Charsets & Encodings

PHP Benelux

Kore Nordmann

29.01.2011

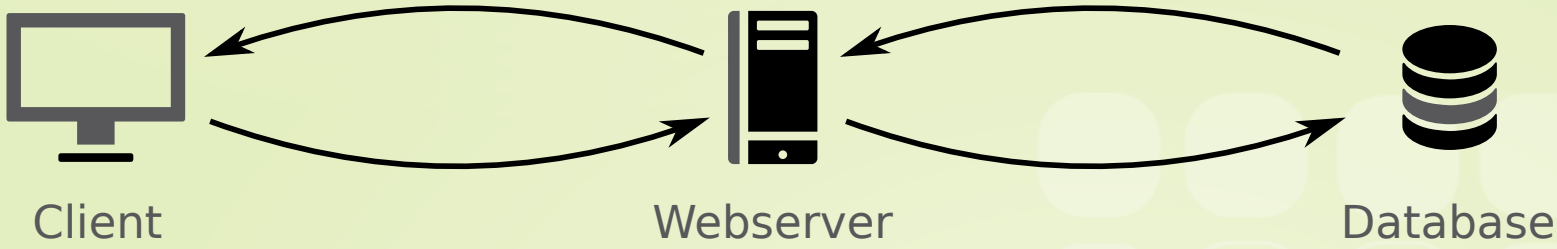
contact@qafoo.com
http://talks.qafoo.com/

About me

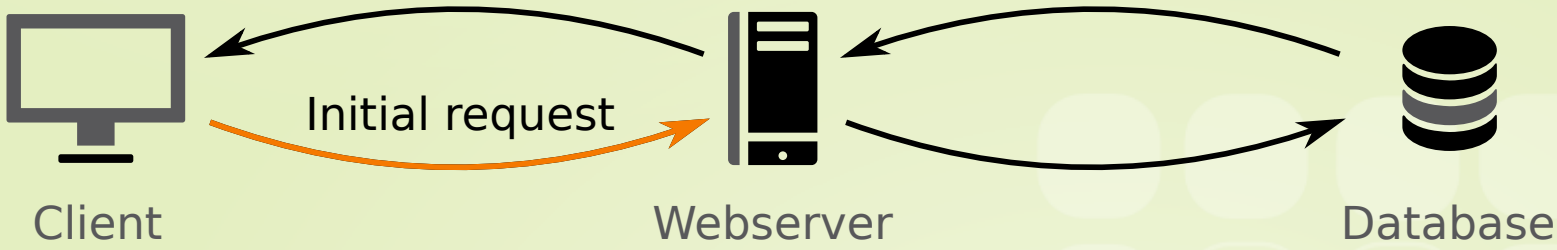
- Kore Nordmann (<kore@php.net>, <kore@apache.org>, <kore@qafoo.com>)
- More than 10 years of professional PHP
- Open source enthusiast
- Contributing to various FLOSS projects
- Founder of Qafoo GmbH
  - Provides training & consulting on PHP software quality tools & processes

Outline

Common request cycle



First request



First request

- Normally a `GET` request
- Normally no “data” transmission
  - . . . unless you are Wikipedia

The browser tells us a lot of things:

```
1  GET /foo.html HTTP/1.1
2  Host: example.com
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:2.0b10pre) Gecko/20110120 Firefox
       -4.0/4.0b10pre
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: chrome://global/locale/intl.properties
6  Accept-Encoding: gzip, deflate
7  Accept-Charset: ISO-8859-1,utf-8;q=1,*;q=0.7
8  Connection: keep-alive
```

First request

- Normally a `GET` request
- Normally no “data” transmission
  - . . . unless you are Wikipedia
- The browser tells us a lot of things:

```
1 GET /foo.html HTTP/1.1
2 Host: example.com
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:2.0b10pre) Gecko/20110120 Firefox
      -4.0/4.0b10pre
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: chrome://global/locale/intl.properties
6 Accept-Encoding: gzip, deflate
7 Accept-Charset: ISO-8859-1,utf-8;q=1,*;q=0.7
8 Connection: keep-alive
```

First request

- Normally a `GET` request
- Normally no “data” transmission
  - . . . unless you are Wikipedia
- The browser tells us a lot of things:

```
1 GET /foo.html HTTP/1.1
2 Host: example.com
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:2.0b10pre) Gecko/20110120 Firefox
      -4.0/4.0b10pre
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: chrome://global/locale/intl.properties
6 Accept-Encoding: gzip, deflate
7 Accept-Charset: ISO-8859-1,utf-8;q=1,*;q=0.7
8 Connection: keep-alive
```

