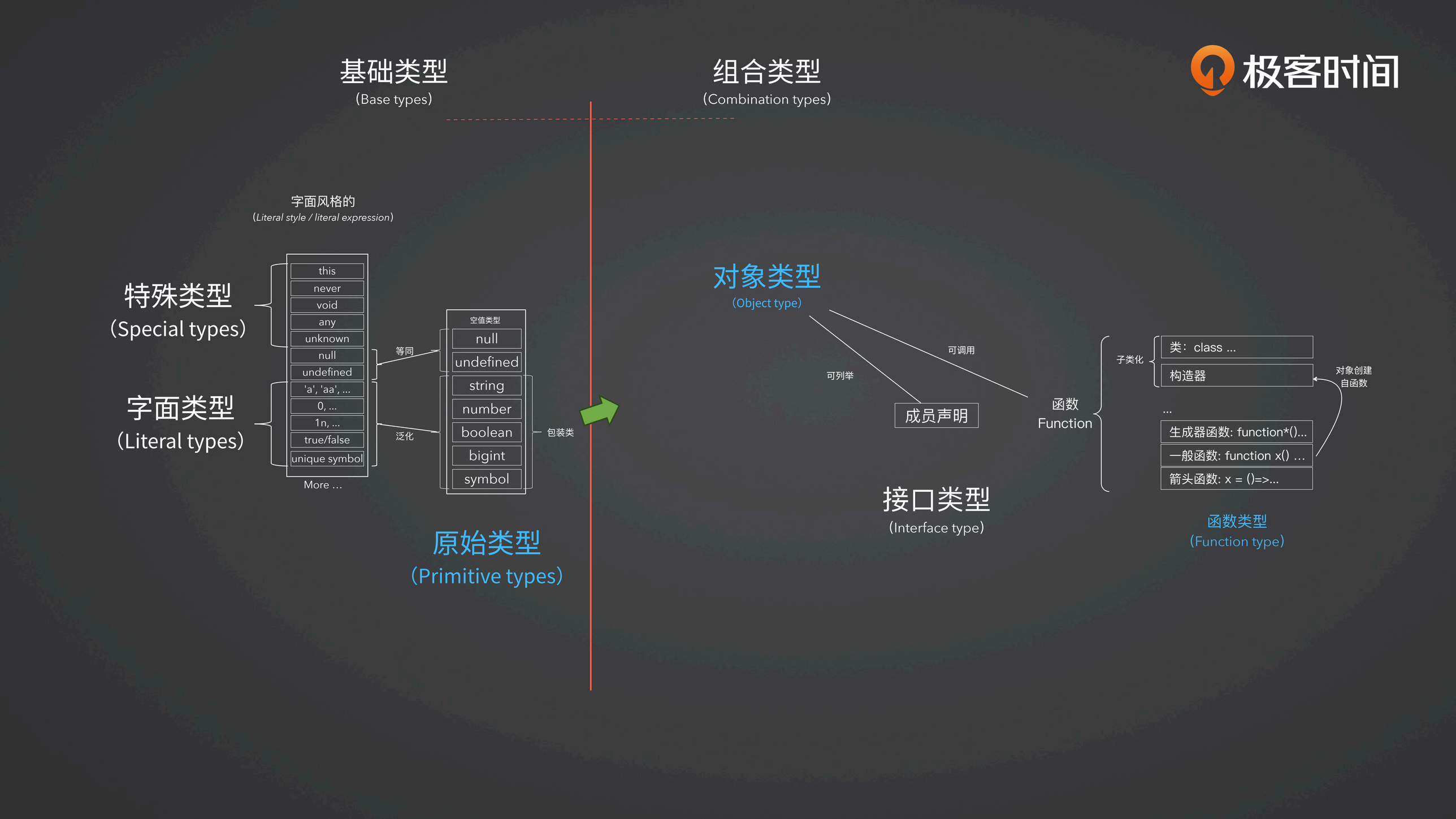
05 | TypeScript 类型系统全景

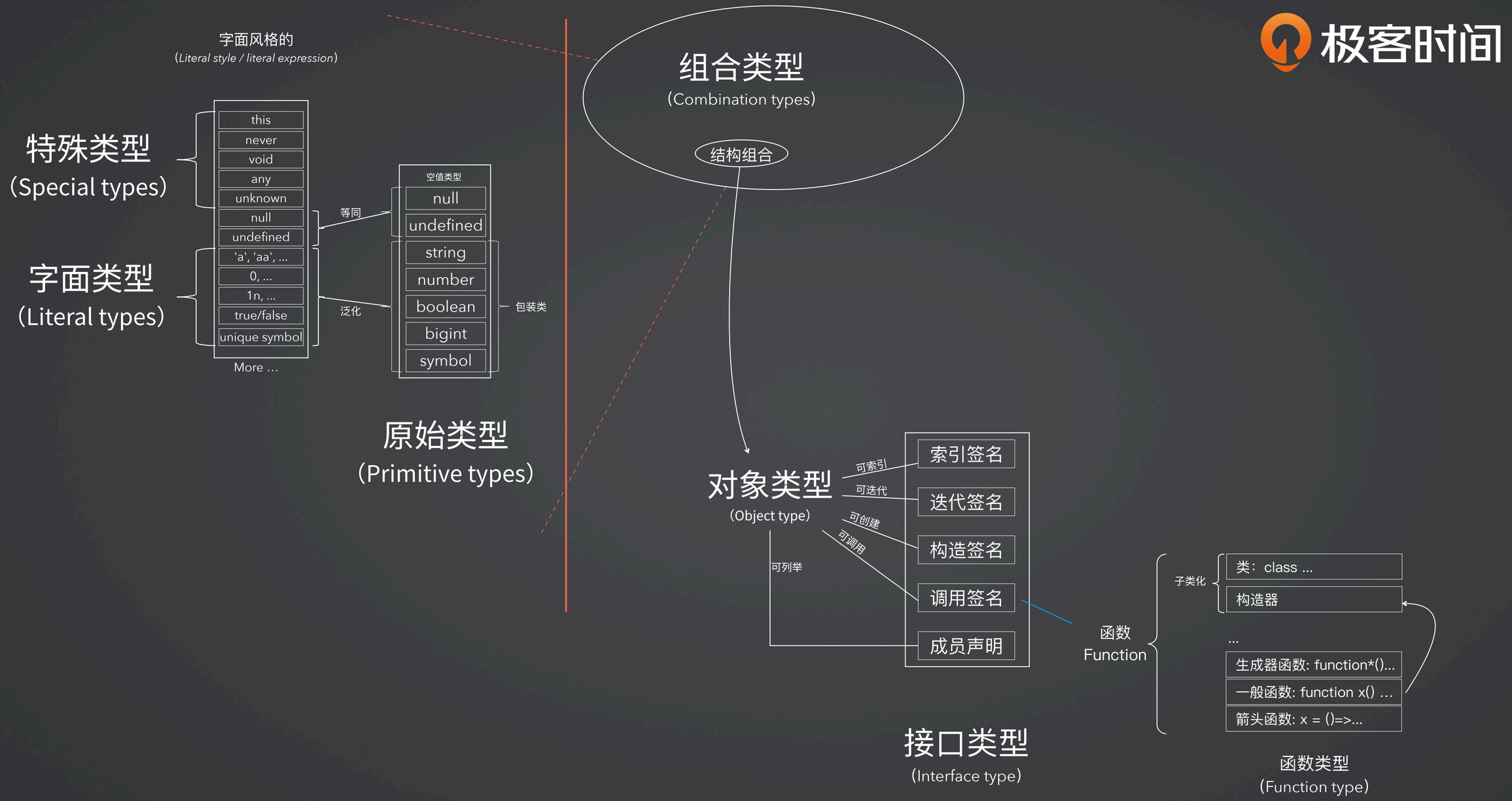
周爱民（Aimingoo）

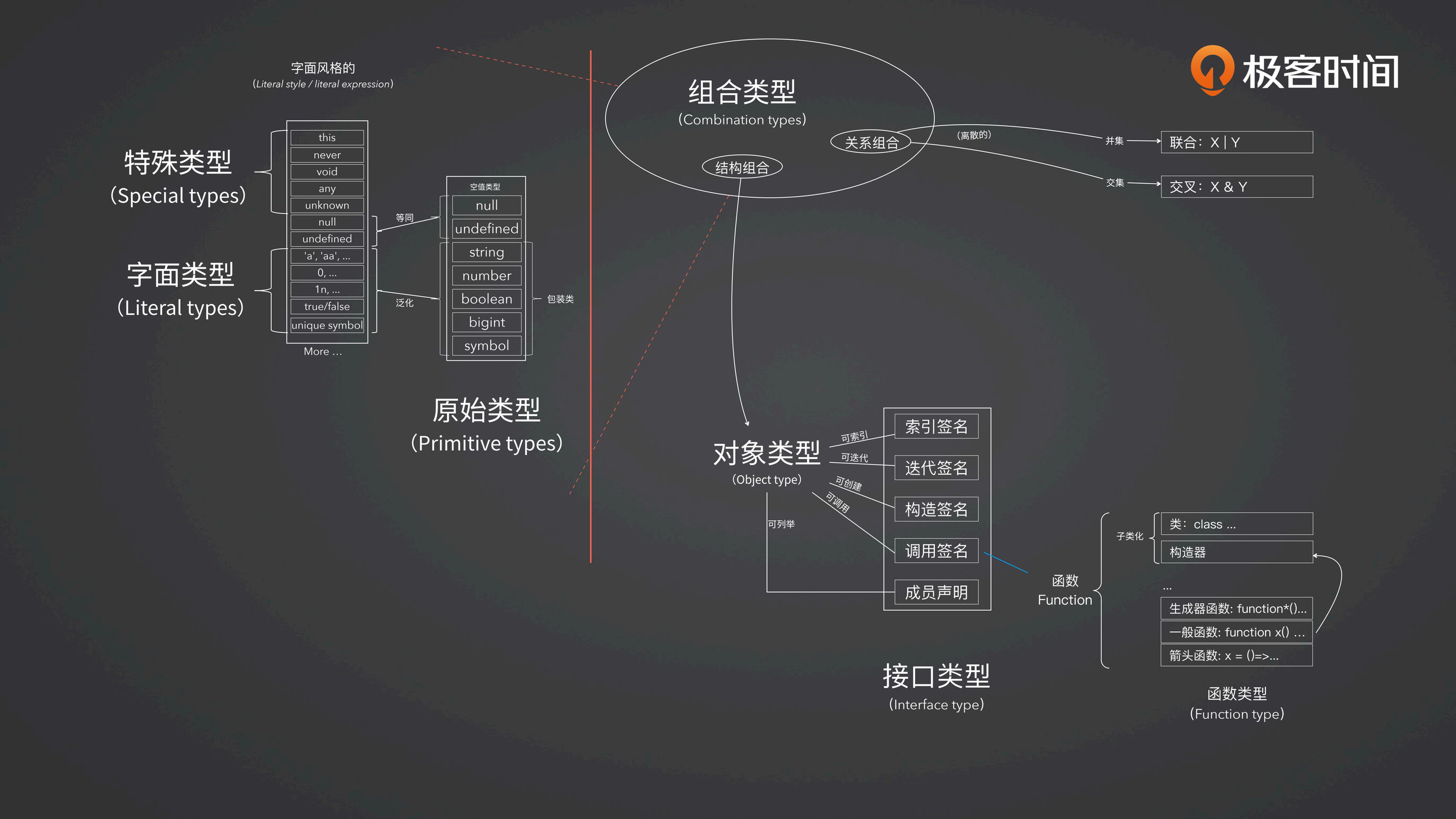
目录

