开发语言解析

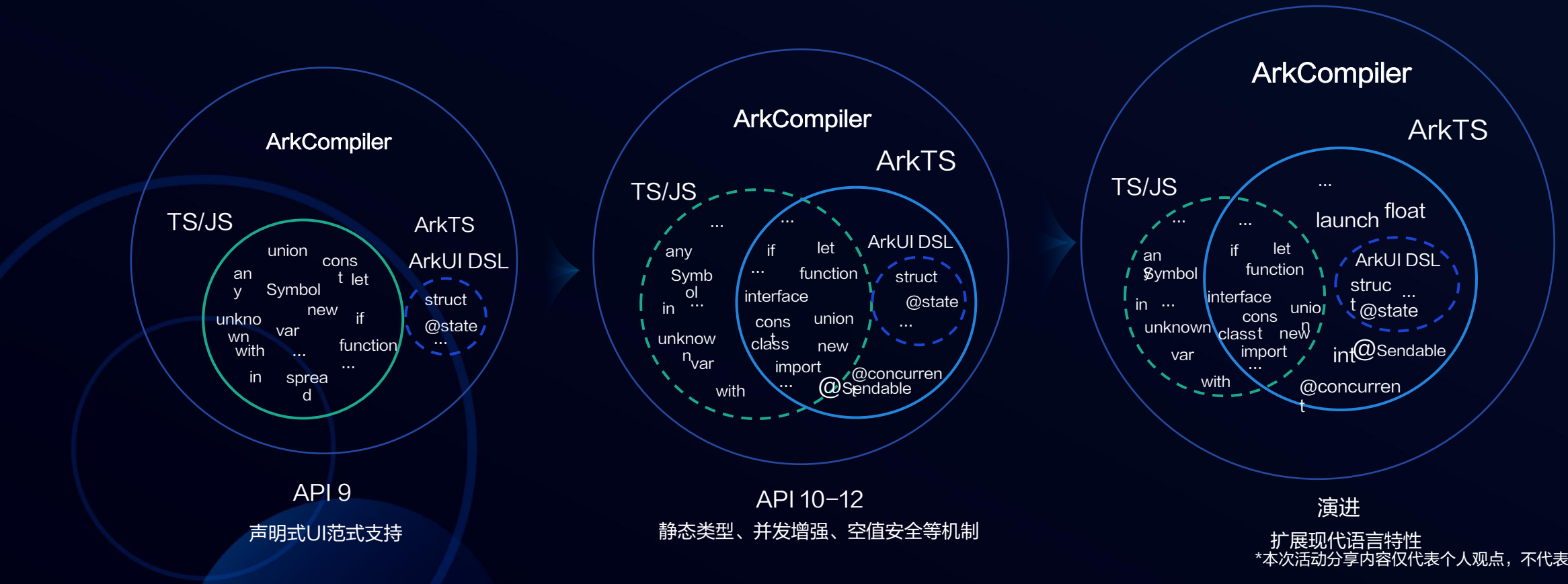
XXX
XXX

*本次活动分享内容仅代表个人观点，不代表华为官方观点

什么是ArkTS?

ArkTS是HarmonyOS生态的应用开发语言。

- ArkTS提供了声明式UI范式、状态管理支持等相应的能力，让开发者可以以更简洁、更自然的方式开发应用。
- 同时，它在保持TypeScript（简称TS）基本语法风格的基础上，进一步通过规范强化静态检查和分析，使得在程序运行之前的开发期能检测更多错误，提升代码健壮性，并实现更好的运行性能，ArkTS同时也支持与TS/JS高效互操作。
- 针对JS/TS并发能力支持有限的问题，ArkTS对并发编程API和能力进行了增强。
- 未来，ArkTS也会结合应用开发/运行的需求持续演进，逐步提供并发能力增强、系统类型增强、分布式开发范式等更多特性。



*本次活动分享内容仅代表个人观点，不代表华为官方观点

基本知识：声明

ArkTS中可以通过关键字let声明变量，使用const声明一个常量，并通过类型注释指定类型

变量声明

```
let count: number = 0;
count = 40;
```

常量声明

```
const MAX_COUNT: number = 100;
```

基本知识：类型

基本类型：string、number、boolean等

```
let name: string = 'Xiaoming';
let age: number = 20;
let isMale: boolean = true;
```

声明字符串类型、数字类型、布尔类型

```
console.log(name); // 输出：Xiaoming
console.log(age.toString()); // 输出：20
console.log(` My name is ${name}, and I am ${age}years old `);
```