Charset vs. Encoding

- *Charset*: A set of characters
- *Encoding*: Mapping of characters to bytes
  - *Multibyte Encoding*: Mapping of a character to a sequence of bytes
- Unicode holds (far) more then 256 Characters.
  - Not possible to map to single bytes
- Unicode, ISO-8859-1, ASCII, ...
- UTF-8, UTF-16, ISO-8859-1, ASCII, ...
- different Unicode encodings
  - UTF-8, encodes full unicode, uses something between 1 and 4 bytes, ASCII compatible
  - UTF-16, does not encode full unicode, always uses 2 bytes
  - UTF-32, encodes full unicode, always uses 4 bytes
  - UTF-7, UCS2, UCS4, ...

Charset vs. Encoding

- *Charset*: A set of characters
- *Encoding*: Mapping of characters to bytes
  - *Multibyte Encoding*: Mapping of a character to a byte sequence

Charset vs. Encoding

- *Charset*: A set of characters
- *Encoding*: Mapping of characters to bytes
  - *Multibyte Encoding*: Mapping of a character to a byte sequence
- Unicode holds (far) more then 256 Characters.
  - Not possible to map to single bytes

Charset vs. Encoding

- *Charset*: A set of characters
- *Encoding*: Mapping of characters to bytes
  - *Multibyte Encoding*: Mapping of a character to a byte sequence
- Unicode holds (far) more then 256 Characters.
  - Not possible to map to single bytes
- Charsets: Unicode, ISO-8859-1, ASCII, ...
- Encodings: UTF-8, UTF-16, ISO-8859-1, ASCII, ...

different Unicode encodings

-8, encodes full unicode, uses something between 1 and 4
s, ASCII compatible
-16, does not encode full unicode, always uses 2 bytes
-32, encodes full unicode, always uses 4 bytes
-7, UCS2, UCS4, ...

Charset vs. Encoding

- *Charset*: A set of characters
- *Encoding*: Mapping of characters to bytes
  - *Multibyte Encoding*: Mapping of a character to a byte sequence
- Unicode holds (far) more then 256 Characters.
  - Not possible to map to single bytes
- Charsets: Unicode, ISO-8859-1, ASCII, ...
- Encodings: UTF-8, UTF-16, ISO-8859-1, ASCII, ...
- There are different Unicode encodings
  - UTF-8, encodes full unicode, uses something between 1 and 4 bytes, ASCII compatible
  - UTF-16, does not encode full unicode, always uses 2 bytes
  - UTF-32, encodes full unicode, always uses 4 bytes
  - UTF-7, UCS2, UCS4, ...

Charset vs. Encoding

- "This use of the term "character set" is more commonly referred to as a "character encoding"." (HTTP 1.1 Specification)

```
Accept-Charset: ISO-8859-1,utf-8;q=1,*;q=0.7
```

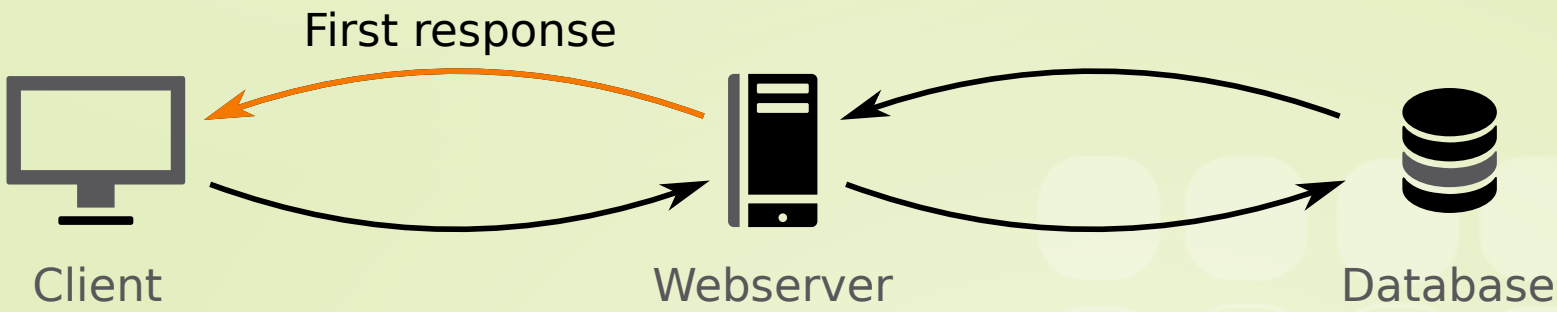
- "This attribute specifies the list of character encodings for input data" (HTML 4.01 Specification)

```
<form action="..." accept-charset="UTF-8">
```

