TypeScript 类型系统

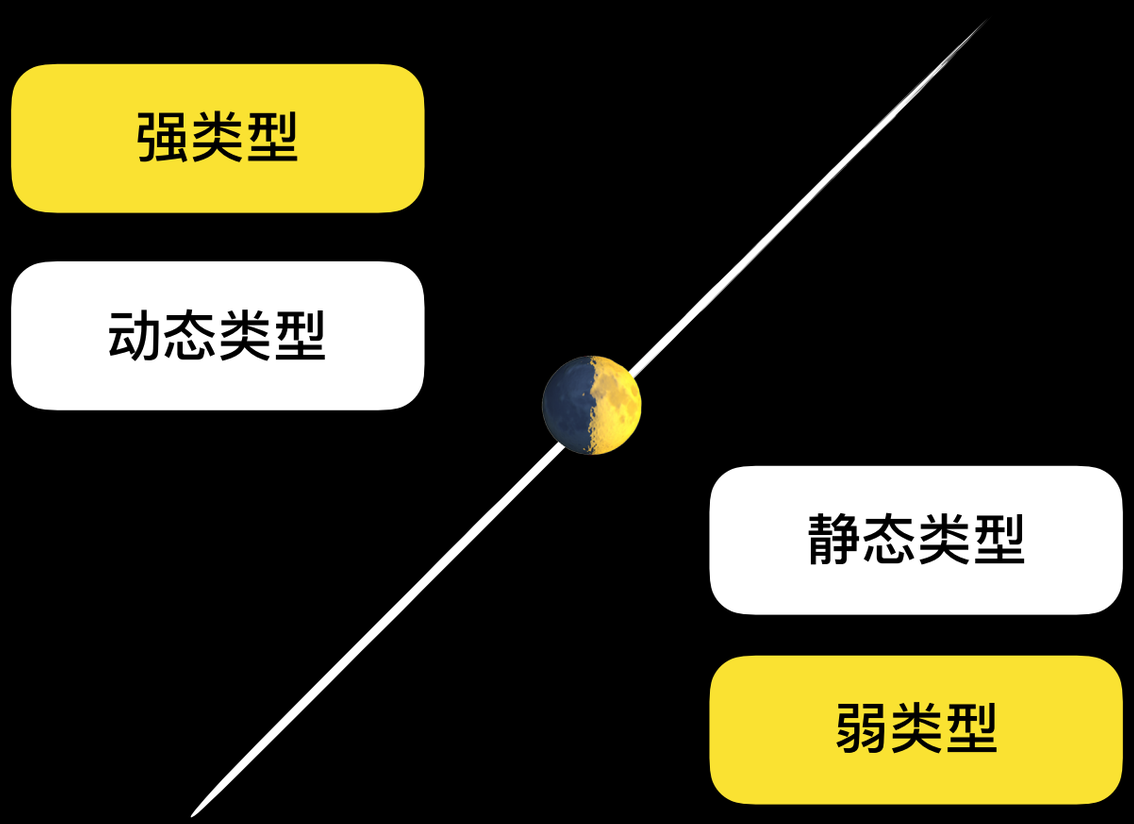
@小胡子哥

概要

- 👍🏻 了解动态类型和静态类型
- 👍🏻 了解强类型和弱类型
- 👍🏻 理解 TypeScript 类型系统及使用
- 👎🏾 TypeScript 使用细节

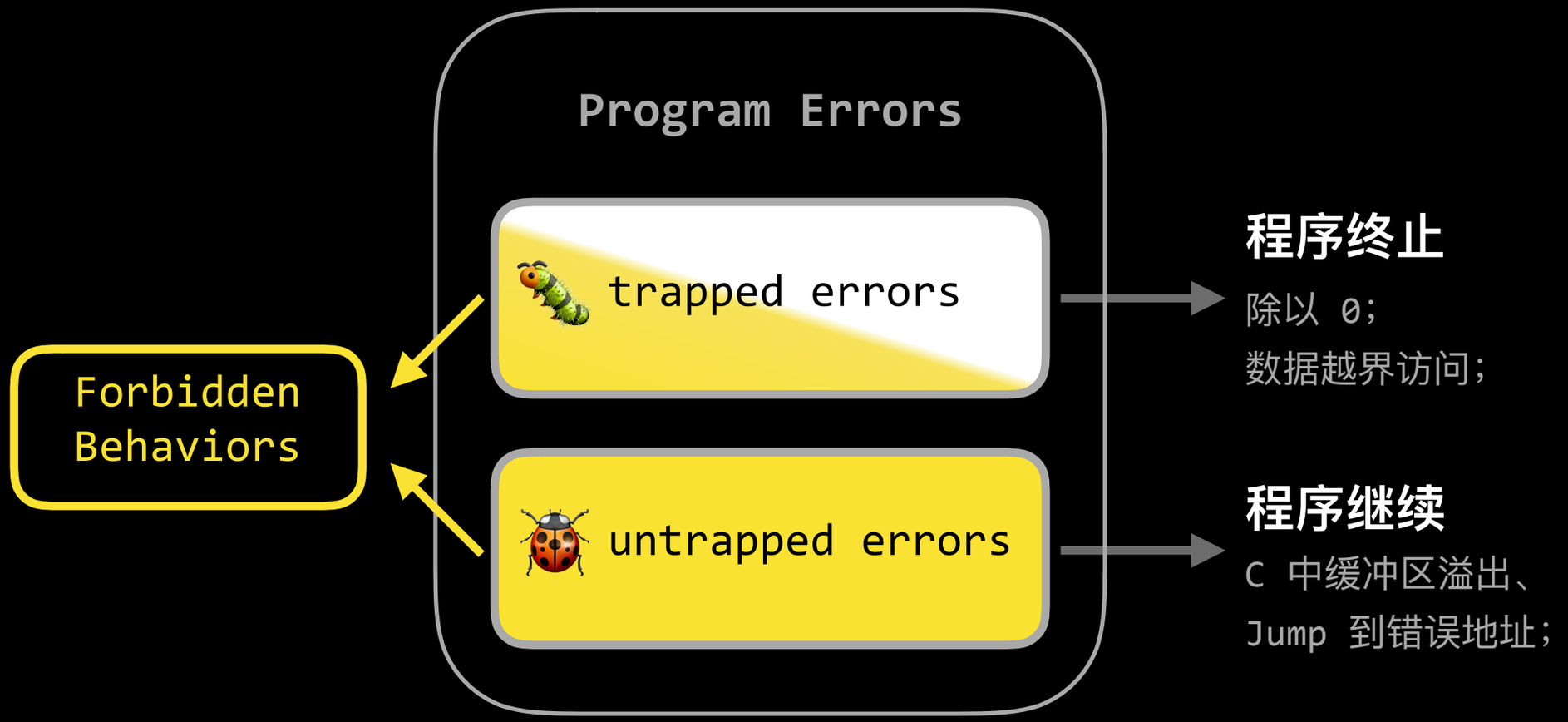
TypeScript

