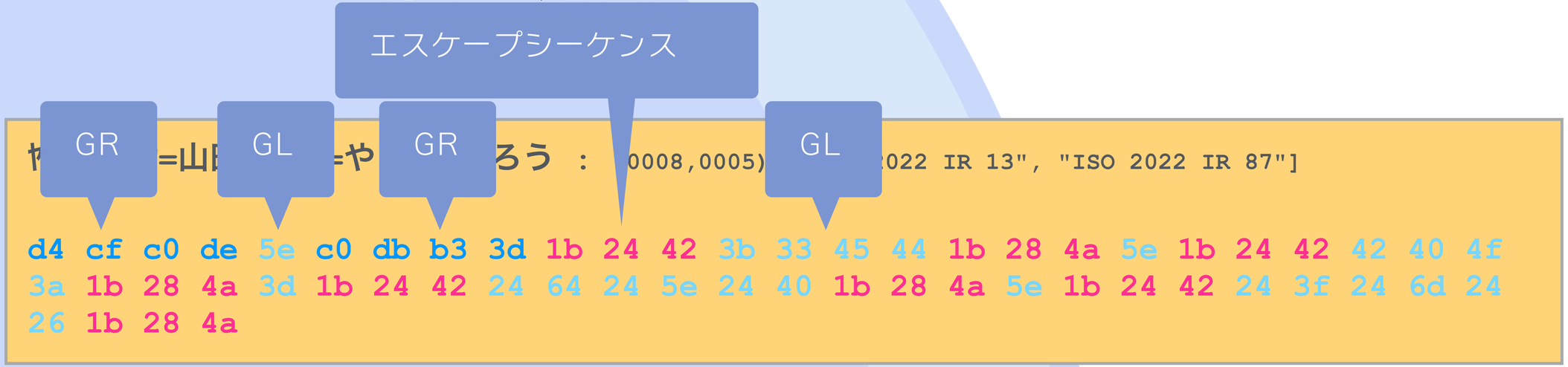
- じっと見る

```
ﾔﾏﾀﾞ^ﾀﾛｳ=山田^太郎=やまだ^たろう : (0008,0005) ["ISO 2022 IR 13", "ISO 2022 IR 87"]

d4 cf c0 de 5e c0 db b3 3d 1b 24 42 3b 33 45 44 1b 28 4a 5e 1b 24 42 42 40 4f
3a 1b 28 4a 3d 1b 24 42 24 64 24 5e 24 40 1b 28 4a 5e 1b 24 42 24 3f 24 6d 24
26 1b 28 4a
```

DICOMの規格書にある患者名の例

作戦

- (エスケープシーケンス)|(GL*)|(GR*) のセグメントに分けて処理すればよいのでは！



できそう！

- できた

```
charset = "ISO 2022 IR 100\\ISO 2022 IR 13"
str = %w(50 6f 6b e9 6d 6f 6e 20 1b 29 49 ce df b9 d3 dd).map(&:hex).pack('C*')
context = DCM_CharSet::Context.new(charset)
ctext = context.convert(str)
pp ctext
```

```
["Pok", "\xE9", "mon ", "\xCE\xDF\xB9\xD3\xDD"]
```

```
pp ctext.map {|x| [x, x.encoding, x.encode('utf-8')]}
```

```
[["Pok", #<Encoding:US-ASCII>, "Pok"],
 ["\xE9", #<Encoding:ISO-8859-1>, "é"],
 ["mon ", #<Encoding:US-ASCII>, "mon "],
 ["\xCE\xDF\xB9\xD3\xDD", #<Encoding:CP50221 (dummy)>, "ﾎﾟｹﾓﾝ"]]
```


Contextという変換器を作って、convertするAPIです

```
# coding: us-ascii
module DCM_CharSet
  class InvalidCharSet < RuntimeError; end
  class NotAllowDefaultCharSet < InvalidCharSet; end

  class Context
    def initialize(dcm_00080005)
      ary = parse_charset(dcm_00080005)

      if ary.size == 1
        @wo_extensions = true
        @encoding = CharactorSetWOExtensions[ary.first]
      else
        @default_encoding = CharactorSet[ary.first]
        @wo_extensions = false
        @allow_encoding = {}
        ary.each do |x|
          CharactorSet[x].each do |y|
            @allow_encoding[y.escape_sequence] = y
          end
        end
        seg = @allow_encoding.keys.map{Regexp.escape(_1)} + ['[\000-\037]+', '[\200-\237]+', '[\040-\177]+', '[\200-\377]+']

        @reg = Regexp.new(seg.join('|'), 0)
      end
    end

    def parse_charset(charset)
      ary = charset ? charset.strip.split('\\').map {|x| x.strip.upcase} : []
      return ['ISO_IR 6'] if ary.empty?
      if ary.size == 1
        raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
        return ary
      end
      ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
      raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
      ary.each do |x|
        raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
      end
```

