Scala Reflection

Andy Huang

Use Case: Define Spark UDF By Reflection

- Motivation:
  - Loading dynamic functions and executing them at runtime to enrich the transform pipeline
- Issues:
  - Being able to handle various parameter/return types
  - Being workable in Spark Environment

Use Case: Define Spark UDF By Reflection

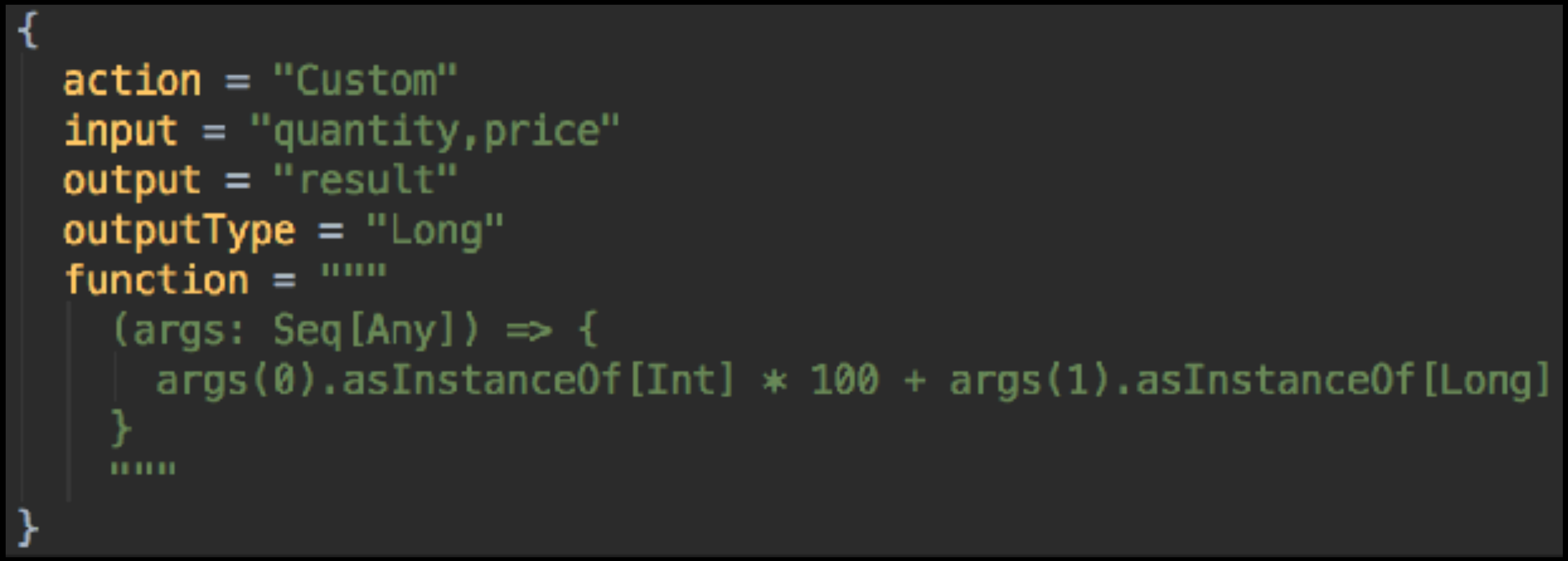
multiple parameters with various type

```
function(arg1: Any, arg2: Any, …): returnType = {

  ……

}
```

various return type

programming section

Use Case: Define Spark UDF By Reflection



```
{
  action = "Custom"
  input = "quantity,price"
  output = "result"
  outputType = "Long"
  function = """
    (args: Seq[Any]) => {
      args(0).asInstanceOf[Int] * 100 + args(1).asInstanceOf[Long]
    }
    """
}
```

```
anonymousFunction(args: Seq[Any]): Long = {
  args(0) * args(1) + args(2)
}
```

