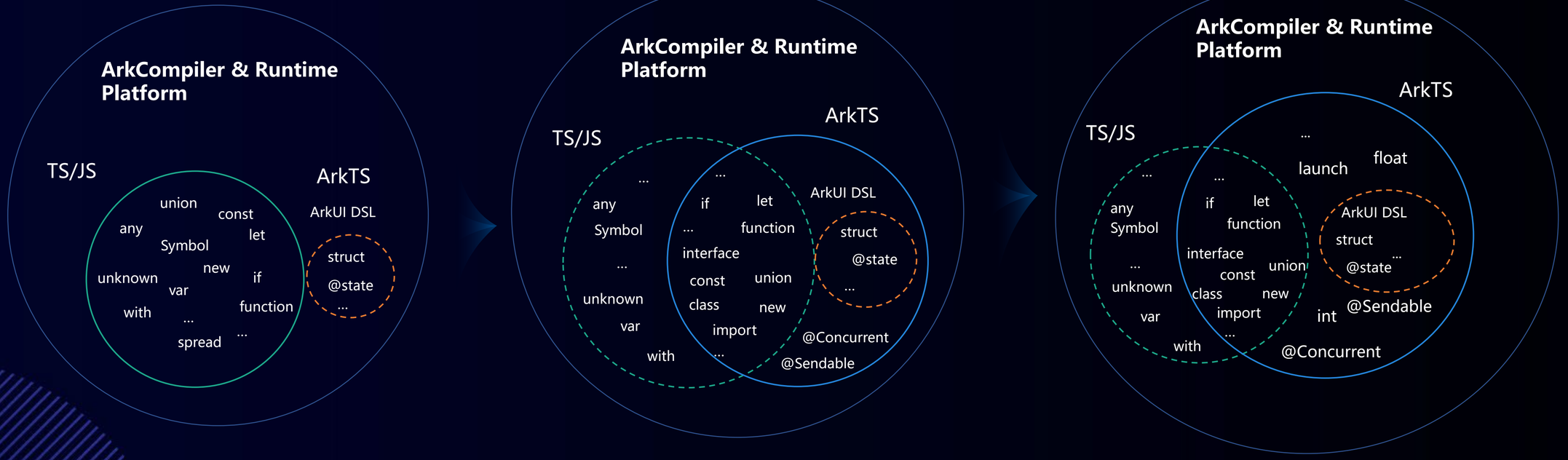
ArkTS Introduction

HUAWEI

What is ArkTS?

ArkTS is the application development language for the HarmonyOS ecosystem where theres’ re 30K+ applications and atomic services developed using ArkTS.

- Provides features to support declarative UI paradigms and state management, enabling developers to create applications in a more concise and natural way.
- In addition to maintaining the basic syntax style of TypeScript, ArkTS further enhances static type checking and analysis, allowing more errors to be detected during the development phase before the program runs. This improves code robustness and achieves better runtime performance. ArkTS also supports efficient interoperability with TS and JS.
- To address the limited support for concurrency in JS/TS, ArkTS has enhanced its concurrency programming APIs and capabilities.
- In the future, ArkTS will continue to evolve based on application development and runtime requirements, gradually providing more features such as improved concurrency capabilities, enhanced type system, and distributed development paradigms.



Declarative UI paradigm support

Static typing, concurrency enhancements, null safety

Extended Language Features

HUAWEI

ArkTS Concurrency Features

HUAWEI

Concurrency API - TaskPool

1. Implementation of Time-Consuming Tasks

```
@Concurrent
async function func(a, b) {
  // Time-consuming business logic
  if (taskpool.Task.isCanceled()) {
     // Terminate time-consuming task early
    return undefined;
  }
  // Time-consuming business logic
  return a + b;
}
```

The Concurrent decorator needs to be written in the ArkTS file.

It supports query-based cancellation, allowing it to be applied between multiple time-consuming operations to return early and avoid executing unnecessary logic.

2. Task Distribution

2.1 Direct Function Distribution (Default Priority)

```
let result = await taskpool.execute(func, 1, 1);
```

Directly executing the function with variable arguments passed in, default priority, and non-cancelable.

2.2 Encapsulate Task and Then Distribute

```
let task = new taskpool.Task(func, 1, 1);
let result = await taskpool.execute(task, taskpool.Priority.HIGH);
```

3. Task Cancellation

```
try {
  taskpool.cancel(task)
} catch (e) {
}
```

Create a task, specify its priority during execution, and cancel it.

After cancellation, the task will not be stopped; instead, the `isCanceled` query returns true, and the task will be rejected back to the main thread upon completion.

In addition to the simple execution interface mentioned above, the system also provides capabilities such as task groups, associated tasks, delayed tasks, periodic tasks, serial tasks, and specifying concurrency levels.