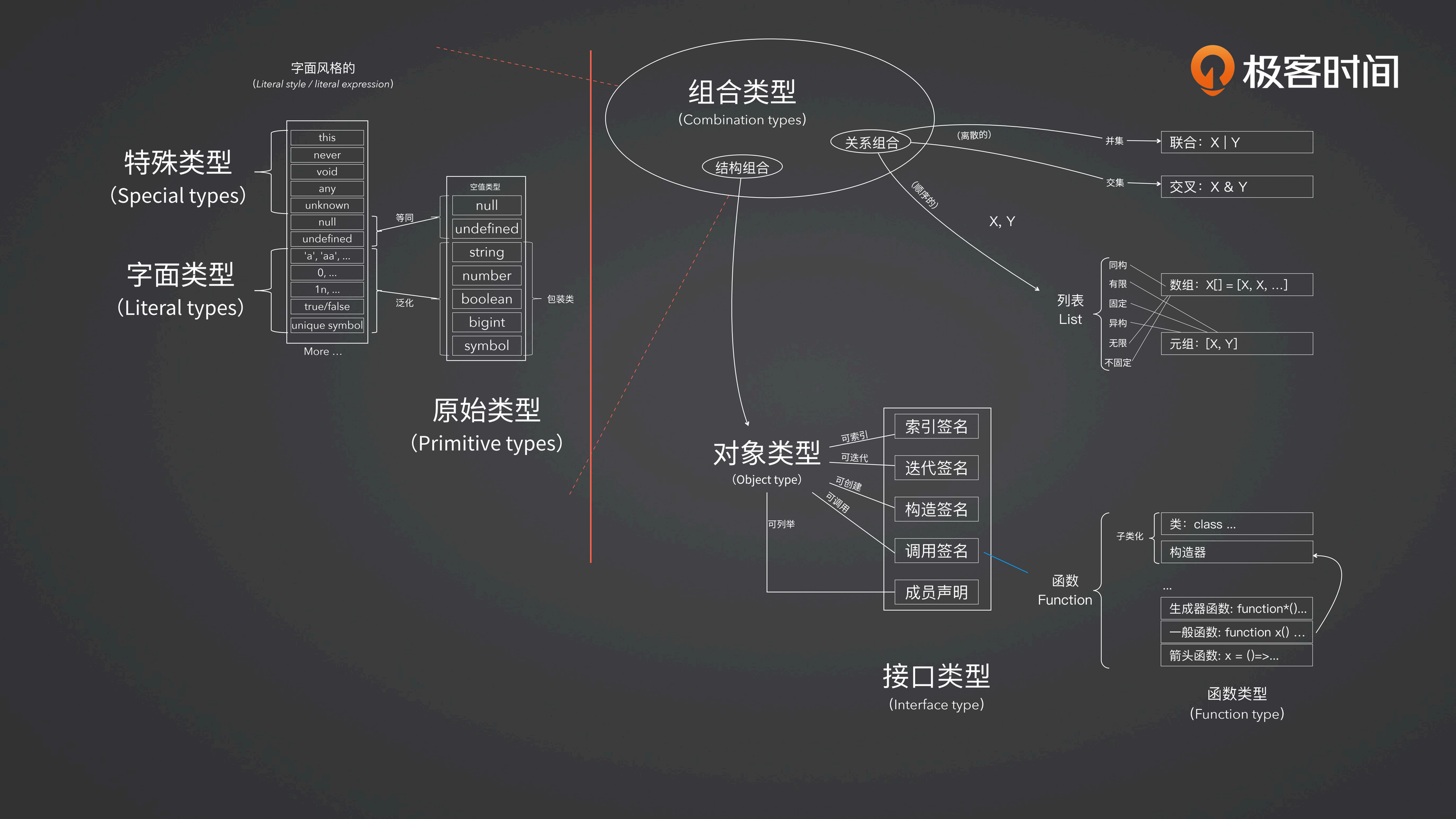
1 TypeScript 类型系统全景

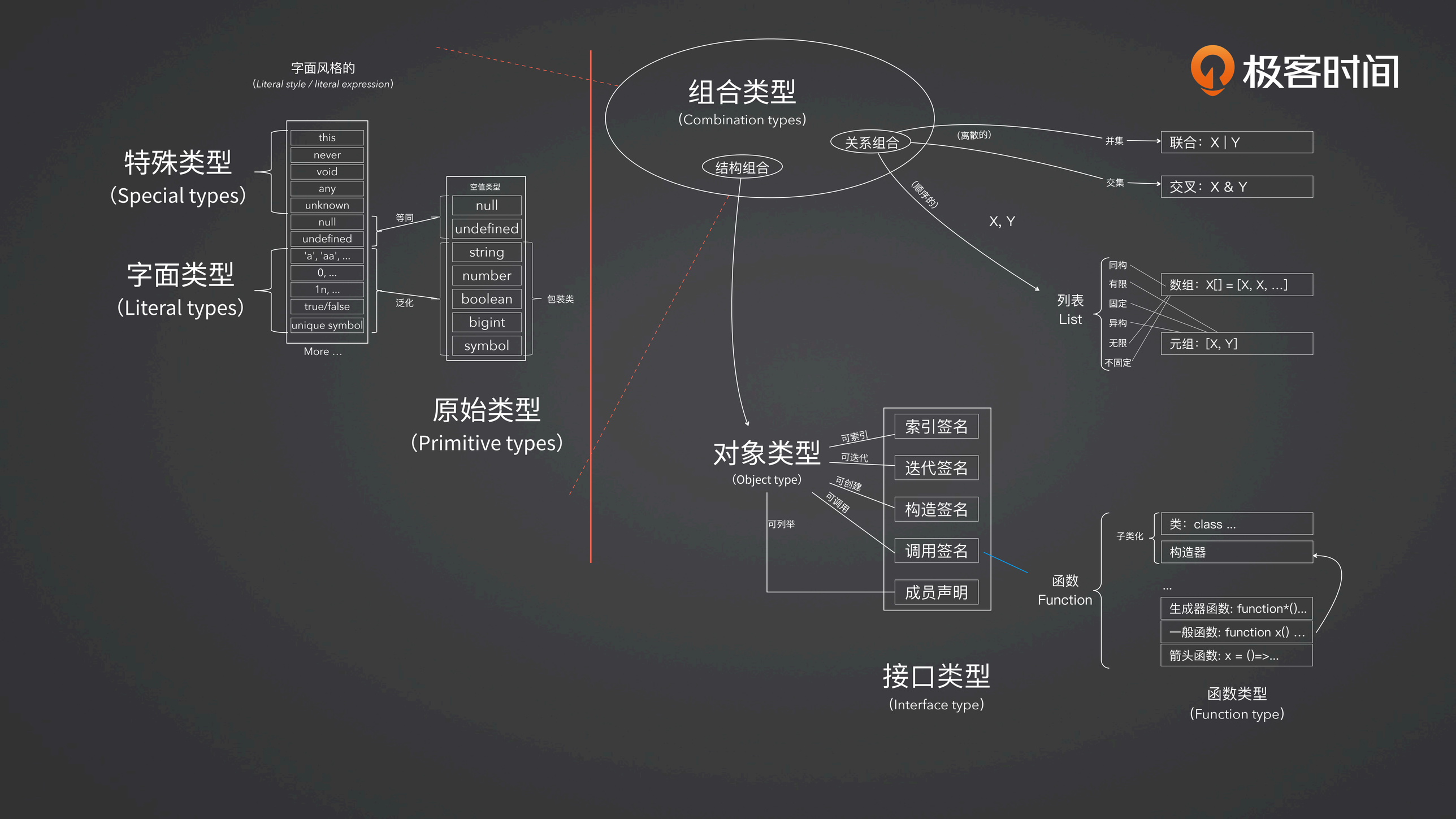
2 总结

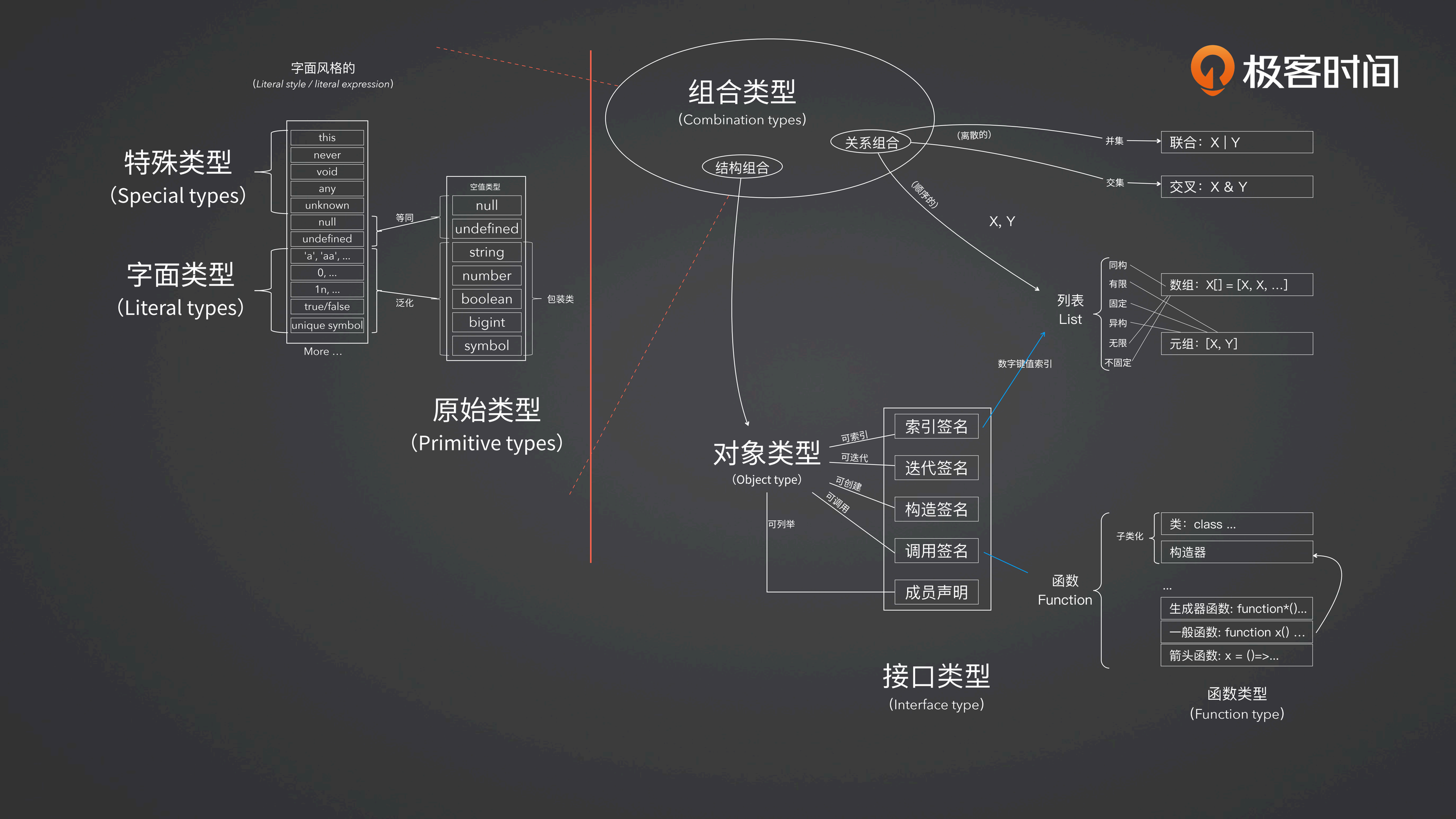

基础类型
(Base types)
组合类型
