现代编程语言概览

从现代多范式静态类型语言看语言发展趋势

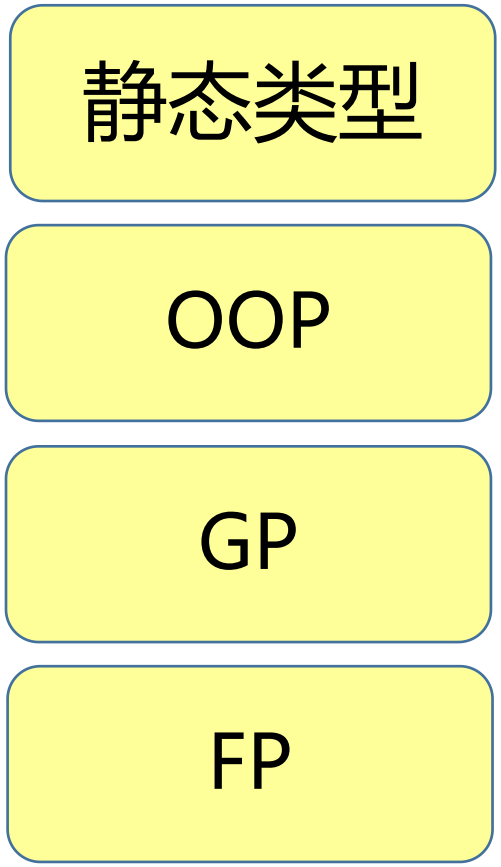
灰蓝天际
github.com/hltj

关于我

- 10年互联网研发
  - 迅雷、新浪、国美互联网
- 关注
  - 编程、架构、技术管理
  - 培训、开源、传统文化
- 开源项目
  - wxMEdit 作者
  - Kotlin 中文站维护人

现代静态类型语言

- Kotlin
- Swift
- Rust
- Scala
- Java 8/9
- C++ 11/14/17



现代静态类型语言

- Kotlin
- Swift
- Rust
- Scala
- Java 8/9
- C++ 11/14/17

- 可空性表达
- trait
- 不可变
- lambda
- 惰性求值
- 高阶函数
- ……

REPL

- 交互式编程环境
  - Read–Eval–Print Loop

语言	第三方/官方 REPL
Kotlin	**kotlinc**
Swift	**swift**
Rust	irust rusti
Scala	**scala**
Java 8/9	javarepl / **jshell**
C++ 11/14/17	cling

类型推断

Java ✗ 不支持常规类型推断

```
[cling]$ auto pi = 3.14159
(double) 3.14159
```

```
scala> val π =3.14159
π : Double = 3.14159
```

```
Welcome to Kotlin version 1.0.3 (JRE 1.7.0_79-b15)
Type :help for help, :quit for quit
>>> val π = 3.14159
```

```
rusti=> #![feature(non_ascii_idents)]
rusti=> let _π = 3.14159;
```

```
Welcome to Swift version 3.0.1 (swift-3.0.1-GM-CANDIDATE).
  1> let pi = 3.14159
pi: Double = 3.1415899999999999
```

空的表达

- Java 的 null

```
public void sendMessageToClient(@Nullable Client client,
                                 @Nullable String message,
                                 @NotNull Mailer mailer) {
    if (client == null || message == null) return;

    PersonalInfo personalInfo = client.getPersonalInfo();
    if (personalInfo == null) return;

    String email = personalInfo.getEmail();
    if (email == null) return;

    mailer.sendMessage(email, message);
}
```

空的表达

- Kotlin 的可空性

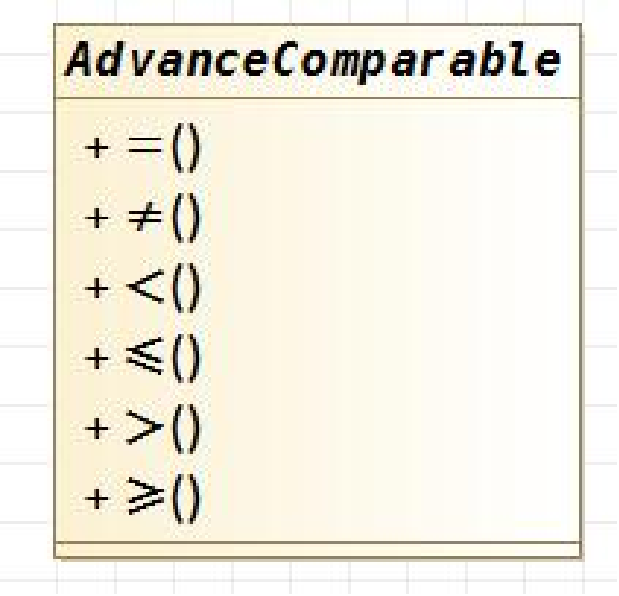
```
fun sendMessageToClient(
        client: Client?, message: String?, mailer: Mailer
) {
    val email = client?.personalInfo?.email?: return
    mailer.sendMessage(email, message?: return)
}
```

