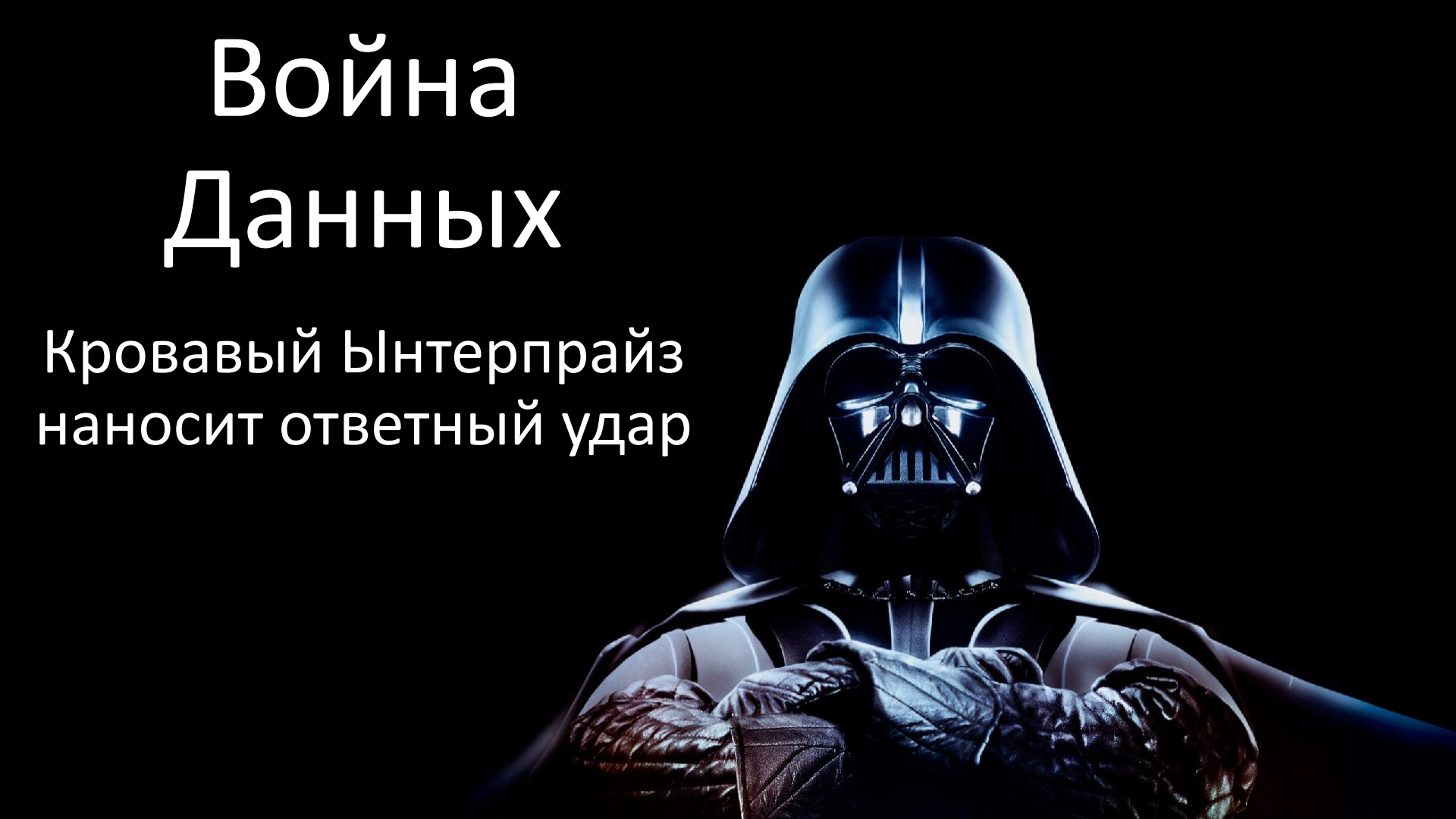
Война Данных

Кровавый ЫИнтерпрайз наносит ответный удар



Виктор Полищук

Давным давно,
в далекой
далекой
галактике...

infopulse

bics

@alkovictor

victor-cr

Смысл? Что? Зачем?