(Combination types)
极客时间
字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
对象类型
(Object type)
可调用
可列举
成员声明
函数
Function
子类化
类：class ...
构造器
对象创建
自函数
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
函数类型
(Function type)
接口类型
(Interface type)

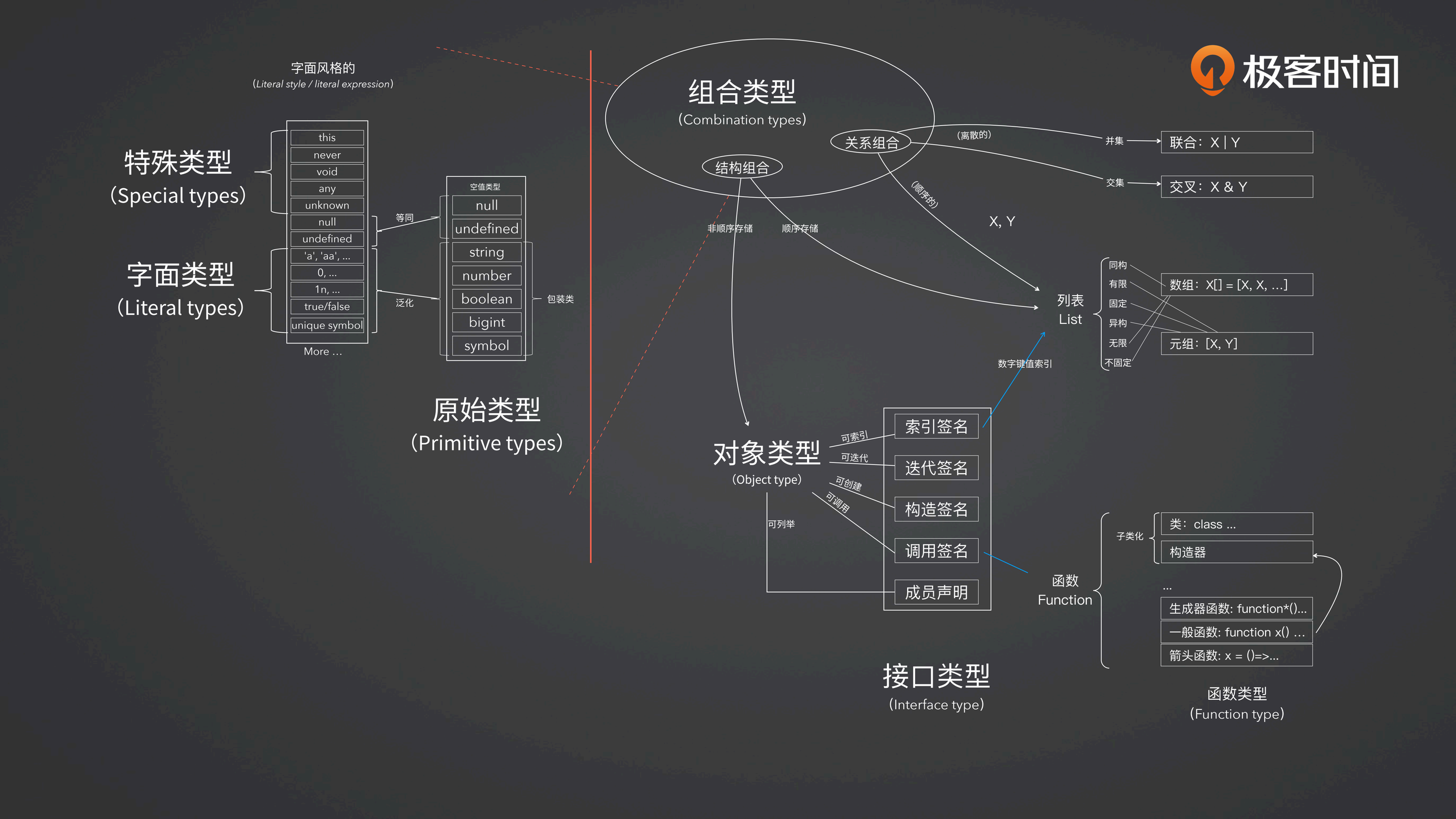

字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
结构组合
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
接口类型
(Interface type)
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
函数类型
(Function type)

