캐스팅에 대해서 알아보세요?

[사전용어]
형 변환은 프로그래밍에서 자료형을 다른 형태로 변경하는 것이다. 필요에 따라서 변경되는 암시적 형변환과 프로그래머가 직접 변경하는 명시적 형변환으로 크게 두 가지로 나눌 수 있다.

간단히 말해서 int -> String 으로 String -> int로 자료형을 변경할 수 있는 것을 의미하고, 프로그래밍적으로는 캐스팅이 굉장히 많이 쓰인다. 하지만 objective -c 에서 캐스팅을 찾아보니… 캐스팅에서 브릿지라는 개념이 있길래 브릿지에 대해서 한번 찾아보았다.

NSType VS CFType

iOS나 OSX 코딩을 좀 해 봤다면 알겠지만, NS로 시작되는 타입과 CF로 시작되는 타입이 있다.
NSType: Objective-C Type / CFType: C Type

NSType의 NS는 NextStep의 약자이다. 다르게 말하자면 Objective-C로 구성된 클래스 타입이라는 의미이다. NSType의 클래스라면 무조건 NSObject를 상속 받고 있다. 이 NSObject 내부에는 레퍼런스 카운트 기능이 구현되어 있다. 따라서 NSType의 리테인과 릴리즈는 메소드 호출 방식으로 이루어진다.

CFType의 CF는 CoreFoundation의 약자이다. CoreFoundation류에는 다수의 C API가 있는데, 결국 C 포인터 타입이라고 의미 할 수 있다. 즉, CFType은 C로 구현된 타입이다. 따라서 클래스 개념이 없다. 대신 C 함수들과 자료구조로 사용되는 구조체(struct) 타입들이 있다. 여기서 사용되는 이 구조체 타입(예를 들어 CFArray의 포인터인 CFArrayRef등등) 내부에 레퍼런스 카운트를 위한 정보가 저장된다고 볼 수 있다.

이 둘 사이에는 리테인과 릴리즈 방법이 좀 다르다.

NSType의 경우 비ARC 환경이라면 아래와 같이 메소드로 리테인과 릴리즈가 구성된다.

```
NSDate *date = [[NSDate alloc] init];  // Retain 발생
...
NSDate *obj = [date retain];
...
[obj release];
...
[date release];
```

이런 식으로 오브젝트 인스턴스 자체에다 retain과 release를 호출한다. 앞서 이야기 한 대로 NSObject에 레퍼런스 카운트 기능이 구현되어 있기 때문이다. 반대로 CF 타입의 경우는 클래스가 없고 대신 자료구조(구조체)와 함수가 분리되어 있다. 그래서 함수를 이용해 리테인과 릴리즈를 해야 한다.

```
CFArrayRef array = CFArrayCreate(...);  // CFRetain 발생
...
CFArrayRef somePtr = CFRetain(array);
...
CFRelease(somePtr);
...
CFRelease(array);
```

위에서 CFRetain() 함수와 CFRelease() 함수가 바로 리테인과 릴리즈를 하는 기능을 제공한다.

자동 브릿지 타입

NSType과 CFType사이에는 비슷한 타입이 다수 존재한다. 예를 들어 NSArray와 CFArrayRef, NSDictionary와 CFDictionaryRef 등등 말이다. 이런 '연관된 두 타입들'은 서로간에 자동으로 브릿지 타입이 있는 타입이다. 쉽게 말해서 타입 캐스팅(Type Casting)을 이용해 서로간의 타입을 왔다 갔다 할 수 있다. 이런 말을 자동으로 브릿지 되어 있는 타입(Automatically Bridged Type)이라고 부른다. 그런데 ARC환경이 되면서 문제가 생겼다. ARC가 포인터의 원래 타입을 모른다는 점이다. 형변환을 했다면 둘 중 하나에 리테인을 거는 등의 처리를 해야 하는데 어디다 해야 할 지 ARC가 혼란해 할 수 밖에 없다.

