Ballerina

Tutorial

Sanjiva Weerawarana, sanjiva@weerawarana.org
Kishanthan Thangarajah, kishanthan@wso2.com

SummerSOC; June 22, 2019

Prerequisites

- Download:
  - Ballerina 0.991.0 from https://ballerina.io
  - Visual Studio Code
  - Optional: Docker Desktop
- Clone GitHub repo: https://github.com/Kishanthan/SummerSOC-2019

Agenda

- Language introduction
- Type system
  - Code: Type system code examples (types directory)
- Sequence diagram concepts & concurrency
  - Code: Concurrency examples (concurrency directory)
- Networking
  - Code: Networking protocol examples (http, http2, websockets, grpc, kafka, rabbitmq directories)
  - Code: SaaS connector examples (gsheets, twitter directories)
- I/O
  - Code: IO examples (io directory)
- Security
  - Security examples (security directory)
- Static & streaming data, and querying
  - Code: Data examples (database, experimental/{tables,streams} directories)
- Transactions
  - Code: Transactions examples (experimental/transactions directory)
- Config, packaging and deployment
  - Code: Deployment examples (docker, kubernetes directories)

Part 1: Language introduction

Ballerina

“A programming language for network distributed applications”

“Programming language”

- Not a DSL
- Full programming language with rich set of general purpose features
- General purpose features are optimized for “network distributed applications”

“network distributed”

- Almost all existing programming languages designed for a world where the normal case is to integrate components running on one machine
- In the cloud world, the normal way for a program to interact with its environment is over the network

“applications”

- Not a systems-level programming language
- Easy to write and modify is more important than squeezing the ultimate compute performance
- Suitable for application programmers

Design principles

- Works for Mort
- Familiar
- Pragmatic
  - not a research language
- Readability
- Design language together with platform

Language vs platform

- Language
  - language means what’s defined in the language spec
  - includes tiny “lang library”
- Platform
  - language
  - standard library
  - Ballerina central
  - project structure, packerina
  - testing, testerina
  - documentation, docerina
  - etc.

What makes Ballerina unique as a programming language?

Two features that work together

- Providing and consuming services
  - first-class language concepts with their own syntax and semantics, not a library
  - inherently concurrent
- Sequence diagrams
  - graphical view of most fundamental aspect of the semantics of a network distributed application
  - sequence diagram view only possible because syntax and semantics of the language were designed to enable it
  - higher-level concurrency abstraction

Ballerina is not object-oriented

- Object means data+code
- OO is wrong model for network distributed applications
- Network transparency doesn’t work
- Don’t want to send code over the network
- Serialization/deserialization of objects is not sending code
    - requires tight coupling between sender and receiver
- Can be a challenge for programmers with strongly OO background
- Ballerina does provide objects for when you really need data+code combo

Typing: strong vs static vs dynamic

- Strong typing means you do not expose the raw memory of values
    - All modern, non-systems languages are strong. Not interesting!
- Static typing means that you check types at compile time
- Dynamic typing means that you check types at run time
- Typing spectrum
    - To what extent are things checked at run-time vs compile-time?
    - How complex are the things that your type system can say?
- Too static leads to inflexibility and/or complexity
- Too dynamic means unreliable and poor IDE experience
- Ballerina has chosen pragmatic point on spectrum

Type system vs schema

- Type system describes values occurring during program execution
- Schema describes messages exchanged between programs
- Type system and schema are usually completely unrelated
- Data binding converts between the two
- Creates a lot of friction for applications that exchange messages

Ballerina’s type system also works as a schema

- Key features
  - Describes shape of a value - structural types
  - Types are just sets of values - semantic subtyping
  - Allows for choice of A or B - untagged unions
  - Extensibility - open records
- Data binding is just a type cast
  - Mutability complicates things a bit
- Most similar to TypeScript

Data structures

- “It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures.” - Alan J. Perlis
- Ballerina has two fundamental data structures
  - lists: ordered list
  - mappings: mappings from strings to values
- Exact match for JSON
- Two ways to type mappings
  - records
  - maps
- Two ways to type lists
  - tuples
  - arrays

Error handling

- Errors are normal part of network programming (one reason why "network transparent" does not work)
- Errors handled as part of normal control flow
- Error is a separate data type
- Possibility of errors described by type system in same way as rest of the language
- Leverages untagged unions

Relationship with Java

- First implementation runs on JVM
- Not a JVM language
  - semantics not influenced by what JVM happens to do
- Designed to be language with multiple implementations
  - Language defined by spec not by implementation

Relationship to open source

- “Open source” is something that applies to code not documentation
- Saying a language is open source makes sense only for a language that is defined by its implementation
- Misleading to say: “Ballerina language is open source”
- Ballerina spec is licensed under Creative Commons, no derivatives license
- jBallerina is open source under Apache 2.0
- Should say: “jBallerina is an open source implementation of the Ballerina language”