コードを説明します

```
        else
          result << graphic['GR'].encode(seg)
        end
      end

      result
    end

    def convert_wo_extensions(str)
      [str.force_encoding(@encoding)]
    end
  end
end

module DCM_CharSet
  class Element
    def initialize(code_element, escape_sequence, encoding)
      @code_element = code_element
      @escape_sequence = escape_sequence.pack('c*')
      @encoding = encoding
    end
    attr_reader :escape_sequence, :encoding, :code_element

    def _encode(str)
      str.dup.force_encoding(@enco
    end

    def encode(str)
      s = _encode(str)
      s.instance_variable_set(:@dicom_encoding_element, self)
      s.freeze
      s
    end

    def inspect
      "#<#{self.class.to_s}:#{@escape_sequence.inspect} #{@encoding}>"
    end
  end

  module E_shift_to_GR
```

DCM_CharSet::Element

Elementはこれ！
"GL"か"GR"を示すcode_elementと
エスケープシーケンスとRubyのencodingの
3つの属性を持つ

Element#encodeで
自分に設定された方法でStringをencodeする
これが主な仕事

変換のはじめ / GLとGRの初期化

```
    return ['ISO_IR 6'] if ary.empty?
    if ary.size == 1
      raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
      return ary
    end
    ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
    raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
    ary.each do |x|
      raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
    end
    ary
  end

  def convert(str)
    return convert_wo_extensions(str) if @wo_extensions

    graphic = @default_encoding.map {|e| [e.code_element, e]}.to_h

    result = []
    str.scan(@reg).each do |seg|
      case seg.bytes.first
      when 033
        e = @allow_encoding[seg]
        graphic[e.code_element] = e
      when 0..037, 0200..0237 # CL, CR
        result << seg
      when 040..0177
        result << graphic['GL'].encode(seg)
      else
        result << graphic['GR'].encode(seg)
      end
    end

    result
  end

  def convert_wo_extensions(str)
    [str].force_encoding(@encoding)]
  end
end
```

GL, GRごとの変換方法の初期化
{
  "GL" => DCMCharSet::Element,
  "GR" => DCMCharSet::Element
}
といったHash

graphicが「操作」される対象だよ

セグメントに分けて処理する

セグメントに分けて処理するイテレータ
scanがぴったりくるぞ！
正規表現@regは後述

セグメントの種類ごとの分岐

```
    return ['ISO_IR 6'] if ary.empty?
    if ary.size == 1
      raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
      return ary
    end
    ary[0] = 'ISO 2022 IR 6' if ary[0] == ''
    raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
    ary.each do |x|
      raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
    end
    ary
  end

                                                    ensions

                                                    _element, e]}.to_h

    result = []
    str.scan(@reg).each do |seg|
      case seg.bytes.first
      when 033
        e = @allow_encoding[seg]
        graphic[e.code_element] = e
      when 0..037, 0200..0237 # CL, CR
        result << seg
      when 040..0177
        result << graphic['GL'].encode(seg)
      else
        result << graphic['GR'].encode(seg)
      end
    end

    result
  end

  def convert_wo_extensions(str)
    [str.force_encoding(@encoding)]
  end
end
```

セグメントに分けて処理する

エスケープシーケンスの場合
graphicの設定を変更する「操作」をする

対応するElementを求めて、
Elementを対応するGL/GRに覚える

```
    return ['ISO_IR 6'] if ary.empty?
    if ary.size == 1
      raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
      return ary
    end
    ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
    raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
    ary.each do |x|
      raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
    end
  e
  d
                                                    ons
                                                    ment, e]}.to_h
      case seg.bytes.first
      when 033
        e = @allow_encoding[seg]
        graphic[e.code_element] = e
      when 0..037, 0200..0237 # CL, CR
        result << seg
      when 040..0177
        result << graphic['GL'].encode(seg)
      else
        result << graphic['GR'].encode(seg)
      end
    end

    result
  end

  def convert_wo_extensions(str)
    [str.force_encoding(@encoding)]
  end
end
```