아래 코드는 그냥 정적으로 타입 캐스팅을 하는 문제가 있는 예이다.

```
CFArrayRef cfArray = CFArrayCreate(...);
NSArray *array = (NSArray *)cfArray;
```

위 예제 두 번째 라인에서 형변환을 하고 있는데 이 코드에서 경고가 발생한다. Xcode에서는 친절하게 브릿지를 넣으라고 알려 줄 것이다.

브릿지 형변환

이제 형변환(Type Casting) 시의 브릿지 방법에 대해 살펴보자. 이런 식으로 쓴다.

```
(BRIDGE TYPE) POINTER
```

바로 와닿지 않은 엉망진창인 표현이다. 그냥 형변환 타입 이름 앞에다 그냥 브릿지 종류 이름을 써 주면 된다. 쓰는 방법을 알기에는 실제 예제가 차라리 좋을 것이니 계속 읽어보자.

브릿지의 경우 아래 3가지가 존재한다:

1) __bridge
2) __bridge_retained (혹은 CFBridgingRetain())
3) __bridge_transfer (혹은 CFBridgingRelease())

일반적인 경우 Objective-C 위주로 코드를 작성할 테니 일반 브릿지(__bridge)가 가장 흔하게 사용된다.
이 세 가지는 분명 브릿지로써 ARC에 리테인과 릴리즈 규칙을 알려주기 위함이지만, 메모리 관리 차원에서 다른 규칙을 가지고 있다. 각각을 살펴보자.

일반적인 브릿지: __bridge

```
(__bridge T) op
```

op를 T타입으로 브릿지 캐스팅을 할 때 사용하는 키워드가 바로 __bridge 이다. 이렇게 사용하기 위해서는 op와 T 사이는 상대 타입이어야 한다. 즉 T가 NSType이면 op는 CFType이라는 말이다. 물론 반대의 경우도 된다. 그렇다면 둘 다 같은 타입이면 안된다는 것도 알 수 있을 것이다.

이 경우 ARC는 op에 별도의 리테인이나 릴리즈를 걸지 않게 된다. 이 말이 뭔가 위험해 보이는가? 그렇다면 아래 코드를 보자.
NSArray *array = (__bridge NSArray *)cfArray;
이 코드에서 __bridge 의 지침대로 ARC는 cfArray 자체에는 어떠한 릴리즈나 리테인을 걸지 않는다. 하지만 그 앞의 array 에 대입하는 코드를 생각해 보자. ARC는 여기에서 리테인을 걸 것이다.

결국 위 예는 일반적인 Objective-C코드로 ARC 하에서 코딩하는 것 처럼 동작하게 된다.

중요한 것을 잊지 말자. CFType에는 별도의 릴리즈 함수가 존재한다는 점을.

```
CFArrayRef cfArray = CFArrayCreate(...);
NSArray *array = (__bridge NSArray *)cfArray;
CFRelease(cfArray); // Create에서 발생한 리테인의 짝
```

혹시나 해서 이야기 하는데, CoreFoundation에서 사용되는 함수 들 중 Create 혹은 Copy라는 이름이 붙은 함수들은 메모리를 할당(Allocation)한 후 내부에서 자동으로 리테인(CFRetain)을 건다. 따라서 이와 짝을 맞추어 릴리즈(CFRelease)를 해 주어야 한다.

이렇게 코딩이 되어 있으면 나머진 NSType인 array가 릴리즈 되어서 사라질 때가 되면 위 cfArray도 레퍼런스 카운트가 0이 되어 함께 사라질 것이다.

물론 반대의 경우도 살펴봐야 할 것이다.
```
NSArray *array = [NSArray array];
CFArrayRef cfArray = (__bridge CFArrayRef)array;
```
NSType인 array는 ARC에 의해 릴리즈가 되므로 좀 더 편해진다.

