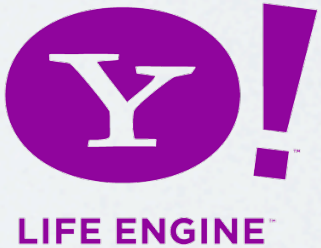
PHP 6 и Unicode

Андрей Змиевский
Yahoo! Inc.

PHPConf.ru, Москва ~ 25 мая, 2006

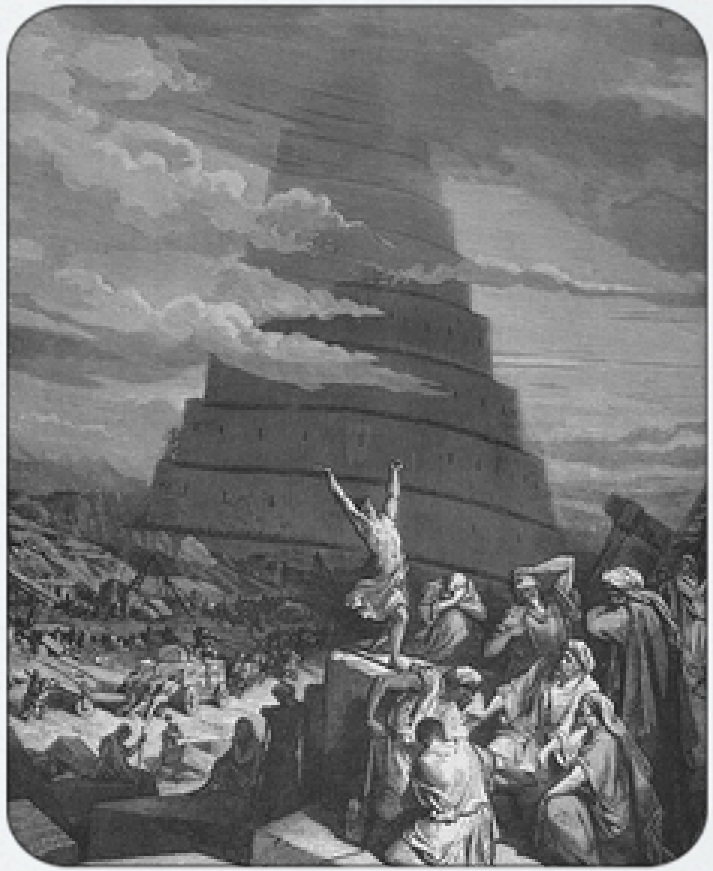


Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Вавилонская башня

“И сказал Господь: сойдем же и смешаем там язык их, так чтобы один не понимал речи другого".
— Бытие 11:7



Вавилонская башня

- История Вавилонской башни насколько–то правда
- Работать с несколькими кодировками достаточно проблематично
- Требуются различные алгоритмы преобразования, опознавания, валидации, обработки...
- Работать с несколькими языками не менее проблематично
- Но этого не избежать в наши дни

Задачи

- Создание приложений на разных языках и для разных культур
- Выполнение поиска, сортировки, разбивки по словам и др. с учётом языка и кодировки
- Поддержка форматов даты, времени, чисел, валюты в конкретной локали
- И многое другое

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

PHP в прошлом

- PHP всегда был бинарным обработчиком
- Тип “строка” байт-ориентирован и используется для всего, от текстов до изображений
- Ядро языка почти ничего не знает про кодировки и обработку мультиязычных данных
- `iconv` и `mbstring` не решают проблему

PHP в прошлом

- Поддержка локалей POSIX
- Зависимость от данных системной локали
- Разнообразные но не стыкующиеся i18n функции

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Unicode

- Unicode предоставляет уникальный численный код для каждого символа, вне зависимости от платформы, приложения и языка

Unicode

- Разработан Unicode Consortium
- Спроектирован так, чтобы обеспечить обработку и отображение текста и символов из всех языков
- Охватывает все основные живые языки
- Версия 4.1 содержит 97,000+ символов
- Общая ёмкость - больше 1 миллиона символов
- Набор символов Unicode = ISO 10646

Unicode Character Set

The primary scripts currently supported by Unicode 4.0 are:

