Chapitre 1

Béatrice CARRE

Introduction

La génération de code se fait en plusieurs passes :
- analyse lexicale et analyse syntaxique de l' idl donnant un ast de type Idl.file.
- vérification des types de l'ast, donnant un nouvel ast de type CIdl.file.
- la génération des fichiers stub java nécessaires pour un appel callback
- la génération à partir de l'ast CIdl.file du fichier .ml
- la génération à partir du CIdl.file du fichier .mli

Ces différentes étapes seront présentées plus en profondeur.

1.1 La syntaxe de l'idl

La syntaxe du langage d'interface est donné en annexe, en utilisant la notation BNF.

Les symboles < et > encadrent des règles optionnelles, les terminaux sont en bleu, et les non-terminaux sont en italique.

1.2 lexing parsing

La première phase est celle d'analyse lexicale et syntaxique, séparant l'idl en lexèmes et construisant l'AST, défini par Idl.file, dont la structure : est définie en annexe

1.3 check

Vient ensuite la phase d'analyse sémantique, analysant l'AST obtenue par la phase précédente, vérifiant si le programme est correct, et construisant une liste de CIdl.clazz, restructurant chaque classe ou interface définie dans l'idl. Le module Cidl définit le nouvel AST allant être manipulé dans les passes de génération de code. Il est décrit en annexe.

1.4 génération stub_file

//TODO

1.5 génération .ml

La génération de ce code se fait en plusieurs passes sur l'ast obtenu après ces précédents phases, le CIdl.file.

1.5.1 schémas de compilation

La génération de code rend une liste de valeurs imprimées (dans le fichier engendrer), en modifiant Nous considérons l'environnement contenant les :

- package (:string) le nom du package courant
- isInterface (:bool) est vrai si la déclaration courante est une interface. Faux si c'ets une classe.
- isCallback (:bool) si la classe courante a l'attribut callback
- isAbstractC (:bool) si la classe courante est abstraite
- classname (:string)
- isAbstractElt (:bool) util ?
- isStaticElt (:bool) util ?
- isFinalElt (:bool) util ?

$[\![package \ < package >^*]\!] \longrightarrow$

$[\![package]\!]$

```
init_env ();
```

$[\![package]\!]$

$[\![decl \ < decl >^*]\!] \longrightarrow$
$[\![package \ qname \ ; \ decl < decl >^*]\!] \longrightarrow$
$[\![class \ qname \ ]\!] \longrightarrow$
$[\![class \ name \ extends \ qname]\!] \longrightarrow$
$[\![class \ name \ extends \ qname]\!] \longrightarrow$
$[\![class \ name \ extends \ qname]\!] \longrightarrow$

1.6 génération .mli

Annexe

BNF

```
class

file ::= package <package>*
       | decl <decl>*

package ::= package qname ; decl <decl>*

decl ::= class
       |interface

class ::= <[attributes]> <abstract> class name
          < extends qname >
          < implements qname <, qname>* >
          { <class_elt ;>* }
class_elt ::= <[ attributes ]> <static> <final> type name
            | <[ attributes ]> <static> <abstract> type name (<args>)
            | [ attributes ] <init> (<args>)

interface ::= <[ attributes ]> interface name
              < extends qname <, qname>* >
              { <interface_elt;>* }
interface_elt ::=
     <[ attributes ]> type name
   | <[ attributes ]> type name (<args>)

args ::= arg <, arg>*
arg ::= <[ attributes ]> type <name>

attributes ::=  attribute <, attribute>*
attribute ::= name ident
            | callback
            | array

type ::= basetype
       | object
       | basetype [ ]
basetype ::= void
           | boolean
           | byte
           | char
           | short
           | int
           | long
           | float
           | double
           | string
object := qname
qname ::= name<.name>*
name ::= ident
```