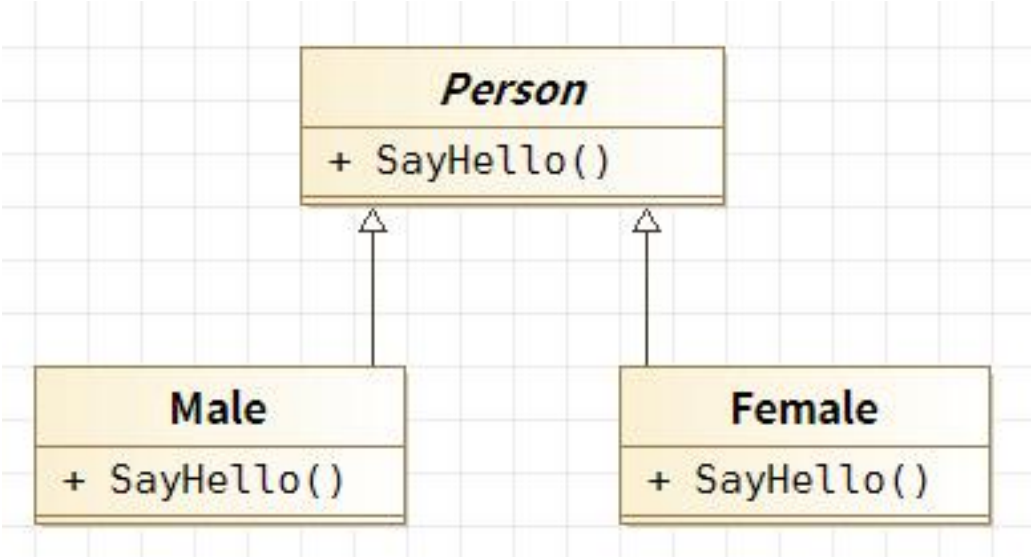
不可变性

	不可变	可变
Rust	**默认**	mut
Kotlin	val	var
Scala	val	var
Swift	let	var
C++	const	默认
Java	✗	默认

trait

- 背景壹
  - 接口能提供默认实现





trait

- 背景贰
  - 能给既有类添加成员

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> otherList = Arrays.asList(1, 2, 3);

Collections.swap(list,
        Collections.binarySearch(list, Collections.max(otherList)),
        Collections.max(list));
```

```
swap(list, binarySearch(list, max(otherList)), max(list));
```

```
list.swap(list.binarySearch(otherList.max()), list.max());
```

trait

- 背景叁
  - 泛型实践

```
typewriter.type()
object.type()

obj3d.translate()
text.translate()
```

trait

- 背景叁
  - 泛型实践
    - 比鸭子类型更好的泛型实践

```
interface HasInformation {
    fun info() = "default information"
}

fun <T: HasInformation> teeInformation(t: T): String {
    println(t.info())
    return t.info()
}
```

lambda表达式

- 匿名类/接口

```
Runnable noArg = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello World");
    }
};
noArg.run();

// 按长度排序
List<String> strings = Arrays.asList("aa", "test", "1");
Collections.sort(strings, new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        return ((Integer)s1.length()).compareTo(s2.length());
    }
});
System.out.println(strings);
```

lambda表达式

- Java 8 的 SAM

```
Runnable noArg = () -> System.out.println("Hello World");
noArg.run();

// 按长度排序
List<String> strings = Arrays.asList("aa", "test", "1");
Collections.sort(strings,
        (s1, s2) -> ((Integer)s1.length()).compareTo(s2.length()));
System.out.println(strings);
```

lambda表达式

- Kotlin 的 lambda