- Arabic
- Armenian
- Bengali
- Bopomofo
- Buhid
- Canadian Syllabics
- Cherokee
- Cypriot
- Cyrillic
- Deseret
- Devanagari
- Ethiopic
- Georgian
- Gothic
- Greek
- Gujarati
- Gurmukhi
- Han
- Hangul
- Hanunóo
- Hebrew
- Hiragana
- Kannada
- Katakana
- Khmer
- Lao
- Latin
- Limbu
- Linear B
- Malayalam
- Mongolian
- Myanmar
- Ogham
- Old Italic (Etruscan)
- Osmanya
- Oriya
- Runic
- Shavian
- Sinhala
- Syriac
- Tagalog
- Tagbanwa
- Tai Le
- Tamil
- Telugu
- Thaana
- Thai
- Tibetan
- Ugaritic
- Yi

Организован по шрифтам

Example Unicode Characters

ASCII	ABCDEFGHIJKLMNOP
Latin-1	ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
Latin-2	āĂăĄąĆćĈĉĊċČčĎďĐ
Greek	ΫABΓΔEZHΘIKΛMNΞO
Cyrillic	рстуфхцчшщъыьэюя
Thai	ภมยรฤลฦวศษสหฬอฮฯ
CJK	北両丢乖两严並丧
Korean	감갑값갓갔강갖갘

Терминология Unicode

- Этот **abstract character** обозначается кодом U+233B4
  
- Его **code point** - 233B4, независимо от кодировки UTF
- В UTF-32 его **code unit** - 233B4
- В UTF-16 у него два 16-битных **code units**:
  - D84C, DFB4 (верхняя и нижняя составляющие)
- В UTF-8 у него четыре 8-битных **code units**:
  - F0, A3, 8E, B4

Unicode можно расширять

- Новые символы можно создавать с помощью композиции

и + ̆ = й

U+0438 + U+0306 = U+0439

a + ̂ + ̣ = ậ

U+0061 + U+0302 + U+0323 = U+1EAD

a + ̨ + ̌ = ą̌

U+0061 + U+0322 + U+030C

Unicode – это будущее

- Многоязычный
- Полный набор характеристик символов
- Стандартные кодировки: UTF-8, UTF-16, UTF-32
- Спецификации алгоритмов делают всё для совместимости приложений
- Но Unicode != i18n

Определения

Интернационализация — I18n

Создание приложения:

- без учёта культурных предположений
- которое можно **эффективно** локализовать

Локализация — L10n

Адаптация приложения под требования конкретного региона, рынка или культуры

Локали

- I18N и L10N зависят от данных локали
- Локаль не означает сами данные, как в POSIX
- Локаль = идентификатор, обозначающий языковые и культурные предпочтения
  - `en_US, en_GB, ru_RU`
- Эти предпочтения могут меняться со временем по культурным и политическим причинам
  - Введение новых валют, как в случае с Euro
  - Изменение порядка сортировки в испанском

Виды данных локали

- Форматы даты/времени
- Форматы чисел/валюты
- Система измерения
- Спецификации сравнения символов
  - сортировка
  - поиск
  - равенство
- Локализованные имена языка, территории, шрифтов, временных зон, валюты,…
- Алфавит и символы, используемые в языке

Общий репозиторий данных локалей (CLDR)

- Поддерживается Unicode Consortium
  - http://www.unicode.org/cldr/
- Цели:
  - Общие, необходимые для всего ПО данные локалей для всех языков
  - Сохранение и управление данными
  - Формат XML для эффективного обмена данными
  - Свободный доступ
- Последний релиз: 2 Июня 2005 (CLDR 1.3)
- 296 локалей: 96 языков и 130 территорий

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Цели

- Введение типа «строка Unicode»
- Отделение типа «бинарная строка»
- Строковые литералы Unicode
- Обновлённая семантика языка
- Изменение существующих функций, вместо создания новых

Цели

- Обратная совместимость
- Сделать простые вещи очень простыми, а сложные - возможными
- Основной фокус – на функциональности
- Соответствие поддержке Unicode и i18n в Java

ICU

- International Components for Unicode
- Почему не наше собственное решение?
  - Требуется много знаний
  - Это было бы изобретение колеса
  - В духе PHP: заимствуй, если возможно; изобретай, если аналогов нет

Почему ICU?

