```
// Maybe => Maybe[Maybe]
const f = a => Just(a.chain(v => Just(v + 2)));
// Maybe => Maybe[Maybe]
const g = a => Just(a.chain(v => Just(v * 2)));
// Maybe[Maybe] ==f==> Maybe[Maybe] ==g==> Maybe[Maybe] ==> Maybe
Maybe(Maybe(3)).chain(f).chain(g)
Maybe(Maybe(3)).chain(x => f(x).chain(g))
```

套一层娃

Maybe

Promise

```
// Promise => Promise[Promise]
const f = a => Promise.resolve(a.then(v => Promise.resolve(v + 2)));
// Promise => Promise[Promise]
const g = a => Promise.resolve(a.then(v => Promise.resolve(v * 2)));
// Promise[Promise] ==f==> Promise[Promise] ==g==> Promise[Promise]
Promise.resolve(Promise.resolve(3)).then(f).then(g)
Promise.resolve(Promise.resolve(3)).then(x => f(x).then(g))
```

正常工作且结果等价 

a.then not defined 

套一层娃

Maybe

```
// Maybe => Maybe[Maybe]
const f = a => Just(a.chain(v => Just(v + 2)));
// Maybe => Maybe[Maybe]
const g = a => Just(a.chain(v => Just(v * 2)));
// Maybe[Maybe] ==f==> Maybe[Maybe] ==g==> Maybe[Maybe] ==> Maybe
Maybe(Maybe(3)).chain(f).chain(g)
Maybe(Maybe(3)).chain(x => f(x).chain(g))
```

正常工作且结果等价 ✓

Promise

```
// Promise => Promise[Promise]
const f = a => Promise.resolve(a.then(v => Promise.resolve(v + 2)));
// Promise => Promise[Promise]
const g = a => Promise.resolve(a.then(v => Promise.resolve(v * 2)));
// Promise[Promise] ==f==> Promise[Promise] ==g==> Promise[Promise]
Promise.resolve(Promise.resolve(3)).then(f).then(g)
Promise.resolve(Promise.resolve(3)).then(x => f(x).then(g))
```

a.then not defined ✗

```
Maybe(Maybe(3)).chain(console.log); // Just {value: 3}
Promise.resolve(Promise.resolve(3)).then(console.log); // 3
```

Promise不是Functor，更不是Monad