- 99% проектов имеют проблемы
- Большая доля этого отводится данным
- Разработка постоянно усложняется
- Часто проблему можно было предотвратить

Деньги, числа, арифметика

Идентификаторы

Текст и строки

Дата и время

Участие Приветствуется

Деньги

Деньги Типы

- Float и Double
- Integer и Long
- BigDecimal

Деньги → Float и Double → Хорошо

Встроенные типы

Большой диапазон значений

- Float: $1.4*10^{-45}...3.4028235*10^{+38}$
- Double: $4.9*10^{-324}...1.7976931348623157*10^{+308}$

Деньги → Float и Double → Беда

Обычно разработчики не знают как вот это представлено в памяти:

- $e = 2.718281828459045$
- $\pi = 3.141592653589793$
- $\tan\left(\frac{\pi}{2}\right) = 1.6331239353195370\boldsymbol{E}16$

Деньги

Float и Double

ЧЗХ #1

Float: 0.6 + 0.1

- = ?

Double: 0.6 + 0.1

- = ?

Деньги

Float и Double

ЧЗХ #1

Float: 0.6 + 0.1

- = 0.70000005

Double: 0.6 + 0.1

- = 0.7

Деньги

Float и Double

stackoverflow.com

6

The problem is not with Java but with the good standard float's (http://en.wikipedia.org/wiki/IEEE_floating-point_standard).

You can either:

- use Double and have a bit more precision (but not perfect of course, it also has limited precision)
- use a arbitrary-precision-library
- use numerically stable algorithms and truncate/round digits of which you are not sure they are correct (you can calculate numeric precision of operations)

share improve this answer

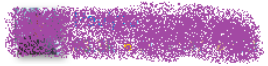
answered Jul 15 '11 at 22:12

2

You could use doubles instead of floats

share improve this answer

answered Jul 15 '11 at 22:08



Деньги

Float и Double

ЧЗХ #2

Float: 0.2 + 0.1

- = ?

Double: 0.2 + 0.1

- = ?

Деньги

Float и Double

ЧЗХ #2

Float: 0.2 + 0.1

- = 0.3

Double: 0.2 + 0.1

- = 0.30000000000000004

Деньги → Float и Double → Чё Внутри

Знак – плюс/минус

Экспонента – степень двойки

Мантисса – двоичное число

Деньги → Float и Double → Чё Внутри

Float (32 бита)

Double (64 бита)

Деньги → Float и Double → И Чё?

0.1f = [0]-[01111011]-[10011001100110011001101]

$0.1f = +\ 2^{123-127} * (1 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + \ldots)$

0.1f = 0.100000001490116119384765625

0.01f = [0]-[01111000]-[01000111101011100001010]

0.01f = 0.009999997764825820922851562

Деньги

Float и Double

Пример

Баварезе

Ø 30 см 450 г

1 шт x 79 грн — **79 грн**

Мексикано

Ø 30 см 450 г

1 шт x 79 грн — **79 грн**

PEPSI

Ø

1 шт x 0.0099999997764826 грн — **0.0099999997764826 грн**

Деньги → Float и Double → ЧЗХ #3

```
+∞     = [0]-[11111111]-[00000000000000000000000]
-∞     = [1]-[11111111]-[00000000000000000000000]
NAN    = [?]-[11111111]-[?????????????????????]
+0.0   = [0]-[00000000]-[00000000000000000000000]
-0.0   = [1]-[00000000]-[00000000000000000000000]
+0.0f == -0.0f?
```

Деньги → Float и Double → Чё Внутри

```
/**
 * Get or create float value for the given float.
 *
 * @param d the float
 * @return the value
 */
public static ValueFloat get(float d) {
    if (d == 1.0F) {
        return ONE;
    } else if (d == 0.0F) {
        // -0.0 == 0.0, and we want to return 0.0 for both
        return ZERO;
    }
    return (ValueFloat) Value.cache(new ValueFloat(d));
}
```

Деньги | Float и Double | Итог

- Очень узкая область применения
- Не очевидны
- Не использовать
- Забыть про существование

Деньги → Integer и Long → Хорошо

Встроенные типы

Целочисленная арифметика

- Нет деления на степень двойки

Деньги → Integer и Long → Как?

Храним деньги в «копейках»

Где-то храним «**scale**»

- например, как глобальную константу

Деньги → Integer и Long → Как?

