

# TypeScript

解决 JavaScript 类型系统的问题

# 内容概要

SUMMARY

- 强类型与弱类型
- 静态类型与动态类型
- JavaScript 自有类型系统的问题
- Flow 静态类型检查方案
- TypeScript 语言规范与基本应用

# 类型系统

强类型与弱类型



```
class Main {  
    static void foo(int num) {  
        System.out.println(num);  
    }  
  
    public static void main(String[] args) {  
        Main.foo(100); // ok  
  
        Main.foo("100"); // error "100" is a string  
  
        Main.foo(Integer.parseInt("100")); // ok  
    }  
}
```

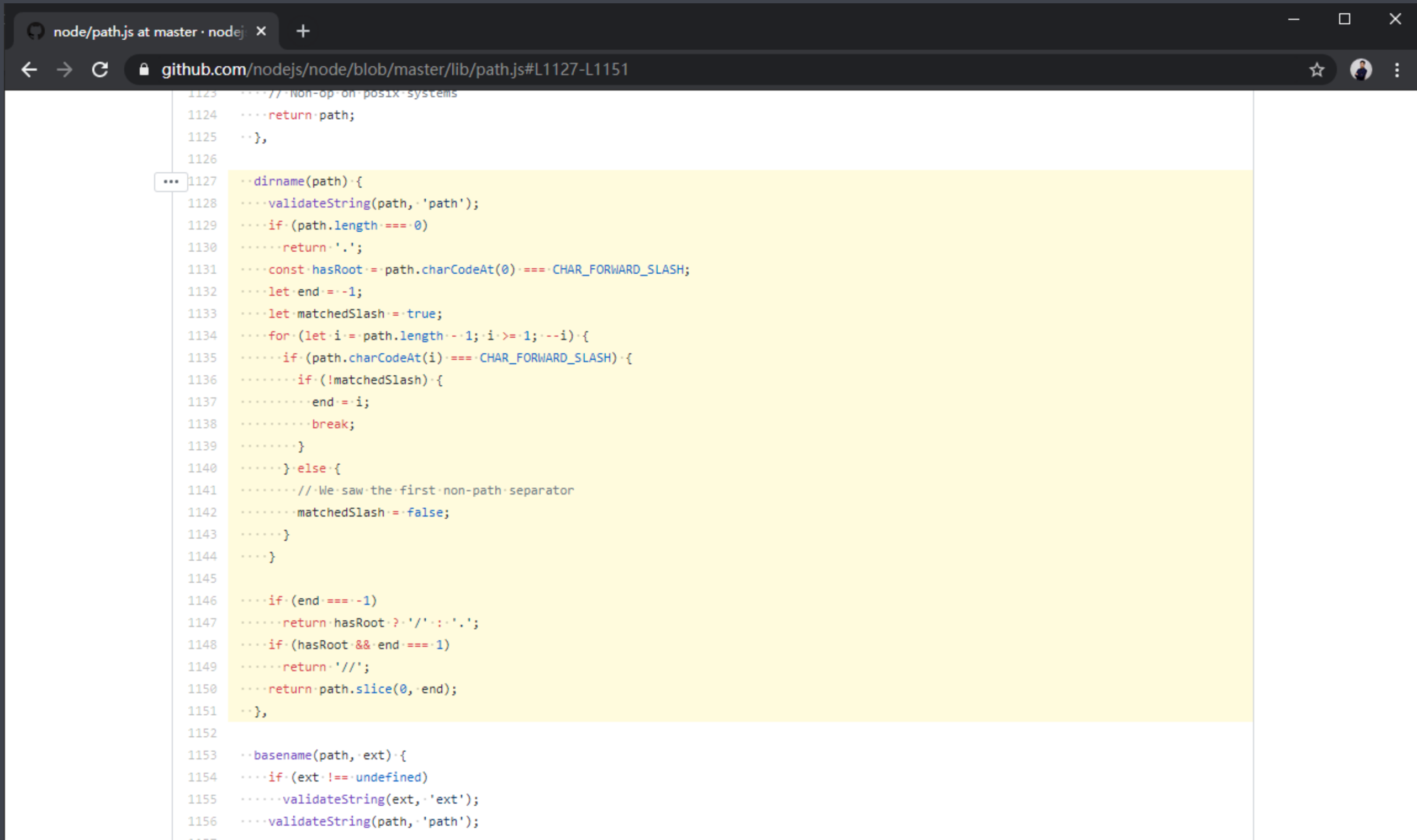


```
function foo (num) {  
  console.log(num)  
}
```

```
foo(100) // ok
```

```
foo('100') // ok
```

```
foo(parseInt('100')) // ok
```



```
1123     ...// Non-ops on posix systems
1124     ...return path;
1125     ...},
1126
1127     ...dirname(path)·{
1128     ...    validateString(path, 'path');
1129     ...    if (path.length === 0)
1130     ...        return '.';
1131     ...    const hasRoot = path.charCodeAt(0) === CHAR_FORWARD_SLASH;
1132     ...    let end = -1;
1133     ...    let matchedSlash = true;
1134     ...    for (let i = path.length - 1; i >= 1; --i) {
1135     ...        if (path.charCodeAt(i) === CHAR_FORWARD_SLASH) {
1136     ...            if (!matchedSlash) {
1137     ...                end = i;
1138     ...                break;
1139     ...            }
1140     ...        } else {
1141     ...            // We saw the first non-path separator
1142     ...            matchedSlash = false;
1143     ...        }
1144     ...    }
1145
1146     ...    if (end === -1)
1147     ...        return hasRoot ? '/' : '.';
1148     ...    if (hasRoot && end === 1)
1149     ...        return '//';
1150     ...    return path.slice(0, end);
1151     ...},
1152
1153     ...basename(path, ext)·{
1154     ...    if (ext !== undefined)
1155     ...        validateString(ext, 'ext');
1156     ...    validateString(path, 'path');
```