セグメントに分けて処理する

```
    return ['ISO_IR 6'] if ary.empty?
    if ary.size == 1
      raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
      return ary
    end
    ary[0] = 'ISO 2022 IR 6' if ary[0] == ''
    raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
    ary.each do |x|
      raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
    end
    ary
  end

  def convert(str)
    return convert_wo_extensions(str) if @wo_extensions

    graphic = @default_encoding.map {|e| [e.code_element, e]}.to_h

    result = []
    str.scan(@reg).each do |seg|
      case seg.bytes.first
      when 033
        e = @allow_encoding[seg]
        graphic[e.code_element] = e
      when 0..037, 0200..0237 # CL, CR
        result << seg
      when 040..0177
        result << graphic['GL'].encode(seg)
      else
        result << graphic['GR'].encode(seg)
      end
    end

    result
  end

  def convert_wo_extensions(str)
    [str.force_encoding(@encoding)]
  end
end
```

CL/CRのときは変換せずに連結

セグメントに分けて処理する

```
    return ['ISO_IR 6'] if ary.empty?
    if ary.size == 1
      raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
      return ary
    end
    ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
    raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
    ary.each do |x|
      raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
    end
    ary
  end

  def convert(str)
    return convert_wo_extensions(str) if @wo_extensions

    graphic = @default_encoding.map {|e| [e.code_element, e]}.to_h

    result = []
    str.scan(@reg).each do |seg|
      case seg.bytes.first
      when 033
        e = @allow_encoding[seg]
        graphic[e.code_element] = e
      when 0..037, 0200..0237 # CL, CR
        result << seg
      when 040..0177
        result << graphic['GL'].encode(seg)
      else
        result << graphic['GR'].encode(seg)
      end
    end

    result
  end

  def convert_wo_extensions(str)
    [str.force_encoding(@encoding)]
  end
end
```

graphicの'GL'に設定されている
Elementでencodeする

'GR'も同様だよ

```
# coding: us-ascii
module DCM_CharSet
  class InvalidCharSet < RuntimeError; end
  class NotAllowDefaultCharSet < InvalidCharSet; end

  class Context
    def initialize(dcm_00080005)
      ary = parse_charset(dcm_00080005)

      if ary.size == 1
        @wo_extensions = true
        @encoding = CharactorSetWOExtensions[ary.first]
      else
        @default_encoding = CharactorSet[ary.first]
        @wo_extensions = false
        @allow_encoding = {}
        ary.each do |x|
          CharactorSet[x].each do |y|
            @allow_encoding[y.escape_sequence] = y
          end
        end
        seg = @allow_encoding.keys.map{Regexp.escape(_1)} + ['[\000-\037]+', '[\200-\237]+', '[\040-\177]+', '[\200-\377]+']

        @reg = Regexp.new(seg.join('|'), 0)
      end
    end

    def parse_charset(charset)
      ary = charset ? charset.strip.split('\\').map {|x| x.strip.upcase} : []
      return ['ISO_IR 6'] if ary.empty?
      if ary.size == 1
        raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
        return ary
      end
      ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
      raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
      ary.each do |x|
        raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
      end
```

@regの準備

```
# coding: us-ascii
module DCM_CharSet
  class InvalidCharSet < RuntimeError; e
  class NotAllowDefaultCharSet < Invalid

  class Context
    def initialize(dcm_00080005)
      ary = parse_charset(dcm_00080005)

      if ary.size == 1
        @wo_extensions = true
        @encoding = CharactorSetWOExtens
      else
        @default_encoding = CharactorSet
        @wo_extensions = false
        @allow_encoding = {}
        ary.each do |x|
          CharactorSet[x].each do |y|
            @allow_encoding[y.escape_sequence] = y
          end
        end
        seg = @allow_encoding.keys.map{Regexp.escape(_1)} + ['[\000-\037]+', '[\200-\237]+', '[\040-\177]+', '[\200-\377]+']

        @reg = Regexp.new(seg.join('|'), 0)
      end
    end

    def parse_charset(charset)
      ary = charset ? charset.strip.split('\\').map {|x| x.strip.upcase} : []
      return ['ISO_IR 6'] if ary.empty?
      if ary.size == 1
        raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
        return ary
      end
      ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
      raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
      ary.each do |x|
        raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
      end
```

