UnicodeとPRECIS Frameworkを用いた国際化文字列の前処理方法(前編)

Takahiro Nemoto, Ph.D.
Institute of Information and Media,
Aoyama Gakuin University

自己紹介

名前: 根本 貴弘(Takahiro NEMOTO)

所属: 青山学院大学 情報メディアセンター

その他の所属や活動:

- 慶應義塾大学メディアデザイン研究所 研究員
- 私立大学情報教育協会情報セキュリティ研究講習会運営委員会 委員
- Internet Society Japan Chapter(ISOC-JP) Officer
- ISOC-JP IETF Education Working Group Chair

Internet Engineering Task Force (IETF)との関わり:

IETF83@Paris ～ IETF97@Seoul

RFC7790 共著者

RFC7790: Mapping Characters for Classes of the Preparation, Enforcement, and Comparison of Internationalized Strings (PRECIS)

本日の内容

インターネットが様々な地域で利用されるようになり，国際化ドメイン名や国際化メールアドレスをはじめASCII文字集合の範囲外の文字を含む識別子の利用が求められるようになっています．本ワークショップでは，Unicodeによる国際化文字列の特徴とIETFで2015年にRFC化されたPRECIS(PREparation and Comparison of Internationalized Strings) Frameworkを用いた国際化文字列の前処理方法の概観及びその適用方法記載したプロトコルプロファイルの事例を紹介します．