类型

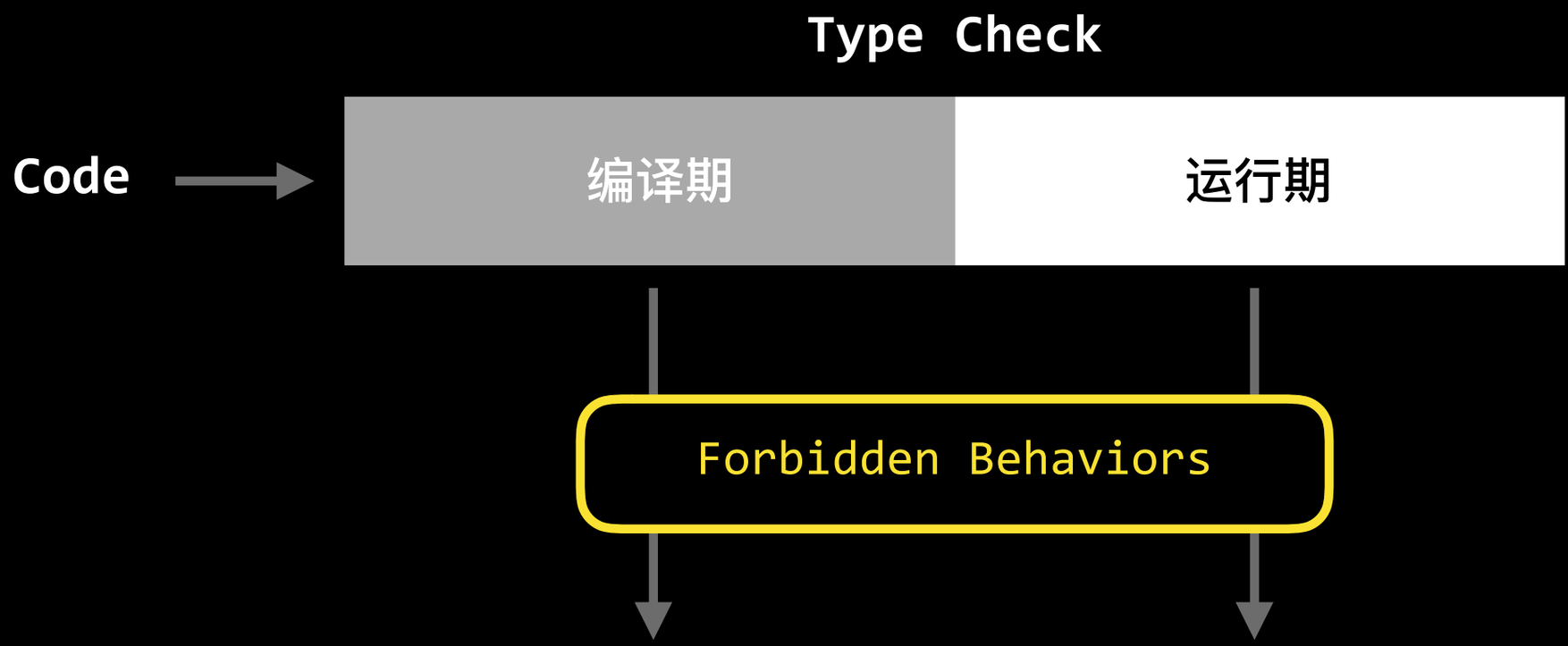


JavaScript 是什么类型，Java 是什么类型?

类型，学术上的定义


Program Errors
trapped errors
untrapped errors
Forbidden Behaviors
程序终止
除以 0;
数据越界访问;
程序继续
C 中缓冲区溢出、
Jump 到错误地址;

