DotNext-2016

Подводные камни Date & Time

Фофанов Илья

Илья Фофанов

ENGINEER SPOCK

http://engineerspock.com

Почему важно уметь работать с датами и временем?

- Источник огромного количества багов
- Говорят, что, познав Date & Time, можно стать **Тёмным Лордом Времени!**

Date & Time Bugs

1996 год, Новая Зеландия.

Плавильные печи остановились в канун НГ.

Date & Time Bugs

2008 год.

MP3-плеер Zune завис в канун НГ.

Date & Time Bugs

Отказы в платежах
по банковским картам.

План доклада

- Проблема измерения времени
- Часовые пояса и их непредсказуемость
- Поставщики данных по часовым поясам и локализация наименований часовых поясов
- Структура DateTime. Проблемы с перспективой
- DateTimeOffset
- Арифметика на датах
- Проблема парсинга
- TimeZoneInfo
- Best Practices
- Обзор Noda Time

Проблемы измерения времени

Астрофизические явления

Человеческий фактор

Грегорианский календарь

- Введён в 1582 году
- Юлианский календарь накапливал ошибку в 1 день каждые 128 лет
- Теперь високосными считаются те вековые годы, число сотен лет которых без остатка делится на 4 (1600, 2000, 2400)
- За 4 октября 1582 года следует 15 октября

А что если…

Создать несуществующую дату?

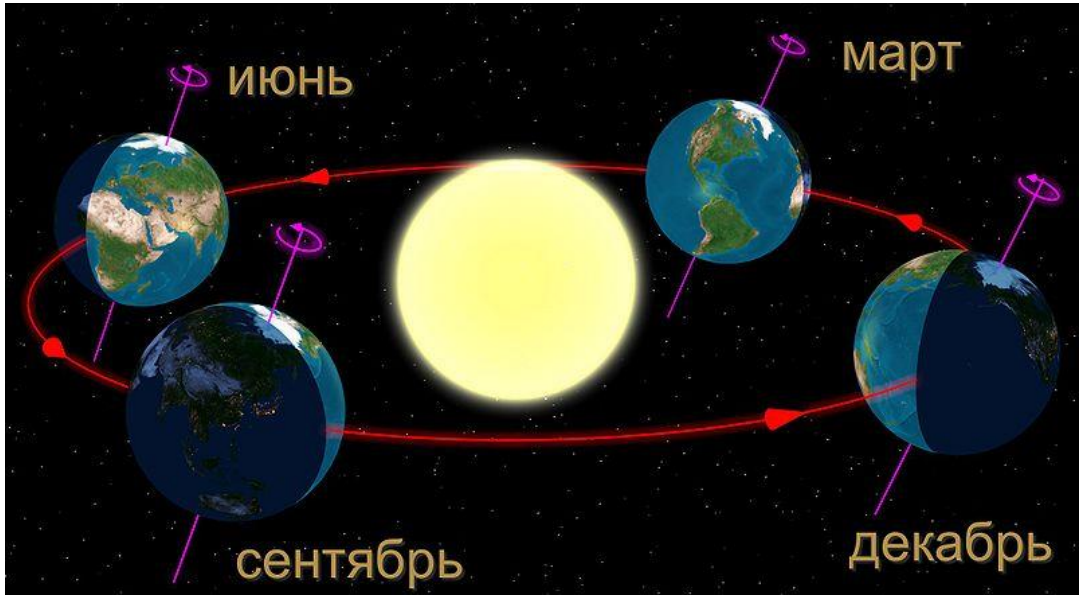
```
new DateTime(1582, 10, 8);//С 5 по 14 дней не существует
```

Всё ОК – объект будет создан,
Грегорианский календарь – **пролептический!**

Не может быть!

Может ли минута длиться
61 секунду?

Может! 31 декабря 2016 после 23:59:59 наступит 23:59:**60**
Будет добавлена **високосная (координационная) секунда!**



Source: https://commons.wikimedia.org/wiki/File:North_season.jpg

Человеческий фактор

Многострадальные часовые пояса!

Универсальное время

- GMT с 1847 года и UTC с 1972 года
- UTC и GMT синхронизированы
- UTC основан на атомном времени
- По умолчанию следует полагаться на UTC
- В Москве UTC+03:00

Зимнее и летнее время

А не сдвинуть ли время
на часочек вперёд?