1. 情報処理における文字列表現
2. Unicodeによる国際化文字列の特徴

←前編で扱った範囲

3. PRECIS Framework

インターネット上のサービスにおける一般的な文字表現

本文：母語で使用する文字

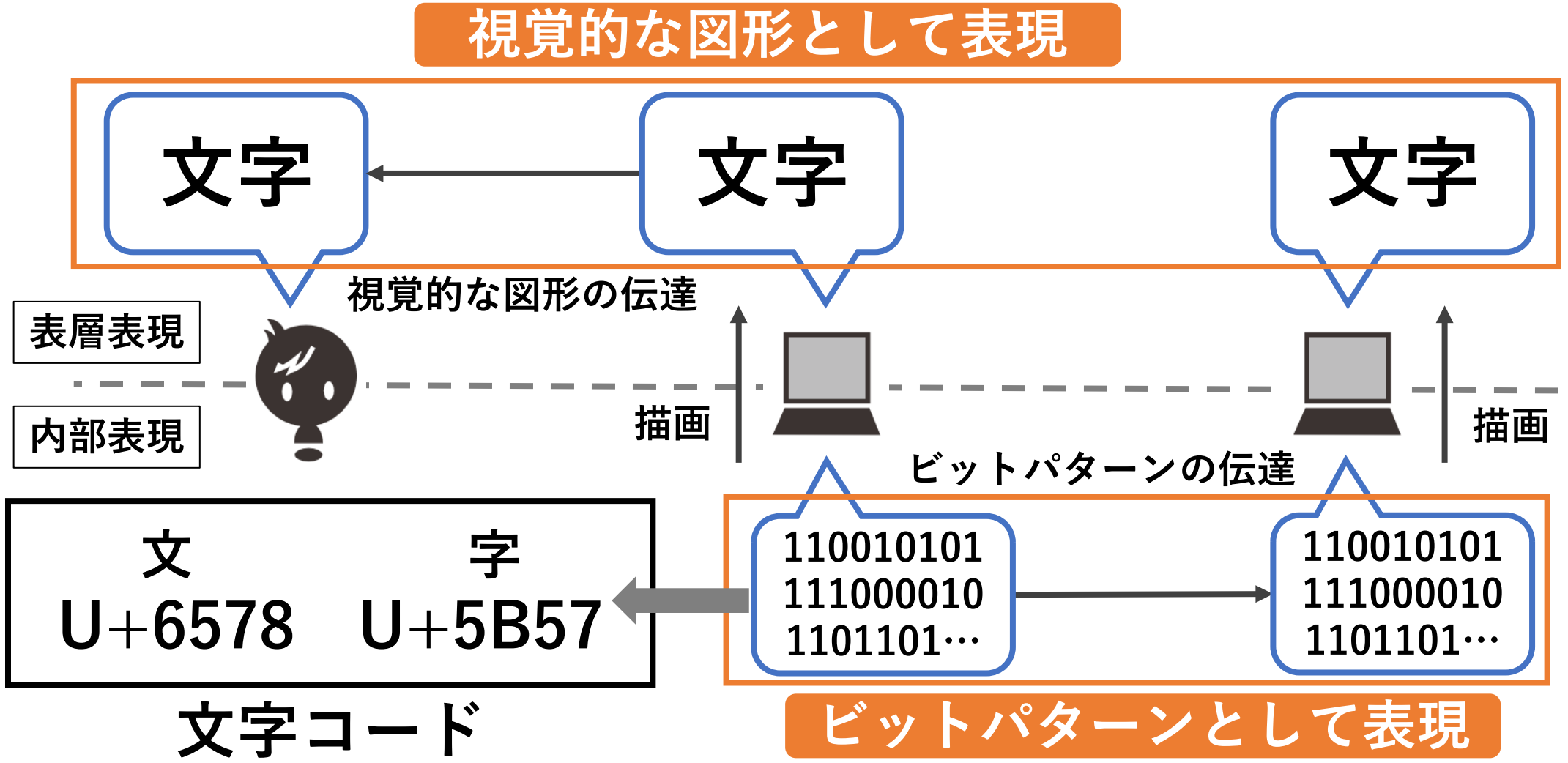
識別子：英数字が中心

画像出典：https://www.isoc.jp/

本文 vs 識別子

- ある集合から一意に区別することが求められるか否か
  - 本文：　一意に区別するすることは求められない
  - 識別子：　一意に区別することが求められる

情報処理における文字列表現



文字コードと文字集合(1/2)

文字コード：文字と対応したビットパターン

文字集合：　文字コードを重複なく集めた集合

- **ASCII文字集合の特徴**
  - 英数字を主とした情報交換を行うことを目的とした文字集合
  - 1文字を7bitで表現
    - 128通りのビットパターンが表現可能
  - インターネット上の識別子として広く使用
    - 比較が容易(大文字・小文字の区別程度)

					b7	0	0	0	0	1	1	1	1
					上位3ビット b6	0	0	1	1	0	0	1	1
					b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		下位4ビット	0	1	2	3	4	5	6	7
0	0	0	0	0		NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2		STX	DC2	"	2	B	R	b	r
0	0	1	1	3		ETX	DC3	#	3	C	S	c	s
0	1	0	0	4		EOT	DC4	$	4	D	T	d	t
0	1	0	1	5		ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6		ACK	SYN	&	6	F	V	f	v
0	1	1	1	7		BEL	ETB	’	7	G	W	g	w
1	0	0	0	8		BS	CAN	(	8	H	X	h	x
1	0	0	1	9		HT	EM	)	9	I	Y	i	y
1	0	1	0	A		LF	SUB	*	:	J	Z	j	z
1	0	1	1	B		VT	ESC	+	;	K	[	k	{
1	1	0	0	C		FF	FS	,	<	L	\	l	|
1	1	0	1	D		CR	GS	-	=	M	]	m	}
1	1	1	0	E		SO	RS	.	>	N	^	n	~
1	1	1	1	F		SI	US	/	?	O	_	o	DEL

ASCII文字集合の文字コード表

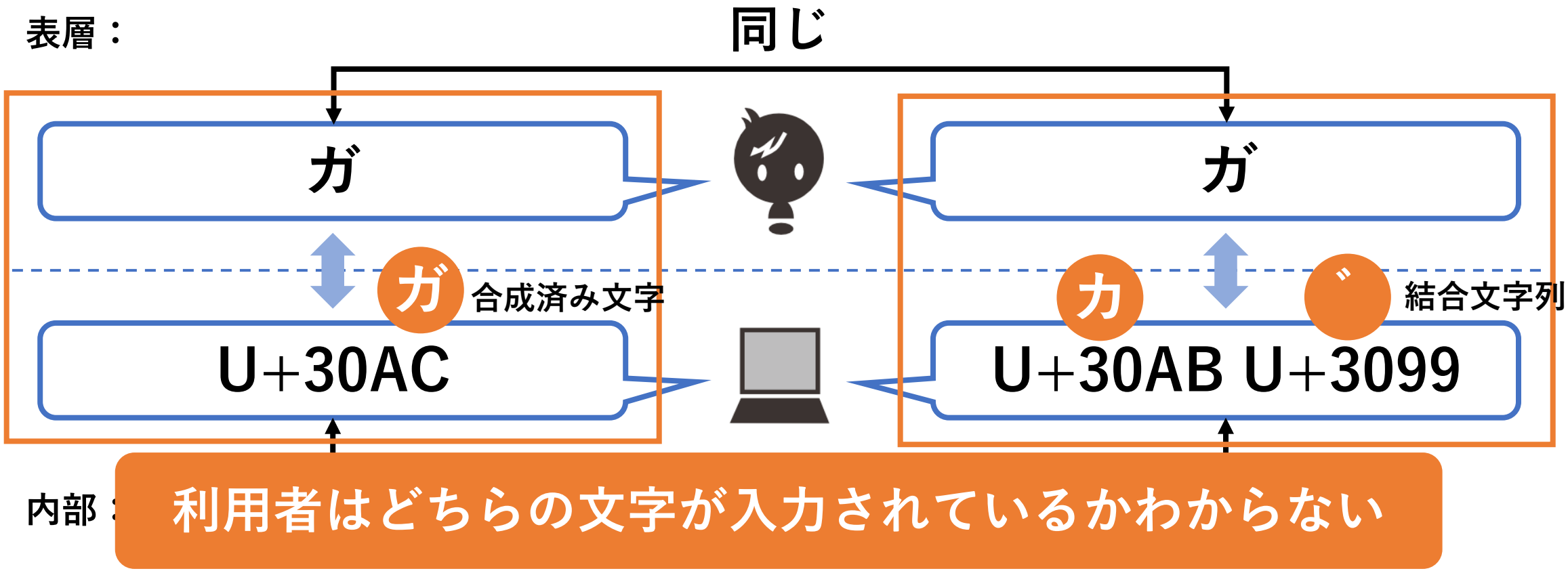
文字コードと文字集合(2/2)