«**Scale**» – это N значащих цифр после запятой

Домножаем или делим на 10^{scale}

Деньги

Integer и Long

ЧЗХ #1

$10_{(2)} * 230_{(2)} + 4_{(2)}$

- = ?

Деньги

Integer и Long

ЧЗХ #1

$10_{(2)} * 230_{(2)} + 4_{(2)}$

- $=0.1*2.3+0.04=27_{(2)}$

Деньги → Integer и Long → Как Надо?

```
@Embeddable
public class Amount implements Serializable {
    private int rate;
    @Transient
    private final int scale;

    public Amount() {
        scale = 6;
    }
    public Amount(int rate, int scale) {
        this.scale = scale;
        setRate(rate);
    }
    …
```

Деньги

Integer и Long

ЧЗХ #2

1.000000*0.150000

- = (int) ?
- = (long) ?

Деньги

Integer и Long

ЧЗХ #2

1.000000*0.150000

- = (int) -323855360
- = (long) 0.150000.000000

Деньги | Integer и Long | Итог

Лучше когда «scale» является частью значения

Лучше когда арифметика и сравнения просто работают

Лучше когда корректность гарантирована

Корректность почти всегда важнее скорости

Деньги → BigDecimal → Хорошо

- Встроенный тип
- Нет предыдущих проблем
- Очень большой диапазон

Деньги > BigDecimal > ЧЗХ #1

1.divide(2)

- =?

Деньги > BigDecimal > ЧЗХ #1

1.divide(2)

- =0.5

2.divide(3)

- =?

2.divide(3)

- =ArithmeticException

Деньги > BigDecimal > ЧЗХ #2

2.divide(3, 3, …)

- =0.666 или 0.667

Деньги BigDecimal ЧЗХ #3

```
1.divide(10).multiply(10).equals(1)
```

- =?

Деньги → BigDecimal → ЧЗХ #3

```
1.divide(10).multiply(10).equals(1)
```

- = false

Деньги BigDecimal ЧЗХ #4

```
BD.valueOf(0.1f).equals(new BD(0.1f))
```

- =?

Деньги BigDecimal ЧЗХ #4

```
BD.valueOf(0.1f).equals(new BD(0.1f))
```

- = false

Деньги BigDecimal ЧЗХ #5

```
BigDecimal i = ONE;

while (i.compareTo(TEN) < 0) {
      i.add(TEN).multiply(i)
.divide(valueOf(3L), 0, ROUND_HALF_EVEN);
}

System.out.println(i);
```

Деньги BigDecimal ЧЗХ #5

41

ArithmeticException

Ничего

19

Деньги BigDecimal ЧЗХ #5

```
BigDecimal i = ONE;

while (i.compareTo(TEN) < 0) {
      i.add(TEN).multiply(i)
.divide(valueOf(3L), 0, ROUND_HALF_EVEN);
}

System.out.println(i);
```

Деньги BigDecimal ЧЗХ #5

41

ArithmeticException

Ничего

19

Деньги BigDecimal ЧЗХ #5

```
BigDecimal i = ONE;

while (i.compareTo(TEN) < 0) {
      i.add(TEN).multiply(i)
.divide(valueOf(3L), 0, ROUND_HALF_EVEN);
}

System.out.println(i);
```

Деньги → BigDecimal → Итог

- Точность и значность известна
- Арифметика и сравнения включены
- Поддерживается в JDK, JDBC и т.д.
- Неплохая производительность