# 类型系统

静态类型与动态类型



```
class Main {  
    public static void main(String[] args) {  
        int num = 100;  
  
        num = 50; // ok  
  
        num = "100"; // error  
  
        System.out.println(num);  
    }  
}
```





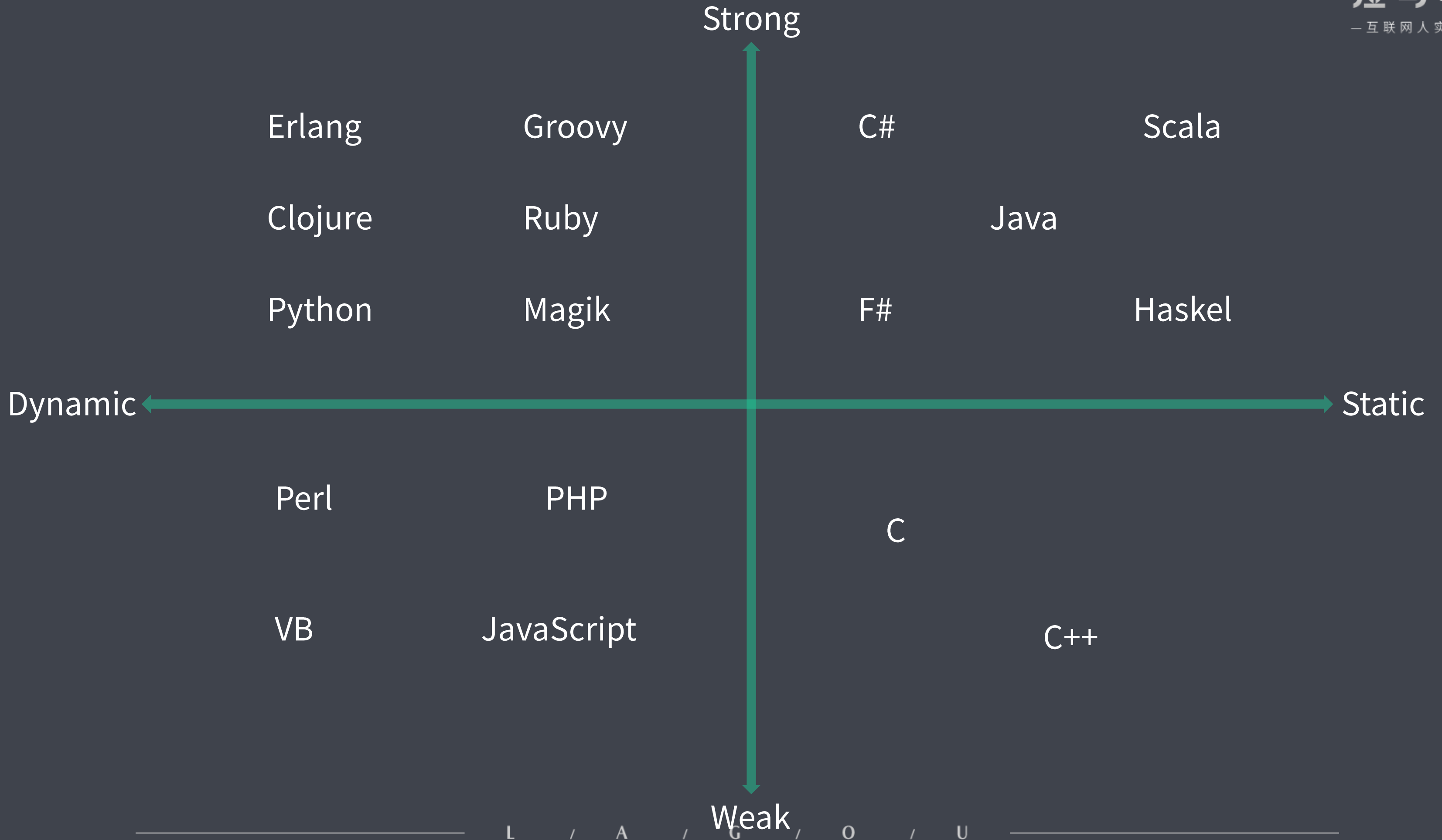
```
var num = 100
```

```
num = 50 // ok
```

```
num = '100' // ok
```

```
num = true // ok
```

```
console.log(num)
```