- 英数字以外の各言語の文字を収録するために様々な国際化文字集合技術が開発
  - ISO/IEC 646，ISO/IEC 2022，ISO/IEC 8859，ISO/IEC 10646
  - 異なる文字集合間では，文字コードの一意性は保証されない (比較が困難)
- Unicode文字集合 (ISO/IEC 10646)
  - 各言語で使用する文字を全て収録することを目標に開発された文字集合
  - 1文字を21bitで表現
    - 1,114,112通りのビットパターンが表現可能
  - アプリケーションプロトコル中の識別子として利用される文字列を国際化する場合，UTF-8による入力をサポートすることが必要(RFC2277)
  - 現在でも改版作業が継続
  - 他の文字集合との互換性を考慮して設計

改版時期	バージョン番号	収録文字数
2002年 3月	3.2.0	95,221
:	:	:
2010年 10月	6.0.0	109,449
2012年 1月	6.1.0	110,181
2012年 9月	6.2.0	110,182
:	:	:
2016年 6月	9.0.0	128,172

表層表現と内部表現の乖離による問題

- 外見ないし意味上同等とみなせる文字が，異なる文字コードとして表現可能



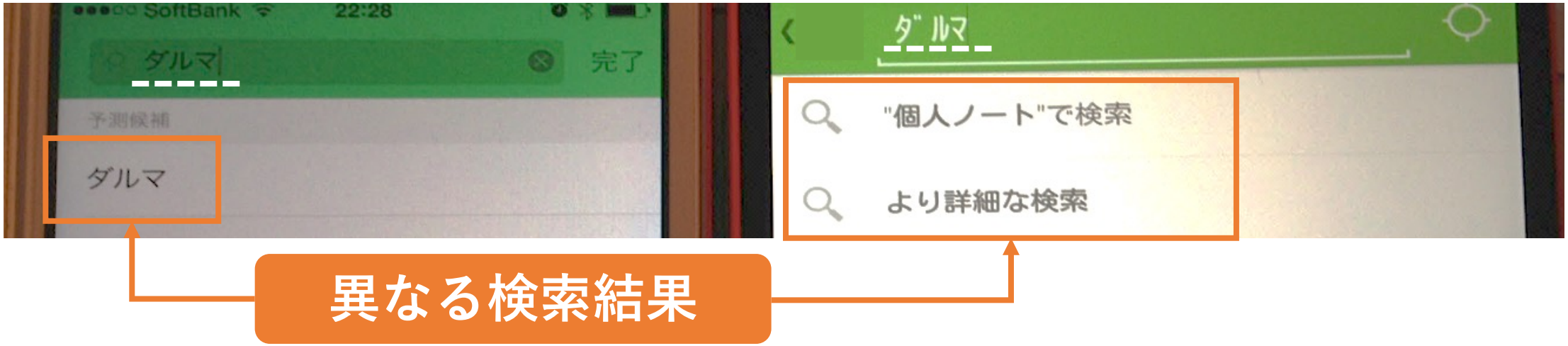
表現の乖離による問題の身近な例(1/2)