Идентификаторы

ID Содержание

Integer и Long

UUID

ID

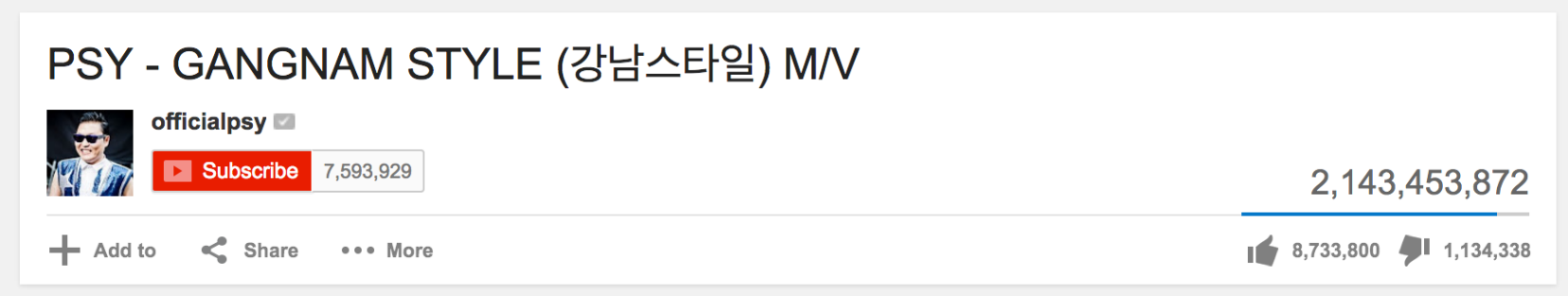
Integer и Long

Integer и Long – конечные типы

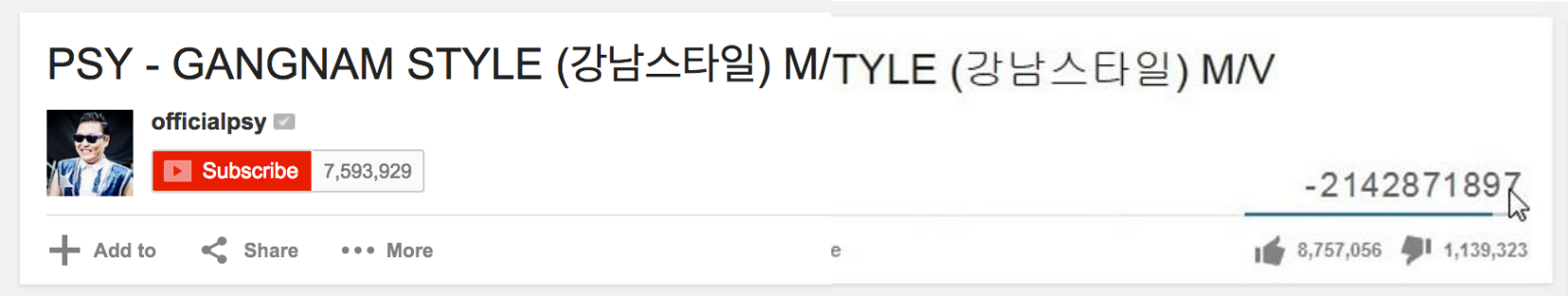
Иногда они переполняются

Они в два раза меньше, чем мы думаем

ID	Integer и Long	Пример



ID	Integer и Long	Пример



ID → Integer и Long → Нежданчик

«Некоторые» языки не умееют работать с 64-битными целыми

Кроме того, «некоторые» языки работают со специальными 32-битными целыми

Разница между DB и API идентификаторами

Если не на 99.9% уверен

- Используй целочисленные ключи в DB
- Используй строковые идентификаторы для API
- Не используй 32- битные типы

ID → UUID → Хорошо

Генерирует ID локально

Довольно мал: 128 бит (16 байт)

Очень маленькая вероятность коллизии

- 1 на 2.71 квинтиллионов (10^{18})
- Вероятность эквивалентна встрече 180 мужчин подряд

