Документация к системе PolarDB

(в свободном стиле)

# Введение

Система PolarDB, по тексту часто просто Поляр, это библиотека решений по структуризации данных предназначенная для построения специализированных систем управления данными, баз данных, систем управления базами данных. Система разработана в ИСИ СО РАН. Разработка выполнена в среде .NET (в настоящее время, в .NET Core). Основным языком программирования является C#, он же является языком демонстрационных примеров.

# Структуры данных

Основой Поляра является специальная типизированная структуризация данных. Это означает, что любое значение Поляра имеет тип. Причем тип, как правило, статически определен, т.е. определен на стадии исходного текста программы. Существуют атомарные типы, совпадающие с традиционным набором: булевский, байт, символ, целое, длинное целое, вещественное. К этому добавляется тип строкового значения. Строка в Поляре соответствует представлению о строках в объектно-ориентированных языках программирования. То есть это набор Unicode-символов, рассматриваемый как единое целое. То есть без возможности прямого изменения элементов или длины строки.

Составные типы формируются через конструкторы: запись, последовательность и объединение. Запись – это фиксированный набор значений разного типа. Последовательность – это набор элементов (ноль или более) заданного типа. Объединение – это пара тег-подзначение. При этом, для каждого варианта (тега) определен свой тип подзначения.

Структурные значения могут существовать в разных формах и как значения в оперативной памяти и как файловые объекты или элементы файловых объектов и как текстовые формулы. Причем возможно использование разных представлений и для объектов в ОЗУ и для файловых (байтовых) и для текстовых форм.

Текстовое представление структурных значений

Текстовое представление атомарных значений, в основном совпадают с представлениями, принятыми в большинстве современных языков и систем программирования: true/false, 127, ‘Z’, 9999, 7.7e-7, “строка в юникоде”. Неопределенности в интерпретации целых чисел как байт или целое или длинное целое не возникает, поскольку тип значение всегда определен на момент его интерпретации.

Текстовое представление составных типов в основном соответствует представлению JSON, т.е фигурные скобки используются для изображения записей: { k1: value1, k2: value2 …}, прямоугольные скобки группируют последовательность элементов: [“abc”, “def”, “ghk”] . Отличительной особенностью Поляровской структуризации является наличие значений объединенного типа. Текстовое представление таких значений: tag^подзначение. Собственно, если нет объединенного типа, то все обычно и похоже на JSON, только ключи в записях изображаются без кавычек – кавычки в ключах JSON это дань динамической типизации.

В силу особенностей Поляровской типизации структурных значений, синтаксис их текстового представления может быть сокращен: идентификаторы ключей в записях могут быть опущены вместе с двоеточиями, теги в объединениях могут быть заменены на численные значения (0, 1,…).

Объектное представление структурных значений

Структурные значения в оперативной памяти представляются следующим образом: атомарные значения – в соответствии с системой программирования, т.е. булевскими, байтами, символами, числами, строками; записи и последовательности – в виде массивов (array) значений элементов; объединения – в виде массива из численного значения тега и значения подэлемента. Все массивы представляются в системе программирования как массивы объектов (object[] в C#).

Пример:

object[] team = new object[] {

new object[] { 7001, "Иванов", 20 },

new object[] { 7001, "Петров", 28 },

new object[] { 7001, "Сидоров", 17 }

};

Поляровские типы как объекты

Структурные значения в Поляре всегда интерпретируются только в контексте явно или неявно заданного типа. Явно заданный тип – это объект класса PType. Поляровский тип, как правило, конструируется в виде формулы, напр.:

PType tp\_persons = new PTypeSequence(

new PTypeRecord(

new NamedType("id", new PType(PTypeEnumeration.integer)),

new NamedType("name", new PType(PTypeEnumeration.sstring)),

new NamedType("age", new PType(PTypeEnumeration.real))));

Соответственно, конструктором элементарных типов является базовый конструктор

PType tp = new PType(PTypeEnumeration.xxx);

Где xxx – один из вариантов: none, boolean, @byte, character, integer, longinteger, real, sstring.

А конструкторы составных типов задают информацию о структуре и типах элементов:

var tp1 = new PTypeSequence(тип\_элементов);

var tp2 = new PTypeRecord(new NamedType(имя\_поля, тип\_поля), …);

var tp3 = new PTypeUnion(new NamedType(имя\_варианта, тип\_варианта), …);

Для типовых значений определена функция интерпретации структурного значения в объектном представлении в виде текста, напр.:

tp\_persons.Interpret(team);

Важно, чтобы структурное значение строго соответствовало типу.

Сериализации

Применяемое структурирование – древовидное. Сериализацией назовем перевод (дерева) объектного структурного значения в последовательность информационных единиц. Это может быть перевод в последовательность символов или перевод в последовательность байтов. В первом случае получается текстовая сериализация, во втором – бинарная. Обратное преобразование из последовательности информационных единиц в объектное представление структурного значения назовем десериализацией.

Модуль TextSerialize содержит статические методы

public static void Serialize(TextWriter tw, object v, PType tp);

public static object Deserialize(TextReader tr, PType tp);

Первый – пишет в поток символов, второй – читает из потока символов и формирует объектное структурное значение.