# 15. 立即执行函数 IIFE 是怎么工作的?

# 立即执行函数 IIFE

## 1. 核心概念 (Core Concept)

立即执行函数表达式（Immediately Invoked Function Expression, IIFE）是一个在定义后**立即执行**的 JavaScript 函数。它是一种设计模式，用于创建独立的作用域，避免变量污染全局作用域。

## 2. 为什么需要它？ (The "Why")

1. **隔离作用域 (Scope Isolation):** JavaScript 中，函数是唯一能创建独立作用域的结构（在 ES6 Module 和 Block Scope ( `let` / `const` ) 出现之前）。IIFE 利用函数作用域，将其内部的变量和函数与外部隔离开，避免命名冲突和意外的全局变量污染，特别在早期模块化不完善的环境下非常重要。
2. **避免污染全局命名空间 (Avoid Global Namespace Pollution):** 在全局作用域中声明的变量会成为全局对象的属性（如 `window` 或 `globalThis` ）。IIFE 将其内部实现细节隐藏在私有作用域中，只暴露必要的内容（如果需要的话），保持全局作用域的整洁。

## 3. API 与用法 (API & Usage)

IIFE 并不是一个新的语法结构，而是利用了 JavaScript 语言中已有的函数表达式和函数调用语法。其基本形式是在一个函数表达式后紧跟一对括号 `()`，表示立即执行该函数。

**基本结构:**

```
(function() {
  // 函数体的代码
})();
```

**工作原理分解:**

1. **函数表达式:**
   - `function() { ... }` 这部分是一个**函数表达式**（Function Expression），而不是函数声明（Function Declaration）。函数声明必须有名称，且不能紧跟一个调用符号 `()`。函数表达式可以匿名，并且可以作为值被处理，例如放置在括号 `()` 中。
   - 将函数表达式放在括号 `()` 中 `(function() { ... })`，是为了强制 JavaScript 解释器将其解析为一个表达式，而不是函数声明。在 JavaScript 语法中，出现在表达式位置的 `function` 会被解析为函数表达式。
2. **立即调用:**

- 紧随函数表达式闭合括号 `)` 后面的一对括号 `()` 表示**函数调用**。当解析器看到 `(...)()` 这样的结构时，它首先评估括号内的函数表达式，得到一个函数对象，然后立即使用后面的 `()` 来调用这个函数对象。

**代码示例 (来自 MDN & 常见用法):**

**示例 1: 创建私有作用域**

```
(function() {
  var privateVariable = "I am private";
  console.log(privateVariable); // 输出: I am private
})();

// console.log(privateVariable); // ReferenceError: privateVariable is not
defined
```

**解释:** `privateVariable` 只存在于 IIFE 的作用域内，外部无法访问。

**示例 2: 在循环中捕获正确的变量值 (早期 JavaScript 常用)**

```
// 常见的错误，所有 setTimeout 引用了最后一次循环时的 i 值
// for (var i = 0; i < 5; i++) {
//   setTimeout(function() {
//     console.log(i); // 都会输出 5
//   }, 100 * i);
// }

// 使用 IIFE 捕获每次循环的 i 值
for (var i = 0; i < 5; i++) {
  (function(index) {
    setTimeout(function() {
      console.log(index); // 输出: 0, 1, 2, 3, 4
    }, 100 * index);
  })(i); // 将当前的 i 作为参数传递给 IIFE，并在 IIFE 内部捕获
}
```

**解释:** 通过将 `i` 作为参数 `index` 传递给每次循环中创建的 IIFE， `index` 在 IIFE 的作用域内形成一个闭包，保存了当次的 `i` 值，避免了异步回调引用到循环结束后的 `i` 值。

**IIFE 的其他变体形式 (用于强制解析为表达式):**

```
// 使用一元运算符（不常用，但语法上有效）
!function() { console.log("Invoked by !"); }();
+function() { console.log("Invoked by +"); }();
-function() { console.log("Invoked by -"); }();
~function() { console.log("Invoked by ~"); }();
```

```
// 前置运算符（不常用）
(function() { console.log("Invoked with leading parenthesis"); }()); // 注意括号位置不同
```

**核心:** 目的都是让跟在 `function` 关键字后面的内容被解析为**表达式**，而不是函数声明。最常见和推荐的还是 `(function(){...})()` 形式。

# 4. 关键注意事项 (Key Considerations)

1. **语法强制为表达式:** enclosing parenthesis `()` around the function expression `(function(){...})` are crucial for the parser to treat it as an expression; otherwise, `function(){}();` would be a syntax error (函数声明后不能直接跟调用)。
2. **返回值:** IIFE 返回的是函数的执行结果。如果函数没有显式 `return` 语句，它会隐式返回 `undefined`。
3. **参数传递:** IIFE 可以接受参数，允许将外部变量“注入”到其独立作用域中，如示例 2 所示。
4. **ES6 及以后:** 随着 ES6 模块 (Modules) 和块级作用域 ( `let`, `const` ) 的引入，IIFE 在现代 JavaScript 开发中的核心作用（隔离作用域和防止全局污染）已部分被替代。模块系统提供了更健壮、标准的模块封装方式，而 `let` / `const` 提供了块级作用域。然而，IIFE 在一些特定场景（如旧项目、立即执行某些初始化代码、某些库的封装模式）仍然可能被使用或需要在阅读代码时理解。

# 5. 参考资料 (References)

- **MDN Web Docs:** Immediately invoked function expression (IIFE)
- **MDN Web Docs:** Functions reference - Function expressions
- **"JavaScript: The Good Parts" by Douglas Crockford:** (章节讨论函数作为实现私有成员的方式)
- **各种 JavaScript 模块模式相关的技术博客文章:** (早期讨论 IIFE 作为模块模式一部分的内容)