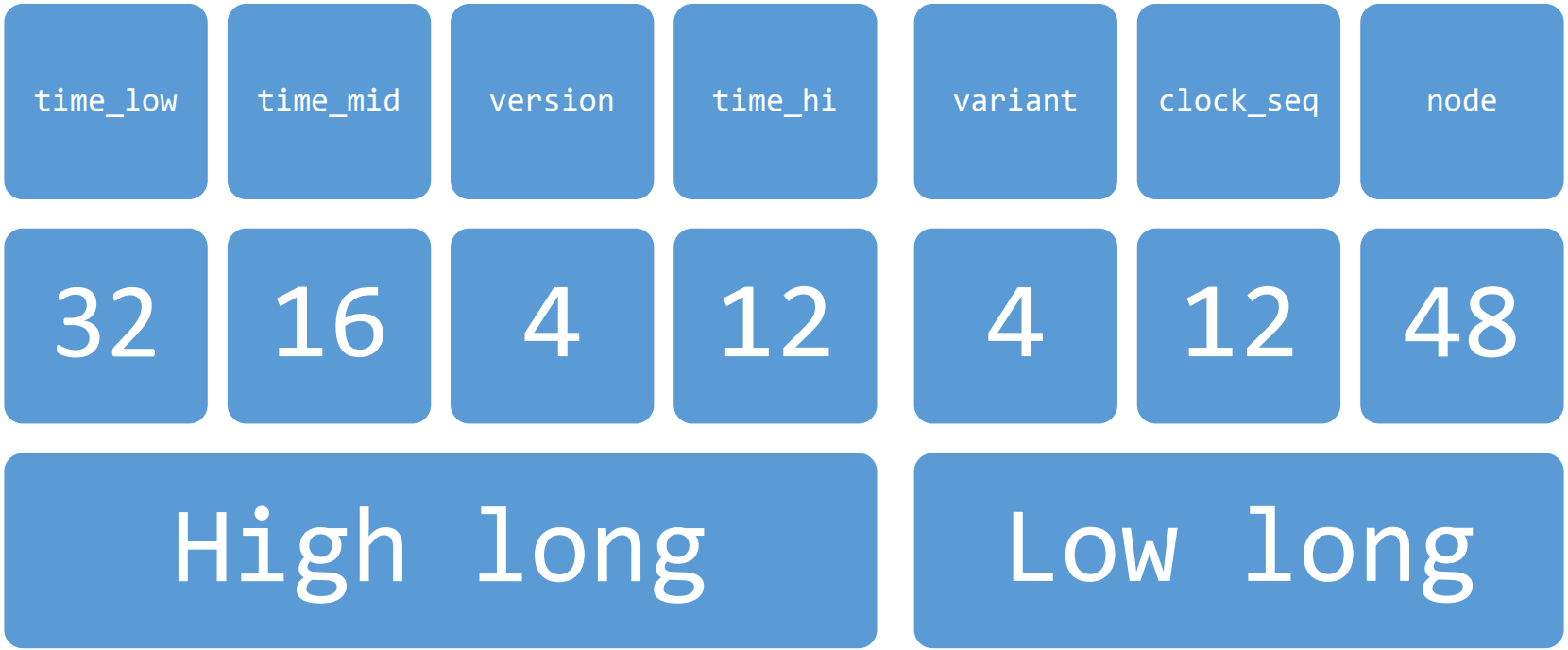
Не гарантированно **глобально** уникальны

Могут быть источником проблем с производительностью

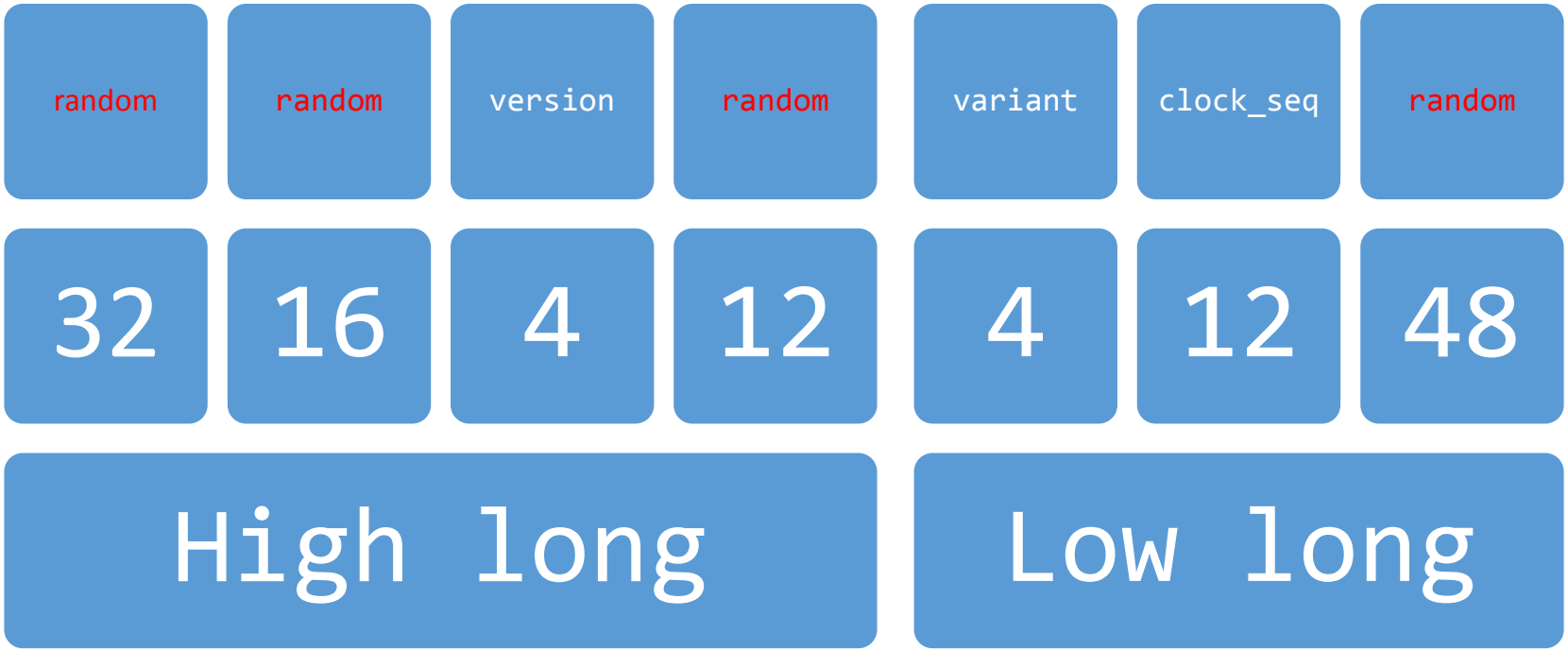
Существует несколько версий с разной степени полезности