Пора переводить стрелки
часов назад!

Часовой пояс

- Ограничен географическим регионом
- Имеет международное юридическое наименование:
  Moscow Standard Time (MSK), Moscow Daylight Time (MSD)
- Имеет стандартное смещение от UTC
  Central Standard Time (CST) = UTC-06:00
- Может иметь летнее время (DST-daylight saving time):
  Mountain Daylight Time = UTC-06:00

Часовой пояс

- Включает правила перехода между летним и зимним временем:
  - Дни перехода с зимнего на летнее и обратно
  - Время перевода часов
  - Величину смещения:

    India Standard Time (IST) = UTC+05:30,
    Nepal Time (NPT) = UTC+05:45
    Netherlands 1909-1937: GMT+00:19:32.13
    1937-1940: GMT+00:20
    1940-1942: UTC+02:00

Часовой пояс

- Включает историю изменений:
  - Правил переходов между летним и зимним временем
  - Абсолютных корректировок

Ещё немного странного

Может ли отсутствовать **30 декабря**
после перехода на Грегорианский календарь?

Может!
В 2011 году Самоа изменила смещение в часовом поясе
с UTC-10 на UTC+14.
Технически было **отменено 30-е декабря.**

БД часовых поясов

IANA

Internet Assigned Numbers Authority

https://www.iana.org/time-zones

- Набор текстовых файлов со списком часовых зон и правил переходов между летним и зимним временем
- Для программного использования текстовые файлы компилируются в бинарные для каждого часового пояса.

Преимущества IANA

Internet Assigned Numbers Authority

https://www.iana.org/time-zones

- Используется большинством стандартов
- Активное коммьюнити, есть мэйлинг-листы
- Полная открытость
- Последнюю версию можно скачать одной командой по FTP
- Поддерживается множеством платформ: .NET через Noda Time

Microsoft Time Zones

- Проприетарная БД
- Хранится в реестре Windows
- Можно опрашивать через WinAPI,
  а в .NET через TimeZoneInfo

Microsoft Time Zones

Недостатки:

- Неадекватное именование часовых поясов
  - Париж – Romance Standard Time
    Нормальное название - Central European Time
- Локализация наименований завязана на локали ОС
- Ограниченная поддержка исторических данных
- Поддерживается только Windows

Локализация наименований часовых поясов

Common Locale Data Repository (CLDR) – www.cldr.unicode.org

- Большой источник различных локализаций:
  для стран, регионов, часовых поясов, месяцев, валют.
- Можно скачать их базу и
  внутри найти файлы в формате XML
- Поддерживает маппинг между
  IANA Time Zones DB и Windows Time Zones DB.

DateTime Structure

Range: 0001-01-01T00:00:00.00000000 –
9999-12-31T11:59:59.99999999

Precision: 1 тик (100 наносекунд)
В реальности 1-15 миллисекунд

DateTime Structure

```
new DateTime(2016, 12, 9, 14, 0, 0);
```

Где протекает это время?

Ответ: нигде! Перспектива не задана.

DateTime Structure

```
new DateTime(2016, 12, 9, 14, 0, 0,
             DateTimeKind.Utc);
```

Перспектива задаётся через DateTimeKind enum.

Три значения DateTimeKind:

- Unspecified
- Local
- Utc

DateTimeKind.Unspecified

```
new DateTime(2016, 12, 9, 14, 0, 0,
             DateTimeKind.Unspecified);
```

«Плавающие» (floating) дата и время.

DateTimeKind.Unspecified

```
var dt = new DateTime(2016, 12, 9, 14, 0, 0);
var localDt = dt.ToLocalTime();
var utcDt = dt.ToUniversalTime();

Console.WriteLine(localDt.ToString("t")); //17:00
Console.WriteLine(utcDt.ToString("t"));   //11:00
```

DateTimeKind.Local

```
new DateTime(2016, 12, 9, 14, 0, 0,
             DateTimeKind.Local);
```

DateTime.Now и DateTime.Today
выставляют DateTimeKind.Local.