- 文字入力環境によって意図した情報資源を参照・検索できない例

登録文字例『ダルマ』

iOS版 某アプリケーション
半角カタカナ利用不可

Android版 某アプリケーション
半角カタカナ利用可



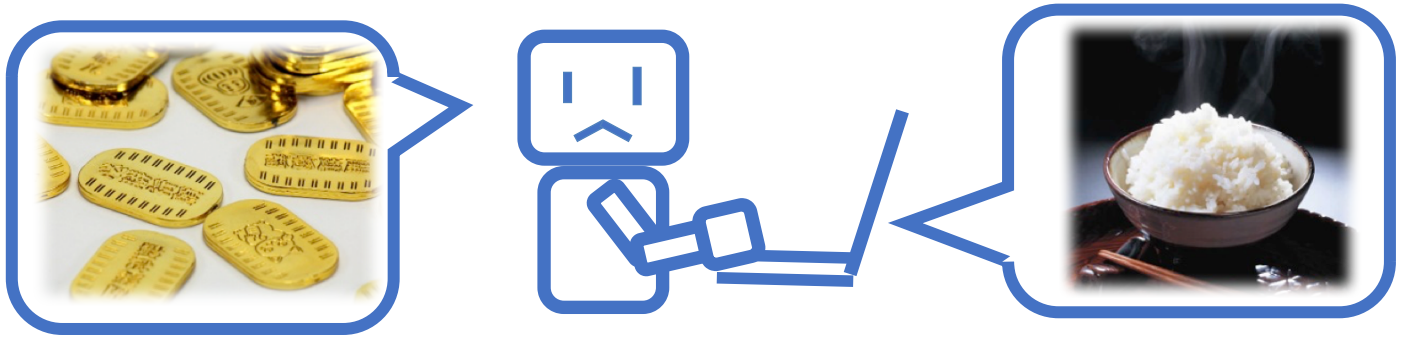
表現の乖離による問題の身近な例(2/2)

- 異なる意味の文字列を同一の文字列として参照・検索してしまう例

MySQLの比較アルゴリズム『utf8_unicode_ci』を使用した場合

Unicode	U+3053	U+3054	U+30B3	U+30B4	U+32D9	U+FF7A
文字	こ	ご	コ	ゴ	㋙	ｺ

Unicode	U+306F	U+3070	U+3071	U+30CF	U+30D0	U+30D1	U+31F5	U+32E9	U+FF8A
文字	は	ば	ぱ	ハ	バ	パ	ㇵ	㋩	ﾊ



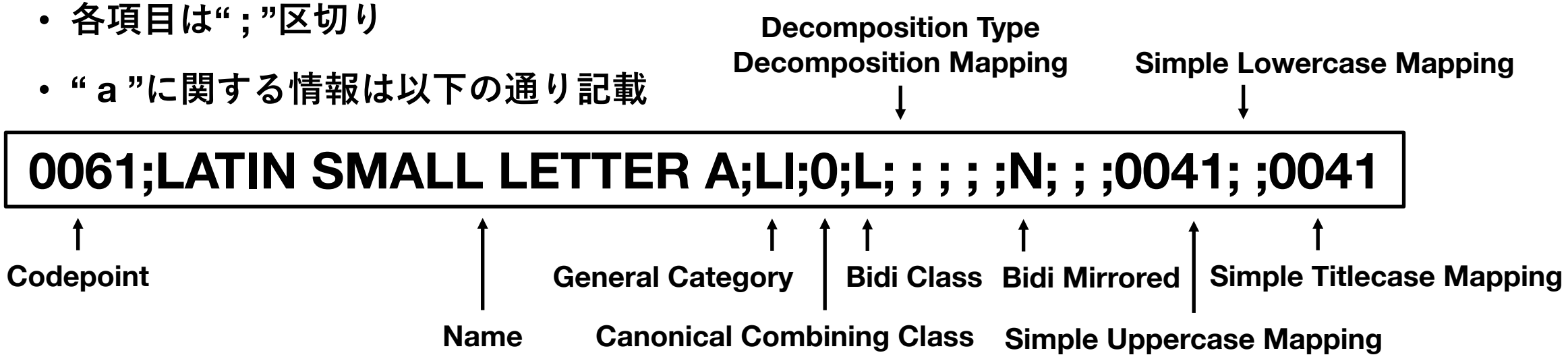
Uniocde コードポイント(Codepoint)

- Uniocdeコードポイント
  - Uniocde文字集合の任意の文字コード値
- 表記方法は“U+16進数”
  - 取りうる値は，U+0000 ~ U+10FFFF
  - 16進数は最低4桁で表記
  - 16進数が5桁以上になる場合は，最上位に0を置かない