```
Runnable { println("Hello World") }.run()

val strings = arrayListOf("aa", "test", "1")
// 按长度排序
strings.sortWith(Comparator<String> {
    s1, s2 -> s1.length.compareTo(s2.length)
})
println(strings)
```

lambda表达式

- Kotlin 的 lambda

```
Runnable { println("Hello World") }.run()

val strings = arrayListOf("aa", "test", "1")
// 按长度排序
println(strings.sortedBy { it.length })
```

惰性求值

```
Logger logger = new Logger();
if (logger.isDebugEnabled()) {
    logger.debug("Look at this: " + expensiveOperation());
}
```

```
val logger = Logger();
logger.debug { "Look at this: " + expensiveOperation() }
```

高阶函数

- 2010 年后出版书的书名，逗号分隔

```
data class Book(val title: String, val year: Int)

val books = arrayListOf(
        Book("Java 8 Lambdas", 2014),
        Book("Thinking in Java 3rd", 2002),
        Book("Thinking in Java 4th", 2004),
        Book("Kotlin in Action", 2017)
)
```

高阶函数

- 2010 年后出版书的书名，逗号分隔
  - 传统方法

```
var bookTitles = ""
for (book in books) {
    if (book.year > 2010) {
        if (!bookTitles.isEmpty())
            bookTitles += ", "
        bookTitles += book.title
    }
}
println(bookTitles)
```

高阶函数

- 2010 年后出版书的书名，逗号分隔
  - 高阶函数

```
val bookTitles = books
        .filter { it.year > 2010 }
        .map { it.title }
        .reduce { acc, s -> acc + ", " + s }

println(bookTitles)
```

高阶函数

- 2010 年后出版书的书名，逗号分隔
  - 高阶函数

```
val bookTitles = books
        .filter { it.year > 2010 }
        .map { it.title }
        .joinToString()

println(bookTitles)
```

高阶函数

- 瓜哥（@2gua）出的一个题目
  - 用你熟悉的语言，统计一个字符串 abcdefghijklmnopqrstuvwxyz...abcdefghijklmnopqrstuvwxyz（1千万个a-z，不可直接a=1千万.....）中每个字母的个数，最后输出类似图示。
  - 要求除了更好的方式（如更加Pythonic的方式），还要计算越快越好，并打印出代码执行时间（打印效果类似图示）

```
a: 10000000, b: 10000000, c: 10000000, d: 10000000, e: 10000000, f: 10000000, g: 10000000,
h: 10000000, i: 10000000, j: 10000000, k: 10000000, l: 10000000, m: 10000000, n: 10000000,
o: 10000000, p: 10000000, q: 10000000, r: 10000000, s: 10000000, t: 10000000, u: 10000000,
v: 10000000, w: 10000000, x: 10000000, y: 10000000, z: 10000000,
```

高阶函数

- Kotlin

```
import kotlin.system.measureTimeMillis

fun main(args : Array<String>) {
    print(measureTimeMillis {
        val letters = ('a'..'z').joinToString(separator="")
        val repeated = letters.repeat(1000_0000)
        println(repeated.groupingBy { it }.eachCount())
    } / 1000.0)
    println('s')
}
```

高阶函数

- 店铺-客户-订单-商品（Kotlin 心印）

```
data class Shop(val name: String, val customers: List<Customer>)

data class Customer(val name: String, val city: City, val orders: List<Order>) {
    override fun toString() = "$name from ${city.name}"
}

data class Order(val products: List<Product>, val isDelivered: Boolean)

data class Product(val name: String, val price: Double) {
    override fun toString() = "'$name' for $price"
}

data class City(val name: String) {
    override fun toString() = name
}
```

高阶函数

- 店铺-客户-订单-商品（Kotlin 心印）

```
fun Shop.getCustomersWhoOrderedProduct(product: Product): Set<Customer> {
    // Return the set of customers who ordered the specified product
    todoCollectionTask()
}

fun Customer.getMostExpensiveDeliveredProduct(): Product? {
    // Return the most expensive product among all delivered products
    // (use the Order.isDelivered flag)
    todoCollectionTask()
}

fun Shop.getNumberOfTimesProductWasOrdered(product: Product): Int {
    // Return the number of times the given product was ordered.
    // Note: a customer may order the same product for several times.
    todoCollectionTask()
}
```

讨论

- 缺少哪些点
- 其他新兴语言特性
- 任何相关话题