Method Dispatch

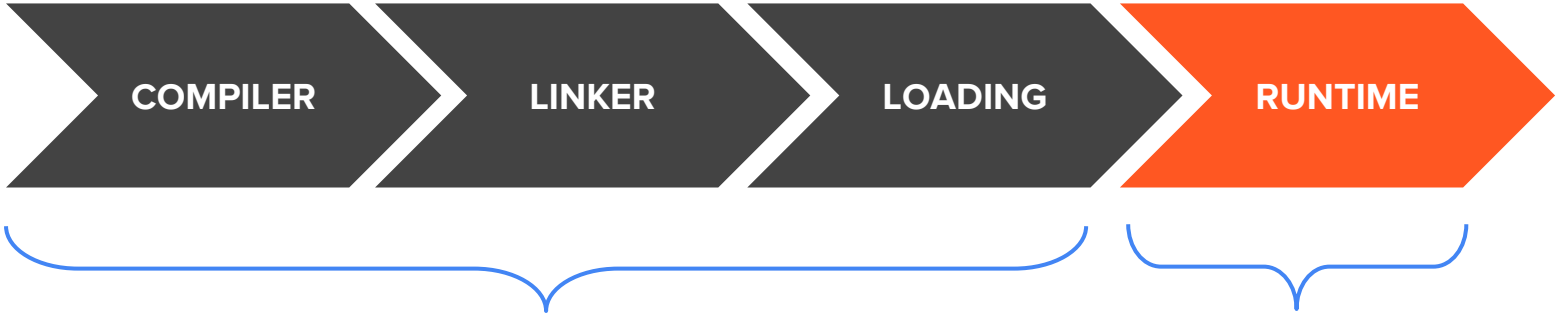
By
Gaurav Keshre

“Method dispatch is the algorithm used to decide which method should be invoked in response to a message.”

Putting things in perspective

Program Execution Pipeline

Execution Pipeline



- Static
- Early Binding
- Compile-time polymorphism

- Compiler's Favorite
- Efficient
- Cheaper*

- Dynamic
- Late Binding
- Run-time Polymorphism

- Developers' Favorite
- Flexible
- Little Expensive*

Demangling the age old Object Oriented Keywords

Binding vs Dispatch

Binding vs Dispatch

Binding is essentially anything (properties, functions) But dispatch is strictly about identifying method implementations

The idea in **dispatch** is following some function pointer to see which method to actually invoke, or object to invoke it on. "Binding" is the idea that the method is "bound" to a particular instance (or class of instances) & that's how you identify it.

Intro

Method Dispatch is how a program selects which instructions to execute when invoking a method. It’s something that happens every time a method is called, and not something that you tend to think a lot about.

- Static
- Table
- Message

Static Dispatch

- Fastest style of method dispatch.
- Result in the fewest number of assembly instructions.
- The compiler can perform all sorts of smart tricks, like inlining code, inserting proper GOTOs, etc

Table Dispatch

- Uses array of function pointers for each method in the class declaration.
- This array is generally referred to as **virtual table**, **dispatch table** or **vtable**
- Swift uses the term **witness table**.

Subclassing - Table dispatch

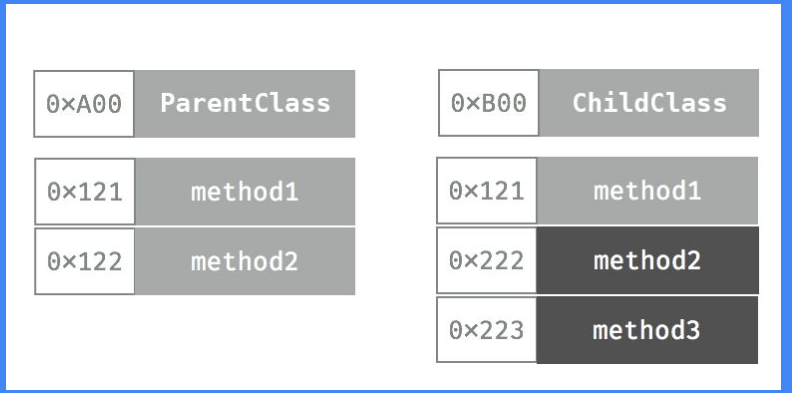
- Every subclass has its own copy of the table with a different function pointer for every method that the class has overridden.
- As subclasses add new methods to the class, those methods are appended to the end of this array.