コードポイント範囲	誤	正
U+0000 ~ U+FFFF	U+41	U+0041
U+10000 ~ U+10FFFF	U+010000	U+10000

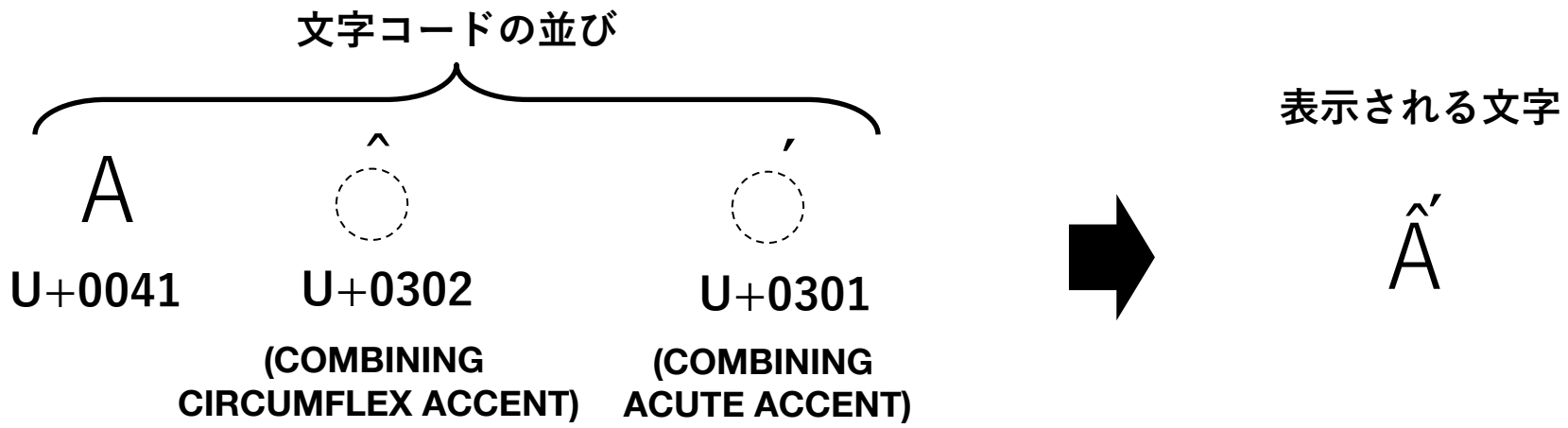
Unicode Character Database(UCD)

- Property Tables (全コードポイントの各種性質情報を定義したデータ群)が登録されている
- 代表的なProperty Tableは“UnicodeData.txt”
  - 1行に1コードポイント分の情報(15項目)が記載
  - 各項目は“ ; ”区切り
  - “ a ”に関する情報は以下の通り記載



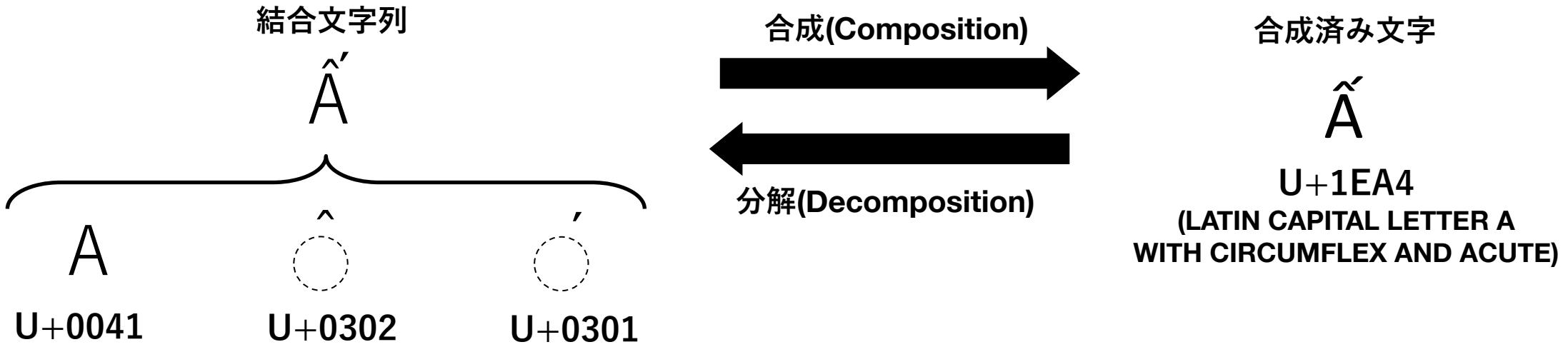
結合文字列 (Combining character sequence)