Language versions

- Language version different from implementation version
  - required to support multiple implementations
  - language version = language spec version
- Language version naming scheme 20xyRn
- 2019 plan
  - 2019R1 - done
  - 2019R2 - end of June; design freeze for jBallerina 1.0
  - 2019R3 - improved spec to coordinate with release for jBallerina 1.0
  - 2019R4 - features that didn’t quite make it into R2/jBallerina 1.0
  - 2019Rn - TBD

Language stability

- Consists of
    - base stability level
    - parts with less than base stability level
- Base stability level for 2019R3
    - stable
    - there will be bugs
    - there will be new language keywords
- Parts with less than base stability
    - Clearly marked in spec
    - XML
    - Tables
- Platform versions everything

Stabilisation plan

- 2019
    - XML
    - Tables
        - language-integrated query
- 2020
    - Streaming query
    - Transactions

Language design process

- Spec maintained in XHTML in GitHub repository

  https://github.com/ballerina-platform/ballerina-spec

- Most design work now happens via issues on GitHub
  - Supplemented by video conferencing
- Best way to provide input or ask a question is to create an issue
  - Please don’t mix different points in a single issue

We are using Ballerina 0.991.0

- Work in progress towards supporting 2019R2
- Some old syntax still there
- Many things still being implemented

Part 2: Type system

Values

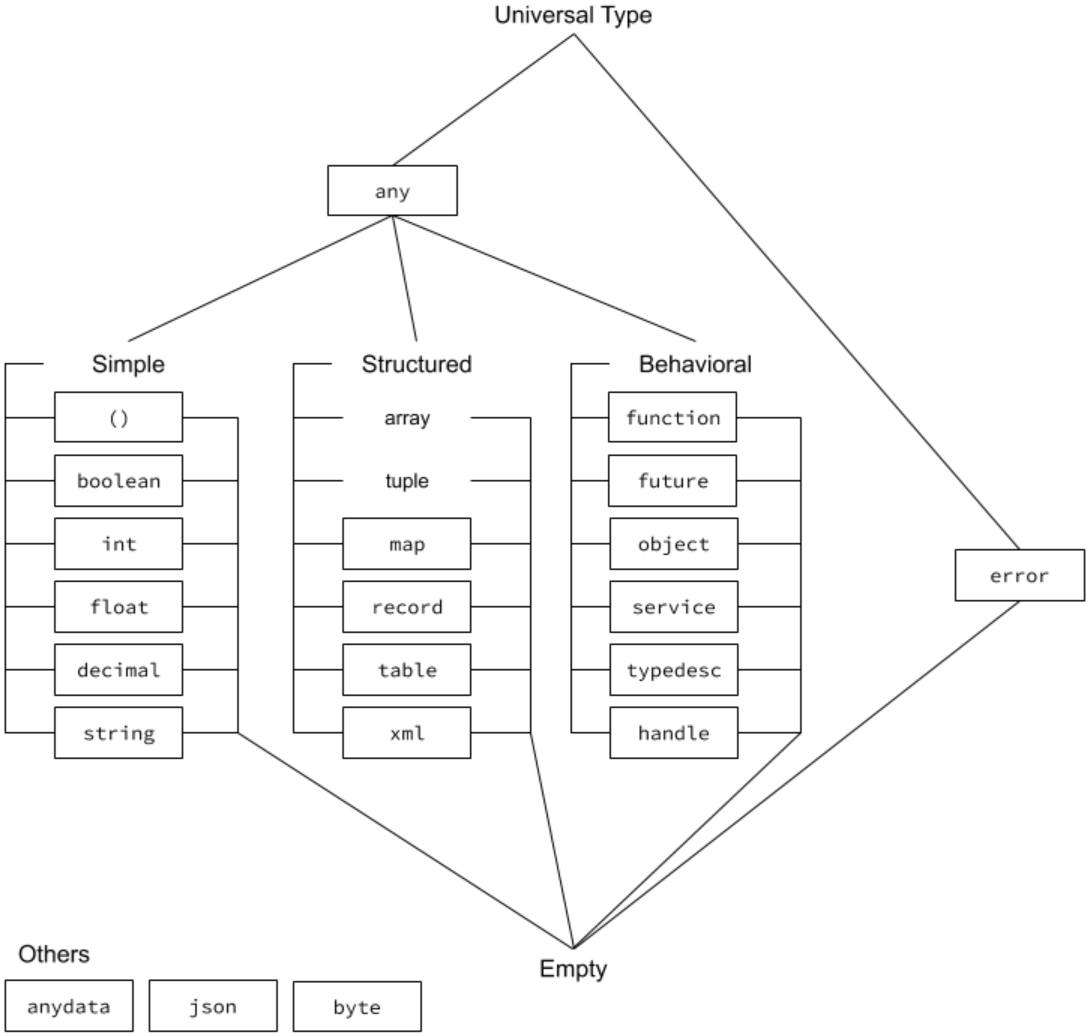
- Values are the universe of things that all Ballerina programs operate on
- 3 kinds of values
  - simple values, like booleans and floating point numbers, which are not constructed from other values
  - structured values, like mappings and lists, which create structures from other values
  - behavioral values, like functions, which allow parts of Ballerina programs to be handled in a uniform way with other values

