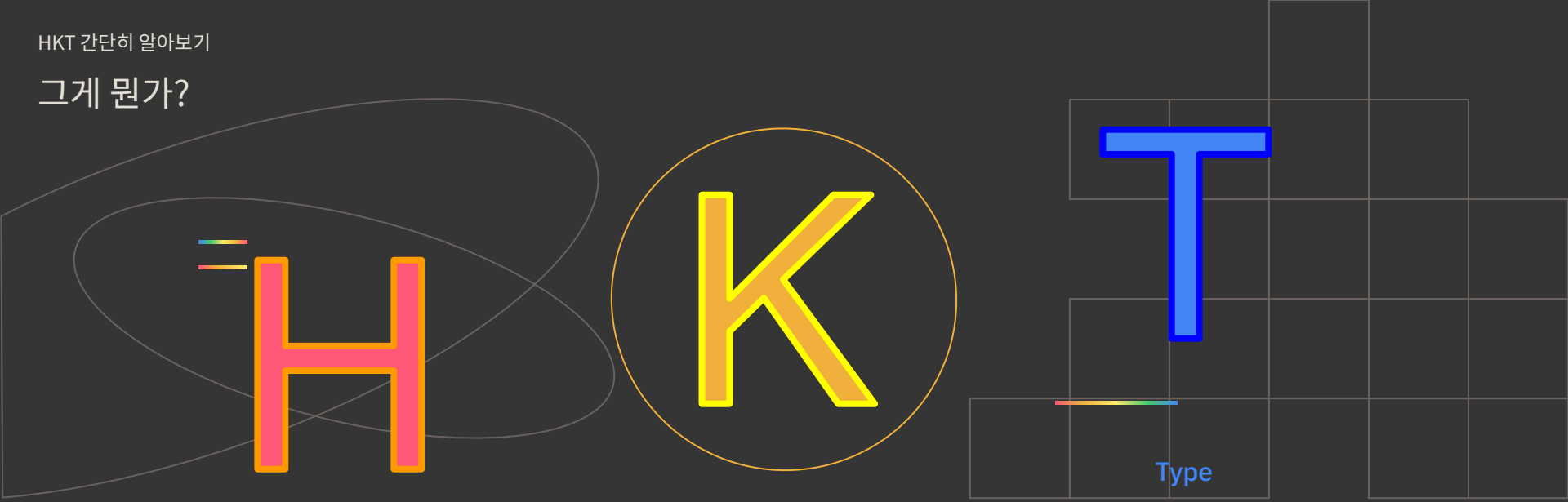
HKT
간단히 알아보기

TypeScript Backend Meetup 2025/10

Contents

그게 뭔가?



Higher

고차

고차함수처럼,
타입 생성자를 받는 타입 생성자나
타입을 받는 타입 생성자 생성자
타입 생성자를 받는 타입 생성자 생성자
등…

Kinded

Kind = 카인드

타입의 종류

Array는 * -> * (타입 생성자)
Array<number>는 * (구체적 타입)

타입

값의 종류

예를 들어 42는 number
“42”는 string

Kind

Higher **Kind**ed Type의 Kind
타입의 종류

*	* -> *	(*, *) -> * * -> * -> *	(* -> *) -> *
구체적인 타입	타입 생성자	인자가 두 개인 타입 생성자, 커링	고차 타입
string number Array<string> 등…	Array Promise 등… Array<T>가 아니라 Array 자체임에 유의!	Map, Record 등 ?	????????????????

Higher?

Higher Order Function
고차 함수

number

구체적인 값

“42”
42
[“42”, “ft”]

등…

(a: number) => number

함수

(a: number) => a + 2
(a: number) => a * 2

등…

(a: A, b: B) => C

인자가 여러개인 함수

(a: A, b: B) => a[b]

등…

(a: A) => (b: B) => C

고차 함수!

함수를 인자로 받는 함수
함수를 리턴하는 함수
= 고차함수

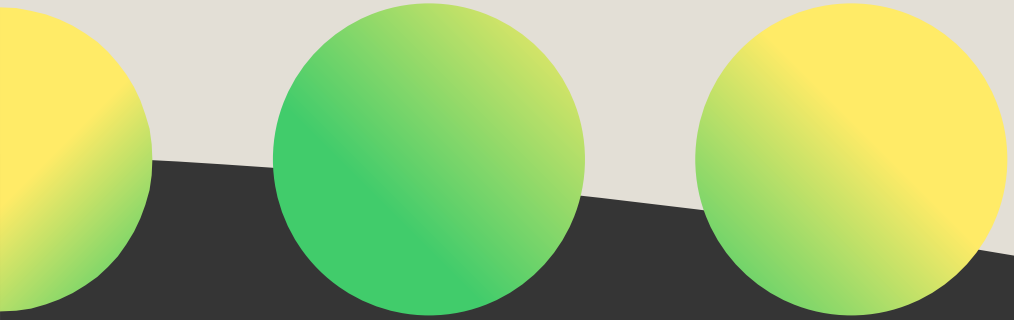
HKT?