Проблемы DateTimeKind.Local

- Local – перспектива локальной машины
- Ничего неизвестно о часовом поясе
- Сбрасывается в Unspecified при любых круговых преобразованиях
  - сериализации/десериализации
  - сохранении в БД/загрузки из неё
  - при передаче 3-й стороне и возврате обратно

DateTimeKind.Local

Можно использовать только для отображения текущего времени пользователя на клиентской машине!

DateTimeKind.Utc

```
new DateTime(2016, 12, 9, 14, 0, 0,
             DateTimeKind.Utc);
```

DateTime.UtcNow выставляет DateTimeKind.Utc.

- Арифметические операции между UTC-значениями безопасны
- UTC недружелюбен к конечному пользователю

DateTimeKind.Utc

```
DateTime dtUtc =
        (DateTime)dataReader[“dateTimeInUtc”];
dtUtc = DateTime.SpecifyKind(dtUtc,
                             DateTimeKind.Utc);
```

DateTime Demo

```
// 2 разных UTC отображающихся на «одно и то же» локальное
var original1 = new DateTime(2014, 10, 25, 21, 30, 00, DateTimeKind.Utc);
var original2 = new DateTime(2014, 10, 25, 22, 30, 00, DateTimeKind.Utc);

// 2014-10-26T01:30:00.0000000+04:00. Kind:Local
var local1 = original1.ToLocalTime();

// 2014-10-26T01:30:00.0000000+03:00. Kind:Local
var local2 = original2.ToLocalTime();


Console.WriteLine(local1 == local2); // True
```

DateTime Demo

```
// 01:30AM до перехода на зимнее
var original1 = new DateTime(2014, 10, 25, 21, 30, 00,
                             DateTimeKind.Utc);
// 2014-10-26T01:30:00.0000000+04:00. Kind:Local
var local1 = original1.ToLocalTime();

var tmpLocal1 = DateTime.SpecifyKind(local1, local1.Kind);

Console.WriteLine(local1.ToUniversalTime()
                        .ToString("t"));      // 21:30
Console.WriteLine(tmpLocal1.ToUniversalTime()
                        .ToString("t"));      // 22:30
```

DateTime

Внутри DateTime время хранится в поле типа ulong (64 бита).
1-62 биты хранят время, прошедшее с 0001-01-01.
63-64 биты хранят значение DateTimeKind.

Есть «секретный» DateTimeKind:

```
const UInt64 KindLocalAmbiguousDst =
                                    0xC000000000000000;
```

DateTimeOffset

Позволяет создать дату и время со смещением от UTC.

Range: 0001-01-01T00:00:00.00000000 –
9999-12-31T11:59:59.99999999

Перспектива технически = Unspecified.
Перспектива семантически = Local, определяемая смещением.

DateTimeOffset

```
var dt1 = new DateTime(2014, 10, 24,
                       15, 30, 00,
                       DateTimeKind.Unspecified);

// from Unspecified
DateTimeOffset dto1 = dt1;

// from Unspecified with Offset
var dto2 = new DateTimeOffset(dt1,
                              TimeSpan.FromHours(6));
```

DateTimeOffset

```
var dt1 = new DateTime(2014, 10, 24,
                       15, 30, 00,
                       DateTimeKind.Local);

// from Local
DateTimeOffset dto1 = dt1;

// from Local with Offset
var dto2 = new DateTimeOffset(dt1,
                              TimeSpan.FromHours(6));
```

DateTimeOffset

```
var dt3 = new DateTime(2014, 10, 24,
                       15, 30, 00,
                       DateTimeKind.Utc);

// from UTC
var dto4 = new DateTimeOffset(dt3);
```

DateTimeOffset

Однозначно определяет момент времени.

На самом деле ничего не знает
о часовом поясе!