DICOM文字列の変換器のクラス
(0008,0005)の文字集合の設定が引数です

parse_charsetで(0008,0005)の設定から文字集合の名前のArrayに分割する

@regの準備

```
# coding: us-ascii
module DCM_CharSet
  class InvalidCharSet < RuntimeError; end
  class NotAllowDefaultCharSet < InvalidCharSet; end

  class Context
    def initialize(dcm_00080005)
      ary = parse_charset(dcm_0008000
      if ary.size == 1
        @wo_extensions = true
        @encoding = CharactorSetWOExtensions[ary.first]
      else
        @default_encoding = CharactorSet[ary.first]
        @wo_extensions = false
        @allow_encoding = {}
        ary.each do |x|
          CharactorSet[x].each do |y|
            @allow_encoding[y.escape_sequence] = y
          end
        end
        seg = @allow_encoding.keys.map{Regexp.escape(_1)} + ['[\000-\037]+', '[\200-\237]+', '[\040-\177]+', '[\200-\377]+']

        @reg = Regexp.new(seg.join('|'), 0)
      end
    end

    def parse_charset(charset)
      ary = charset ? charset.strip.split('\\').map {|x| x.strip.upcase} : []
      return ['ISO_IR 6'] if ary.empty?
      if ary.size == 1
        raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
        return ary
      end
      ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
      raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
      ary.each do |x|
        raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
      end
```

extensionなしのケースは割愛

@regの準備

```
# coding: us-ascii
module DCM_CharSet
  class InvalidCharSet < RuntimeError; end
  class NotAllowDefaultCharSet < InvalidCharSet; end

  class Context
    def initialize(dcm_00080005)
      ary = parse_charset(dcm_00080005)

      if ary.size == 1
        @wo_extensions = true
        @encoding = CharactorSetWOExtensions[ary.first]
      else
        @default_encoding = CharactorSet[ary.first]
        @wo_extensions = false
        @allow_encoding = {}
        ary.each do |x|
          CharactorSet[x].each do |y|
            @allow_encoding[y.escape_sequence] = y
          end
        end
        seg = @allow_encoding.keys.map{Regexp.escape(_1)} + ['[\000-\037]+', '[\200-\237]+', '[\040-\177]+', '[\200-\377]+']

        @reg = Regexp.new(seg.join('|'), 0)
      end
    end

    def parse_charset(charset)
      ary = charset ? charset.strip.split('\\').map {|x| x.strip.upcase} : []
      return ['ISO_IR 6'] if ary.empty?
      if ary.size == 1
        raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
        return ary
      end
      ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
      raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
      ary.each do |x|
        raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
      end
```

CharactorSetは文字集合の名前からElementを引くHash（後述）

この文書で使用可能なElementを集めて表（@allow_encoding）を作る。エスケープシーケンスからElementを引くHashである

@regの準備

```
# coding: us-ascii
module DCM_CharSet
  class InvalidCharSet < RuntimeError; end
  class NotAllowDefaultCharSet < InvalidCharSet; end

  class Context
    def initialize(dcm_00080005)
      ary = parse_charset(dcm_00080005)

      if ary.size == 1
        @wo_extensions = true
        @encoding = CharactorSetWOExtensions[ary.first]
      else
        @default_encoding = CharactorS
        @wo_extensions = false
        @allow_encoding = {}
        ary.each do |x|
          CharactorSet[x].each do |y|
            @allow_encoding[y.escape_s
          end
        end
        seg = @allow_encoding.keys.map{Regexp.escape( 1)} + ['[\000-\037]+', '[\200-\237]+', '[\040-\177]+', '[\200-\377]+']

        @reg = Regexp.new(seg.join('|'), 0)
      end
    end

    def parse_charset(charset)
      ary = charset ? charset.strip.split('\\').map {|x| x.strip.upcase} : []
      return ['ISO_IR 6'] if ary.empty?
      if ary.size == 1
        raise(InvalidCharSet.new(ary.first)) unless CharactorSetWOExtensions.include?(ary.first)
        return ary
      end
      ary[0] = 'ISO 2022 IR 6' if ary[0].empty?
      raise(NotAllowDefaultCharSet.new(ary[0])) if MultibyteCharactorSet.include?(ary[0])
      ary.each do |x|
        raise(InvalidCharSet.new(x)) unless CharactorSet.include?(x)
      end
```