Higher Kinded Type

= Type with Higher Kind

= 고차 kind인 타입

= 타입인데, 그 kind가 고차



타입스크립트는 HKT를 직접 지원하지 않습니다.

HKT를 “직접” 지원하지 않습니다.

```ts
interface HKT {
    input: unknown;
    output: unknown;
}

interface ToArrayHKT extends HKT {
    output: Array<this['input']>;
}

type Transform<T, K extends HKT> = ({ input: T } & K)['output'];

type Result = Transform<42, ToArrayHKT>;
//   ^? type Result = 42[]
```

```ts
interface HKT {
    input: unknown;
    output: unknown;
}

interface ToArrayHKT extends HKT {
    output: Array<this['input']>;
}

type Transform<T, K extends HKT> = ({ input: T } & K)['output'];

type Result = Transform<42, ToArrayHKT>;
//    ^? type Result = 42[]
```

어떻게 쓰나?


Step 1
Step 2
Step 3
Step 4
Goal
interface 정의
제네릭 인자 대신
다른 필드로 받고,
결과 타입도 필드로
그 interface 상속
결과 타입이
this를 활용해
다른 필드의 타입으로
계산되도록
상속된 interface에
intersection으로
제네릭 인자를 대신할
다른 필드의 타입을 주입
그 intersection에서
결과물 타입을
나타내는 필드에서
타입을 꺼내면…!
HKT 흉내내기 성공!

HKT 인코딩을 위한 interface 만들기

```
interface MyHKT {
    a: unknown;
    b: unknown;
    c: string;
    output: number;
}
// type MyHKTImpl<A, B, C extends string> = ...
```

HKT 인코딩하기

```
8   // type MyHKTImpl<A, B, C extends string> = ...
9
10  interface MyHKTImpl extends MyHKT {
11      output: MyHKTImplImpl<this['a'], this['b'], this['c']>;
12  }
13
```

```ts
type CallMyHKT<HKT extends MyHKT, A, B, C extends string>
    = ({ a: A, b: B, c: C } & HKT)['output'];


type X = CallMyHKT<MyHKTImpl, number, 42, "ft">;
//   ^? type X = 1 & { __extra: "ft"; }
//      = MyHKTImplImpl<number, 42, "ft">

type Y = CallMyHKT<MyHKTImpl, "42", string, "test">;
//   ^? type Y = 1 & { __extra: "test"; }
//      = MyHKTImplImpl<"42", string, "test">

type Z = CallMyHKT<MyHKTImpl, boolean, number, string>;
//   ^? type Z = 0 & { __extra: string; }
//      = MyHKTImplImpl<boolean, number, string>;
```

그래서 그게 도대체 왜 필요한가?

1. 함수형 프로그래밍

Functor, Monad, map, flatMap이나
composeK, chain, sequence, traverse 등등…

타입마다 비슷한 타입을 매번 만들지 않으려면 필요

2. type-safe eDSL 커스텀 연산

예를 들면…

type-safe eDSL validation이 있는 경우
타입마다 validator를 별도로 만들어야 함

커스텀 타입 지원을 추가하려면
커스텀 타입의 validator를 인자로 받아야 함

그런데 여기서 validator가 타입 생성자

```ts

type Merge2<A extends object, B extends object> = A & B;
type Merge<T extends object[], R extends object = {}> =
    0 extends T['length'] ? R :
    T extends [infer First extends object, ...(infer Rest extends object[])]
        ? Merge<Rest, Merge2<R, First>>
        : R

type Join2<A extends string, B extends string> = `${A}, ${B}`;
type Join<T extends string[], R extends string = ""> =
    0 extends T['length'] ? R :
    T extends [infer First extends string, ...(infer Rest extends string[])]
        ? Join<Rest, Join2<R, First>>
        : R


// 이상하다...? 중복이 많은 것 같은데?