Арифметика на датах

28 января 2017 + 1 месяц + 1 день = 1 марта 2017
29 января 2017 + 1 месяц + 1 день = 1 марта 2017

1 марта 2017 – 1 месяц – 1 день = 31 января 2017

Арифметика на датах

```
var dt = new DateTime(2016, 2, 29);
dt = new DateTime(dt.Year + 1,
                  dt.Month,
                  dt.Day);

Console.WriteLine(dt.ToString("o"));
```

Арифметика на датах

```
var dt = new DateTime(2016, 2, 29);
dt = dt.AddYears(1);
dt = dt.AddMonths(1);
dt = dt.AddDays(-1);
dt = dt.AddHours(-1);
```

Арифметика на времени

```
var dt = new DateTime(2014, 10, 26,
                      00, 00, 00,
                      DateTimeKind.Local);
var dt2 = dt.AddHours(2);

// 2014-10-26T02:00:00.00000000+03:00
Console.WriteLine(dt2.ToString("o"));
```

Арифметика на времени

```
var source = new DateTime(2014, 10, 26,
                          00, 00, 00,
                          DateTimeKind.Local);
var universal = source.ToUniversalTime();
var updatedUniversal = universal.AddHours(2);

var result = updatedUniversal.ToLocalTime();

//2014-10-26T01:00:00.00000000+03:00
Console.WriteLine(dt2.ToString("o"));
```

Парсинг

При парсинге строки, по умолчанию используется текущая культура потока, на котором исполняется парсинг.

Парсинг

```
var dateString = "03/01/2009 10:00 AM";
var culture = CultureInfo.CreateSpecificCulture("en-US");
var dtEnUs = DateTime.Parse(dateString, culture);
var dtRuRu = DateTime.Parse(dateString);
// Prints: En-Us Month:3
Console.WriteLine("En-Us Month:{0}", dtEnUs.Month);
// Prints: Ru-Ru Month:1
Console.WriteLine("Ru-Ru Month:{0}", dtRuRu.Month);
```

Парсинг

Если знаете формат строки – предпочитайте ParseExact.

Методы парсинга принимают DateTimeStyles enum.

DateTimeStyles.RoundtripKind

DateTimeStyles.RoundtripKind

```
string str = "2016-12-09T15:30:00Z";

DateTime dtParsed = DateTime.Parse(str);
Console.WriteLine(dtParsed.Kind); // Unspecified
```

Сохраняя перспективу:

```
DateTime dtParsed = DateTime.Parse(str,
                    CultureInfo.InvariantCulture,
                    DateTimeStyles.RoundtripKind);
Console.WriteLine(dtParsed.Kind); // Utc
```

Парсинг в DateTimeOffset

Если строка содержит смещение,
лучше парсить в `DateTimeOffset`.

```
DateTimeOffset dtoParsed =
                    DateTimeOffset.Parse(input);

PassString(dtoParsed.ToString("o"));
```

Парсинг в DateTime

```
//пример запускаем в Московском часовом поясе
var dt1 = new DateTime(2016, 12, 9,
                       15, 00, 00,
                       DateTimeKind.Unspecified);

// 2016-12-09T15:00:00.0000000+06:00
var dto1 = new DateTimeOffset(dt1,
                              TimeSpan.FromHours(6));

var dtParsed = DateTime.Parse(dto1.ToString("o"));

// 2016-12-09T12:00:00.0000000+03:00, Local
Console.WriteLine($"After: {dtParsed:o}, {dtParsed.Kind}");
```

TimeZoneInfo

Тип `TimeZoneInfo` поддерживает
только Microsoft Time Zones.

Поддерживает конверсию:

- В локальное и из локального времени
- В UTC и из UTC
- Между любыми часовыми поясами

Из UTC в часовой пояс

```
DateTime dtUtc = DateTime.UtcNow;

string zone = "Russian Standard Time";
TimeZoneInfo tzi = TimeZoneInfo
                       .FindSystemTimeZoneById(zone);
DateTime local = TimeZoneInfo
                       .ConvertTimeFromUtc(dtUtc, tzi);

Console.WriteLine($"Kind: {local.Kind}"); // Unspecified
```

Из UTC в часовой пояс

```
DateTimeOffset dtUtc = DateTimeOffset.UtcNow;

string zone = "Russian Standard Time";
TimeZoneInfo tzi = TimeZoneInfo
                       .FindSystemTimeZoneById(zone);

DateTimeOffset local = TimeZoneInfo
                           .ConvertTime(dtUtc, tzi);
```

Best Practices

Потеря перспективы

Смещение или перспектива не должны теряться при преобразованиях, особенно круговых.