- Уже существует
- Полнофункциональная
- Надёжная
- Быстрая
- Проверенная
- Портабельная
- Расширяемая
- Open Source
- Поддерживается авторами

Особенности ICU

- ✓ Свойства символов Unicode
- ✓ Обработка строк Unicode
- ✓ Транформации текста (нормализация, изменение регистра и др.)
- ✓ Обработка границ текста (Итераторы по символам, словам и предложениям)
- ✓ Конверторы для 500+ кодировок
- ✓ Поиск и сортировка с учётом языка
- ✓ Регулярные выражения с поддержкой Unicode
- ✓ Многопоточность
- ✓ Форматирование: Дата/Время/Числа/Валюта
- ✓ Календари и временные зоны
- ✓ Поддержка 230+ локалей
- ✓ Resource Bundles
- ✓ Транслитерация (50+ пар алфавитов)
- ✓ Расположение текста для арабского, иврита, хинди и тайского
- ✓ Национальные доменные имена и веб-адреса
- ✓ Java model for locale-hierarchical resource bundles. Multiple locales can be used at a time

Основные этапы

- Добавление поддержки Unicode в ядро
- Поддержка Unicode в существующих модулях
- Создание функций-врапперов для ICU API

Да будет Unicode!

- INI-директива `unicode_semantics`
- Возможность включения в пределах виртуального хоста
- Никаких изменений в приложении не требуется, если эта директива отключена
- Отключение не означает полное отсутствие Unicode!

Строковые типы

- Строковые типы в PHP 4/5
  - только один, используется для всего
- Строковые типы в PHP 6
  - Unicode: текстовые данные в UTF-16
  - Бинарный: текстовые данные в других кодировках и бинарные данные

Строковые литералы

- Если `unicode_semantics=off`, строковые литералы – это обычные 8-битные строки
- 1 символ = 1 байт

```
$str = "Hello, world!"; // кодировка ASCII
echo strlen($str);      // результат – 13


$jp = "検索オプション";   // кодировка UTF-8
echo strlen($str);      // результат – 21
```

Строковые литералы Unicode

- При `unicode_semantics=on`, строковые литералы переводятся в Unicode
- 1 символ может быть > 1 байт

```
// unicode_semantics = on
$str = "Hello, world!"; // строка Unicode
echo strlen($str);      // результат – 13


$jp = "検索オプション";   // строка Unicode
echo strlen($str);      // результат – 7
```

- Для получение длины строки в байтах следует использовать отдельную функцию

Бинарные строковые литералы

- Бинарные литералы требуют нового синтаксиса
- Содержание, т.е. последовательность байтов между разделителями, зависит от кодировки скрипта

```
// пусть исходный код написан в KOI8-R

$str = b'beer';    // байты: 62 65 65 72

$str = b"пиво";    // байты: D0 C9 D7 CF

$str = b<<<EOD
   раз\x01два
EOD;               // D2 C1 D8 0A C4 D7 C1
```