- 複数の文字コードを組み合わせて1文字を表現可能
  - 収録されている文字数を増やさずに，発音記号(アクセントやウムラウト等)が付く文字等を表現可能とする仕組み
  - “(基底文字(base character)) ＋ 結合文字(combining character)”
    - 結合文字に該当する文字は，General Categoryの値が“Mc”，“Me”，“Mn”のいずれかを取る
    - 結合位置は，Canonical Combining Classの値によって決まる



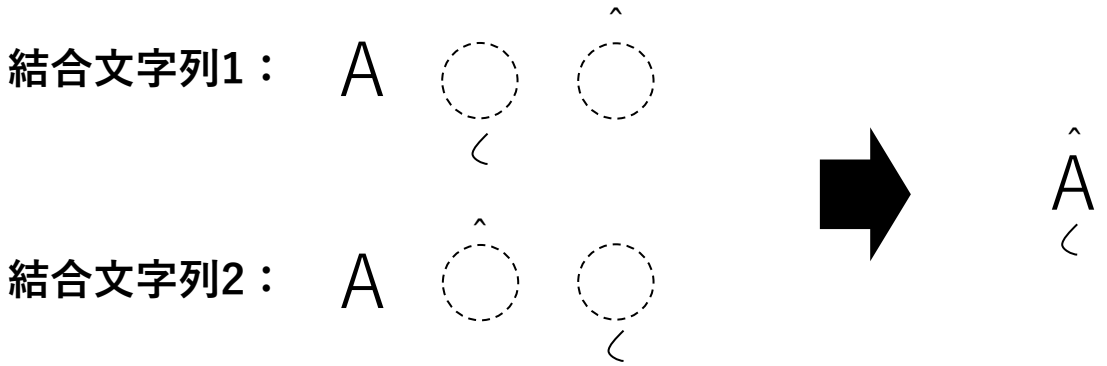
合成済み文字(Precomposed character)

- 結合文字列を成す各文字コードを合成し，1文字コードで表した文字
  - 合成済み文字を分解すると，結合文字列(基底文字＋結合文字)になる(＝等価)

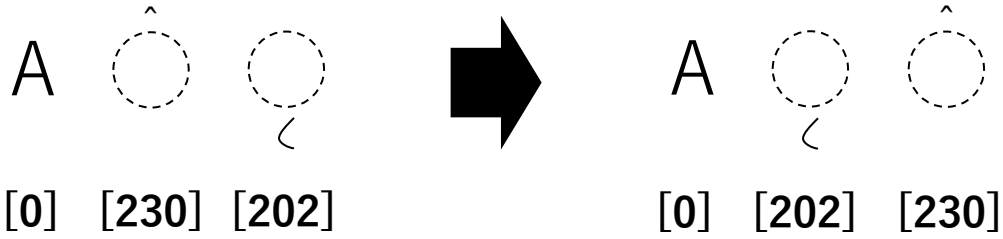


正規整列(Canonical ordering)

- Canonical Combining Classの値に従い文字コードを並び替える仕組み
  - Canonical Combining Classの値が異なる結合文字を2文字以上用いる結合文字列は，<u>異なる文字コードの並びで見かけ上同じ文字が表現可能</u>



  - このままでは，文字列の比較が不便なので， Canonical Combining Classの値でソート



分解(Decompsition)

- 分解対象となるコードポイントに対して2種類の分解方法を定義している
  - 正規分解 (Canonical Decompsition)
    - 機能的に等しく視覚的にも識別が困難な文字に分解
    - 正規分解結果が互いに等しい場合，正規等価性がある
  - 互換分解 (Compatibility Decomposition)
    - 視覚的に異なり機能的にも異なる可能性のある文字に分解
    - 互換分解結果が互いに等しい場合，互換等価性がある
    - 互換等価であるが正規等価でない文字があるので要注意 (全角半角文字等)
  - 上記いずれかの分解を再帰的に行い，正規整列を行った結果の文字コードの並びがUCDに登録されている

正規分解と互換分解の例

	比較文字	正規分解結果	互換分解結果
例1: 正規等価及び 互換等価が 成り立つ文字	ガ U+30AC	ガ U+30AB + U+3099	ガ U+30AB + U+3099
	ガ U+30AB + U+3099	ガ U+30AB + U+3099	ガ U+30AB + U+3099
例2: 互換等価のみ 成り立つ文字	ガ U+30AC	ガ U+30AB + U+3099	ガ U+30AB + U+3099
	ｶﾞ U+FF76 + U+FF9E	ｶﾞ U+FF76 + U+FF9E	ガ U+30AB + U+3099