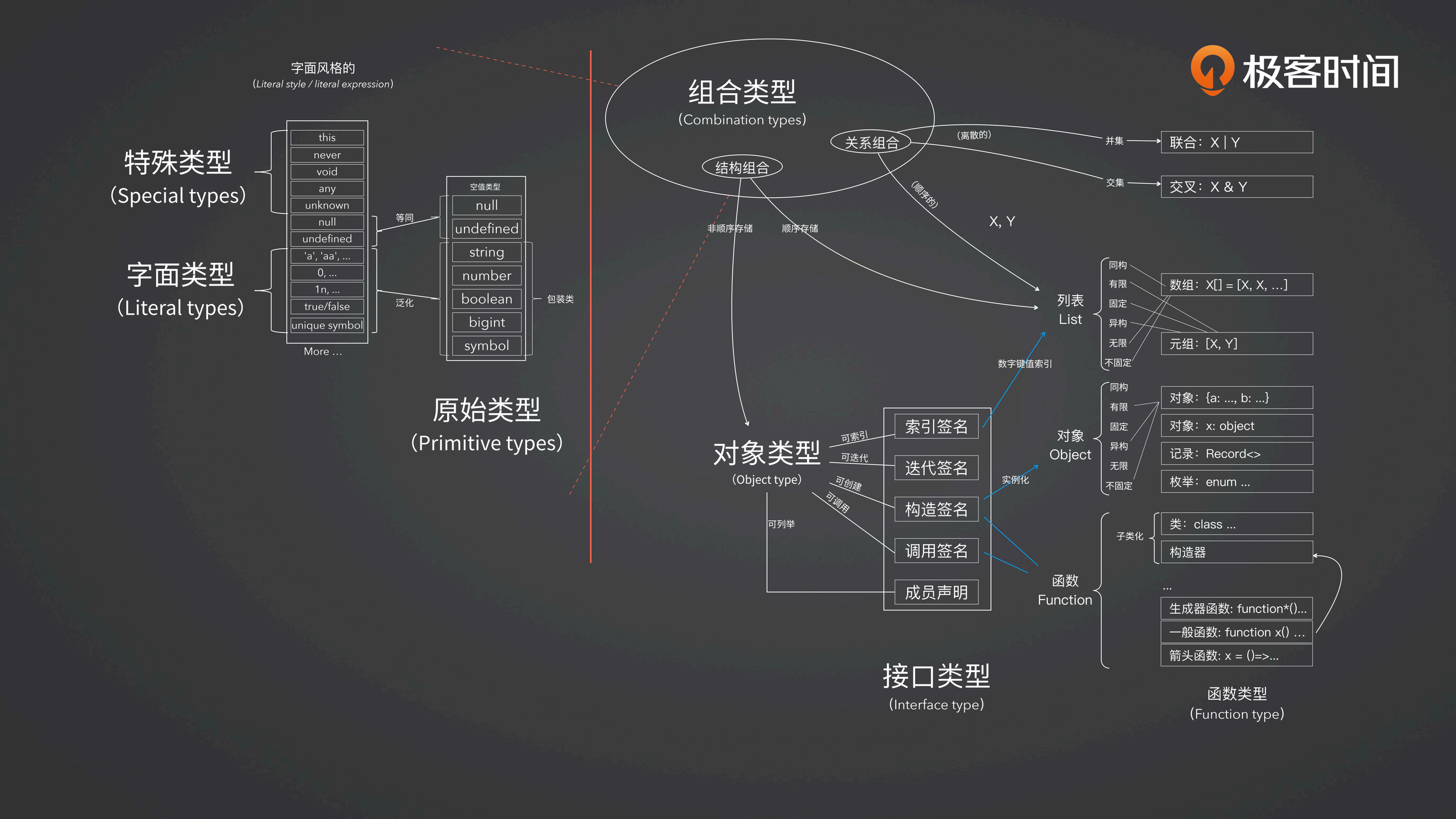

字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
并集
联合：X | Y
交集
交叉：X & Y
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
接口类型
(Interface type)
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
函数类型
(Function type)
极客时间