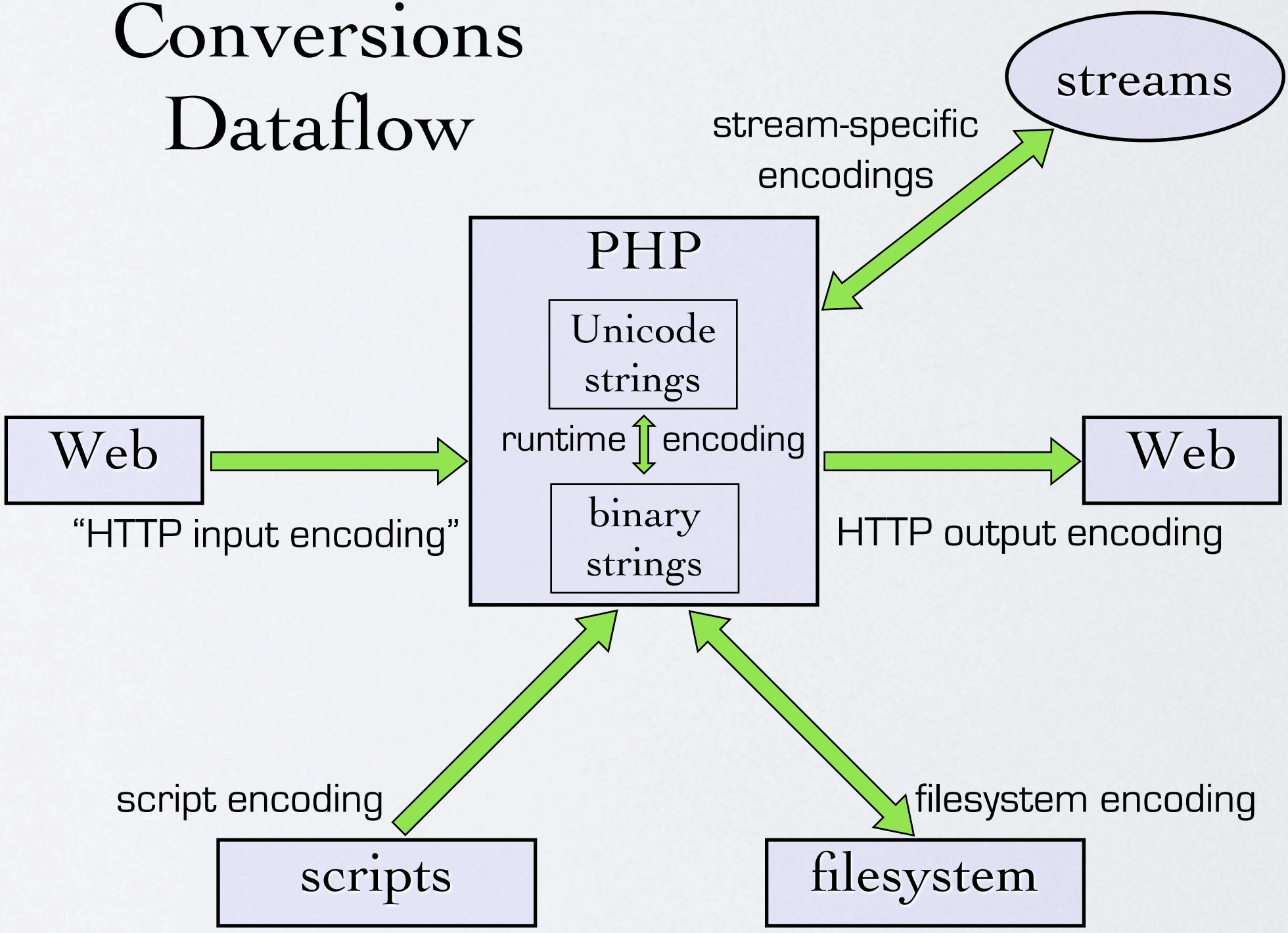
Escape Sequences

- Внутри строк типа Unicode можно использовать escape sequences `\uXXXX` и `\UXXXXXXX` для обозначения символов численным способом

```
// эти строки эквивалентны
$str = "Hebrew letter alef: א";
$str = "Hebrew letter alef: \u05D0";

// эти тоже
$str = 'tetragram: 𝌨';
$str = 'tetragram: \U01D328';
```

Conversions Dataflow



Кодировка в процессе исполнения

- Указывает кодировки, которая используется при конвертации бинарных строк в Unicode и наоборот

```
// unicode.runtime_encoding = koi8-r

$uni = "пиво";          // Unicode
$str = (binary)$str;   // бинарная строка в KOI8-R
$uni = (unicode)$uni; // обратно в Unicode
```

- Также используется при работе с функциями, которые еще не поддерживают строки в Unicode

```
$str = uniqid(); // бинарная строка в KOI8-R
```

Кодировка скрипта

- На данный момент скрипты могут быть в разных кодировках: ISO-8859-1, Shift-JIS, UTF-8 и др.
- Ядро должно знать кодировку скрипта, чтобы обрабатывать его правильным образом
- Кодировка может быть указана в **php.ini** или с помощью оператора `declare()`
- Влияет на интерпретацию идентификаторов и строк

Кодировка скрипта

- Какая бы ни была кодировка у скрипта, строковые литералы так или иначе будут в Unicode

```
// unicode.script_encoding = koi8-r
$uni = "пиво"; // строка Unicode

// unicode.script_encoding = utf-8
$uni = "пиво"; // также строка Unicode
```