Shape and type

- Shape of a value: focus on structure of the value
  - abstracts value
  - for simple values, shape and value are the same
  - for structured and behavioral values, shape ignores storage location and mutability
    - e.g.: record { int a; string b; } and record { string b; int a; } have same shape
- Type
  - set of shapes
  - same shape can be in many sets
- Type descriptor
  - describes a type (membership test for the set)

Subtyping

- Subtype means a subset of shapes
- “Semantic subtyping”



Universal Type

any

Simple

- ()
- boolean
- int
- float
- decimal
- string

Structured

- array
- tuple
- map
- record
- table
- xml

Behavioral

- function
- future
- object
- service
- typedesc
- handle

error

Empty

Others

- anydata
- json
- byte

Time for code

Part 3: Sequence diagrams & concurrency

Sequence diagrams

- Natural way to represent and explain concurrency
  - but not used to program concurrency
- Ballerina concurrency model is built on sequence diagrams
  - every Ballerina program is a collection of sequence diagrams
  - diagram is the code, not a picture of the code

Sequence diagrams

- Many sequence diagrams, one per unit of work or collapsed to one
- Each actor / line is a sequence flow of logic
  - graphically a flow chart
- Naturally concurrent – just add parallel actors
- Active actors (code you write) and passive actors (external participants: network services)
- Not meant to be used for low-code / no-code or by non-programmers
  - meant to make it easier to understand complex distributed interactions

Easy concurrency

- Every executable entity (function, method, resource) is a collection of workers
  - default worker
  - named workers
- Each worker is a independent execution context
- All workers of a function execute concurrently when the function is called

Strands

- Strands are independent execution stacks
  - will be scheduled into a thread
  - (not a thread because of it offers a blocking programming model by scheduling to a thread)
  - represented as a future
- Function calls within a strand are regular stack based invocation
  - default worker of a called function runs in the same strand as the calling worker
- Concurrency is across strands
  - communication between workers only, not worker to/from future
  - (to prevent deadlocks)

Worker to worker communication

- Via anonymous channels, (non-)blocking for send, blocking for receive
- Error and panic propagation
- Interaction patterns and deadlock prevention via session types

```
isig :=
  type-descriptor ->        // send a message of this type
  | type-descriptor <-      // receive a message of this type
  | isig , isig             // sequence
  | isig | isig             // choice
  | isig *                  // repeat zero or more
  | ( isig )                // group
```

Concurrency control

- Concurrency reality for business use cases and cloud native computing
  - small number of actual cores
- Lock construct (work in progress)
  - implements two phase locking
- Immutable values
  - constants
  - frozen copies
- More work TBD to provide better concurrency safety
  - Rust: ownership types
  - Swift: uniqueness types
  - Software transactional memory

Time for code

Part 4: Networking

Introduction

- Most languages treat network interactions as just communications over another kind of I/O stream
  - Based on Berkeley socket library
- Ballerina models network interactions as a connection + interaction protocol at language level

Concepts

- Endpoints
- Connectors
- Listeners
- Services

Endpoints

- Endpoints represent network termination points for incoming network interactions or outgoing network interactions plus set of typed interactions that are allowed
- Ingress vs. egress endpoints
- Egress endpoints represent remote systems
    - offers a set of actions for interaction with them
- Ingress endpoints are network entry points to a Ballerina runtime via a registered service
    - calls are delivered to a particular resource in a service
    - offers incoming endpoint list of actions to reply/message
- Modeled as an actor in sequence diagram

Connectors

- Client objects
  - objects with some methods that are marked as “remote”
- Remote methods are network interactions
- Allows a “stub” to wrap another “stub” and propagate network awareness

Listeners

- Listeners open network entry points to a Ballerina program
- Dispatch incoming requests to one or more services based on some listener-specific criteria