For more details, see: https://gitcode.com/openharmony/docs/blob/master/zh-cn/application-dev/reference/apis-arkts/js-apis-taskpool.md

HUAWEI

Comparison between Worker and TaskPool

Technology	TaskPool	Worker
Memory Model	Isolation	Isolation
Parameter Passing Mechanism	Structured Clone	Structured Clone
Parameter Passing	Direct Passing	Unique Message Object Parameter
Method Invocation	Direct Invocation	Message passing is parsed and the corresponding method is implemented in the Worker.
Return Value	Asynchronous call returns.	Message returned, needs to be parsed and assigned in onmessage.
Lifecycle	TaskPool manages itself; no need to worry about the load level of background tasks.	Developers manage the task load, memory reclamation, and the number of workers themselves.
Priority	Supported	Not Supported
Cancel	Support	Not Support

Summary:

Concurrent APIs with different granularities

Worker is more like a Thread or Service dimension.
Task is the single task dimension.
In most cases, it is recommended to use TaskPool; for long-running tasks, consider using Worker.

Advantages of TaskPool:

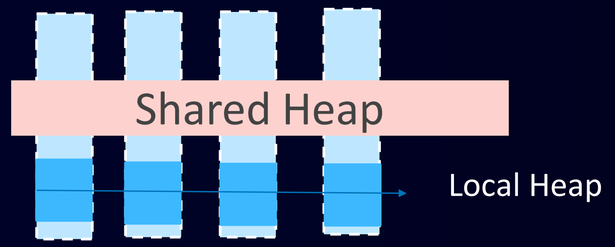
- Simplifies developers' work on concurrent programming.
- More features support advanced needs: priority/task cancellation.
- Unified management saves system resources and optimizes scheduling.

Note: At the underlying level of concurrent instances and interactions, both essentially use a memory isolation model, with consistent parameter and range value restrictions, and also incur overhead. (Pay attention to the granularity of concurrent tasks.)

HUAWEI

Memory Sharing Capability - Sendable Specification

Sendable and Shared Heap



Shared Heap:

Stores Sendable objects and immutable objects, supports multi-threaded access.

Local Heap:

Stores non-Sendable objects, which cannot be accessed in a multi-threaded environment.

```
@Sendable
class SendableA{
  ...
}
//A Object is allocated to Shared Heap.
let a: SendableA = newSendableA();

// Type B objects are non-Sendable objects
class B {
  ma: SendableA | null = null;
  ...
}
// Object b is allocated on the Local Heap
let b: B = new B();
// Object b can hold sendable object a
b.ma = a;
```

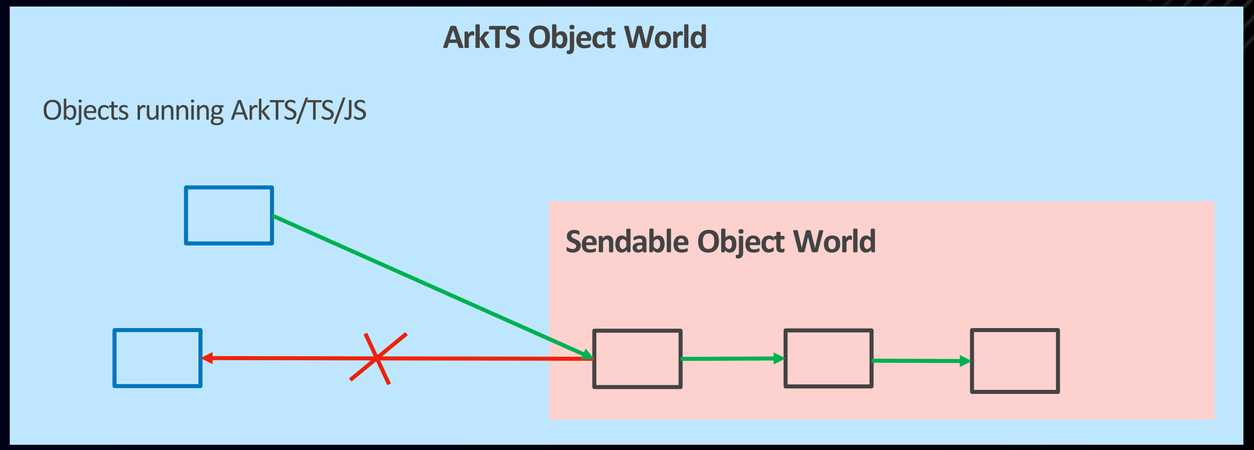
Sendable Objects

Objects created by the Sendable class are allocated in the Shared Heap by default, and their default semantics in concurrent communication is reference passing.

Note: Immutable objects such as string objects and BigInt objects (which do not have data race issues) are also allocated in the Shared Heap by default.