- В случае парсинга строки, содержащей время в UTC используйте DateTimeStyles.RoundtripKind.
- При неизбежной потере перспективы, постарайтесь её восстановить через вызов DateTime.SpecifyKind(DateTimeKind.Utc);

Замеры истёкшего времени

```
DateTime start = DateTime.Now;
…
DateTime finish = DateTime.Now;
TimeSpan period = finish – start;
```

Полагайтесь на `DateTime.UtcNow` или `DateTimeOffset`!

Высокоточные замеры

Используйте `Stopwatch` вместо `DateTime`.

- Stopwatch зависит от оборудования и ОС вкупе
- Stopwatch базируется на QueryPerformanceCounter в Windows
- Если замеряете операции, длящиеся более 1 ms. – всё ОК

Планирование событий

- Надо отличать перспективу будущего события!
- Если событие определяется абсолютным временем, то его можно планировать в UTC.
  Событие через 128 часов таки должно случиться через 128 часов!

Планирование событий

Если событие определяется локальной перспективой, то необходимо хранить локальное время и часовой пояс.

Локальное время «уедет» в случае перехода между зимним и летним временем,
если будет выражено через UTC.

Планирование событий

Храним:

- Локальное время и ID часового пояса (минимум)
- Паттерн повторения: ежедневно, еженедельно и т.д.
- Если требуется, то ещё время по UTC

Планирование событий

Запланированное время попадает в переход на зимнее время:

- Событие возникает при первом попадании
- Событие возникает при втором попадании
- Событие возникает дважды

Планирование событий

Запланированное время попадает в переход на летнее время:

- Событие возникает на час позже
- Событие вообще не возникает

Серверное время

Нельзя полагаться на часовой пояс сервера:

- Сервер может переехать
- Могут выставить UTC+00
- Может потребоваться координация данных по времени с разных серверов

Используйте UTC или DateTimeOffset!

Определение возраста

```
void CalcAge(DateTime birth, DateTime momentInTime) {
    double diff = (momentInTime - birth).TotalDays;
    int age = (int)(diff / 365.242199);
    Console.WriteLine("Age:{0}", age);
}

CalcAge(new DateTime(2007, 3, 2),
        new DateTime(2008, 3, 2)); // 1

CalcAge(new DateTime(2008, 3, 2),
        new DateTime(2009, 3, 2)); // 0
```

Определение возраста

```
// Днём рождения тех кто родился 29-го февраля,
// в не високосные годы считается 28-е февраля
void CorrectCalcAge1(DateTime birthday, TimeZoneInfo targetTimeZone)
{
   DateTime today = TimeZoneInfo.ConvertTimeFromUtc(DateTime.UtcNow,
                                                    targetTimeZone).Date;
   int age = today.Year - birthday.Year;
   if (birthday.Date.AddYears(age) > today) age--;
}
```

Определение возраста

```
// Днём рождения тех кто родился 29-го февраля,
// в не високосные годы считается 1-е марта
void CorrectCalcAge2(DateTime birth, TimeZoneInfo targetTimeZone)
{
   DateTime today = TimeZoneInfo.ConvertTimeFromUtc(DateTime.UtcNow,
                                                    targetTimeZone).Date;
   int age = today.Year - birth.Year;
   if (birth.Month > today.Month ||
      (birth.Month == today.Month && birth.Day > today.Day)) age--;
}
```

Определение возраста

Ребёнок родился 31 декабря 2016 года.
Сколько лет будет ребёнку 1 января 2017?

Правильный ответ: зависит от счёта.
В европейском исчислении 0 лет.
В восточноазиатском в некоторых странах – 2 года.

Noda Time

Jon Skeet

Open-Source, Free.
Разработан по мотивам Joda Time из Java

nodatime.org

Noda Time

Преимущества:

- Заставляет думать о смысле действий
  над датами и временем
- Исключает сюрпризы и не делает предположений
- Более «многословный» API. Да, это плюс.
- Поддерживает IANA и Microsoft Time Zones
  + маппинг между ними!
- Поддерживает работу с
  датами и временем по отдельности

Noda Time

```
// Получаем Unix-epoch time
Instant now = SystemClock.Instance.Now;

// Берём локальный часовой пояс
DateTimeZone tz = DateTimeZoneProviders
                                  .Bcl
                                  .GetSystemDefault();

// Конвертируем Unix-epoch в часовой пояс
ZonedDateTime localNow = now.InZone(tz);
```