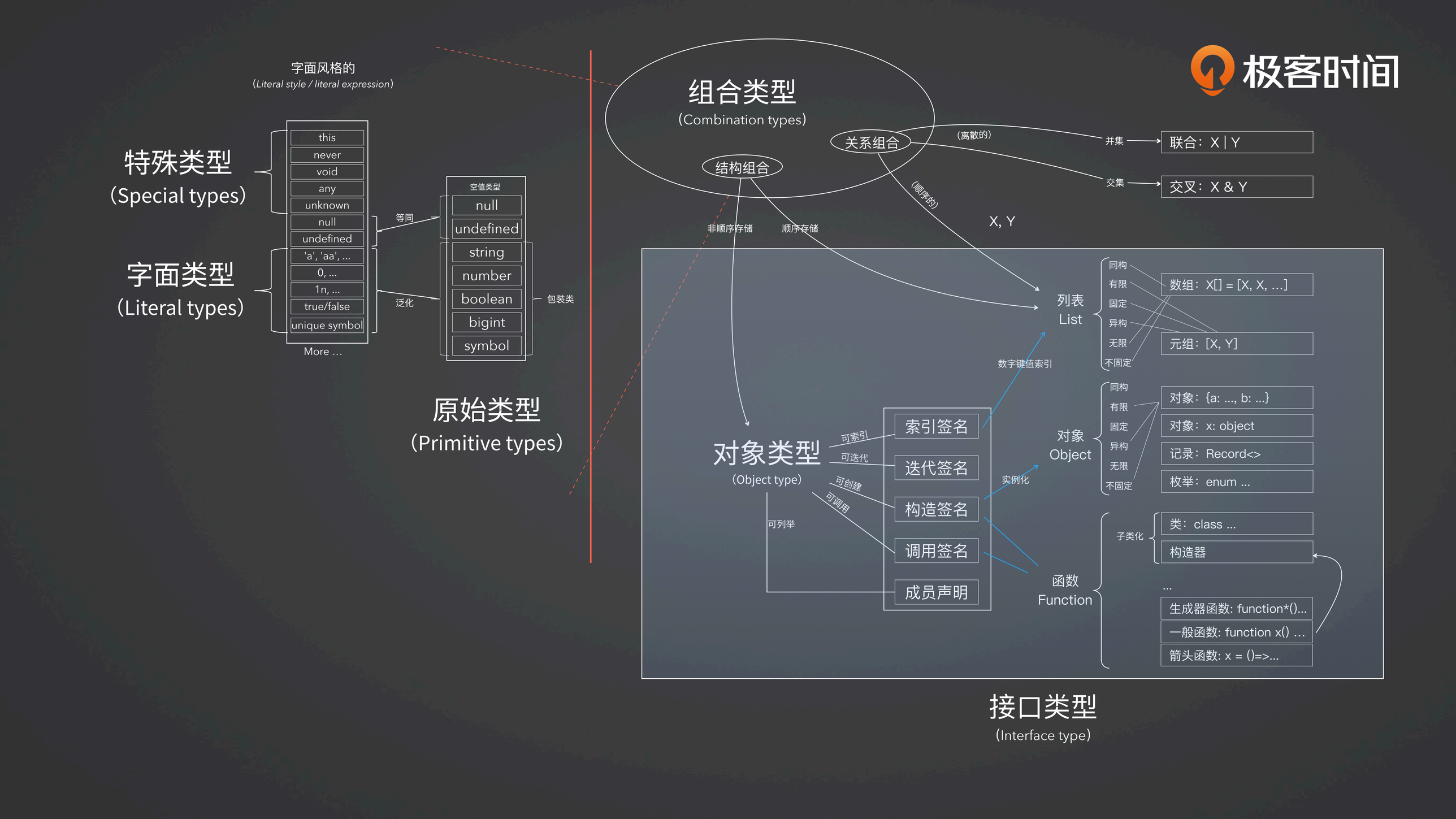

字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
并集
联合：X | Y
交集
交叉：X & Y
(顺序的)
X, Y
列表
List
同构
有限
固定
异构
无限
不固定
数组：X[] = [X, X, ...]
元组：[X, Y]
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
接口类型
(Interface type)
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
函数类型
(Function type)


字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
(顺序的)
并集
交集
联合：X | Y
交叉：X & Y
X, Y
列表
List
同构
有限
固定
异构
无限
不固定
数组：X[] = [X, X, ...]
元组：[X, Y]
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
接口类型
(Interface type)
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
函数类型
(Function type)


字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
并集
联合：X | Y
交集
交叉：X & Y
(顺序的)
X, Y
列表
List
同构
有限
固定
异构
无限
不固定
数组：X[] = [X, X, ...]
元组：[X, Y]
数字键值索引
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
接口类型
(Interface type)
函数类型
(Function type)
极客时间


字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
(顺序的)
并集
交集
联合：X | Y
交叉：X & Y
非顺序存储
顺序存储
X, Y
列表
List
同构
有限
固定
异构
无限
不固定
数组：X[] = [X, X, ...]
元组：[X, Y]
数字键值索引
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
接口类型
(Interface type)
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
函数类型
(Function type)


字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
(顺序的)
并集
交集
联合：X | Y
交叉：X & Y
非顺序存储
顺序存储
X, Y
列表
List
同构
有限
固定
异构
无限
不固定
数组：X[] = [X, X, ...]
元组：[X, Y]
数字键值索引
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
实例化
对象
Object
对象：{a: ..., b: ...}
对象：x: object
记录：Record<>
枚举：enum ...
函数
Function
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
接口类型
(Interface type)
函数类型
(Function type)