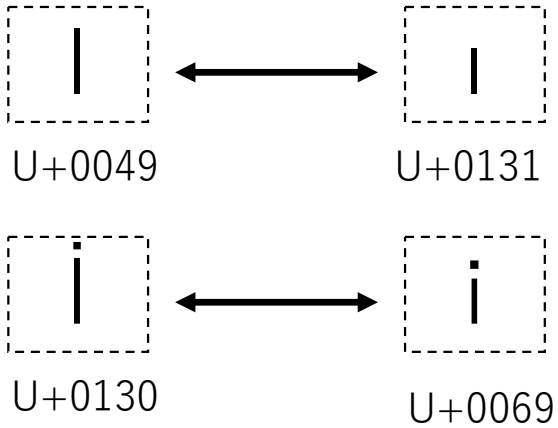
Unicode正規化形式 (Unicode Normalization Form)

- 4種類定義している
  - NFD:　Canonical Decomposition + Canonical Ordering
  - NFKD:　C(K)ompatibility Decomposition + Canonical Ordering
  - NFC:　NFD + Canonical Composition
  - NFKC:　NFKD + Canonical Composition

  - 頭文字で正規化形式を区別する際，CanonicalとCompatibilityを便宜上区別するためにCompatibilityはKompatibilityとされている

文字種変換(Casing)

- 文字種(大文字/小文字/タイトル文字)
- 2種類定義されている
  - Simple Casing
    - “UnicodeData.txt”で定義された文字種変換
  - Special Casing
    - “SpecialCasing.txt” で定義された文字種変換
    - ロケールや文脈に依存した文字種変換を必要をが定義されている
      - ロケール依存: トルコ語やアゼルバイジャン語で用いられる“İ (U+0130)”，“ı (U+0131)”
      - 文脈依存: ギリシャ語で用いられる“Σ (U+03A3)”，“σ (U+03C3)”，“ς (U+03C2)”



文字の表記方向

- “Bidi Class”として全てのコードポイントに対して文字の表記方向が定義

文字の方向	例
左から右（ラテン文字）	**nemoto**
右から左（アラビア文字）	نَيْمُوتو
右から左（ターナ文字）	Rނޯސްޖީޓަރ
双方向（アラビア文字・数字）	نَيْمُ0 تو
双方向（アラビア文字・数字）	يْمُو١٠

アラビア文字の字形とコードポイントの並び

- アラビア文字は，前後のコードポイントの並びで表示される字形が異なる

例:

ب

U+0628

記述位置	字形	コードポイント
語頭	بـ	**U+0628** U+0640 U+0640
語中	ـبـ	U+0640 **U+0628** U+0640
語末	ـب	U+0640 U+0640 **U+0628**
独立系	ب	**U+0628**

- 通常左から右に表記する文字を，右から左に表記するためにはRLO (Right-to-Left Override) と呼ばれる制御文字 (U+202E)を使用
  - この制御文字を利用すると見かけ上の文字列「annexe.txt」のコードポイントの並びが「ann(U+202E)txt.exe」 であったりするので要注意

識別子を国際化するにあたり考慮すべき文字集合の問題

- 人間にとって 視覚的ないし機能的に同等にみなせる文字が複数ある
  - ASCII文字集合は文字種（大文字・小文字）程度

項目	例	
文字種（大文字・小文字）	A (U+0041)	a (U+0061)
文字幅（全角・半角）	ｱ (U+FF71)	ア (U+30A2)
合成済文字・結合文字列	カ゛ (U+30AB U+3099)	ガ (U+30AC)
文脈依存文字	σ (U+03C3)	ς (U+03C2)
言語依存文字	İ (U+0130)	i (U+0069)

コンピュータがこれらの文字を同等な文字として扱えることは，

利便性や安全性の観点から重要

識別子を国際化するにあたり考慮すべきプロトコルにおける問題(1/2)

- プロトコルによって文字列の比較時に行う前処理が異なる
  - 大文字小文字を区別しない
  - 文字幅を一般的な文字幅に統一する
  - 結合文字列は可能な限り合成する
  - 制御文字は削除
  - 文字の向きの確認を行う

RFC番号	プロトコル	正規化	区切り文字	文字種統一	その他
3490	IDNA	-	○	-	-
3491	IDNA	○	-	○	-
3722	iSCSI	○	-	○	-
4314	IMAP	○	-	-	○
4518	LDAP	○	-	○	○

○ = 必要な処理
- = 不必要な処理

- 共通の前処理が異なるプロトコルプロファイルとして存在

共通する前処理をフレームワーク化しておくことは,

各プロトコルで使用するために重要

識別子を国際化するにあたり考慮すべきプロトコルにおける問題(2/2)