Мгновенное время в Noda Time

Instant:

- Точка на временной шкале, представленная через Unix-epoch
- Аналог: DateTime с перспективой Utc

OffsetDateTime:

- Аналог: DateTimeOffset

ZonedDateTime:

- Измеряется с учётом смещения, определяемого часовым поясом
- Нет прямого аналога в BCL: DateTimeOffset + TimeZoneInfo

Календарные типы в Noda Time

LocalDate:

- Полная дата на календаре
- Аналог DateTime с Kind = Unspecified с игнором времени

LocalTime:

- Время дня
- Аналог: DateTime с Kind = Unspecified с игнором даты

LocalDateTime:

- LocalDate + LocalTime
- Аналог: DateTime с Kind = Unspecified

Другие типы в Noda Time

Duration:

- Промежуток времени
- Аналог TimeSpan

Period:

- Промежуток календарного времени (1 месяц)

Interval:

- Формирует полуоткрытый интервал: [start, end)
- Выражен через два Instant значения внутри

Итоги

Недостатки DateTime

- Local – обманчивая перспектива,
  означает локальную машину.
- Local сбрасывается в Unspecified
  при круговой конвертации
- API DateTime имеет скрытые
  недокументированные «фичи»
- Сравнение DateTime не учитывает смещения
- Арифметика на DateTime
  может оказаться забагованной

DateTimeOffset

Основной недостаток: тип ничего не знает о часовом поясе.

Плюс: однозначно определяет момент времени.

Хорошо применять для:

- Арифметических операций между значениями даты и времени
- Конвертации между зонами с использованием TimeZoneInfo

TimeZoneInfo

Основной недостаток: базируется на Microsoft Time Zones

Как следствие:

- Странное именование часовых поясов
- Неглубокая история изменений

Выводы

- В BCL отсутствует тип, который однозначно определяет момент времени и привязывает часовой пояс
- Нет типов, которые представляют дату и время по отдельности
- API стандартных типов запутано и скрывает «тухлые помидоры»
- В большинстве случаев стандартных типов хватает.
  В противном случае:
  NodaTime, Quartz Enterprise Scheduler .NET.

Ссылки

https://www.pluralsight.com/courses/date-time-fundamentals
Date and Time Fundamentals by Matt Johnson

http://aakinshin.net/en/blog/dotnet/datetime/
DateTime Under the Hood – Андрей Акиньшин

http://aakinshin.net/en/blog/dotnet/stopwatch/
Stopwatch Under the Hood – Андрей Акиньшин

https://codeblog.jonskeet.uk/2014/09/30/the-mysteries-of-bcl-time-zone-data/
Баги в TimeZoneInfo – Jon Skeet

http://blog.nodatime.org/2011/08/what-wrong-with-datetime-anyway.html
Что не так с DateTime - Jon Skeet

Ссылки

https://en.wikipedia.org/wiki/February_29#Holidays_and_observancesDate – Проблема ДР 29-го февраля

https://goo.gl/LBxNIg - Восточноазиатский счёт возраста

http://nodatime.org/ - NodaTime

http://www.quartz-scheduler.net/ - Quartz .NET Scheduler

https://youtu.be/l3nPJ-yK-LU - Jon Skeet talk on Dates and Times

https://goo.gl/KQnjL1 Zune Software Leap-Year Bug

https://goo.gl/9vPclu Правила исчисления возраста в РФ

Ссылки

https://goo.gl/GKxCyM –
DateTime and DateTimeOffset in .NET: Good practices and common pitfalls

http://stackoverflow.com/questions/4331189/datetime-vs-datetimeoffset
DateTime vs DateTimeOffset

http://stackoverflow.com/questions/246498/creating-a-datetime-in-a-specific-time-zone-in-c-sharp-fx-3-5#246529
Creating a DateTime in a specific Time Zone in c# fx 3.5

https://channel9.msdn.com/Shows/Going+Deep/Bart-De-Smet-Rx-20-RC-Time-Error-Handling-SafeSubscribe-and-More
Time, Error Handling, Event Subscription in Rx

Спасибо за внимание!

Илья Фофанов

ENGINEER SPOCK

http://engineerspock.com

https://www.udemy.com/user/eliasfofanov/

https://habrahabr.ru/users/engineerspock/