- В обоих случаях `$uni` – это Unicode строка, содержащая 4 символа: `U+043F(п)`, `U+0438(и)`, `U+0432(в)` и `U+043e(о)`

Кодировка скрипта

- Кодировка может меняться с помощью оператора
- Оператор должен быть указан в самой первой строке скрипта
- Это не влияет на файлы, подключаемые с помощью include()

```
// unicode.script_encoding = utf-8

declare(encoding="koi8-r");
$uni = "пиво";// читается как строка KOI8-R
              // так как declare() замещает
              // INI-директиву

// содержимое файла читается как UTF-8
include "myfile.php";
```

Кодировка вывода

- Указывает кодировку, которая используется для вывода данных
  - `echo`, `print`, `var_dump()`, output buffering, и др.
- Вывод скрипта перекодируется на лету
- Бинарные строки не перекодируются

```
// unicode.output_encoding = koi8-r
// unicode.script_encoding = utf-8

$uni = "пиво"; // строка Unicode
echo $uni;     // конвертирует $uni на KOI8-R

echo b"пиво";  // без конвертации
```

Кодировка HTTP запросов

- При включенном режиме Unicode, необходимо перекодировать входящие данные HTTP-запросов в Unicode
- Запросы GET не имеют кодировки, а в запросах POST она указывается очень редко
- Определение кодировки – вещь ненадёжная
- Правильное декодирование HTTP запросов – это всё еще нерешенная проблема

Кодировка HTTP запросов

- Предполагается, что браузеры посылают данные в той же кодировке, в которой были данные на странице формы
- PHP будет пытаться декодировать их, используя `unicode.output_encoding`
- Можно также учитывать заголовок `Accept-Charset`
- Если перекодировка завершается с ошибкой, PHP будет считать, что к нему пришли пустые данные
- Приложения могут запрашивать повторную пересылку данных, используя другую кодировку

Кодировка файловой системы

- Указывает кодировку имен файлов и директорий
- Функции, работающие с файлами будут перекодировать имена файлов автоматически

```
// unicode.filename_encoding = koi8-r

$dh = opendir("/tmp/инвентаризация");
while (false !== ($file = readdir($dh)) {
   echo $file, "\n";
}
```

Кодировка по умолчанию

- Эта кодировку будет использоваться, если другие кодировки не указаны
- Упрощает конфигурирование
- Значение по умолчанию - UTF-8
- Если приложение работает только с данными в KOI8-R:

```
unicode.fallback_encoding = koi8-r
```

Конвертирование строк

- Unicode и бинарные строки могут быть конвертированы друг в друга явно или неявно
- При конверсии используется `unicode.runtime_encoding`
- Явные конверсии = приведение типов
  - `(binary)` приводит к бинарной строке
  - `(unicode)` приводит к строке Unicode
  - `(string)` приводит к строке Unicode, если `unicode_semantics` включена, если нет, то к бинарной строке

Конвертирование строк

- Неявные конверсии = конкатенация, сравнение, передача параметров
- Лучше конвертировать в Unicode: точность больше

```
// unicode.runtime_encoding = koi8-r

$uni = "Золотой";          // Unicode строка
$str = (binary)$uni;       // бинарная KOI8-R строка

$sum = $str . "телёнок";// $str конвертируется в Unicode
                           // и соединяется с литералом

if ($str < "слиток")      // $str конвертируется в Unicode
                           // и сравнивается с литералом

// предположим str_word_count() ещё не переписан
$n = str_word_count($sum); // $sum конвертируется в бинарную
                           // KOI8-R строку
```

Проблемы конвертации

- Unicode содержит все символы из множества старых кодировок
- Однако, многие символы Unicode не могут быть отображены в старых кодировках
  - Буква иврита `ט (U+05D8)` отсутствует в кодировке ISO-8859-1
- Строки могут содержать битые данные или нестандартные последовательности байтов

Поведение при ошибках конвертации

- Вы можете сами настроить поведение PHP при ошибках конвертации
- Глобальные установки по умолчанию применяются ко всем конвертациям
- Можно менять с помощью:

```
unicode_set_error_mode(direction, mode)
unicode_set_subst_char(character)
```