- XML does it "right":

```
<?xml encoding="UTF-8"?>
<doc/>
```

First request



First response

- You *should* select output encoding based on `Accept-Charset`

First response

- You *should* select output encoding based on `Accept-Charset`
  - But ISO-8859-1 and UTF-8 will “always” work.

First response

- You *should* select output encoding based on `Accept-Charset`
  - But ISO-8859-1 and UTF-8 will “always” work.
  - With UTF-8 you do not need any HTML entities any more.

First response

- You *should* select output encoding based on `Accept-Charset`
  - But ISO-8859-1 and UTF-8 will "always" work.
  - With UTF-8 you do not need any HTML entities any more.
- Report used encoding *consistently* to client
  - HTTP header `Content-type: text/html; charset=utf-8`
  - HTML meta tag `Content-Type`

First response

- You *should* select output encoding based on `Accept-Charset`
  - But ISO-8859-1 and UTF-8 will “always” work.
  - With UTF-8 you do not need any HTML entities any more.
- Report used encoding *consistently* to client
  - HTTP header `Content-type: text/html; charset=utf-8`
  - HTML meta tag `Content-Type`

Handling encodings in PHP

- PHPs strings are byte arrays
- PHPs string functions are binary safe

Handling encodings in PHP

- PHPs strings are byte arrays
- PHPs string functions are binary safe
  - PHP does not care about encodings (or charsets)

String functions are *not* "multibyte-safe"

Handling encodings in PHP

- PHPs strings are byte arrays
- PHPs string functions are binary safe
  - PHP does not care about encodings (or charsets)
  - PHPs string functions are *not* “multibyte-safe”

Handling multibyte encodings in PHP

- Use iconv or mbstring

```
1  <?php
2
3  $string = 'öää';
4
5  var_dump(
6      strlen( $string ),
7      iconv_strlen( $string, 'UTF-8' )
8  );
9
10 // int(6)
11 // int(3)
```

Converting between encodings

- Encoding conversions in PHP
  - `iconv()`
  - `mb_convert_encoding()()`
  - `utf8_encode()` / `utf8_decode()`

Converting between encodings

- Encoding conversions in PHP
  - `iconv()`
  - `mb_convert_encoding()()`
  - `utf8_encode()` / `utf8_decode()`
- Character set conversions in PHP
  - `iconv()`
  - pecl/translit (by Derick Rethans)

Converting

```php
<?php

var_dump( iconv( 'UTF-8', 'ASCII', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(0) ""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(8) "oau test"

setlocale( LC_ALL, 'de_DE.utf8' );

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(11) "oeaeue test"

var_dump( iconv( 'UTF-8', 'ASCII//IGNORE', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(5) " test"

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü <snowman> test' ) );
// string(13) "oeaeue ? test"
```

Converting

```php
<?php

var_dump( iconv( 'UTF-8', 'ASCII', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(0) ""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(8) "oau test"

setlocale( LC_ALL, 'de_DE.utf8' );

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(11) "oeaeue test"

var_dump( iconv( 'UTF-8', 'ASCII//IGNORE', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(5) " test"

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü <snowman> test' ) );
// string(13) "oeaeue ? test"
```

Converting

```php
<?php

var_dump( iconv( 'UTF-8', 'ASCII', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(0) ""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(8) "oau test"

setlocale( LC_ALL, 'de_DE.utf8' );

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(11) "oeaeue test"

var_dump( iconv( 'UTF-8', 'ASCII//IGNORE', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(5) " test"

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü <snowman> test' ) );
// string(13) "oeaeue ? test"
```