- プロトコルによって使用禁止確認対象となるコードポイントが異なる

RFC番号	プロトコル	空白文字	制御文字	私用文字	非文字	サロゲート	平常文不適	正規表現不適	表示特性変更	タグ文字
3491	IDNA	△	△	○	○	○	○	○	○	○
3722	iSCSI	○	○	○	○	○	○	○	○	○
4314	IMAP	△	○	○	○	○	○	○	○	○
4518	LDAP	-	-	○	○	○	△	-	○	-

○＝確認対象　△＝一部確認対象　-＝確認対象外

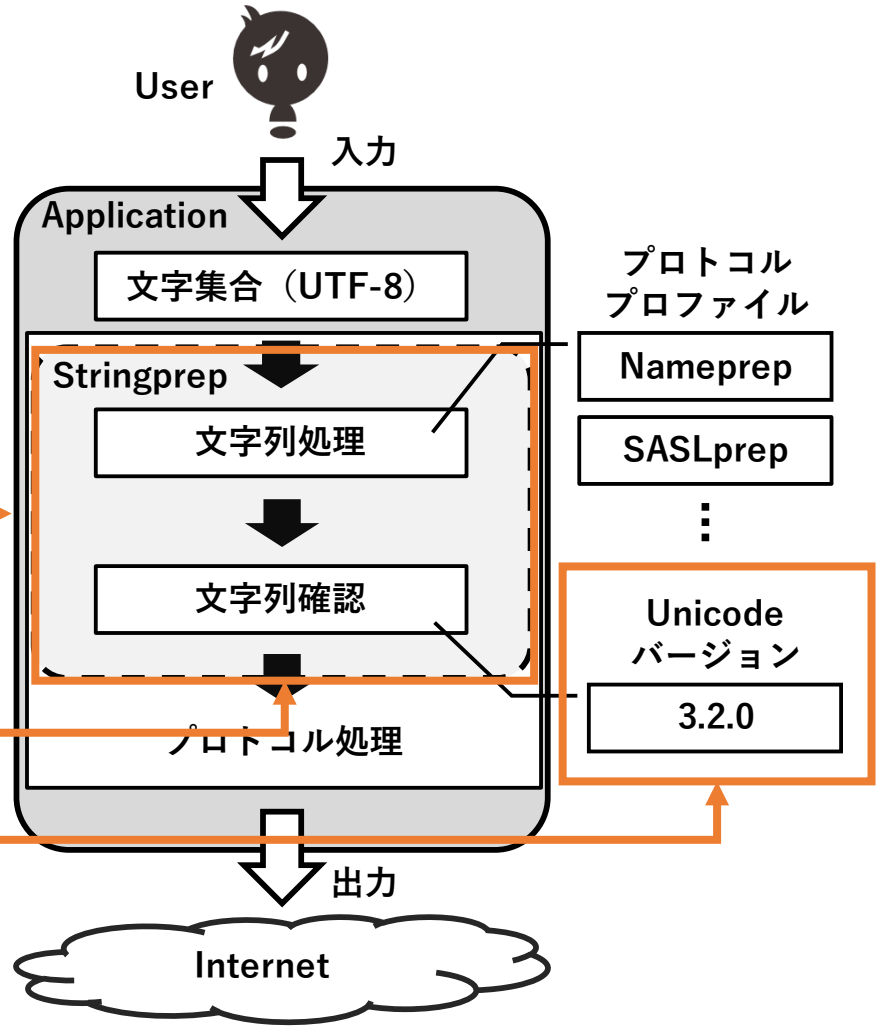
UCDに登録された文字の性質に基づき，
プロトコルで使用可能な文字を分類する方法は，

文字集合の改版に追従するために重要

従来の国際化技術: Stringprep

- 各プロトコルで国際化文字列を扱うための文字列処理の枠組み
  - 文字列変換処理と使用禁止文字を定義
  - 各プロトコルは必要な処理を選択して利用
- 課題
  - 文字列変換処理の問題(正規化形式)
  - 文字列確認方法の問題(双方向性，禁止文字)
  - 特定の文字集合のバージョンに依存

    (明治，大正，昭和，平成，~~□(新元号)~~)



従来の国際化技術: IDNA2008

- ドメイン名で国際化文字列を扱うための文字列処理の枠組み
  - 文字列変換処理と使用許可文字分類方法を定義
  - 各Unicodeの改版に適応可能
- 課題
  - 他のプロトコルで使用不可(変換処理，分類方法)