Режимы обработки ошибок конвертации

- `direction` = `FROM_UNICODE / TO_UNICODE`
- Возможные режимы: остановить конвертацию, пропустить символ, заменить символ или экранировать символ

```
// unicode.runtime_encoding = koi8-r
unicode_set_error_mode(FROM_UNICODE, U_CONV_ERROR_SUBST);
unicode_set_subst_char('*');
$uni = "< 力 >";

$str = (binary) $uni; // результат: “< * >”
```

Режимы обработки ошибок конвертации

```
// unicode.runtime_encoding = iso-8859-1
$str = (binary)“< 力 >”;
```

Режим	Результат
stop	“”
skip	“< >”
substitute	“< * >”
escape (Unicode)	“< {U+30AB} >”
escape (ICU)	“< %U30AB >”
escape (Java)	“< \u30AB >”
escape (XML decimal)	“< カ >”
escape (XML hex)	“< カ >”

Режимы обработки ошибок конвертации

```
// unicode.runtime_encoding = utf-8
$str = (unicode)b"< \xe9\xfe >"; // нестандарт. UTF-8
```

Режим	Результат
stop	""
skip	""
substitute	""
escape (Unicode)	"< %XE9%XFE >"
escape (ICU)	"< %XE9%XFE >"
escape (Java)	"< \xE9\xFE >"
escape (XML decimal)	"< é&#254 >"
escape (XML hex)	"< éþ >"

Режимы обработки ошибок конвертации

- PHP будет всегда останавливатся на битых или неверных данных, если не используется один из режимов экранирования
- Сообщение, которое вы получите, будет выглядеть таким образом:

```
Warning: Could not convert Unicode string to binary string
(converter KOI8-R failed on character {U+DC00} at offset 2)
```

- Или, если конвертировать бинарную строку в Unicode:

```
Warning: Could not convert binary string to Unicode string
(converter UTF-8 failed on bytes (0xE9) at offset 2)
```

Обычные конвертации

- Приведение типов – это просто лаконичный способ перекодировать строку с использованием runtime-кодировки
- Для других кодировок используйте следующие функции

```php
$uni = "Yahoo! по–русски";

// $str – бинарная строка в KOI8-R
$str = unicode_encode($uni, 'koi8-r', U_CONV_ERROR_SUBST);

// $res должна быть идентична $uni
$res = unicode_decode($uni, 'koi8-r');
```

Поддержка операторов

- Помните что смещения в строках работают не с байтами, а с символами!

```
$str = "大学";    // 2 символа
echo $str[1];    // результат – 学
$str[0] = 'サ';  // строка теперь – サ学
```

- Нет необходимости что-либо менять в коде, если вы работаете только с однобайтовыми кодировками, как ASCII или ISO-8859-1

Идентификаторы в Unicode

- В PHP будут разрешены идентификаторы с использованием Unicode
- Можно начать с старого, доброго..

```
class Component {
    function process { ... }
}

$schema = array();
$schema['part'] = new Component();
```

Идентификаторы в Unicode

- Несколько ударных символов не помешают..

```
class Bâtiment {
   function créer { ... }
}

$schématique = array();
$schématique['premier'] = new Bâtiment();
```

Идентификаторы в Unicode

- Потом вы узнаете еще пару языков...
- ...и начинается настоящее буйство

```
class コンポーネント {
    function ᓱᓕᓐ ᐊᓪᓚᑲᒃᑵᖅ      { ... }
    function சிவாஜி கனேசன்   { ... }
    function ᥖᥭᥰᥒᥬᥰᥛᥭᥱ         { ... }
}

$プロバイダ = array();
$プロバイダ[‘רַעְיוֹלוּחַ שַׁנַה’] = new コンポーネント();
```

Текстовые итераторы

- Оператор смещения `[]` для доступа к символам – довольно медленный (пока)
- Привыкайте к использованию `TextIterator`
- Это значительно быстрее и даёт вам доступ к различным частям текста, используя общий интерфейс
- Вы можете итерировать code units, code points, combining sequences, символы, слова, строки и предложения вперед и назад