```

```ts
interface FoldHKT<T> {
  acc: T;
  curr: T;
  type: T;
}
type ArrayFold<T, V extends T[], HKT extends FoldHKT<T>, Init extends T>
  = V extends [infer F extends T, ...infer R extends T[]]
    ? ArrayFold<T, R, HKT, ({ acc: Init, curr: F } & HKT)['type']>
    : Init;

interface Merge2K extends FoldHKT<object> {
  type: this['acc'] & this['curr'];
}
interface Join2K<Sep extends string> extends FoldHKT<string> {
  type: `${this['acc']}${Sep}${this['curr']}`;
}

```

```ts

type Merge<T extends object[]> = // prettify
  ArrayFold<object, T, Merge2K, {}> extends
  infer I ? { [K in keyof I]: I[K] } : never;

type ObjList = [{ a: 1 }, { b: 2 }, { c: 3 }];
type Merged = Merge<ObjList>;
//   ^? type Merged = { a: 1; b: 2; c: 3; }

type Join<T extends string[], Sep extends string = ', '> =
  ArrayFold<string, T, Join2K<Sep>, ''> extends
  `${Sep}${infer I}` ? I : ""; // remove leading sep

type Names = ['x', 'y', 'z'];
type Joined = Join<Names>;
//   ^? type Joined = "x, y, z"

```

```ts
type TypeMap = {
  "INT": number;
  "STRING": string;
};

type FunctionMap = {
  intToString: {
    inputAutocomplete: ["INT"];
    outputHKT: IntToStringGetOutputHKT;
  }
};

interface IntToStringGetOutputHKT extends GetOutputHKT {
  output: this['input'] extends ["INT"] ? Ok<"STRING"> : Err<`intToString accepts [INT], got [${Joir
}

type Fail = CallFunctionResult<TypeMap, FunctionMap, "intToString", []>;
//   ^? type Fail = Err<"intToString accepts [INT],…

type Pass = CallFunctionResult<TypeMap, FunctionMap, "intToString", ["INT"]>;
//   ^? type Pass = Ok<"STRING">
```

주의사항

```ts
type Test = number[] & [1, 2, 3];


type Length = Test['length'];
//   ^? type Length = 3

type X = Test[0];
//   ^? type X = 1


type First = Test extends
//   ^? type First = 3 | 1 | 2
        [infer I, ...infer _] ? I : never;

type Rest = Test extends
//   ^? type Rest = unknown[]
        [infer _, ...infer I] ? I : never;


type What = Test extends
//   ^? type What = 1
        [infer I, ...unknown[]] ? I : never;

```

(배열 & 튜플) + infer = ???

```ts
export interface Transformer<From, To> {
    from: From;
    to: To;
}

export interface Join
    extends Transformer<number[], string> {
    to: JoinRaw<this['from']>;
}

export type JoinRaw<
    T extends number[],
    Acc extends string = never
> = T extends [
        infer First extends number,
        ...infer Rest extends number[]
    ] ? JoinRaw<
        Rest,
        [Acc] extends [never]
            ? `${First}`
            : `${Acc}, ${First}`
    > : [Acc] extends [never] ? "" : Acc;

type Result = (Join & { from: [1, 2, 3] })['to'];
//   ^? type Result = "1" | "3" | "2"

```

결과가 이상해질 수 있음

```ts
export type Pipe2<
    T,
    From extends T,
    X extends Transformer<From, unknown>,
    Y extends Transformer<X["to"], unknown>
> = Transform<
    X["to"],
    Y["to"],
    Y,
    Transform<T, X["to"], X, From>
>;

type Test1 = Pipe2<
    string,
    "Hello",
    ToTupleLengthOf<string, 3>,
//  Type 'ToTupleLengthOf<string, 3>' does not satisfy th
    ArrayMap<string, string, DoubleString>
>;
type Test2 = Pipe2<
    string,
    "Hello",
    ToTupleLengthOf<"Hello", 3>,
    ArrayMap<string, string, DoubleString>
//  Type 'ArrayMap<string, string, DoubleString>' does n
>;