基本知识：类型

引用类型：Interface、Object、Function、 Array、Class、Tuple 、Enum等

```
interface StudentType {
  name: string;
  age: number;
}
```

```
let student: StudentType = {
  name: 'Xiaoming',
  age: 18
}
```

```
let newName: string = ' ';

function printInfo(student: StudentType): void {
  newName = students.name;
  console.info(newName);
}
```

```
class User {
  name: string;
  id: number;
}
```

```
let user: User = new User();
```

```
let tuple: [string, number, boolean];
tuple = ['hello world', 100, true];
```

```
let students: Array<string> = ['Xiaoming', 'Xiaozhang', 'Xiaowang', 'Xiaoli'];
let students: string[] = ['Xiaoming', 'Xiaozhang', 'Xiaowang', 'Xiaoli'];
```

基本知识：类型

枚举类型：Enum

```
enum Color {
  Red,
  Blue,
  Green
}
```

声明Color的枚举类型

```
let favouriteColor: Color = Color.Red;
```

基本知识：类型

联合类型：Union 允许变量的值为多个类型

```
class Cat {
  name: string = 'cat';
  // ...
}
class Dog {
  name: string = 'dog';
  // ...
}
class Frog {
  name: string = 'frog';
  // ...
}
type Animal = Cat | Dog | Frog | number;
// Cat、Dog、Frog是一些类型（类或接口）

let animal: Animal = new Cat();
animal = new Frog();
animal = 42;
// 可以将类型为联合类型的变量赋值为任何组成类型的有效值
```

基本知识：语句

条件语句　用于基于不同的条件来执行不同的动作，根据判断条件的执行结果（true 或 false）来决定执行的代码块。

```
let isValid: Boolean = false;
if (Math.random() > 0.5) {
  isValid = true;
} else {
  isValid = false;
}
```

生成[0.0, 1.0)随机浮点数

条件表达式　`let isValid = Math.random() > 0.5 ? true : false;`

基本知识：语句

循环语句　　用于重复执行相同的一组语句，提高效率、简化代码

```
let students: string[] = ['Xiaoming', 'Xiaozhang', 'Xiaowang', 'Xiaoli'];
```

for循环语句

```
for (let i = 0; i < students.length; i++) {
  console.log(students[i]);
}
```

for…of 循环语句

```
for (let student of students) {
  console.log(student);
}
```

while/do while循环语句

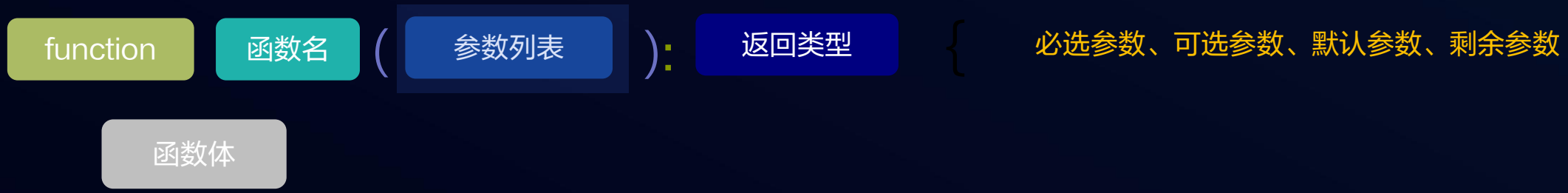
```
let index = 0;
while (index < students.length) {
  console.log(students[index]);
  index++;
};

do {
  console.log(students[index]);
  index++;
} while (index < students.length);
```

```
app Log: Xiaoming
app Log: Xiaozhang
app Log: Xiaowang
app Log: Xiaoli
```

函数的声明和使用

函数是一组一起执行多条语句的组合，形成可重用的代码块
通过function关键字声明要告诉编译器函数的名称、返回类型和参数以及执行的内容




```
function printStudentsInfo(students: string[]): void {
  for (let student of students) {
    console.log(student);
  }
}

printStudentInfo(['Xiaoming', 'Xiaozhang', 'Xiaowang', 'Xiaoli']);
printStudentInfo(['Xiaoming', 'Xiaozhang', 'Xiaoli']);
```

函数的声明和使用：必选参数和可选参数

- 必选参数：指的是必须要传入的参数
- 可选参数：参数是可选的，即在调用函数时可以选择性传入的参数

```
function func1(a: number, b?: number): number {
  return b ? a + b : a;
}

func1();     // Expected 1-2 arguments, but got 0.

func1(1); // 输出 1
func1(1, 2); // 输出 3
```

函数的声明和使用：默认参数和剩余参数

- 默认参数：允许开发者为参数指定默认值，在调用函数时若未传递相应的参数，则使用默认值
- 剩余参数：允许开发者将函数的多个独立参数收集起来，并打包成一个数组

```
// 默认参数
function func2(a: number, b: number = 2): number {
  console.info('a + b：', a + b);
    return a + b;
}

func2(1); // 使用默认值，输出 3
func2(1, 3); // 重新传值，输出 4
```

```
// 剩余参数
function func3(a: number, ...Args: number[]): void {
    console.info('Args：', JSON.stringify(Args));
}

func3(1, 2, 3, 4, 5); // [2,3,4,5]
```