类型，动静判断



JavaScript 无编译期，故为**动态类型**

Java 编译期会暴露 FB，故为**静态类型**

类型，强弱判断

	强类型	弱类型
“6” + 1	❌	✅
`Some Forbidden Behaviors`	❌	✅

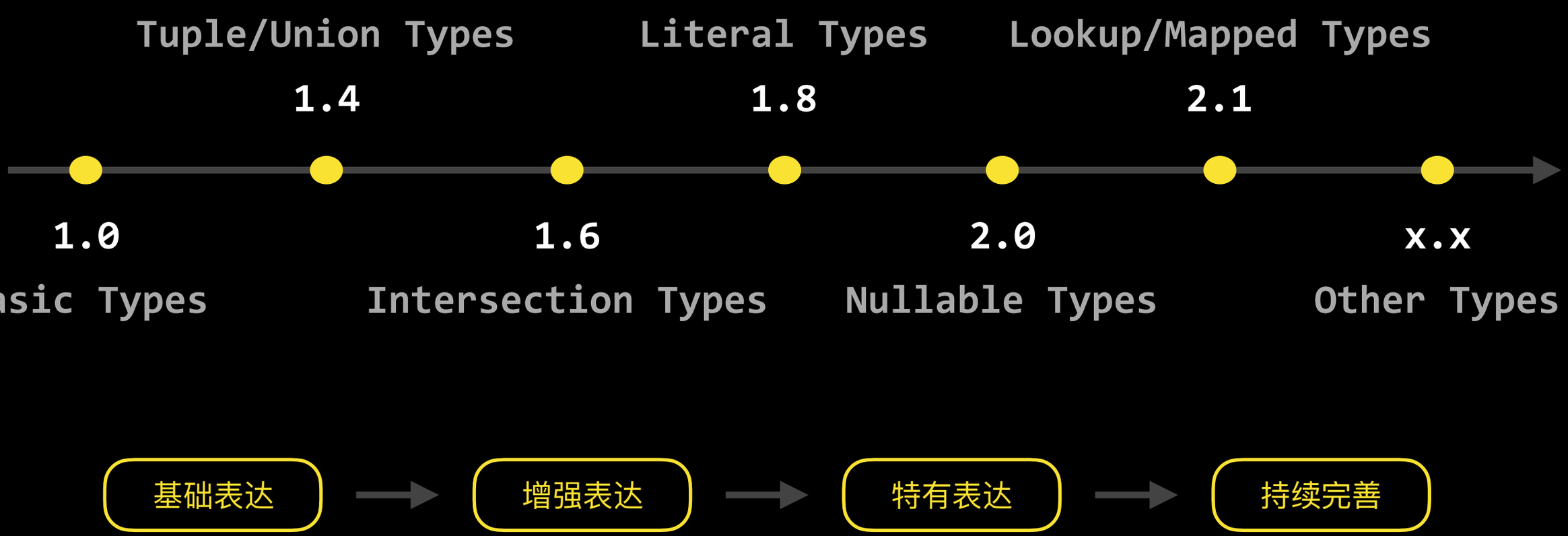
不讨论强弱类型，可以看是否偏向于容忍隐式类型转换

类型，TypeScript

👉TypeScript 具备动态检查特征，编译期也具备静态检查操作；具备弱类型特征，也存在强类型约束。

那它到底是什么类型？这并不重要，理解意思就好👌

TypeScript 类型系统（2012.10~now）



Basic Types　　Intersection Types　　Nullable Types　　Other Types

基础表达 → 增强表达 → 特有表达 → 持续完善

ECMAScript Improving!

TypeScript?

基础类型

Basic Types

基本类型（Basic Typing）

```ts
let isDone: boolean = false; // Boolean
let decimal: number = 6;     // Number
let color: string = "blue";  // String

let list: number[] = [1, 2, 3]; // Array
let x: [string, number];        // Tuple
enum Color {Red, Green, Blue}   // Enum

function warnUser(): void {     // Void
  console.log("This is my warning message");
}
class Animal { type = 1 }       // Class
```

🔑 编程基础数据单元，静态类型约束

Basic Types

渐进定型（Gradual Typing）

```
function padLeft(value: string, padding: any) {
  if (typeof padding === "number") {
      return Array(padding + 1).join(" ") + value;
  }
  if (typeof padding === "string") {
      return padding + value;
  }
  throw new Error(`Expected string or number, got '${padding}'.`);
}

padLeft("Hello world", 4); // returns "    Hello world"
```

🔑 用户输入、三方库等未知类型场景，静态类型和动态类型混合

Basic Types

结构定型（Duck Typing）

```
function printLabel(labeledObj: { label: string }) {
  console.log(labeledObj.label);
}