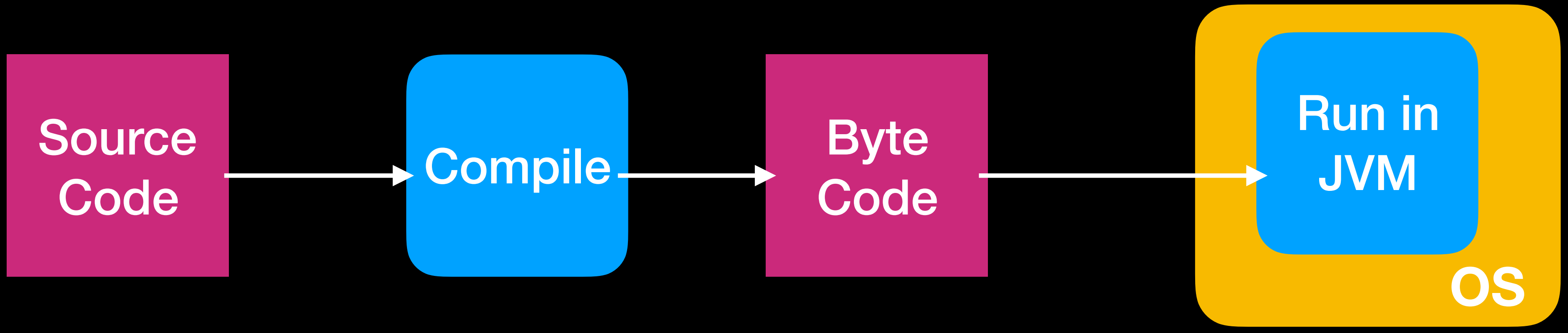
id	quantity	price	tax
1	10	99.9	9.9
2	100	88.8	88

id	quantity	price	tax	result
1	10	99.9	9.9	989.1
2	100	88.8	88	8792

Meta Programming

- Ability to treat programs as data.
  - Reading program structure and act on that knowledge
  - Modifying program itself while running
- The ability of language to be its own metalanguage is called Reflection

Reflection



- Run time reflection: program introspection and hot loading code e.g. json mapping, string to program, etc

- Compile time reflection: programs modify themselves at **compile time**, e.g. program transformer, code generator etc

Reflection

- How to make program change/write themselves?
  1. runtime class reloading
     - Dynamic class reloading/JRebel
     - Runtime compiling/scala.tools.reflect.ToolBox
  2. Invocation

Reflection

- Runtime Reflection
  - Inspection of classes, fields and methods at runtime
  - Instantiation of new objects at runtime
  - Invocation of methods at runtime
  - Runtime compiling

Scala Reflection

- Universes
    - Run Time Reflection
    - Compile Time Reflection(macros)
- Mirrors
    - Class Loader
    - Invoker