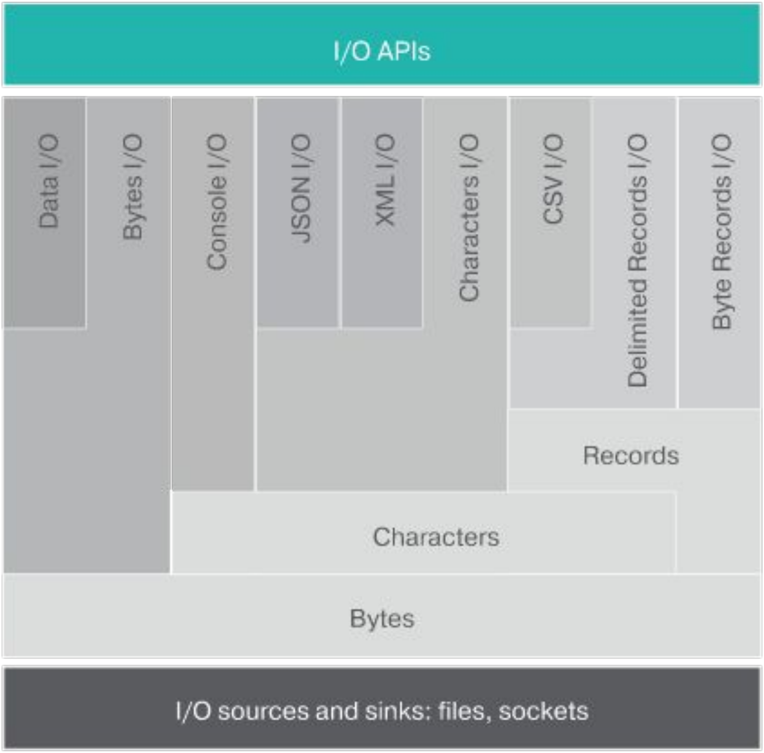
Services

- Behavioral value
- Service is a collection of resource functions and regular functions
- Resource functions are entry points to which incoming network requests are dispatched
    - no return value - need to send responses explicitly via incoming connection reference

Time for code

Part 5: I/O

I/O system architecture



I/O

- Blocking programming model with non-blocking performance
  - blocks worker (and therefore strand), not thread
- Comprehensive I/O package with flexibility to extend
  - bytes -> character -> record

Time for code

Part 6: Security

Security

- Goal is to make programs secure by default
  - force programmer to think about security
  - make it easy to establish security
- Security means
  - not trusting data from the network
  - knowing who is calling you (authentication)
  - checking authority to execute (authorization)

Taint checking

- Data from untrusted sources is considered “tainted”
- Touching tainted data taints you
  - propagates across function calls
- Untainting requires explicit programmer action to sanitize
- Done via annotations
  - Current version has untaint operator; will be changing

Authentication

- Not in language per se
- All network connectors & listeners support appropriate protocols
- Common concept of principal

Authorization

- Permission model based on scopes
  - borrowed from OAuth

Time for code

Part 7: Data & querying (WIP)

Kinds of data

- Tabular data
  - table data type
- Graph data
  - maps & records
- Streaming data
  - stream data type

Tables & SQL-like querying

- First class data type in language
  - collection of records
  - keys
- Integrated query syntax to query tables directly
  - Similar to C# LINQ
- Future: mirroring tables from databases, mapping to network tables, CSV

Graph data

- Maps and records
- Exploring GraphQL

Streaming data

- stream type
  - distribute events to listeners
- Streaming queries
  - permanently block worker and execute streaming SQL like queries

Time for code

Part 8: Transactions

Transactions

- Goal is to make correct transactional programming natural and easy
- Support for
  - local & distributed transactions
  - 2PC and compensation
- No external transaction coordinator required

Local transactions

- Language construct for transactions

```
transaction {

} onretry {

} committed {

} aborted {

}
```

Distributed transactions

- New Ballerina microtransaction protocol
  - designed by Frank Leymann
- Goal is to make microservice transactions work smoothly
  - Initiator creates a coordinator
  - Participants have transaction infected to them transparently
  - Works OOTB for Ballerina distributed transactions and with sidecar for others
- Compensation
- WIP

Time for code

Part 9: Config & deployment

Configuration

- Applications need to externalize configuration
- Two layers
  - config API
  - language support
- Language support for configuration
  - mark module level variables/attributes as configurable
  - externally refer to those and provide values
  - language startup system will read and population values before user code starts
  - (inspired by X Window System config model)

Deployment

- Gone are the days of compiling and running programs
- Now
  - build
  - create Docker image
  - write YAML files
  - deploy
  - start
- Programmer is usually out of the loop on all of these
- Ballerina compiler supports extensibility for compilation process to include additional functions into compilation process

Time for code

Summary

Summary

- Ballerina is a programming language for network distributed applications
  - goal is to make writing such applications at least 3-5x faster, better, cheaper
- Brings fundamental concepts of networking, concurrency, data, security, deployment all together in an approach that offers both equivalent textual and graphical syntaxes
- Current implementation uses JVM but native compilation will come in 2020
- Language still evolving but stable core ready to go
- Already used in production for commercial products
  - jBallerina 1.0 in July/August