函数的声明和使用：箭头函数 / lambda表达式

箭头函数 / lambda表达式　　简化函数声明，通常用于需要一个简单函数的地方

(参数列表): 返回类型 => { 函数体 }

箭头函数的返回类型可以省略，省略时，返回类型通过函数体推断

```
const printInfo = (name: string): void => { console.log(name) };
```

执行体只有一行的情况下可以省略花括号

```
const printInfo = (name: string) => console.log(name);
printInfo('Xiaoming');
```

箭头函数常用于作为回调函数

```
let students: string[] = ['Xiaoming', 'Xiaozhang', 'Xiaowang', 'Xiaoli'];
students.forEach((student: string) => console.log(student));
```

函数的声明和使用：闭包函数

一个函数可以将另一个函数当做返回值，保留对内部作用域的访问

```
function outerFunc(): () => string {
  let count = 0;
  return (): string => {
    count++;
    return count.toString();
  }
}

let invoker = outerFunc();
console.log(invoker()); // 输出：1
console.log(invoker()); // 输出：2
```

返回一个函数

类的声明和使用

类的声明

ArkTS支持基于类的面向对象的编程方式，定义类的关键字为 class，后面紧跟类名

类的声明描述了所创建的对象共同的属性和方法

类的声明和使用：类的创建

类的创建

```
enum Gender {
  male,
  female
}

class Person {
  name: string = 'Xiaoming';
  age: number = 20;
  isMale: Gender = Gender.male;
}

const person = new Person();
console.log(person.name); // 输出：Xiaoming

const person: Person = { name: 'Xiaozhang', age: 29, isMale: Gender.male };
console.log(person.name); // 输出：Xiaozhang
```

new实例创建

字面量创建

类的声明和使用：构造器

constructor用于实例化时进行初始化操作

```
class Person {
  name: string = 'Xiaoming';
  age: number = 20;
  isMale: Gender = Gender.male;

  constructor(name: string, age: number, isMale: Gender) {
    this.name = name;
    this.age = age;
    this.isMale = isMale;
  }
}
```

```
const person = new Person('Xiaozhang', 32, Gender.female);
console.log(person.name); // 输出: Xiaozhang
```

通过传入参数实例化

类的声明和使用：封装

将数据隐藏起来，只对外部提供必要的接口来访问和操作数据，确保数据的一致性和安全性

```
class Person {
  public name: string = 'xiaoming';
  private _age: number = 20;
  isMale: Gender = Gender.male;

  ... // 省略构造器和方法内容

  public set age(value: number) {
    this._age = value;
  }

  public get age(): number {
    return this._age;
  }
}

const person: Person = new Person('Xiaozhang', 28, Gender.male);
console.log(person._age.toString()); // 无法直接访问私有属性
console.log(person.age.toString());  // 实际访问的是get方法
```

可见性修饰符包括：
private、protected和public。
默认可见性为public

私有变量通过getter和setter
进行访问控制

类的声明和使用：继承

子类继承父类的特征和行为，使得子类具有父类相同的行为

ArkTS中允许使用继承来扩展现有的类，对应的关键字为extends

```
class Employee extends Person {
  department: string;
  constructor(name: string, age: number, isMale: Gender, department: string) {
    super(name, age, isMale);
    this.department = department;
  }
}
```

通过super关键字访问父类

```
const employee: Employee = new Employee('Xiaozhang', 28, Gender.male, 'xxCompany');
employee.printInfo(); // 输出：Xiaozhang is a boy, and he is 28 years old
```

子类继承父类的方法

类的声明和使用：多态

子类继承父类，并可以重写父类方法，使不同的实例对象对同一行为有不同的表现

```
class Employee extends Person {
  department: string;

  constructor(name: string, age: number, isMale: Gender.male, department: string) {
    super(name, age, isMale);
    this.department = department;
  }

  public printInfo(): void {
    super.printInfo();
    console.log(`working in ${this.department}`);
  }
}
```

重写父类方法

```
const person: Person = new Person('Xiaozhang', 28, Gender.male);
person.printInfo(); // 输出：Xiaozhang is a boy，and he is 28 years old

const employee: Employee = new Employee('Xiaozhang', 28, Gender.male, 'xxCompany');
employee.printInfo();
// 输出：Xiaozhang is a boy，and he is 28 years old
// 输出：working in xxCompany
```

子类与父类相同行为不同表现

谢 谢

*本次活动分享内容仅代表个人观点，不代表华为官方观点