Странно, но RDBMS редко поддерживают UUID/GUID типы

ID UUID Чё внутри



ID UUID Чё внутри




```
$ uuidgen
0c8aa0f6-9f6f-4fad-9662-1b683f2f4a0d
$ uuidgen
1ee09695-3a04-4e7a-8bab-e67dabc4b5a2
$ uuidgen -t
3770f4d0-88b3-11e6-bba6-005056bb68cb
$ uuidgen -t
3b14dd54-88b3-11e6-8c53-005056bb68cb
```

ID UUID v4

- Зависит от генератора энтропии
- Виртуальные машины
- Контейнеры

ID Итог

Используйте внешнее строковое представление (API)

Внутреннее представление должно быть:

- DB счетчик (Long/BigInteger)
- UUID + DB счетчик (Long/BigInteger)
- UUID
- Twitter Snowflake (Long) – не развивается
- *Flake (128 bit)

Строки

Строки

Содержание

Кодировка строк в Java

JDBC драйвера и DB типы

Строки

Кодировка

Java использует UTF-16 для строк

Кодовая таблица UTF-16: 0x0000..0x10FFFF

Строки используют char[] (byte[] in JDK9)

Возможные значения char: 0x0000..0xFFFF

Как представить символы 0x100000..0x10FFFF используя char?

Строки > Кодировка > ЧЗХ #1

Определим «суррогатный» диапазон: 0xD800..0xDFFF [0x800] [2048]

Поделим его пополам:

- «High»: 0xD800..0xDBFF [0x400] [1024]
- «Low»: 0xDC00..0xDFFF [0x400] [1024]

Используем пары (High;Low) для получения недостающих символов

...

PROFIT!!!

Строки > Кодировка > ЧЗХ #2

```
String x = new String(new char[]{
        'z',0xD801,0xDC05,'a','b','c'
});
System.out.println(x.length());
System.out.println(x);
System.out.println(x.substring(0,2));
System.out.println(x.substring(2));
```

Строки > Кодировка > ЧЗХ #2

```
б
z0abc
z?
?abc
```

Строки → Кодировка → Решение

```
Character.isLowSurrogate(...);
Character.isHighSurrogate(...);
```

Строки > Кодировка > Итог

Хорошего решения нет

Надо об этом помнить

Строки JDBC vs DB

- Отображение JDBC типов на DB
- Некоторые примеры
- BLOB или CLOB
- Несовместимые кодировки

Строки

JDBC vs DB

Отображение

```
public enum JDBCType implements SQLType {
   CHAR(Types.CHAR),
   VARCHAR(Types.VARCHAR),
   LONGVARCHAR(Types.LONGVARCHAR),
   ...
   BLOB(Types.BLOB),
   CLOB(Types.CLOB),
   ...
   NCHAR(Types.NCHAR),
   NVARCHAR(Types.NVARCHAR),
   LONGNVARCHAR(Types.LONGNVARCHAR),
   NCLOB(Types.NCLOB),
```