```

Pipe, Compose 등
유틸을 못 쓸 수도 있음

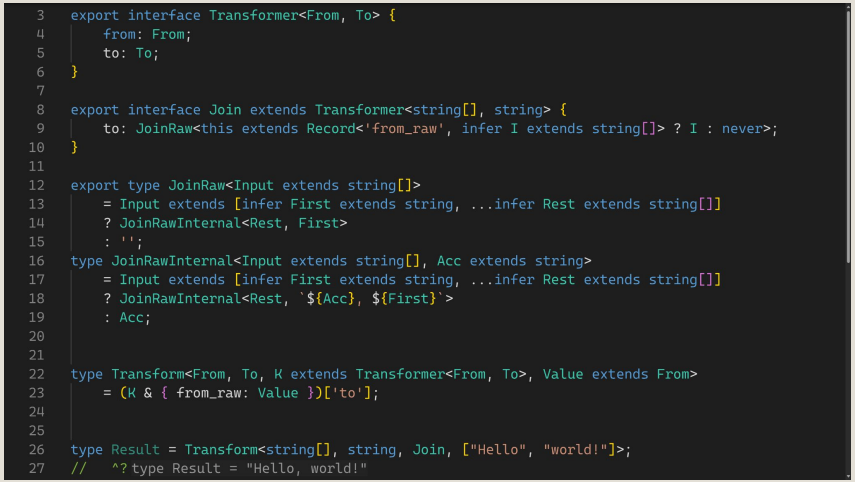
TSBM 2025/10

```ts

export type InvokeHKT<
  OutputFieldName extends string,
  OutputType,
  KindBase extends Record<OutputFieldName, OutputType>,
  Kind extends KindBase,
  Input extends Omit<KindBase, OutputFieldName>
> = (Kind & Input)[OutputFieldName];

export interface Transformer<From, To> {
    from: From;
    to: To;
}

export interface DoubleString extends Transformer<string, string> {
    to: `${this["from"]}${this["from"]}`;
}

export type Transform<
    From,
    To,
    K extends Transformer<From, To>,
    Value extends From
> = InvokeHKT<"to", To, Transformer<From, To>, K, { from: Value }>;

type Test4 = Transform<string, string, DoubleString, Transform<string, string, DoubleString, "test">>;
//   ^? type Test4 = "testtesttesttest"


```

```ts
export type Transform<
    From,
    To,
    K extends Transformer<From, To>,
    Value extends From
> = InvokeHKT<"to", To, Transformer<From, To>, K, { from: Value }>;

export interface ArrayMap<From, To, K extends Transformer<From, To>> extends Transformer<From[], To[]> {
    to: ArrayMapRaw<From, To, K, this['from']>;
}

export type ArrayMapRaw<From, To, K extends Transformer<From, To>, Arr extends From[], Acc extends To[] = []>
    = Arr extends [infer First extends From, ...infer Rest extends From[]]
    ? ArrayMapRaw<From, To, K, Rest, [...Acc, Transform<From, To, K, First>]>
    : Acc;


type Test = ["A", "B"];


type Ordinary = ArrayMapRaw<string, string, DoubleString, Test>;
//   ^? type Ordinary = ["AA", "BB"]

type Strange = Transform<string[], string[], ArrayMap<string, string, DoubleString>, Test>;
//   ^? type Strange = ["AA" | "BB" | "AB" | "BA"]


type Why = ArrayMapRaw<string, string, DoubleString, string[] & Test>;
//   ^? type Why = ["AA" | "BB" | "AB" | "BA"]

```

```ts
export type Transform<
    From,
    To,
    K extends Transformer<From, To>,
    Value extends From
> = InvokeHKT<"to", To, Transformer<From, To>, K, { from: Value }>;

export interface ArrayMap<From, To, K extends Transformer<From, To>> extends Transformer<From[], To[]> {
    to: ArrayMapRaw<From, To, K, this['from']>;
}

export type ArrayMapRaw<From, To, K extends Transformer<From, To>, Arr extends From[], Acc extends To[] = []> =
    Arr extends [infer First extends From, ...infer Rest extends From[]]
        ? ArrayMapRaw<From, To, K, Rest, [...Acc, Transform<From, To, K, First>]>
        : Acc;

export interface ToTupleLengthOf<T, Length extends number> extends Transformer<T, T[]> {
    to: ToTupleLengthRaw<T, Length, this['from']>;
}

export type ToTupleLengthRaw<T, Length extends number, V extends T, Acc extends T[] = []> =
    Acc['length'] extends Length ? Acc : ToTupleLengthRaw<T, Length, V, [...Acc, V]>;

export type Pipe2<T, From extends T, X extends Transformer<T, unknown>, Y extends Transformer<X['to'], unknown>> =
    Transform<X['to'], Y['to'], Y, Transform<T, X['to'], X, From>>;

type Test = Pipe2<string, "Hello", ToTupleLengthOf<string, 3>, ArrayMap<string, string, DoubleString>>;
// Type 'ArrayMap<string, string, DoubleString>' does not satisfy the constraint 'Transformer<[string, string, strin
```