Module Idl

```
(**  module Idl  *)

type ident = {
    id_location: Loc.t;
    id_desc: string
  }
type qident = {
    qid_location: Loc.t;
    qid_package: string list;
    qid_name: ident;
  }
type type_desc =
    Ivoid
  | Iboolean
  | Ibyte
  | Ishort
  | Icamlint
  | Iint
  | Ilong
  | Ifloat
  | Idouble
  | Ichar
  | Istring
  | Itop
  | Iarray of typ
  | Iobject of qident
and typ = {
    t_location: Loc.t;
    t_desc: type_desc;
  }
type modifier_desc =
  | Ifinal
  | Istatic
  | Iabstract
and modifier = {
    mo_location: Loc.t;
    mo_desc: modifier_desc;
}
type ann_desc =
  | Iname of ident
  | Icallback
  | Icamlarray
and annotation = {
    an_location: Loc.t;
    an_desc: ann_desc;
}
type arg = {
    arg_location: Loc.t;
    arg_annot: annotation list;
    arg_type: typ
}
type init = {
    i_location: Loc.t;
    i_annot: annotation list;
    i_args: arg list;
}
type field = {
    f_location: Loc.t;
    f_annot: annotation list;
    f_modifiers: modifier list;
    f_name: ident;
    f_type: typ
}
type mmethod = {
    m_location: Loc.t;
    m_annot: annotation list;
    m_modifiers: modifier list;
    m_name: ident;
    m_return_type: typ;
    m_args: arg list
}
type content =
    | Method of mmethod
    | Field of field
type def = {
    d_location: Loc.t;
    d_super: qident option;
    d_implements: qident list;
    d_annot: annotation list;
    d_interface: bool;
    d_modifiers: modifier list;
    d_name: ident;
    d_inits: init list;
    d_contents: content list;
}
type package = {
    p_name: string list;
    p_defs: def list;
}
type file = package list
```

Module CIdl

```
(**  module CIdl  *)
type typ =
  | Cvoid
  | Cboolean (** boolean -> bool *)
  | Cchar (** char -> char *)
  | Cbyte (** byte -> int *)
  | Cshort (** short -> int *)
  | Ccamlint (** int -> int<31> *)
  | Cint (** int -> int32 *)
  | Clong (** long -> int64 *)
  | Cfloat (** float -> float *)
  | Cdouble (** double -> float *)
  | Ccallback of Ident.clazz
  | Cobject of object_type (** object -> ... *)
and object_type =
  | Cname of Ident.clazz (** ... -> object *)
  | Cstring (** ... -> string *)
  | Cjavaarray of typ (** ... -> t jArray *)
  | Carray of typ (** ... -> t array *)
  | Ctop

type clazz = {
    cc_abstract: bool;
    cc_callback: bool;
    cc_ident: Ident.clazz;
    cc_extend: clazz option; (* None = top *)
    cc_implements: clazz list;
    cc_all_inherited: clazz list; (* tout jusque top ... (et avec les
        interfaces) sauf elle-meme. *)
    cc_inits: init list;
    cc_methods: mmethod list; (* methodes + champs *)
    cc_public_methods: mmethod list; (* methodes declarees + celles
        heritees *)
    cc_static_methods: mmethod list;
  }
and mmethod_desc =
  | Cmethod of bool * typ * typ list (* abstract, rtype, args *)
  | Cget of typ
  | Cset of typ
and mmethod = {
    cm_class: Ident.clazz;
    cm_ident: Ident.mmethod;
    cm_desc: mmethod_desc;
  }
and init = {
    cmi_ident: Ident.mmethod;
    cmi_class: Ident.clazz;
    cmi_args: typ list;
  }
type file = clazz list
```

module Ident

```
(* module Ident  *)
(* le type des identifiants de classe de l'IDL *)
type clazz = {
    ic_id: int;
    ic_interface: bool;
    ic_java_package: string list;
    ic_java_name: string;
    ic_ml_name: string;
    ic_ml_name_location: Loc.t;
    ic_ml_name_kind: ml_kind;
  }
type mmethod = {
    im_java_name: string;
    im_ml_id: int; (** entier unique pour une nom ml *)
    im_ml_name: string;
    im_ml_name_location:Loc.t;
    im_ml_name_kind: ml_kind;
  }
```

idl_camlgen.make ast

Type jni
    MlClass.make_jni_type
Class type
    MlClass.make_class_type
Cast JNI
    MlClass.make_jniupcast
    MlClass.make_jnidowncast
Fonction d'allocation
    MlClass.make_alloc
    MlClass.make_alloc_stub
Capsule / souche
    MlClass.make_wrapper
Downcast utilisateur
    MlClass.make_downcast
    MlClass.make_instance_of
Tableaux
    MlClass.make_array
Fonction d'initialisation
    MlClass.make_fun
Classe de construction
    MlClass.make_class
fonctions / methodes static
    MlClass.make_static