字面风格的
(Literal style / literal expression)
特殊类型
(Special types)
字面类型
(Literal types)
this
never
void
any
unknown
null
undefined
'a', 'aa', ...
0, ...
1n, ...
true/false
unique symbol
More ...
等同
泛化
空值类型
null
undefined
string
number
boolean
bigint
symbol
包装类
原始类型
(Primitive types)
组合类型
(Combination types)
关系组合
结构组合
(离散的)
(顺序的)
并集
交集
联合：X | Y
交叉：X & Y
X, Y
非顺序存储
顺序存储
对象类型
(Object type)
可索引
可迭代
可创建
可调用
可列举
索引签名
迭代签名
构造签名
调用签名
成员声明
数字键值索引
实例化
列表
List
对象
Object
函数
Function
同构
有限
固定
异构
无限
不固定
数组：X[] = [X, X, ...]
元组：[X, Y]
对象：{a: ..., b: ...}
对象：x: object
记录：Record<>
枚举：enum ...
子类化
类：class ...
构造器
...
生成器函数: function*()...
一般函数: function x() ...
箭头函数: x = ()=>...
接口类型
(Interface type)

课程进度

1. 有关类型的补充

- 补充讲述联合与交叉在接口类型（亦即是结构类型）上的表现
- 补充讲述接口类型的签名，并以 4 种签名为基础讲述列表（数组与元组）、函数、类、构造器等类型
- 补充讲述枚举等类型

2. 有关 TypeScript 语言特性的讲述

- 讲述 TS 对 JS 运行时的影响与交互
- 讲述 TS 作为一门语言的基础元素：语句、名字（包括同名策略）与表达式等

总结

1. 分类法

- 一、基础类型 + 复合类型
- 二、特殊类型 + 字面类型 + 原始类型 + 对象类型（列表+对象+函数）与接口类型
- 三、（特殊类型 + 字面类型 + 接口类型）+（对象类型 + 函数类型 + 原始类型 + 类/构造器类型）

2. 接口是对象的外观与表现：表达对象的成员列表

- “对象是类的实例”、“接口是对象的外观”与“类实现接口并以此为外观来交付实例”在概念上是重叠的，这导致“对象类型”和“接口类型”在语法和语义上近似。
- 注意 `type` 和 `interface` 关键字的用法

3. 接口用签名来表达对象的各种性质

名词/概念

名词、术语

可列举、可调用、可创建、可存取：	Enumerable, Callable, Creatable, Accessible
可构造的、可迭代的、固定的、异构的：	Constructible, Iterable, Fixed/Fixed layout, Heterogeneous
结构、结构化的、复合类型：	Struct, Constructed,, Composite/compound types
组合类型、 结构组合、关系组合：	Combination types, Constructed combination, Relational combination
匿名类型、具名类型、别名：	Anonymous/Unnamed types、Named types、Alias (Alias of type)
结构化数据类型、顺序存储、非顺序存储：	Structured data types, contiguously stored, Noncontiguously stored
结构类型系统（*in TypeScript*）：	Structural–typing type system

概念

- 复合类型（Composite）是由其它类型按组合规则复合而成（*be constructed using primitive data types and other composite types*），也称为结构或聚合（*structure or aggregate type*）。复合是组合规则之一，侧重于表达静态的、结构化的组合关系。注意，在某些情况下，TypeScript强调它是一个“结构类型的（Structural-typing）”类型系统，在语义上说的是它的类型兼容性处理方式，而与具体的类型（例如Struct、object, or array）无关。
  @see: https://en.wikipedia.org/wiki/Composite_data_type
  @see: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-func.html#structural-typing
- 组合（Combination）是在既有类型上得到新类型的方法，组合规则既可以是静态声明的，也可以是逻辑计算的。
- 数据类型包括它的逻辑组成与存储它的方式（*mode of storage associated with it*），结构化数据类型是由基础的标量（*scalar data types, or base data types*）构成，例如数组、记录等（*Structured data types are made of the scalar types, ex.*）。这些数据类型的存储方式包括顺序的或非顺序的（连续或非连续存储， Contiguously/Noncontiguously stored），也称为值类型的或引用类型的。
  @see: https://www.geeksforgeeks.org/difference-between-contiguous-and-noncontiguous-memory-allocation
  @see: https://www.pilotlogic.com/sitejoom/index.php/wiki/104-wiki/pascal-basics/chapter-1/115-pascal-data-types.html
- 匿名类型是指任何非系统内建名字（built-in names）的类型，例如所有的字面类型。具名类型只是为了给予类型一个名字（*Named types just give a name to a type*），这有两种方式：1、使用type关键字创建别名（*type alias*）；2、使用interface、class等关键字进行具名声明（Declaration）。
  @ https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-func.html

Q&A

Q: 枚举类型为什么会归类到“对象(object)”中?

A: 在TypeScript中，枚举在类型系统中表现为联合，但它是作为对象来实现的。这会在“第12讲 | 枚举类型”中详细讲述。

Q: “记录：Record<>”是一个什么类型?

A: 一般来说，它也被称为“结构（Struct）”，不过tc39也在提出自己的关于“struct”这个类型的提案。此外，“Record<>”这种写法，指的是TypeScript中的泛型工具，有些时候它直接称为“Record类型”，也有些时候它写作“Record<T>”这样的带参数的形式。

Q: 为什么官方手册（或其它资料）中没有提到“关系组合”

A: 这个概念是将动态类型与静态类型分隔开的时候的一个讲法，关系组合更倾向于（通过运算）来表达类型之间的关系。并且，从这个角度上出发，类型（X、Y、...）之间会存在的关系描述有三种：X | Y、X & Y，和 X, Y，正好对应几种TypeScript类型表达式的书面记法。此外，TypeScript内部实现中也确实是使用“关联（Relation / Relation Cache）”来描述类型之间的兼容逻辑。


极客时间
THANKS