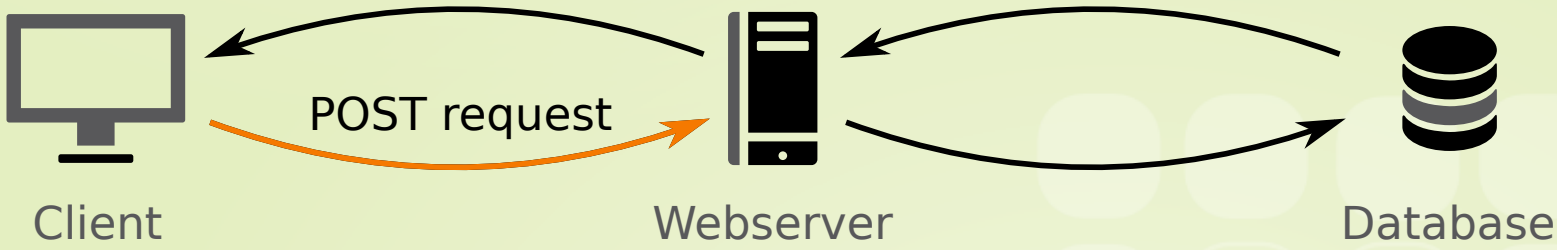
Converting

```php
<?php

var_dump( iconv( 'UTF-8', 'ASCII', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(0) ""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(8) "oau test"

setlocale( LC_ALL, 'de_DE.utf8' );

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(11) "oeaeue test"

var_dump( iconv( 'UTF-8', 'ASCII//IGNORE', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(5) " test"

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü <snowman> test' ) );
// string(13) "oeaeue ? test"
```

Converting

```php
<?php

var_dump( iconv( 'UTF-8', 'ASCII', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(0) ""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(8) "oau test"

setlocale( LC_ALL, 'de_DE.utf8' );

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü test' ) );
// string(11) "oeaeue test"

var_dump( iconv( 'UTF-8', 'ASCII//IGNORE', 'öäü test' ) );
// Notice: iconv(): Detected an illegal character in input string
// string(5) " test"

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', 'öäü <snowman> test' ) );
// string(13) "oeaeue ? test"
```

First request



First response

- Browsers send data in site encoding, if consistently specified
  - `accept-charset` != site encoding just fucks it up.

First response

- Browsers send data in site encoding, if consistently specified
  - `accept-charset` != site encoding just fucks it up.
- Handling input data
  - Store encoding used for the user in session (with a sane default)
  - Handle broken encodings in input layer
    - Reject or fix?

First response

- Browsers send data in site encoding, if consistently specified
  - `accept-charset` != site encoding just fucks it up.
- Handling input data
  - Store encoding used for the user in session (with a sane default)
  - Handle broken encodings in input layer
    - Reject or fix?
- External data sources (file uploads, services)
  - Detection of input encoding?
    - This is *impossible*
    - Implement custom data specific heuristics
    - Try `mb_detect_encoding()`

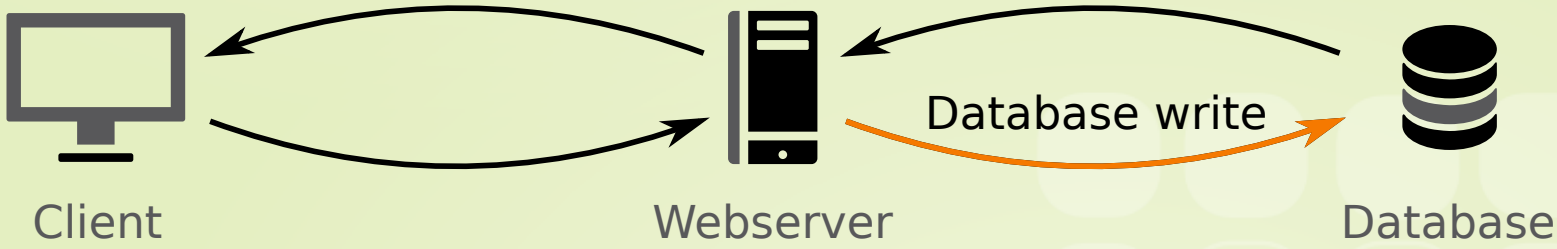
First response

- Browsers send data in site encoding, if consistently specified
  - `accept-charset` != site encoding just fucks it up.
- Handling input data
  - Store encoding used for the user in session (with a sane default)
  - Handle broken encodings in input layer
    - Reject or fix?
- External data sources (file uploads, services)
  - Detection of input encoding?
    - This is *impossible*
    - Implement custom data specific heuristics
    - Try `mb_detect_encoding()`

