CHAPTER 1

What Is Scope?

Biên dịch lý thuyết

Điều này có thể là bình thường hoặc ngạc nhiên, phụ thuộc vào những ngôn ngữ khác nhau bạn đã học, nhưng mặc dù Javascript được liệt kê vào ngôn ngữ biên dịch~~. Nó không được biên dịch ở bậc cao hơn so với nhiều ngôn ngữ biên dịch truyền thống khác~~.

Tuy nhiên, javascript engine thi hành nhiều bước giống nhau, mặc dù cách thức phức tạp hơn những gì chúng ta thấy thông thường so với ngôn ngữ biên dịch truyền thống khác.

Trong tiến trình biên dịch ngôn ngữ truyền thống, 1 khúc dữ liệu của source code trải qua 3 bước điển hình trước khi được thực thi, nói thẳng ra là “sự biên dịch”:

***Tokenizing/Lexing***

Tách 1 đoạn string từ những kí tự có nghĩa của 1 đoạn dữ liệu được gọi là tokens. Chẳng hạn như var a = 2; , đoạn code này sẽ dc tách nhỏ thành những đoạn tokens như sau: var, a, =, 2, ;. Khoảng trắng sẽ không dc tính.

 Sự khác biệt tokenizing và lexing là khó hình dung và có tính hàn lâm, nhưng nó tập trung vào những tokens đó có được định nghĩa theo hướng phi trạng thái hay có trạng thái hay không. ~~Để đơn giản, nếu thuật toán tách từ (tokenizer) .. để tính toán có hay không nên được xem xét....~~

***Parsing***

Nhận vào 1 chuỗi tokens và trả về 1 cấu trúc cây các phần tử lồng nhau miêu tả cấu trúc ngữ pháp của chương trình. Cái cây này được gọi là “AST” (abstract syntax tree).

Cấu trúc cây của var a = 2; có thể bắt đầu ở nút cao nhất được gọi là *VariableDeclaration*, với nút con tiếp theo được gọi là Indentifer (giá trị của nó là a), nút con ở giai đoạn tiếp theo nữa được gọi là *AssignmentExpresion* , bản thân nó có 1 đứa con được gọi là *NumericLiteral* (nó có giá trị bằng 2).

***Code-Generation***

Tiến trình nhận 1 AST và trả về bên trong đoạn code có thể chạy được.

Javascript engine rộng lớn , phức tạp hơn 3 bước trên như những ngôn ngữ biên dịch khác. Chẳng hạn , trong quá trình parsing và code-generation, chắc chắn là những bước để tối ưu hóa hiệu suât chương trình

Một điều nữa, Javascript engines không có????

Understanding Scope

Khi bạn thấy chương trình var a = 2; , hầu như bạn nghĩ đây là 1 câu lệnh. Nhưng thực tế đây là là 2 câu lệnh rõ ràng, 1 cái là cho sự biên dịch, 1 cái cho việc thực thi.

Phân tích var a = 2;

Việc đầu tiên biên dịch sẽ làm với chương trình này sẽ là tiến hành chia nhỏ thành các tokens.

Một giả thiết hợp lý là trình biên dịch sẽ tạo ra code bằng việc sử dụng pseudocode: “ phân chia bộ nhớ cho 1 biến, gán nó là a, sau đó gắn value là 2 cho biến đó”.

Tóm lại: có 2 hành động rõ ràng cho viến gán 1 biến. Đầu tiên, trình biên dịch sẽ khai báo 1 biến (nếu trước đó chưa khai báo) ở trong scope hiện tại. Thứ 2 khi thi hành, Engine sẽ tìm kiếm biến trong scope và gán giá trị cho nó nếu tìm thấy.

**Compiler Speak**

Chúng ta cần 1 ít dữ liệu bit để biên dịch nhanh hơn ..

Khi Engine thi thực thi code mà trình biên dịch tạo ra ở bước 2, nó phải tìm kiếm biến a để xem nếu nó dc khai báo, bước tìm kiếm này là hỏi ý kiến scope . Nhưng kiểu tìm kiếm của Engine mà thi hành sẽ ảnh hưởng kết quả của việc tìm kiếm.

Trong trường hợp của chúng ta, điều này nói rằng Engine đang thực hiện 1 sự tìm kiếm LHS (lefthand side). Loại tìm kiếm khác được gọi là RHS(righthand side).

Side ... của cái gì? Của toán tử gán

Hay nói cách khác, 1 LHS look-up hoàn thành là khi 1 biến xuất hiện ở phía bên trái của 1 toán tử gán, RHS look-up hoàn thành khi 1 biến xuất hiện ở phía bên phải của 1 toán tử gán.