この文書で使用するエスケープシーケンス、CL, CR, GL, GRの正規表現を | で連結して、@reg を作る

@regの準備

CharactorSet

DICOMの文字集合の名前から、対応する操作（Element）のリストを引くHash

```
  'ISO_IR 127' => 'iso-8859-6',
  'ISO_IR 126' => 'iso-8859-7',
                                    IXME
}

AsciiElement = Element.new('GL', [0x1B, 0x28, 0x42], 'ascii')

CharactorSet = {
  'ISO 2022 IR 6' => [AsciiElement],

  'ISO 2022 IR 100' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x41], 'iso-8859-1')
  ],

  'ISO 2022 IR 101' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x42], 'iso-8859-2')
  ],

  'ISO 2022 IR 109' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x43], 'iso-8859-3')
  ],

  'ISO 2022 IR 110' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x44], 'iso-8859-4')
  ],

  'ISO 2022 IR 144' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x4C], 'iso-8859-5')
```

CharactorSet

```
  'ISO_IR 127' => 'iso-8859-6',
  'ISO_IR 126' => 'iso-8859-7',
  'ISO_IR 138' => 'iso-8859-8',
  'ISO_IR 148' => 'windows-1254', # FIXME
  'ISO_IR 203' => 'iso-8859-15',
  'ISO_IR 13' => 'shift-jis', #FIXME
  'ISO_IR 166' => 'tis-620',
  'ISO_IR 192' => 'utf-8',
  'GB18030' => 'GB18030',
  'GBK' =>
}

AsciiElement = Element.new('GL', [0x1B, 0x28, 0x42], 'ascii')

CharactorSet = {
  'ISO 2022 IR 6' => [AsciiElement],

  'ISO 2022 IR 100' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x41], 'iso-8859-1')
  ],

  'ISO 2022 IR 101' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x42], 'iso-8859-2')
  ],

  'ISO 2022 IR 109' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x43], 'iso-8859-3')
  ],

  'ISO 2022 IR 110' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x44], 'iso-8859-4')
  ],

  'ISO 2022 IR 144' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x4C], 'iso-8859-5')
```

呼び出し先

エスケープシーケンス

Rubyのencoding

asciiへの操作はなんども出るのでメモしとく

CharactorSet

```
  'ISO_IR 127' => 'iso-8859-6',
  'ISO_IR 126' => 'iso-8859-7',
  'ISO_IR 138' => 'iso-8859-8',
  'ISO_IR 148' => 'windows-1254', # FIXME
  'ISO_IR 203' => 'iso-8859-15',
  'ISO_IR 13' => 'shift-jis', #FIXME
  'ISO_IR 166' => 'tis-620',
  'ISO_IR 192' => 'utf-8',
  'GB18030' => 'GB18030',
  'GBK' => 'gbk'
}

AsciiElement = Element.new('GL', [0x1B, 0x28, 0x42], 'ascii')

CharactorSet = {
  'ISO 2022 IR 6' => [AsciiElement],

  'ISO 2022 IR 100' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x41], 'iso-8859-1')
  ],

  'ISO
    As
    El                                    o-8859-2')
  ],

  'ISO
    As
    El                                    o-8859-3')
  ],

  'ISO 2022 IR 110' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x44], 'iso-8859-4')
  ],

  'ISO 2022 IR 144' => [
    AsciiElement,
    Element.new('GR', [0x1B, 0x2D, 0x4C], 'iso-8859-5')
```

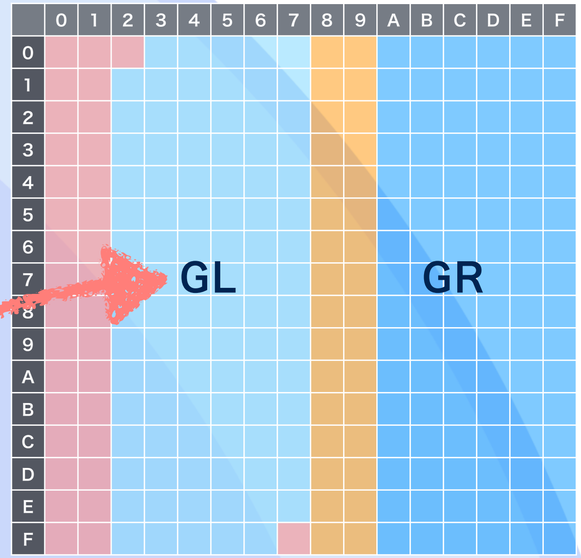
ISO 2022 IR 100は
GLにasciiを呼び出す操作
GRにIR 100を呼び出す操作
で構成される