First response

- Browsers send data in site encoding, if consistently specified
  - `accept-charset` != site encoding just fucks it up.
- Handling input data
  - Store encoding used for the user in session (with a sane default)
  - Handle broken encodings in input layer
    - Reject or fix?
- External data sources (file uploads, services)
  - Detection of input encoding?
    - This is *impossible*
    - Implement custom data specific heuristics
    - Try `mb_detect_encoding()`

Databases



Databases

- Too many different options, but for MySQL and PostgreSQL:
  - The connection (client) encoding is relevant
  - The DBMS converts from and to table encoding

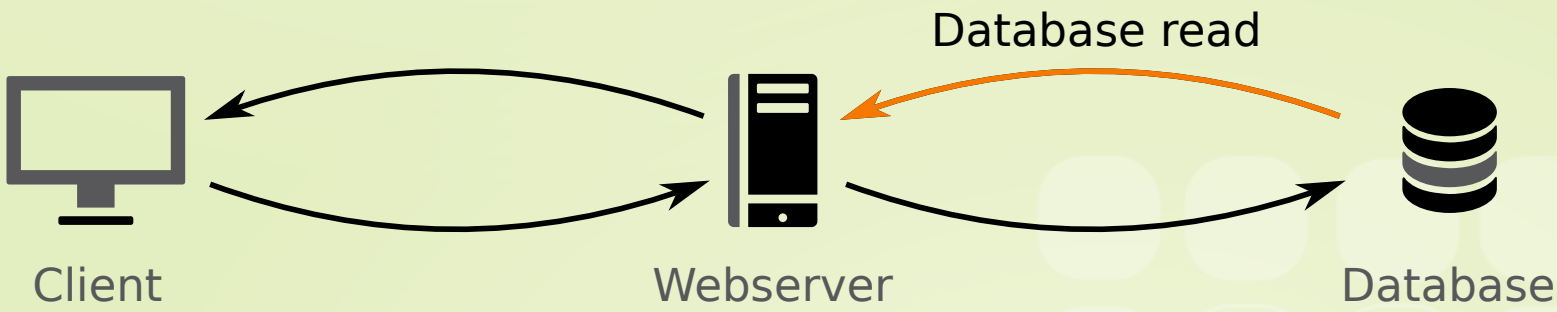
Databases

- Too many different options, but for MySQL and PostgreSQL:
  - The connection (client) encoding is relevant
  - The DBMS converts from and to table encoding
  - Mind that the charsets are compatible

Databases

- Too many different options, but for MySQL and PostgreSQL:
  - The connection (client) encoding is relevant
  - The DBMS converts from and to table encoding
  - Mind that the charsets are compatible
  - Setting the encoding: `SET NAMES 'UTF-8';`
    - `mysql_real_escape_string()` will *not* be aware of this.

Databases



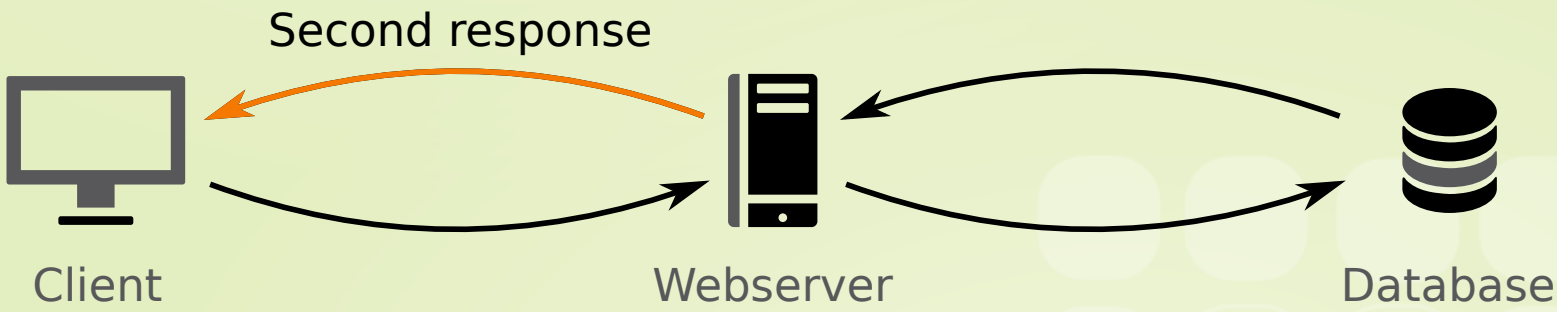
Collations

- Basically the same as for writing, but:
  - Additionally to the charset and encoding there are "Collations"
  - Define sorting order