Cụ thể hơn, 1 RHS look-up là không thể phân biệt được, đơn giản là nó tìm kiếm giá trị cho biến , còn LHS là tìm 1 biến gán lại cho chính biến đó.

scope From Function

function scope khuyến khích ý tưởng là tất cả các biến thuộc về function, có thể dc sử dụng và tái sử dụng từ đầu đến cuối trong toàn bộ function . điều này rất

Hiding in Plain Scope (che giấu trong scope đơn giản)

Cách truyền thống nghĩ  về 1 function là khai báo 1 function và sau đó thêm code vào bên trong nó. Nhưng hãy nghĩ ngược lại là sự tiện dụng và hữu ích: tùy biến bất kì phần code nào mà bạn viết ra và bao bọc chúng bên trong function bạn khai báo, cái mà có tức dụng "ẩn" phần code .

Thực ra là khai báo biến hoặc 1 hàm bên trong 1 function khác, nó che giấu những variable và function đó vì chỉ có function bao đóng nó mới sử dụng dc chúng, còn bên ngoài thì không. Chúng còn được gọi là closure.

Việc ẩn variable và function là 1 kỹ xảo. Chúng được dùng để thiêt kế design patter, hoặc API cho 1 module hoặc object. Chỉ phô ra những gì cần thiết , những thứ còn lại thì lên giấu đi.

Nếu tất cả các variable và function được đặt trong global scope, chúng có thể kết nối với bất kì nested scope nào. Nhưng điều này sẽ vi phạm tính riêng tư.

Collision Avoidance (tránh sự xung đột)

Một lợi ích khác của việc ẩn variable và function bên trong 1 scope là tránh sự xung đột không biết trước được giữa những định danh khác nhau với tên tương tự nhưng có mục đích khác. Kết quả của sự xung đột thường nằm trong sự ghi đè value không mong muốn.

function foo() {  
 function bar(a) {  
 i = 3; // changing the `i` in the enclosing scope's  
// for-loop  
 console.log( a + i );  
 }  
 for (var i=0; i<10; i++) {  
 bar( i \* 2 ); // oops, inifinite loop ahead!  
 }  
}  
foo();

Trong đoạn code trên, i = 3 dc gán bên trong function bar( ), thật không may i cũng được khai báo ở vòng lặp for trong function foo( ). Những trong trường hợp này , kết quả là vòng lặp vô hạn, bởi vì i được set giá trị cố định là 3 và mãi mãi < 10.

Để giải quyết ta chỉ cần khai báo biến local var i =3; bên trong function bar( ), bất chấp tên định danh được chọn là gì.

function bar(a) {  
 var i = 3; // changing the `i` in the enclosing scope's  
// for-loop  
 console.log( a + i );  
 }

Trong 1 chương trình, bạn hoàn toàn có thể sử dụng tên định danh giống nhau, việc tận dụng scope  để ẩn các khai báo bên trong là sự lựa chọn tốt nhất và duy nhất.

Global namespaces

Một ví dụ rất đặc biệt của sự xung đột giữa các biến xảy ra ở trong global scope. Nhiều library được load trong chương trình của bạn khá dễ xung đột với nhau nếu chúng không ẩn thực sự các function và các variable nội bộ / bảo mật.

Điển hình  library sẽ tạo ra 1 sự khai báo biến đơn, thường là 1 object, với tên gọi là duy nhất, ở trong global scope. Object đó dc sử dụng như 1 *namespace*cho thư viện đó. Tất cả các chức  năng cụ thể của libarary sẽ được tạo ra như những property của object đó.

var MyReallyCoolLibrary = {  
 awesome: "stuff",  
 doSomething: function() {  
// ...  
 },  
 doAnotherThing: function() {  
// ...  
 }  
};

Module management (Trình quản lý module)

Tùy chọn khác để tránh sự xung đột là hướng đến việc sử dụng bất kỳ trình quản lý module nào yêu thích. Tùy vào mục đích sử dụng mà sẽ dùng các gói dependency manager tùy thích.

Functions as Scopes

Cho đoạn code sau

var a = 2;  
function foo() { // <-- insert this  
 var a = 3;  
 console.log( a ); // 3  
} // <-- and this  
foo(); // <-- and this  
console.log( a ); // 2

Bình thường khi chúng ta chúng ta khai báo 1 function và phải invoke function đó thì khối code bên trong function mới đc thực thi. Nếu thật hay nếu ko cần invoke, xem đoạn code sau:

var a = 2;  
(function foo() { // <-- insert this  
 var a = 3;  
 console.log(a);  
})(); // <-- and this  
console.log(a);

Vậy có điều gì  khác biệt ở đây, toàn bộ function được bao bọc trong ( ), đây thật sự là 1 sự thay đổi quan trọng. Thay vì sự thực thi của function như 1 sự khai báo bình thường, thì function thực thi như 1 function - expression.

.