Non-Sendable Objects

Non-sendable objects are by default allocated in the Local Heap, which is the thread-private heap.



@Sendable Class (Layout Non-Modifiable)

- **A Sendable class can only inherit from a Sendable class**
- **Members of a Sendable object must be primitive objects or of Sendable type**

Requires runtime sharing of the HiddenClass (internal) of primitive objects.
Requires runtime sharing of the Class (internal) of Sendable objects (static type).

The default semantics for inter-thread communication of @sendable class objects is reference passing.

Object Reference Relationship

- Sendable Class objects can be attached to type-variant objects.
- Type-variant objects (JS Objects) cannot be attached to the member variables of Sendable Class objects.
- Sendable Class objects can be used in TaskPool and Wo

HUAWEI

Memory Sharing Capability - Sendable Usage

Define Sendable Module

```
// Shared module sharedModule.ets
import { ArkTSUtils } from '@kit.ArkTS';

// Declare the current module as a shared module,
which can only export Sendable data
"use shared"

// Shared module, SingletonA is globally unique
@Sendable
class SingletonA {
private count_: number = 0;
lock_: ArkTSUtils.locks.AsyncLock = new
ArkTSUtils.locks.AsyncLock()
  public async getCount(): Promise<number> {
    return this.lock_.lockAsync(() => {
      return this.count_;
    })
  }
  public async increaseCount() {
    await this.lock_.lockAsync(() => {
      this.count_++;
    })
  }
}
export let singletonA = new SingletonA();
```

Multithreading Usage

```
// Thread 1
import { singletonA } from './sharedModule.ets';

@Concurrent
async function increaseCount() {
await singletonA.increaseCount();
  console.info("SharedModule: count is: " + await singletonA.getCount());
}
```

```
// Thread 2
import { singletonA } from ‘./sharedModule.ets’

@Concurrent
async function printCount() {
  console.info("SharedModule: count is:" + await singletonA.getCount());
}
```

HUAWEI

Related proposals in TC39

Structs, Shared Structs and Synchronization Primitives

https://tc39.es/proposal-structs/#intro (Stage 2)

Similar to ***Sendable*** in ArkTS

Deferred Imports Evaluation (Stage 3)

https://tc39.es/proposal-defer-import-eval/

Similar to "***Lazy import***" in ArkTS

JavaScript Structs: Fixed Layout Objects and Some Synchronization Primitives

Stage: 2

Author: Shu-yu Guo (@syg)

Champion: Shu-yu Guo (@syg), Ron Buckton (@rbuckton)

Introduction

This proposal introduces four (4) logical features:

- Structs, or unshared structs, which are fixed-layout objects. They behave like `class` instances, with more restrictions that are beneficial for optimizations and analysis.
- Shared Structs, which are further restricted structs that can be shared and accessed in parallel by multiple agents. They enable shared memory multithreading.
- Mutex and Condition, which are higher level abstractions for synchronizing access to shared memory.

Deferring Module Evaluation

previously known as "Lazy Module Initialization"

Status

Champion(s): *Nicolò Ribaudo*

Author(s): *Yulia Startsev, Nicolò Ribaudo and Guy Bedford*

Stage: 3

Slides:

- 2021-01 - Stage 1 (notes)
- 2022-11 - Take two: Defer Module Evaluation (notes)
- 2023-07 - Deferred import evaluation for Stage 2 (notes)
- 2024-04 - Deferred import evaluation for Stage 2.7
- 2024-06 - Deferred import evaluation for Stage 2.7 (2)
- 2025-01 - `import defer` for Stage 3

HUAWEI

Other Differences Between ArkTS and TS/JS

Extended features in ArkTS comparing to TS/JS:

- Final classes and Final methods. Reason: performance, better OOP design
- Native functions, methods and constructors. Reason: performance, interop
- Named constructors. Reason: better control of object initialization
- Explicit type annotaions in for-of loop. Reason: performance, better design
- Array creation expressions. Reason: safety
- FixedArray type. Reason: performance
- Trailing lambdas. Reason: UI
- Functions and lambdas with receiver (extension functions): UI
- Implicit overloading. Reason: performance, developer productivity
- Explicit overload declaration. Reason: performance, better control
- Extended set of numeric types: byte, short, int, long, float, double. Reason: performance, interop with C++
- Type 'char'. Reason: better type check
- Default value for some predefined types. Reason: developer productivity
- Annotations - for compile-time settings, AOP, changing semantic of declarations in compile-time, bytecode and run-time

ArkTS has more strict rules for:

- Undefined safety: No implicit use of 'undefined' value. Reason: safety
- Array: arrays are not sparse. Reason: performance
- Tuples and Arrays are different entities. Reason: performance, better design
- Late init (TS definite assignment) is more strict

<Thank You>

HUAWEI