Subclassing
Table dispatch

- Every subclass has its own copy of the table with a different function pointer for every method that the class has overridden.

- As subclasses add new methods to the class, those methods are appended to the end of this array.

```
class ParentClass {
    func method1() {}
    func method2() {}
}
class ChildClass: ParentClass {
    override func method2() {}
    func method3() {}
}
```



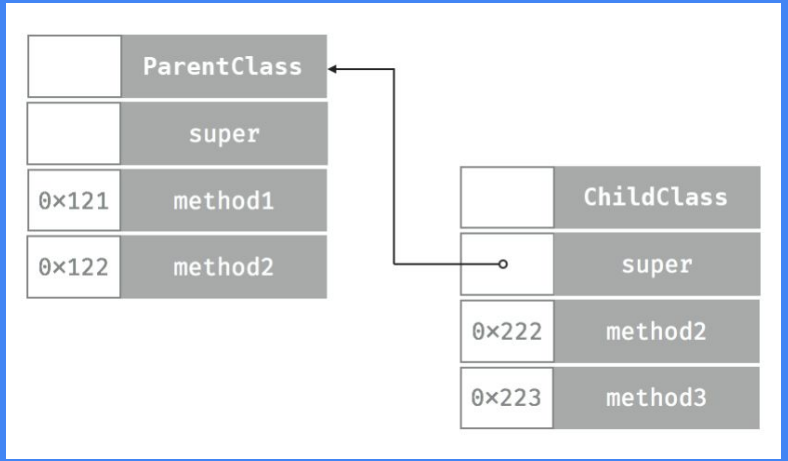
Message Dispatch

- Most dynamic method of invocation available.
- Powers the techniques like KVO, UIAppearance.
- Allows developers to modify the dispatch behavior at runtime (*via* swizzling)
- Slow to some extent.
- Objective-C Sports this dispatch all the time.

Subclassing
Table dispatch

- When a message is dispatched, the runtime will crawl the class hierarchy to determine which method to invoke.

```
class ParentClass {
    func method1() {}
    func method2() {}
}
class ChildClass: ParentClass {
    override func method2() {}
    func method3() {}
}
```



What's in for me?
To write Performant Code

We have just scratched the surface with these description. The motive was to bring everyone's attention towards the concept and give its due attention.

Candidates of improvement

Method dispatch usually takes place in the realm of following constructs.

- Classes
- Structs
- Enums
- Protocol
- Extensions (of all types)

Classes

- Table Dispatch by default
- Can be controlled by using suitable access specifiers or visibility control.
- Use private, fileprivate, final appropriately to restrict the dynamic dispatch when not needed.

Extension

- Static Dispatch
- This is cool.
- One of the primary reason we cannot have stored properties in extensions.

Struct & Enums

- Static Dispatch
- Still use `private`, `fileprivate`, `final` appropriately to write beautiful and reasonable code

Extension

- Static Dispatch

Protocols

- Dynamic Dispatch
-

Extension

- **Requirement Methods** are dynamically Dispatched

- **Non-Requirement Methods** are statically Dispatched
- Resolved from the type of the caller.

Visibility and Access specifiers

private, fileprivate and final

- **Static dispatch**
- Don’t hesitate to overuse it.

dynamic

- Makes it visible go Objc runtime AND
- Forced to use **ObjC Dynamic Table Dispatch**
-
- Use only when needed.
- Eg: If you want to override methods in extensions.

@objc

- Makes it visible go Objc runtime
- No Effect on dispatch
- Don’t hesitate to overuse it.

@nonobjc

- Makes a Swift declaration unavailable in Objective-C
- Enforces Swift

ZAP
SUBSCRIBE

Why pay for a car if you can subscribe and earn ?

https://www.zoomcar.com/zap/subscribe

Why pay for a dynamizm when you don’t need it?

Apple: Increasing Performance by Reducing Dynamic Dispatch

Thanks!

Contact us:

Email:
gauravkesh.re@gmail.com

Twitter:
@gauravkeshre

Github:
@gauravkeshre

Links

https://swiftunboxed.com/interop/objc-dynamic/

https://blog.untitledkingdom.com/objective-c-vs-swift-messages-dispatch-9d5b7fd58327

https://stackoverflow.com/a/41036133/751026

https://www.raizlabs.com/dev/2016/12/swift-method-dispatch/

https://medium.com/@vhart/protocols-generics-and-existential-containers-wait-what-e2e698262ab1