The underestimated power of KeyPaths

Vincent Pradeilles (@v_pradeilles) – Worldline

KeyPaths?

KeyPaths

KeyPaths were introduced with Swift 4

They are a way to defer a call to the getter/setter of a property

```
let countKeyPath: KeyPath<String, Int> = \String.count // or \.count

let string = "Hello dotSwift!"

string[keyPath: countKeyPath] // 15
```

They perform the same job than a closure, but with less `$0` hanging around

How can they be leveraged?

```
people.filter({ $0.isOverEighteen })
      .map({ $0.lastName })
```

Let’s introduce an operator!

Let’s introduce an operator!

```
prefix operator ^

prefix func ^<Type, Property>(_ keyPath: KeyPath<Type, Property>)
                                -> (Type) -> Property {

    return { obj in obj[keyPath: property] }
}
```

Let’s introduce an operator!

```
prefix operator ^

prefix func ^<Type, Property>(_ keyPath: KeyPath<Type, Property>)
                                -> (Type) -> Property {

    return { obj in obj[keyPath: property] }
}
```

Let’s introduce an operator!

```
prefix operator ^

prefix func ^<Type, Property>(_ keyPath: KeyPath<Type, Property>)
                              -> (Type) -> Property {

    return { obj in obj[keyPath: property] }
}
```

Let’s introduce an operator!

```
prefix operator ^

prefix func ^<Type, Property>(_ keyPath: KeyPath<Type, Property>)
                                -> (Type) -> Property {

    return { obj in obj[keyPath: property] }
}
```

Let’s introduce an operator!

```
prefix operator ^

prefix func ^<Type, Property>(_ keyPath: KeyPath<Type, Property>)
                                -> (Type) -> Property {

    return { obj in obj[keyPath: property] }
}
```

Let’s introduce an operator!

```
prefix operator ^

prefix func ^<Type, Property>(_ keyPath: KeyPath<Type, Property>)
                                -> (Type) -> Property {

    return { obj in obj[keyPath: property] }
}
```

```
people.filter({ $0.isOverEighteen })
      .map({ $0.lastName })
```

```
people.filter(^\.isOverEighteen)
      .map(^\.lastName)
```

It’s not always this simple…

```
people.sorted(by: { $0.lastName < $1.lastName })
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

Just write another helper

```
func their<Type, T: Comparable>(_ keyPath: KeyPath<Type, T>,
                                comparedWith comparator: @escaping (T, T) -> Bool = (<))
                                -> (Type, Type) -> Bool {
    return { obj1, obj2 in
        return comparator(obj1[keyPath: keyPath], obj2[keyPath: keyPath])
    }
}
```

```
people.sorted(by: { $0.lastName < $1.lastName })
```

```
people.sorted(by: their(\.lastName))
```

```
people.sorted(by: their(\.lastName, comparedWith: >))
```

Conclusion

Conclusion

KeyPaths are great, use them!

Many other possibilities, like type-safe predicates:

```
people.select(where: \.age <= 22 && \.lastName.count > 10)
```

You want to see more?

https://github.com/vincent-pradeilles/KeyPathKit