­­Direct dispatch, table dispatch and message dispatch

Mục đích của dispatch là để báo cho CPU vị trí của executable code trong memory.

1. **Direct Dispatch (static dispatch)**

Là loại dispatch **nhanh nhất**. Cho phép compiler xác định vị trí của instructions, tại **compile time**. Khi function được gọi hệ thống có thể trỏ ngay đến **memory address** của function đó.

Đặc điểm: nhanh nhưng hạn chế về flexibility, không thể override được

1. **Dynamic dispatch**

* Programe sẽ không biết nên áp dụng implementation nào cho đến **runtime**
* Flexibility cao, hỗ trợ OOP tốt
* Swift hỗ trợ 2 dynamic dispatch: **table dispatch** và **message dispatch**
  1. **Table dispatch**
* Class sẽ được liên kết với **virtual table** (vtable – xem phần SIL) bao gồm 1 array các function pointers, dùng để chỉ đến implementation tương ứng của class đó.
* Vtable được xây dựng ở compile time. Thus, there are only two additional instructions (read and jump) as compared to static dispatch. So the dispatch should be theoretically pretty fast.
  1. **Message dispatch**
* Trong IOS, message dispatch được cung cấp bởi Objectvie-C. Mỗi lần Objective-C method được gọi, the invocation is passed to **objc\_msgSend** which handles the look ups. Quá trình này bắt đầu từ class hiện tại và lặp lại theo hierarchy để xác định implementation.
* Không giống với Table dispatch, message passing dictionary có thể sửa đổi tại runtime, cho phép ta thay đổi hành vi của program.
* Trong 3 loại dispatch, message dispatch là dispatch dynamic nhất. Đánh đổi lại là performance sẽ chậm hơn.
* KVO is implemented using swizzling

Chú ý:

* Function của struct và các kiểu dữ liệu đều là static dispatch.
* Final function là static dispatch
* Các extension (không có final, dynamic, @objc) là direct dispatch. Mà direct dispatch thì không override được.

**extension Animal {**

**func eat() { }**

**@objc dynamic func getWild() { }**

**}**

**class Dog: Animal {**

**override func eat() { } *// Compiled error!***

**@objc dynamic override func getWild() { } *// Ok :)***

**}**

**protocol Noisy {**

**func makeNoise() -> Int *// TABLE***

**}**

**extension Noisy {**

**func makeNoise() -> Int { return 0 } *// TABLE***

**func isAnnoying() -> Bool { return true } *// STATIC***

**}**

**class Animal: Noisy {**

**func makeNoise() -> Int { return 1 } *// TABLE***

**func isAnnoying() -> Bool { return false } *// TABLE***

**@objc func sleep() { } *// Still TABLE***

**}**

**extension Animal {**

**func eat() { } *// STATIC***

**@objc func getWild() { } *// MESSAGE***

**}**

**let animal1 = Animal()**

**print(animal1.isAnnoying()) *// Value: false***

**let animal2: Noisy = Animal()**

**print(animal2.isAnnoying()) *// Value: true***

* Noisy.isAnnoying() và Animal.getWild() là static dispatch bởi vì nó là extensions.
* Noisy.makeNoise() là table dispatch kể cả khi nó có default implementation
* animal2.isAnnoying() sử dụng implementation của protocol extensions bởi vì nó là direct method

**let animal1 = Animal()**

**print(animal1.****makeNoise()) *// Value: 1***

**let animal2: Noisy = Animal()**

**print(animal2.makeNoise()) *// Value: 1***

* animal1.makeNoise() và animal2.makeNoise() cung cấp cùng 1 kết quả bởi vì nó đều là table dispatch
* @objc của sleep có nghĩa là nó sẽ visible với Objective-C (SIL sẽ build cả swift và Objective-C method cho nó). Nó không có nghĩa là program sẽ luôn chọn Objective-C method để execute mà nó sẽ ưu tiên chọn swift method hơn.

Độ ưu tiên của các dispatch với compiler:

1. Direct dispath
2. Nếu như cần override thì table dispatch sẽ được chọn
3. Nếu cần cả override và Objective-C thì sẽ chọn message dispatch

|  | **Direct** | | | **Table** | **Message** |
| --- | --- | --- | --- | --- | --- |
| **Explicitly enforced** | | final, static | — | | dynamic |
| **Value type** | | all methods | — | | — |
| **Protocols** | | extensions | initial declaration | | — |
| **Class** | | extensions | initial declaration | | extensions with @objc |

Property được khai báo **dynamic** khi được set giá trị sẽ tự động gọi các hàm **willChangeValue** và **didChangeValue**. 1 property không khai báo **dynamic** mà muốn **observe** ta có thể tự gọi hàm **willChangeValue** và **didChangeValue** khi thay đổi giá trị của property hoặc dùng hàm **setValue** để set giá trị mới cho property (do khi t)

Link:

* <https://www.rightpoint.com/rplabs/switch-method-dispatch-table>

* <https://trinhngocthuyen.github.io/posts/tech/method-dispatch-in-swift/>

* <https://medium.com/flawless-app-stories/static-vs-dynamic-dispatch-in-swift-a-decisive-choice-cece1e872d>