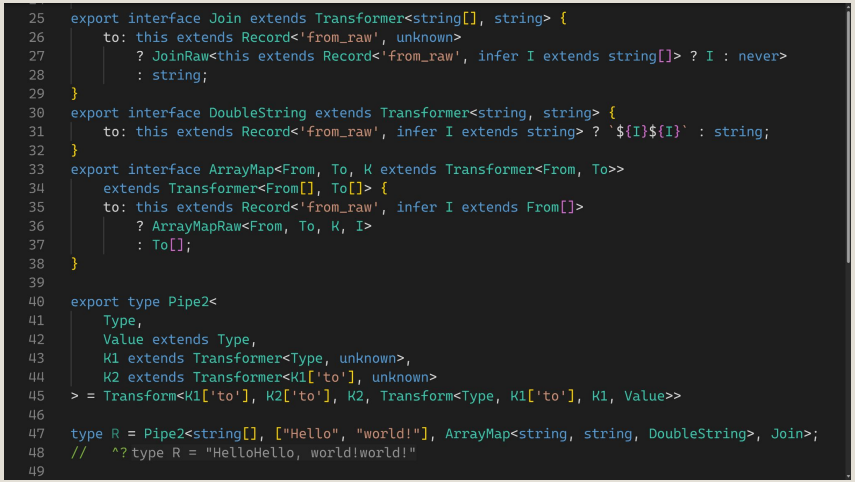
해결 방법



```ts
export interface Transformer<From, To> {
    from: From;
    to: To;
}

export interface Join extends Transformer<string[], string> {
    to: JoinRaw<this extends Record<'from_raw', infer I extends string[]> ? I : never>;
}

export type JoinRaw<Input extends string[]>
    = Input extends [infer First extends string, ...infer Rest extends string[]]
    ? JoinRawInternal<Rest, First>
    : '';
type JoinRawInternal<Input extends string[], Acc extends string>
    = Input extends [infer First extends string, ...infer Rest extends string[]]
    ? JoinRawInternal<Rest, `${Acc}, ${First}`>
    : Acc;


type Transform<From, To, K extends Transformer<From, To>, Value extends From>
    = (K & { from_raw: Value })['to'];


type Result = Transform<string[], string, Join, ["Hello", "world!"]>;
//   ^? type Result = "Hello, world!"
```

input은 타입 검사용, 실제 타입은 input_raw에

-> (배열 & 튜플) infer 문제로 인한 이상한 결과 방지



```ts
export interface Join extends Transformer<string[], string> {
    to: this extends Record<'from_raw', unknown>
        ? JoinRaw<this extends Record<'from_raw', infer I extends string[]> ? I : never>
        : string;
}
export interface DoubleString extends Transformer<string, string> {
    to: this extends Record<'from_raw', infer I extends string> ? `${I}${I}` : string;
}
export interface ArrayMap<From, To, K extends Transformer<From, To>>
    extends Transformer<From[], To[]> {
    to: this extends Record<'from_raw', infer I extends From[]>
        ? ArrayMapRaw<From, To, K, I>
        : To[];
}

export type Pipe2<
    Type,
    Value extends Type,
    K1 extends Transformer<Type, unknown>,
    K2 extends Transformer<K1['to'], unknown>
> = Transform<K1['to'], K2['to'], K2, Transform<Type, K1['to'], K1, Value>>

type R = Pipe2<string[], ["Hello", "world!"], ArrayMap<string, string, DoubleString>, Join>;
//   ^? type R = "HelloHello, world!world!"
```

input_raw가 없으면 output을 그대로

호출할 때에만 output 계산 (호출 시에만 input_raw 있음)
그 외에는 output을 타입 검사용으로 그대로 두기

-> Pipe, Compose같은 유틸 사용 가능

결론

고급 타입을 다루다보면 HKT가 필요할 때가 있는데,

간단합니다! 꼭 써 보세요!

참고

함수형 프로그래밍이나 HKT에 대해 더 깊게 알아보고 싶으시다면…?
참고로 보면 좋을 라이브러리들!

fp-ts

함수형 프로그래밍을 위한 라이브러리
타입 클래스 기반 고급 추상화 제공

effect

fp-ts를 확장한 프레임워크
비동기 처리나 자원 관리 등을 다루기 좋음

hotscript

HKT를 활용한 다양한 연산 등을 제공

hkt-core

HKT의 핵심 기능을 간단하게 구현

~~hktu~~

~~제가 만든 라이브러리~~

~~(이름 미정)~~

~~TSBM Studio 2기에서 만들 라이브러리~~
~~많관부~~

감사합니다

다음 TSBM Studio 2기에서 HKT를 활용하는
재밌는 프로젝트를 진행할 예정입니다

많관부!

질의응답