Text Iterator

Поработаем с code points

```
$text = "la\u0300-bas"; // по настоящему là-bas
foreach (new TextIterator($text) as $u) {
    var_inspect($u);
}
```

Результат

```
unicode(1) "l" { 006c }
unicode(1) "a" { 0061 }
unicode(1) "`" { 0300 }
unicode(1) "-" { 002d }
unicode(1) "b" { 0062 }
unicode(1) "a" { 0061 }
unicode(1) "s" { 0073 }
```

Text Iterator

А теперь с символами

```
$text = "la\u0300-bas"; // по настоящему là-bas
foreach (new TextIterator($text,
          TextIterator::CHARACTER) as $u) {
    var_inspect($u);
}
```

Результат

```
unicode(1) "l" { 006c }
unicode(2) "à" { 0061 0300 }
unicode(1) "-" { 002d }
unicode(1) "b" { 0062 }
unicode(1) "a" { 0061 }
unicode(1) "s" { 0073 }
```

Text Iterator

Попросим слова в обратном порядке

```
$text = "Не подскажите как пройти к библиотеке? Спасибо.";
foreach (new ReverseTextIterator($text,
                    TextIterator::WORD) as $u) {
    if ($u != " ") echo($u),"\n";
}
```

Результат

```
.
Спасибо
?
библиотеке
к
пройти
как
подскажите
Не
```

Text Iterator

Теперь попросим границы строк

```
$text = "Pouvez-vous me dire quelle heure il est ? Merci.";
foreach (new TextIterator($text, TextIterator::LINE) as $u) {
    if ($u != " ") echo($u),"\n";
}
```

Результат

```
Pouvez-
vous
me
dire
quelle
heure
il
est ?
Merci.
```

Локали

- Забудьте про `setlocale()`
- Поддержка Unicode в PHP использует только локали ICU
- Локаль по умолчанию можно изменить/узнать с помощью:

```
locale_set_default()
locale_get_default()
```

Collation

- Collation (коллэйшн) – это процесс упорядовачения частей текстовой информации
- Зависит от определённой локали, языка, и документа

English: ABC...RSTUVWXYZ

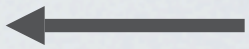
German: AÄB...NOÖ...SßTUÜV…YZ

Swedish/Finnish: ABC...RSTUVWXYZÅÄÖ

Collation

- Text в языках мира сортируется более чем одним способом
  - Немецкий: словарь или телефонная книга
  - Японский: радикал–черта или черта–радикал
  - Испанский: традиционный или современный
- Операторы сравнения в PHP не используют коллэйшн

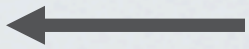
```
if ("côte" < "coté") {
    echo "less\n";
} else {
    echo "greater\n";
}
```



Collation

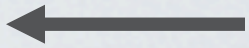
- Попробуем с коллэйшн

```
$coll = new Collator("fr_CA");
if ($coll->compare("côte", "coté") < 0) {
    echo "less\n";
} else {
    echo "greater\n";
}
```



- Можно не беспокоиться об ударных символах

```
$coll = new Collator("fr_CA");
$coll->setStrength(Collator::PRIMARY);
if ($coll->compare("côte", "coté") == 0) {
    echo "same\n";
} else {
    echo "different\n";
}
```



Collation

- Можно также сортировать массивы

```
$strings = array(
        "cote", "côte", "Côte", "coté",
        "Coté", "côté", "Côté", "coter ");
$coll = new Collator("fr_CA");
print(implode("\n", $coll->sort($strings));
```

```
cote
côte
Côte
coté
Coté
côté
Côté
coter
```

Collation

- Существует коллэйтор, который связан с локалью по умолчанию
- Его можно достать/узнать с помощью:

```
collator_get_default()
collator_set_default()
```

- Если поменять локаль по умолчанию, то этот коллэйтор меняется тоже
- Его можно установить отдельно от локали

Collation

- Полное API – очень мощное и настраиваемое
- Можно менять уровень коллэйшн, использовать численное упорядовачение, отменить или использовать уровень заглавных букв или знаков препинания, и многое другое
- Вы можете полагаться на то что коллэйшн алгоритм и данные будут обновляться наряду с каждой новой версией Unicode