ちょっと苦労したとこ

```
  Element.new('GR', [0x1B, 0x2D, 0x46], 'iso-8859-7')
],

'ISO 2022 IR 138' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x48], 'iso-8859-8')
],

'ISO 2022 IR 148' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x4D], 'iso-8859-9')
],

'ISO 2022 IR 203' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x62], 'iso-8859-15')
],

'ISO 2022 IR 13' => [
  Element.new('GL', [0x1B, 0x28, 0x4A], 'cp50221'),
  Element.new('GR', [0x1B, 0x29, 0x49], 'cp50221')
],

'ISO 2022 IR 166' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x54], 'tis-620')
],

'ISO 2022 IR 87' => [
  Element.new('GL', [0x1B, 0x24, 0x42], 'euc-jp').extend(E_shift_to_GR)
],

'ISO 2022 IR 159' => [
  Element.new('GL', [0x1B, 0x24, 0x28, 0x44], 'euc-jp').extend(E_shift_to_GR)
],

'ISO 2022 IR 149' => [
  Element.new('GR', [0x1B, 0x24, 0x29, 0x43], 'euc-kr')
],
```

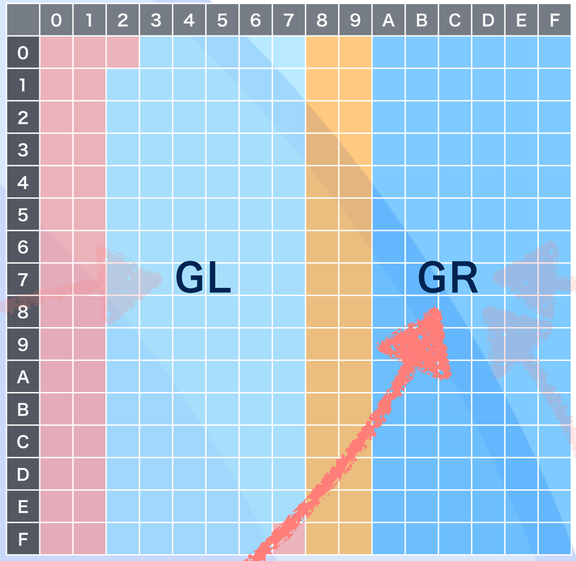
IR 87, 159はGLなんだけど、GRへシフトして
euc-jpとして処理することにした

