[TS] 5. Generics in TypeScript

by jbee

`Generics`는 자바스크립트 개발자에게 친숙하지 않은 용어일꺼라고 생각됩니다. 하지만 정적 타이핑에 있어서 큰 부분을 차지하고 있는 Generics syntax에 대해 알아봅니다.

Contents

Generics?

이전에 다뤘던 인터페이스가 어떠한 '틀'을 지정하여 함수 또는 클래스를 정의했다면, 제네릭을 통하여 함수 또는 클래스에 '틀'을 '주입'하여 그 확장성을 보다 높일 수 있습니다. 즉, 제네릭의 목적은 재사용성(reusable)을 높이고 함수나 클래스의 확장성을 높여 중복된 코드를 방지하기 위함이라고 할 수 있습니다.

Generics to Class

어떠한 장바구니(Basket)에 물건(Item)을 넣어야 한다고 가정을 해봅시다. 이때, 하나의 장바구니에는 모두 동일한 타입의 물건이 들어있어야 합니다. 이 경우 모든 경우의 수에 해당하는 장바구니를 각각의 클래스로 만들어줘야 하는 문제점이 발생합니다.

```
class BookBasket { ... }
class CupBasket { ... }
class DollBasket { ... }
//...
```

이럴 경우, 제네릭을 사용하여 하나의 클래스에서 타입을 **주입받아** 모든 경우의 수에 해당하는 장바구니를 만들 수 있습니다.

```
class Basket<T> {
  private item: T[];

  getItem(index: number): T {
    return this.item[index];
  }
}
```

아주 간단한 `Basket` 클래스를 만들어 보았습니다. 일반 클래스를 정의할 때와는 다르게 `<T>` 라는 것이 클래스 이름 옆에 추가된 것을 확인하실 수 있는데요, 해당 클래스 내에서 사용할 타입을 `T` 라는 값으로 받을 수 있게 되는 것입니다. 여기서 `T` 는 별 뜻이 있는게 아니라 Type의 약자입니다. 이렇게 정의한 클래스를 이용하여 여러 경우에 해당하는 장바구니를 생성할 수 있습니다.

```
const bookBasket = new Basket<Book>();
const cupBasket = new Basket<Cup>();
const dollBasket = new Basket<Doll>();
```

`new` 키워드를 통해 인스턴스를 생성할 때 정의할 때와 마찬가지로 `<>` 와 함께 타입을 지정하여 인스턴스를 생성해줍니다. (`Book` , `Cup` , `Doll` 에 대한 인터페이스 정의는 생략합니다.)

Generics to Function

제네릭스는 클래스 뿐만 아니라 함수에도 적용될 수 있는데요, 이전에 살펴보았던 AjaxUtils를 예제로 제네릭에 대해 알아보겠습니다.

```
export async function fetchData(param: fetchDataParam): P
    //...
}
```

위 함수는 `fetchDataParam` 이라는 타입의 인자만 받을 수 있으며 `Promise<DataFormat>` 이라는 타입의 반환만 할 수 있게 설계되어 있습니다. 그러나 함수의 body가 여러 곳에서 중복된다면 `fetchDataParam` 또는 `DataFormat` 을 위 함수에 주입하여 여러 상황에 대응할 수 있는 함수로 변환시킬 수 있습니다. 이 때, 제네릭이 사용됩니다.

위 함수에 제네릭을 적용하여 함수의 재사용성을 높여보겠습니다.

```
async function fetchDataOf<T, U>(param: T): Promise<U> {
    const response = await fetch(`${param}`);
    return response.json();
}
```

arrow function에도 마찬가지로 적용할 수 있습니다.

```
const fetchDataOf = async <T, U>(param: T): Promise<U> =>
    const response = await fetch(`${param}`);
    return response.json();
}
```

이렇게 정의를 하면 다음과 같이 호출할 수 있습니다. 여기서 T와 U는 그냥 type의 종류를 나타내는 character라고 생각하시면 됩니다.

```
const data: DataFormat = await fetchDataOf<string, DataFo
```

`param` 으로 넘겨주게 되는 인자의 타입이 다르거나, 반환하게 되는 값의 타입이 다를 경우에도 `<>` 에 해당 타입을 지정하여 하나의 함수를 사용할 수 있게 됩니다.

물론 의 형식으로도 사용할 수 있지만(이럴바에 차라리 자바스크립트를 사용하는 것이 생산성 측면에서 더 좋을 것 같습니다.) 특정 타입을 받아 해당 타입으로 반환한다는 측면에 있어서 안정성과 확장성 두 마리 토끼를 모두 잡을 수 있습니다.

마무리

실제 개발해서보다는 라이브러리를 개발할 때 많이 사용할 수 있는 Generics 에 대해 알아봤습니다. 해당 포스팅 외 다른 타입스크립트 포스팅은 여기에서 보실 수 있으며 예제에 사용된 코드는 여기에서 확인하실 수 있습니다.
감사합니다.

5. Generic in TypeScript end