Функции

- В PHP существуют тысячи функций
- Каждую из них надо проверить и выяснить нужно ли её переписывать на поддержку Unicode, и если да, то как именно
- API по чтению параметров поможет с автоматическим конвертированием в течении этого процесса

Функции

- Эта переделка фунций – довольно продолжительный процесс и потребует помощи авторов расширений
- Текущую статистику по переделке можно найти здесь:

http://www.php.net/~scoates/unicode/

Примеры

- `strtoupper()` и подобные выполняют правильную трансформацию заглавных букв

```
$str = strtoupper("fußball"); // результат: FUSSBALL

$str = strtolower("ΣΕΛΛΆΣ");  // результат: σελλάς
```

- `strip_tags()` понимает Unicode

```
$str = strip_tags("雅<span>虎</span>通"); // результат: 雅虎通
```

- `strrev()` сохраняет combining sequences

```
$u = "Vie\u0302\u0323t Nam";  // Việt Nam
$str = strrev($u);            // result is maN tệiV
```

Stream I/O

- В PHP система ввода/вывода основана на streams
- Обобщенные процедуры по работе с файлами, сетью, сжатием данных, и так далее
- PHP не может предполагать что данные на другом конце stream находятся в определённой кодировке
- Необходимо применить конвертации

Stream I/O

- По умолчанию, stream будет в бинарном режиме и никакой перекодировки происходить не будет
- Приложения могут перекодировать вручную:

```
$data = file_get_contents('mydata.txt');
$unidata = unicode_decode($data, 'CP1251');
```

- Но так как мы программисты народ ленивый, пусть это за нас делают streams

Stream I/O

- `t` режим - это не только для окончаний строк на Windows!
- Использует `encoding` настройку в контексте по умолчанию, значение которой – UTF-8, но её можно менять
- Чтение из тектового файла в кодировке UTF-8:

```
$fp = fopen('somefile.txt', 'rt');
$str = fread($fp, 100); // возвращает 100 символов Unicode
```

- Запись в такой же текстовый файл :

```
$fp = fopen('somefile.txt', 'wt');
fwrite($fp, $uni); // пишет данные в кодировке UTF-8
```

Stream I/O

- Если вы работаете в основном с файлами в другой кодировке, измените stream контекст

```
stream_default_encoding('CP1251');
$data = file_get_contents('somefile.txt', FILE_TEXT);
// ... работа над $data ...
file_put_contents('somefile.txt', $data, FILE_TEXT);
```

- Или создайте свой собственный контекст

```
$ctx = stream_context_create(NULL,
                array('encoding' => 'KOI8-R'));
$data = file_get_contents('somefile.txt', FILE_TEXT, $ctx);
// ... работа над $data ...
file_put_contents('somefile.txt', $data, FILE_TEXT, $ctx);
```

Stream I/O

- Если у вас есть stream который открыли в бинарном режиме, всё равно можно автоматизировать перекодировку

```
$fp = fopen('http://www.yahoo.com/', 'r');
stream_encoding($fp, 'utf-8');
$data = fgetss($fp, 1024); // читает 1024 символа Unicode
```

- Функция `fopen()` и ей подобные умеют определять кодировку ответа из заголовков
- Stream функции используют глобальные установки при ошибках конвертации, но это можно поменять своими параметрами контекстов

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

Повестка дня

- Вавилонская башня
- PHP в прошлом
- Unicode и локали
- PHP в будущем
- Текущее состояние

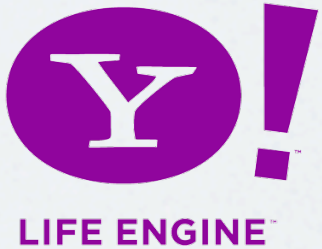
Что уже сделано?

- Весь код находится в главном архиве CVS
- PHP 6 snapshots доступны сегодня
- Работа над большинством из описанной функциональности уже завершена
- Часть ICU API тоже открыта
- Всё ещё идет активная разработка

Что осталось сделать?

- Закончить функциональность ядра
- Переписать приложенные модули
- Сделать все сервисы ICU доступными PHP
- Обновить руководство по PHP
- Оптимизировать скорость
- Обучать, обучать, обучать – всех!

Спасибо!



http://www.gravitonic.com/talks/