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 modules

camlgen :
check :
common :
javagen :
jnihelpers :
parser :

1.2 lexing parsing

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

```
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
```

1.3 check

Vient ensuite une phase, prenant l'AST obtenue par la phase précédente, construisant une liste de CIdl.clazz, structurant chaque classe ou interface déninie dans l'idl. Le module Cidl définit l'AST allant être manipulé dans les passes de génération de code.

```
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  *)
(* 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;
  }
```

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.

```
(** Fonction idl_camlgen.make *)

  let str_list = [] in
  (** Type jni *)
  let str_list = (MlClass.make_jni_type c_file) :: str_list in
  (** Class type *)
  let class_type = MlClass.make_class_type ~callback:false c_file in
  let str_list = match class_type with
  | [] -> str_list
  | list -> <:str_item< class type $MlGen.make_rec_class_type class_type$
      >> :: str_list in
  let class_type = MlClass.make_class_type ~callback:true c_file in
  let str_list = match class_type with
  | [] -> str_list
  | list -> <:str_item< class type $MlGen.make_rec_class_type class_type$
      >> :: str_list in
  (** cast JNI *)
  let str_list = (MlClass.make_jniupcast c_file) :: str_list in
  let str_list = (MlClass.make_jnidowncast c_file):: str_list in
  (** fonction d'allocations *)
  let str_list = (MlClass.make_alloc c_file) :: str_list in
  let str_list = (MlClass.make_alloc_stub c_file) :: str_list in
  (** capsule/souche *)
  let wrapper = [] in
  let wrapper = List.append (MlClass.make_wrapper ~callback:true c_file)
      wrapper in
  let wrapper = List.append (MlClass.make_wrapper ~callback:false c_file)
      wrapper in
  let str_list = match wrapper with
    | [] -> str_list
    | _ ->
        let list = MlGen.make_rec_class_expr wrapper in
        <:str_item< class $list$ >> :: str_list
  in
  (** downcast 'utilisateur' *)
  let str_list = (MlClass.make_downcast c_file) :: str_list in
  let str_list = (MlClass.make_instance_of c_file) :: str_list in
  (** Tableaux *)
  let str_list = (MlClass.make_array c_file) :: str_list in
  (** fonction d'initialisation *)
  let str_list = (MlInit.make_fun ~callback:false c_file) :: str_list in
  let str_list = (MlInit.make_fun ~callback:true c_file) :: str_list in
  (** classe de construction *)
  let str_list = (MlInit.make_class ~callback:false c_file) :: str_list in
  let str_list = (MlInit.make_class ~callback:true c_file) :: str_list in
  (** fonctions / mehodes static *)
  let str_list = (MlMethod.make_static c_file) :: str_list in
  List.rev str_list
```

make_jni_type :

$[\![file]\!] \longrightarrow$

String.concat $[\![clazz]\!]$ file

$[\![clazz]\!] \longrightarrow$

```
"type␣_jni_"^clazz.cc_ident.ic_ml_name^"␣=␣Jni.obj;;"
```

make_class_type

$[\![clazz]\!]_{callback=false} \longrightarrow$

```
"class␣type␣"^clazz.cc_ident.ic_ml_name^"␣="
"object"
```

$[\![clazz.cc\_exends]\!]_{callback=false}$
$[\![clazz.cc\_implements]\!]_{callback=false}$

```
"method␣"^_get_jni_"^clazz.cc_ident.ic_ml_name^" : "^_jni_^"clazz.
    cc_ident.ic_ml_name
```

$[\![clazz.cc\_methods]\!]_{callback=false}$

```
"end"
```

$[\![clazz]\!]_{callback=true} \longrightarrow$

```
"class␣type␣virtual␣_stub_"^clazz.cc_ident.ic_ml_name
"object"
```

$[\![clazz.cc\_exend]\!]_{callback=true}$
$[\![clazz.cc\_all\_inherited]\!]_{callback=true}$

```
 "method␣_get_jni_"^clazz.cc_ident.ic_ml_name^"␣:␣_jni_"^clazz.cc_ident.
     ic_ml_name
```

$[\![clazz.cc\_public\_methods]\!]_{callback=true}$

```
"end"
```

$[\![cc\_extend]\!]_{callback=false} \longrightarrow$

```
match cl.cc_extend with
  None -> "inherit␣JniHierarchy.top"
 | Some super -> "inherit␣"^super.cc_ident.ic_ml_name
```

$[\![cc\_extend]\!]_{callback=true} \longrightarrow$

```
"inherit␣JniHierarchy.top"
```

$[\![cc\_implements]\!]_{callback=false} \longrightarrow$

```
List.map (fun interface -> "inherit␣"^interface.cc_ident.ic_ml_name) cl.
    cc_implements
```

$[\![cc\_all\_inherited]\!]_{callback=true} \longrightarrow$

```
List.map (fun cl ->
"method␣_get_jni_" ^ cl.cc_ident.ic_ml_name^"␣:␣_jni_"^cl.cc_ident.
    ic_ml_name ) cl.cc_all_inherited
```

$[\![cc\_methods]\!]_{callback=false} \longrightarrow$

```
List.map ( fun m ->
  match m.cm_desc with
   | Cmethod (abstract, rtype, args) ->
      let typ = (args, rtype) in
      "method␣"^m.cm_ident.im_ml_name^"␣:␣"^
```

$[\![typ]\!]$

```
   | Cset typ ->
      let typ = ([typ], Cvoid) in
      "method␣"^m.cm_ident.im_ml_name"␣:␣"^
```

$[\![typ]\!]$

```
   | Cget typ ->
      let typ = ([], typ) in
      "method␣"^m.cm_ident.im_ml_name^"␣:␣"^
```

$[\![typ]\!]$

```
 ) cc_methods
```

$[\![cc\_public\_methods]\!]_{callback=true} \longrightarrow$

```
List.map ( fun ->
  match m.cm_desc with
    | Cset _
    | Cget _ -> acc
| Cmethod (abstract,rtyp, targs) ->
  let ml_stub_name = Ident.get_method_ml_stub_name m.cm_ident in
  let sign = match targs with
    | [] -> MlType.ml_jni_signature_of_type rtyp
    | targs -> MlType.ml_jni_signature targs rtyp in
    <:class_sig_item< method $lid:ml_stub_name$ : $sign$ >> :: acc
```

1.6 génération .mli