BLOB – это не текст

И никогда не будет

Строки > JDBC vs DB > BLOB/CLOB

Binary Large OBject

- Массив байт

Character Large OBject

- Массив символов в «некоторой» кодировке

Строки > JDBC vs DB > Отображение

```
public enum JDBCType implements SQLType {
   CHAR(Types.CHAR),
   VARCHAR(Types.VARCHAR),
   LONGVARCHAR(Types.LONGVARCHAR),
   ...
   CLOB(Types.CLOB),
   ...
   NCHAR(Types.NCHAR),
   NVARCHAR(Types.NVARCHAR),
   LONGNVARCHAR(Types.LONGNVARCHAR),
   NCLOB(Types.NCLOB),
```

Строки

JDBC vs DB

CHAR и VARCHAR

Тип	Oracle	MySQL	MS SQL
CHAR	2000 b	255 char	8000 b
VARCHAR	4000 b	255 char 65535 b	8000 b ~ 2 Gb

Строки JDBC vs DB Чё внутри?

```
setStringInternal(int var1,String var2) throws SQLException {
   ...
   int var6 = var2 != null?var2.length():0;
   ...
   if(var6 <= this.maxVcsCharsSql) {
       this.basicBindString(var1, var2);
   } else if(var6 <= this.maxStreamNCharsSql) {
       this.setStringForClobCritical(var1, var2);
   } else {
       this.setStringForClobCritical(var1, var2);
   }
```

Строки > JDBC vs DB > CHAR/NCHAR

CHAR, VARCHAR, CLOB…

- Использует кодировку DB

NCHAR, NVARCHAR, NCLOB…

- Использует указанную (национальную) кодировку

Строки > JDBC vs DB > Совместимость

- Иногда у вас нет проблем с кодировками
- Иногда «умный» драйвер все портит
- Иногда кодировки несовместимы
- Используйте Unicode семейство в DB

Строки | JDBC vs DB | Итог

- Используй Unicode кодировку DB
- Если уже поздно пользуйся N* DB типами
- Не храни текст как BLOB
- Знай специфику драйвера и DB
- Символ – это не байт, и часто даже не два

Дата и Время

Дата

Содержание

Базис: Instant и Date

DST и високосная магия

Временные зоны

Дата Базис Инфо

Instant

- Представляет некое временное событие отсчитаное от точки отсчета в равномерных интервалах

DateTime

- Кортеж, неравномерных интервалов приятных для человеческого восприятия, чаще всего привязанных к циклам солнца и/или луны. Год-месяц-день и т.д.

Дата Базис И Чё?

Instant

- это «абсолютное» земное время

DateTime

- это представление времени, конвертируемое в Instant, но не всегда однозначно.

Дата Базис Итог

Instant

- используй если важна причинность, и если даже [милли-]секунда важна

DateTime

- используй во всех остальных случаях

Дата

DST и Магия

Пропущеные и лишние часы

«Високосные» секунды

Дата → DST и Магия → Вычисления

24 часа в дне

60 минут в часе

60 секунд в минуте

Итого: 24 * 60 * 60 * 1000 мс/день

Дата DST и Магия ЧЗХ #1

26.03.2017-27.03.2017 EET

- 23 часа (нет 26.03.2017 03:00-04:00)

29.10.2017-30.10.2017 EET

- 25 часа (два 29.10.2017 03:00-04:00)

Дата | DST и Магия | ЧЗХ #2

31.12.2016-01.01.2017 MSK

- 24 часа

01.01.2017-02.01.2017 MSK

- 24 часа и 1 секунда

Дата → DST и Магия → Високосные секунды

Было: 31.12.2016 23:59:60 (UTC)

Пока нет точного графика

Виновата Земля, Луна, Солнце и т.д.