# JavaScript 类型系统特征

弱类型 且 动态类型

# 弱类型的问题

# 强类型的优势

# Flow

JavaScript 的类型检查器

```
function square (n) {  
    return n * n  
}
```

```
square('100')
```

```
function square (n: number) {  
    return n * n  
}
```

```
square('100')
```



```
function square (n: number) {  
    return n * n  
}
```

```
square('100')
```

```
function sum (a, b) {  
    return a + b  
}
```

```
function sum (a: number, b: number) {  
    return a + b  
}
```

```
function sum (a: number, b: number) {  
    return a + b  
}
```

```
sum(100, 50)
```

```
function sum (a: number, b: number) {  
    return a + b  
}
```

```
sum(100, 50)
```

```
sum('100', 50)
```

```
function sum (a: number, b: number) {  
  return a + b  
}
```



Babel

```
function sum (a, b) {  
  return a + b  
}
```

```
function sum (a: number, b) {  
    // a: number  
    // b: any  
    return a + b  
}
```

# 快速上手

Flow



# 编译

通过编译移除「类型注解」

# 开发工具插件

Flow Language Support

# 类型推断

Type Inference

# 类型注解

Type Annotations

# 原始类型

Primitive Types

# 数组类型

Array Types

# 对象类型

Object Types

# 函数类型

Function Types



# 特殊类型

# Mixed & Any

任意类型

# 类型小结

# 运行环境 API

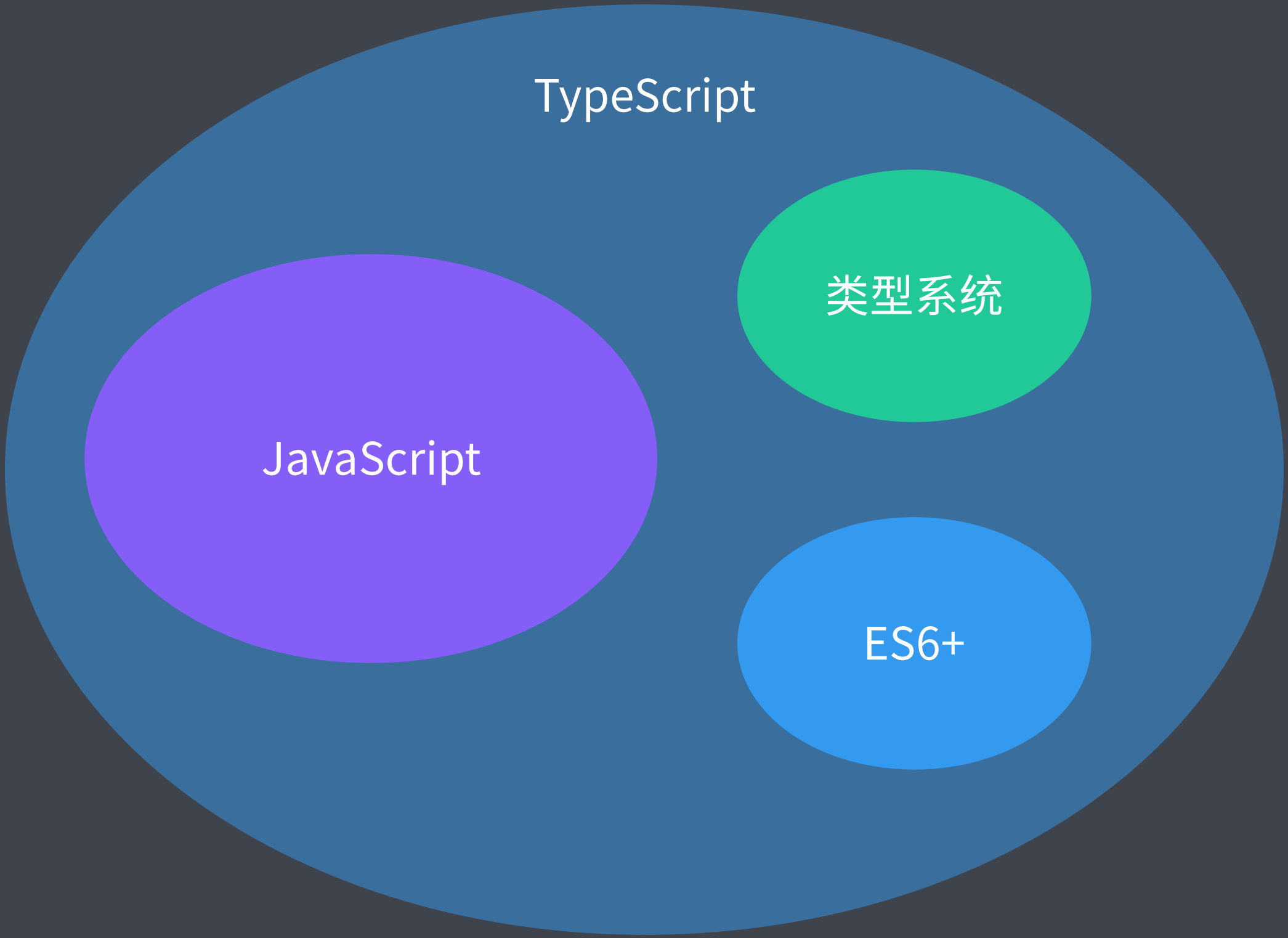
内置对象

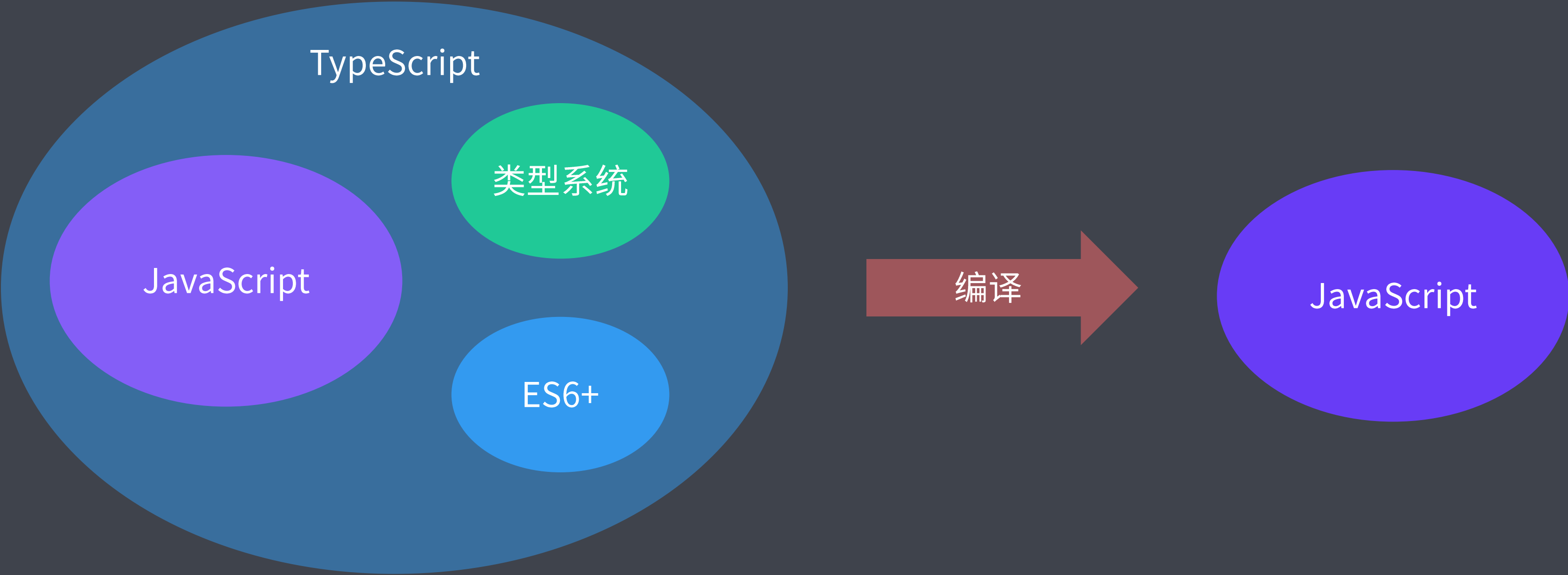
# TypeScript

JavaScript 的超集 (superset)



JavaScript







# 快速上手

TypeScript

# 配置文件

# 原始类型

Primitive Types

# 标准库声明

内置对象类型

# 中文错误消息

# 作用域问题

# Object 类型

Object Types

# 数组类型

Array Types



# 元组类型

Tuple Types

# 枚举类型

Enum Types

# 函数类型

Function Types

# 任意类型

Any Types

# 隐式类型推断

Type Inference

# 类型断言

Type assertions

# 接口

Interfaces

# 接口

可选成员、只读成员



类

Classes

# 类

## 访问修饰符

类

只读属性

# 类

## 类与接口

类

抽象类

# 泛型

Generics

# 类型声明

Type Declaration