Scala Reflection - Universes

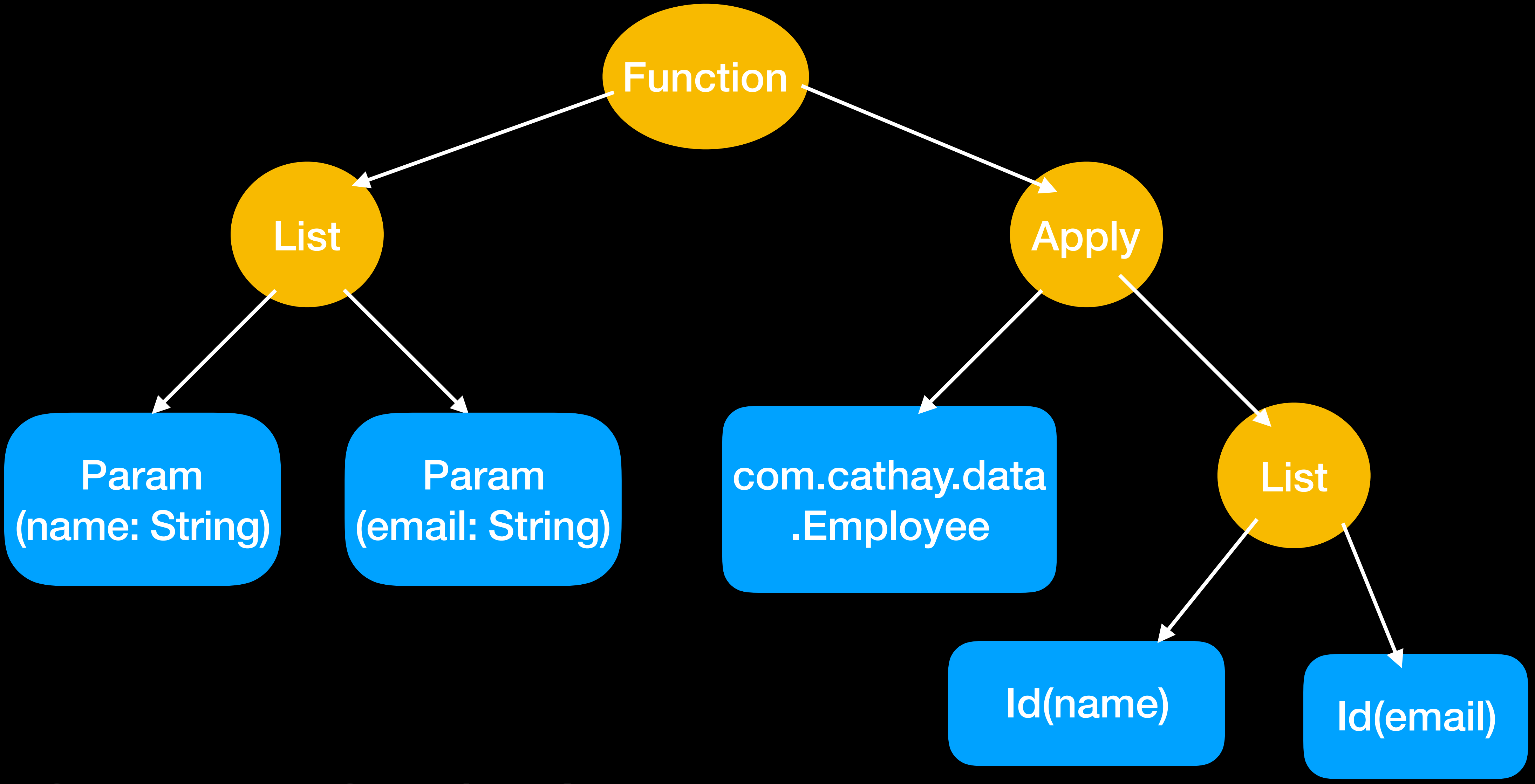
- Symbols: Binding between a name and the entity it refers to, i.e. the declaration of an entity(class, object, trait, etc.) or member(val s, var s, def s, etc.)
- Types: Representing the information about the type of a corresponding symbol
- Trees: Representing programs which also called Abstract Syntax Trees

Scala Reflection - Universes

- Type Symbols representing type, class, and trait declarations, as well as type parameters.
  - e.g. typeOf[Employee].typeSymbol.asType.typeParams
- Term Symbols representing val, var, def, and object declarations as well as packages and value parameters.
  - e.g. typeOf[Employee].decl(TermName(“name")).asTerm.isLazy

Scala Reflection - Universes

Abstract Syntax Tree



```
(name: String, email: String) => {
  com.cathay.data.Employee(name, email)
}
```

Scala Reflection - Mirrors

- Mirrors: The set of entities that we have reflective access to.
- The entities accessible through **runtime reflection** are made available by a Classloader mirror
- A Classloader mirror can create Invoker mirrors(such as InstanceMirror, MethodMirror, etc)

References

- Java dynamic class reloading
- Java Reflection
- Scala Reflection
- Scala ToolBox Example
- Define spark UDF by Reflection