Разработчики тоже виноваты

Дата Зоны

Временные зоны

- позволяют работать с локальным временем

Правила конвертации

- позволяют преобразовывать в Instant
- позволяют преобразовывать в другие зоны

Дата Зоны Запись

```
PreparedStatement.setTimestamp(
      1,
      new Timestamp(...),
      Calendar.getInstance(
            TimeZone.getTimeZone("UTC")
      )
);
```

Дата Зоны Чтение

```
Timestamp ts = ResultSet.getTimestamp(
      1,
      Calendar.getInstance(
            TimeZone.getTimeZone("...")
      )
);
```

Дата Зоны ЧЗХ #3

Сохранили 2017-03-26 12:00+00:00

- UTC =?
- Москва =?
- Новосибирск =?
- *BD* =?

Дата Зоны ЧЗХ #3

Сохранили 2017-03-26 12:00+00:00

- UTC = 2017-03-26 15:00+03:00 [0]
- Москва = 2017-03-26 12:00+03:00 [-3]
- Новосибирск = 2017-03-26 08:00+03:00 [-7]
- *BD* = 2017-03-26 12:00:00.0

Дата Зоны ЧЗХ #3

Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the timestamp if the underlying database does not store timezone information.

- **columnIndex** - the first column is 1, the second is 2, ...
- **cal** - the java.util.Calendar object to use in constructing the timestamp

Дата Зоны А если надо?

TIMESTAMP

TIMESTAMP WITH TIME ZONE

DATETIMEOFFSET

Дата Зоны ЧЗХ #4

Сохранили 2017-03-26 12:00+00:00

- MS SQL v4 =?
- Oracle v10 =?
- Oracle v12 =?

Дата Зоны ЧЗХ #4

Сохранили 2017-03-26 12:00+00:00

- MS SQL v4 = 2017-03-26 12:00+00:00
- Oracle v10 = 2017-03-26 12:00+00:00
- Oracle v12 = 2017-03-26 12:00+03:00

Дата Зоны oracle.sql.TIMESTAMP

```
public static byte[] toBytes(Timestamp var0, Calendar var1) {
    if (var0 == null) { return null; }
    int var2 = var0.getNanos();
    byte[] var3 = var2 == 0 ? new byte[7] : new byte[11];
    Calendar var4 = var1 == null ? Calendar.getInstance() : Calendar.getInstance(var1.getTimeZone());

    var4.setTime(var0);
    int var5 = getOracleYear(var4);
    var3[0] = (byte) (var5 / 100 + 100);
    var3[1] = (byte) (var5 % 100 + 100);
    var3[2] = (byte) (var4.get(2) + 1);  // Calendar.MONTH
    var3[3] = (byte) var4.get(5);        // Calendar.DATE (DAY_OF_MONTH)
    var3[4] = (byte) (var4.get(11) + 1); // Calendar.HOUR_OF_DAY
    var3[5] = (byte) (var4.get(12) + 1); // Calendar.MINUTE
    var3[6] = (byte) (var4.get(13) + 1); // Calendar.SECOND
    if (var2 != 0) {
        var3[7] = (byte) (var2 >> 24);
        var3[8] = (byte) (var2 >> 16 & 255);
        var3[9] = (byte) (var2 >> 8 & 255);
        var3[10] = (byte) (var2 & 255);
    }
    return var3;
}
```

Дата Зоны ЧЗХ #4

Сохранили 2017-03-26 12:00+00:00

- MS SQL v4 = 2017-03-26 12:00+00:00
- Oracle v10 = 2017-03-26 12:00+00:00
- Oracle v12 = 2017-03-26 12:00+00:00

Дата Зоны Что делать?

Использовать только UTC

Использовать “странный” API

- void setObject(int, Object, int, int)

Использовать java.time API или Calendar

- ZonedDateTime and etc

Не верить имплементаторам JDBC

Дата Зоны Вопросы

- JavaScript клиент с непонятной зоной
- Java сервер с +03:00 зоной
- Как передавать даты?

Дата Зоны Ответы

Дата не определена в спецификации JSON

Long – это плохая идея для JavaScript

String как ISO 8601 меньшее зло

Дата Зоны Итог

Используй единую временную зону

Тестируй драйвер на ожидаемое поведение

- Например: https://github.com/victor-cr/jdbc-test

*Не используй setTimestamp(…) для Instant

Публикуй дату как строку в JSON API

Дата Итог

- Используй UTC как можно чаще
- Помни про разницу даты и Instant
- Работай с датами по закону Мерфи
- 24*60*60*1000 – это небезопасное упрощение
- Используй специальные библиотеки для работы с датами

Спасибо