let myObj = {size: 10, label: "Size 10 Object"};
printLabel(myObj);
```

```
// ====== TypeScript Code ======
interface LabeledValue {label: string;}
function printLabel(labeledObj: LabeledValue) {
  console.log(labeledObj.label);
}
let myObj = {size: 10, label: "Size 10 Object"};
printLabel(myObj);
```

🔑 通过结构定型，让程序设计更具鲁棒性

高级类型

Generic

约束函数入参和返回

```ts
type myType = string | number;
function loggingIdentity<T extends myType>(arg: T[]): T[] {
  console.log(arg.length);
  return arg;
}
loggingIdentity([1, 2, 'str']);
// Error: Type '{}' is not assignable to type 'string | number'.
loggingIdentity([1, 2, {}]);

// ====== Complex Generic ======
function getProperty<T, K extends keyof T>(obj: T, key: K) {
  return obj[key];
}
```

Generic

约束类成员和方法

```ts
// 创建一个泛型类
class Queue<T> {
  private data :T[] = [];
  push = (item: T) => this.data.push(item);
  pop = (): T | undefined => this.data.shift();
}

// 简单的使用
const queue = new Queue<number>();
queue.push(0);
queue.push('1'); // Error: 不能推入一个 `string`，只有 number 类型被允许
```

🔑 泛型增强约束类、函数，让组件设计具备更强的兼容性、可复用性和一致性

More…

Intersection/Union/Literal/Lookup/Mapped/Nullable/Condition/…

类型系统增强：交叉、合并、反射、判断、别名…

TypeScript 的类型系统是图灵完备的!

https://www.reddit.com/r/programming/comments/6r2vmq/typescripts_type_system_is_turing_complete/

TypeScript 使用感受

- 👍🏻 “闭着眼睛”写代码，不翻文档、不看其他模块代码
- 👍🏻 自动生成文档，RPC 服务类文档参数细节清晰
- 👎🏾 较大的旧项目迁移成本高，依赖库的 ts 问题

动态类型一时爽，

代码重构火葬场！

TypeScript! 好!