iso-2022-jp



G0 G1 G2 G3

euc-jp



G0

G1

G2

G3

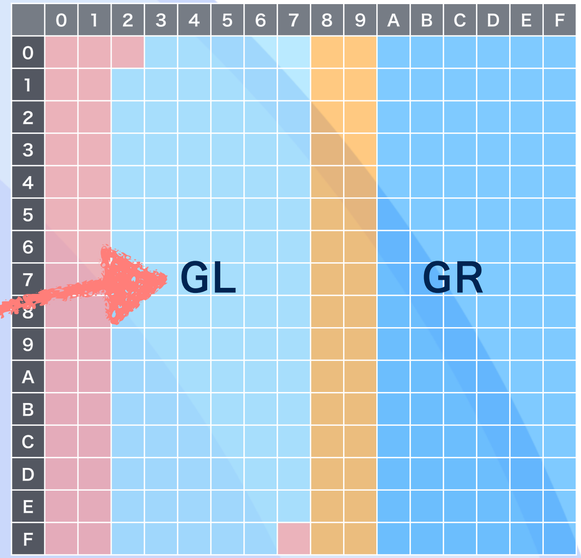
ASCIIの左

漢字
JIS X 0208

カタカナ

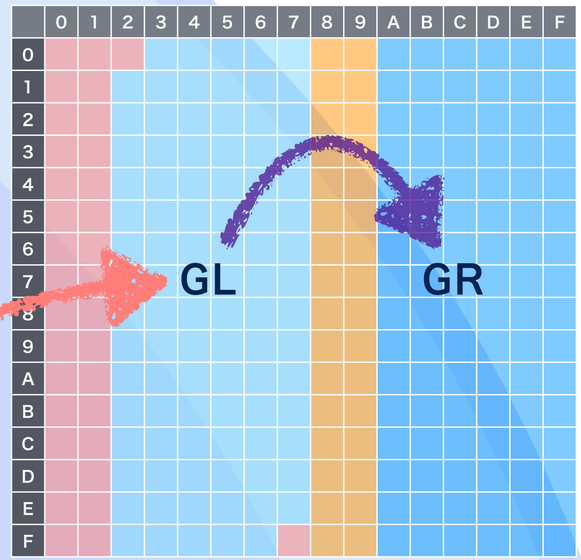
漢字
JIS X 0212

iso-2022-jp



G0 G1 G2 G3

GRへ移動してeuc-jpとして扱う



G0　G1　G2　G3

```
  Element.new('GR', [0x1B, 0x2D, 0x46], 'iso-8859-7')
],

'ISO 2022 IR 138' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x48], 'iso-8859-8')
],

'ISO 2022 IR 148' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x4D], 'iso-8859-9')
],

'ISO 2022 IR 203' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x62], 'iso-8859-15')
],

'ISO 2022 IR 13' => [
  Element.new('GL', [0x1B, 0x28, 0x4A], 'cp50221'),
  Element.new('GR', [0x1B, 0x29, 0x49], 'cp50221')
],

'ISO 2022 IR 166' => [
  AsciiElement,
  Element.new('GR', [0x1B, 0x2D, 0x54], 'tis-620')
],

'ISO 2022 IR 87' => [
  Element.new('GL', [0x1B, 0x24, 0x42], 'euc-jp').extend(E_shift_to_GR)
],

'ISO 2022 IR 159' => [
  Element.new('GL', [0x1B, 0x24, 0x28, 0x44], 'euc-jp').extend(E_shift_to_GR)
],

'ISO 2022 IR 149' => [
  Element.new('GR', [0x1B, 0x24, 0x29, 0x43], 'euc-kr')
],
```

ちょっと苦労したとこ

IR 87, 159はGLなんだけど、GRへシフトして
euc-jpとして処理することにした

ちょっと苦労したとこ

```
    end
  end
end

module DCM_CharSet
  class Element
    def initialize(code_element, escape_sequence, encoding)
      @code_element = code_element
      @escape_sequence = escape_sequence.pack('c*')
      @encoding = encoding
    end
    attr_reader :escape_sequence, :encoding, :code_element

    def _encode(str)
      str.dup.force_encoding(@encoding)
    end

    def encode(str)
      s = _encode(str)
      s.instance_variable_set(:@dicom_encoding_element, self)
      s.freeze
      s
    end

    def inspect
      "#<#{self.class.to_s}:#{@escape_sequence.inspect} #{@enco
    end
  end

  module E_shift_to_GR
    def _encode(str)
      str.each_byte.map {|x| x > 0x20 ? x | 0x80 : x}.pack('c*').force_encoding(@encoding)
    end
  end

  CharactorSetWOExtensions = {
    'ISO_IR 6' => 'ascii',
    'ISO_IR 100' => 'windows-1252',
    'ISO_IR 101' => 'iso-8859-2',
    'ISO_IR 109' => 'iso-8859-3',
```

moduleの中身はこれ。x | 0x80

ここまでのまとめ

- DICOMの複数文字集合な文字列をRubyで読めるようになった
  - GL/GRは独立して変わるワナがある
  - scanでセグメントにわけたらできたよ

やり残してることがある

- 変換結果がイケテナイ

```
["Pok", "\xE9", "mon ", "\xCE\xDF\xB9\xD3\xDD"]
```

  - Stringのように見えるラッパーが要る？
  - X11/Xtのcompound textとか参考になるのか！？

- 出力（DICOMへの変換）がない
  - utf-8への変換はできるけど、任意の(0008,0005)で変換できるようにしたい

○●● おわり

- DICOMとISO/IEC 2022を少しだけ紹介して複数文字集合の文字列をRubyで読む、というお話をしました
- 身近にあるちょっとした問題を解くのは楽しい

前提の規格を調べて説明するのはちょっと大変だった

今後

- textbringerが複数文字集合を直接扱えるようになり、令和のmuleとして人気が出るのを願っています