Collations

- Basically the same as for writing, but:
- Addtionally to the charset and encoding there are “Collations”
  - Specify the sorting order

Final output



Final suggestions

- Use UTF-8 consistently in your application
  - Always convert input
  - Convert to output encoding as late as possible

Outline

Unicode character equality

```php
<?php

$char1 = "\x65\xcc\x81"; // U+0065 + U+0301
$char2 = "\xc3\xa9"; // U+00E9

var_dump( $char1 . $char2 );
// string(5) éé""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', $char1 . $char2 ) );
// string(2) "ee"
```

Unicode character equality

```php
<?php

$char1 = "\x65\xcc\x81"; // U+0065 + U+0301
$char2 = "\xc3\xa9"; // U+00E9

var_dump( $char1 . $char2 );
// string(5) éé""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', $char1 . $char2 ) );
// string(2) "ee"
```

Normalize Unicode strings before comparison

And that for usernames

Unicode character equality

```php
<?php

$char1 = "\x65\xcc\x81"; // U+0065 + U+0301
$char2 = "\xc3\xa9"; // U+00E9

var_dump( $char1 . $char2 );
// string(5) éé""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', $char1 . $char2 ) );
// string(2) "ee"
```

Unicode character equality

```php
<?php

$char1 = "\x65\xcc\x81"; // U+0065 + U+0301
$char2 = "\xc3\xa9"; // U+00E9

var_dump( $char1 . $char2 );
// string(5) éé""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', $char1 . $char2 ) );
// string(2) "ee"
```

- Always normalize Unicode strings before comparison

Unicode character equality

```php
<?php

$char1 = "\x65\xcc\x81"; // U+0065 + U+0301
$char2 = "\xc3\xa9"; // U+00E9

var_dump( $char1 . $char2 );
// string(5) éé""

var_dump( iconv( 'UTF-8', 'ASCII//TRANSLIT', $char1 . $char2 ) );
// string(2) "ee"
```

- Always normalize Unicode strings before comparison
  - Mind that for usernames

Regular expressions

- Using UTF-8 you can use nice PCRE features: Unicode character classes

Regular expressions

- Using UTF-8 you can use nice PCRE features: Unicode character classes
  - `(\p{L}+)u` Matches all letters
  - `(\p{P}+)u` Matches all punctuation characters
  - `(\p{S}+)u` Matches all symbols
  - `(\p{Sc}+)u` Matches all currency symbols

http://www.php.net/manual/en/regexp.reference.unicode.php

Regular expressions

- Using UTF-8 you can use nice PCRE features: Unicode character classes
  - `(\p{L}+)u` Matches all letters
  - `(\p{P}+)u` Matches all punctuation characters
  - `(\p{S}+)u` Matches all symbols
  - `(\p{Sc}+)u` Matches all currency symbols

`http://www.php.net/manual/en/regexp.reference.unicode.php`

Regular expressions

- Using UTF-8 you can use nice PCRE features: Unicode character classes
  - `(\p{L}+)u` Matches all letters
  - `(\p{P}+)u` Matches all punctuation characters
  - `(\p{S}+)u` Matches all symbols
  - `(\p{Sc}+)u` Matches all currency symbols

http://www.php.net/manual/en/regexp.reference.unicode.php

Regular expressions

- Using UTF-8 you can use nice PCRE features: Unicode character classes
  - `(\p{L}+)u` Matches all letters
  - `(\p{P}+)u` Matches all punctuation characters
  - `(\p{S}+)u` Matches all symbols
  - `(\p{Sc}+)u` Matches all currency symbols

Regular expressions

- Using UTF-8 you can use nice PCRE features: Unicode character classes
  - `(\p{L}+)u` Matches all letters
  - `(\p{P}+)u` Matches all punctuation characters
  - `(\p{S}+)u` Matches all symbols
  - `(\p{Sc}+)u` Matches all currency symbols
- See `http://php.net/manual/en/regexp.reference.unicode.php`

Thanks for listening

- More about me:
  - `http://kore-nordmann.de` / @koredn
- More about us:
  - `http://qafoo.com` / @qafoo
- *Please* rate this talk:
  - `http://joind.in/2488`
- PHP Charset & Encoding FAQ:
  `http://kore-nordmann.de/blog/php_charset_encoding_FAQ.html`