CFType으로 바꿀 때: __bridge_retained

앞의 일반적인 브릿지에 대해 이해했다면 이제부터 설명이 조금 쉬워질 것 같다. 이제 __bridge_retained 에 대해 살펴보자.

```
(__bridge_retained T) op
```

__bridge_retained 는 NSType 포인터(op)를 CFType 포인터(T)로 바꿀 때 사용한다. 아래 예를 보자.

```
NSArray *array = [[NSArray alloc] init];
CFArrayRef cfArray = (__bridge_retained CFArrayRef)array;
```

혹은 위 코드는 아래 처럼 쓸 수도 있다.

```
NSArray *array = [[NSArray alloc] init];
CFArrayRef cfArray = (CFArrayRef)CFBridgingRetain(array);
```

이 코드는 일반적인 브릿지를 쓴다고 가정하면 아래와 같이 풀어 쓸 수 있다.

```
NSArray *array = [[NSArray alloc] init];
CFArrayRef cfArray = (__bridge CFArrayRef)array;
CFRetain(cfArray);
```

즉 __bridge_retained 는 NSType을 CFType으로 변환하면서 CFType 쪽으로 리테인을 걸어주게 만들어 준다. 따라서 원래의 NSType이 릴리즈 되어도 CFType 쪽으로 레퍼런스 카운트가 하나 더 있기 때문에 CFType이 살아있는 동안은 메모리가 해제되지 않는다. 물론 이 경우 사용이 끝나면 CFRelease()로 릴리즈 해 주는 것을 잊어서는 안된다.

NSType으로 바꿀 때: __bridge_transfer

이제 반대로 CFType 포인터를 NSType 포인터로 바꿀 때만 쓸 수 있는 __bridge_transfer 에 대해 살펴보자.

```
(__bridge_transfer T) op
```

이 코드는 CFType 포인터인 op를 NSType 인 T 타입으로 캐스팅 할 때 사용한다. 아래는 예제이다.

```
CFArrayRef cfArray = CFArrayCreate(...);
NSArray *array = (__bridge_transfer NSArray *)cfArray;
```

아래도 위 예와 동일한 예제이다.
```
CFArrayRef cfArray = CFArrayCreate(...);
NSArray *array = (NSArray *)CFBridgingRelease(cfArray);
```

이 경우는 자동으로 릴리즈를 추가하는 방식으로 동작한다는 점이 다르다. __bridge_retained 를 일반 브릿지를 이용해 아래와 같은 식으로 바꾸어 설명 가능하다.

```
CFArrayRef cfArray = CFArrayCreate(...);
NSArray *array = (__bridge NSArray *)cfArray;
CFRelease(cfArray);
```

아주 안정적인 모습이 되는 것을 볼 수 있다. 이제 NSArray 타입 인스턴스인 array는 ARC 제어 하에서 갑작스런 메모리 해제가 발생하지 않게 된다.

정리

NSType과 CFType 사이에는 형변환(Type Casting)이 가능하다. 물론 이 타입들 간에는 형변환이 가능한 타입이 미리 정의되어 있다. 그리고 이 둘 사이를 형변환 할 때 ARC에서 어떤 식으로 처리할 것인지 알려줘야 하며 이 알려주는 역활을 브릿지가 담당한다. 사실은 __bridge 만 써도 된다. 라인 수를 하나라도 줄이고 싶다면 다른 것을 써도 되겠지만, 어떻게 보면 __bridge 는 CFType 쪽 코드를 투명하게 보여주기 때문에 오히려 문제 발생 가능성을 줄일 수도 있기 때문이다.

****

나의 정리.
캐스팅을 알아보다가 언어간에 캐스팅과 관련된 브릿지를 알게 되었다. 사실 실제 코딩에서 어떻게 쓰일지는 한번 해봐야 알 것 같고, 스위프트로 넘어가서는 쓰이는 일이 많이 없다고는 하지만… 그래도 캐스팅을 통해서 언어간에 캐스팅 또한 가능하게 구현해 둔 objective -c 가 조금 대단한 것 같다. 이런 일에 캐스팅이 쓰이는 것 같다. ㅎ