UNIT –II

Extending Ruby: Ruby Objects in C, the Jukebox extension, Memory allocation, Ruby Type System, Embedding Ruby to Other Languages, Embedding a Ruby Interpreter

Extending Ruby

- It is easy to extend Ruby with new features by writing code in Ruby. Once you start adding in low-level code written in C, however, the possibilities are endless.
- Extending Ruby with C is pretty easy.
- But before we can get Ruby and C to work together, we need to see what the Ruby world looks like from the C side.

Ruby Objects in C

- how to represent and access Ruby data-types within C.
- Everything in Ruby is an object, and all variables are references to objects.
- In C, this means that the type of all Ruby variables is VALUE, which is either a pointer to a Ruby object or an immediate value (such as Fixnum).
- This is how Ruby implements object-oriented code in C: a Ruby object is an allocated structure in memory that contains a table of instance variables and information about the class. The class itself is another object (an allocated structure in memory) that contains a table of the methods defined for that class. On this foundation hangs all of Ruby.

VALUE as a Pointer

- When VALUE is a pointer, it is a pointer to one of the defined Ruby object structures—you can't have a VALUE that points to an arbitrary structure.
- The structures for each built-in class are defined in “ruby.h” and are named R*Classname*, as in RString and RArray.
- You can check to see what type of structure is used for a particular VALUE in a number of ways.
- The macro TYPE(*obj*) will return a constant representing the C type of the given object: T_OBJECT, T_STRING, and so on. Constants for the built-in classes are defined in “ruby.h”.

- use the macro Check_Type, which will raise a TypeError exception if *value* is not of the expected *type* (which is one of the constants T_STRING, T_FLOAT, and so on):
- Check_Type(VALUE *value*, int *type*)
- If speed is an issue, there are faster macros that check specifically for the immediate values Fixnum and nil.
- FIXNUM_P(*value*) non-zero if value is a Fixnum NIL_P(*value*) non-zero if value is nil RTEST(*value*) non-zero if value is neither nil nor false

- Again, note that we are talking about “type” as the C structure that represents a particular built-in type. The class of an object is a different beast entirely. The class objects for the built-in classes are stored in C global variables named rb_c*Classname* (for instance, rb_cObject); modules are named rb_m*Modulename*.
- It wouldn't be advisable to mess with the data in these structures directly, however—you may look, but don't touch unless you are fond of debuggers. You should normally use only the supplied C functions to manipulate Ruby data (we'll talk more about this in just a moment).

- However, in the interests of efficiency you may need to dig into these structures to obtain data. In order to dereference members of these C structures, you have to cast the generic VALUE to the proper structure type. ruby.h contains a number of macros that perform the proper casting for you, allowing you to dereference structure members easily. These macros are named R*CLASSNAME*, as in RSTRING or RARRAY. For example:

VALUE str, arr;

RSTRING(str)->len length of the Ruby string

RSTRING(str)->ptr pointer to string storage

RARRAY(arr)->len length of the Ruby array

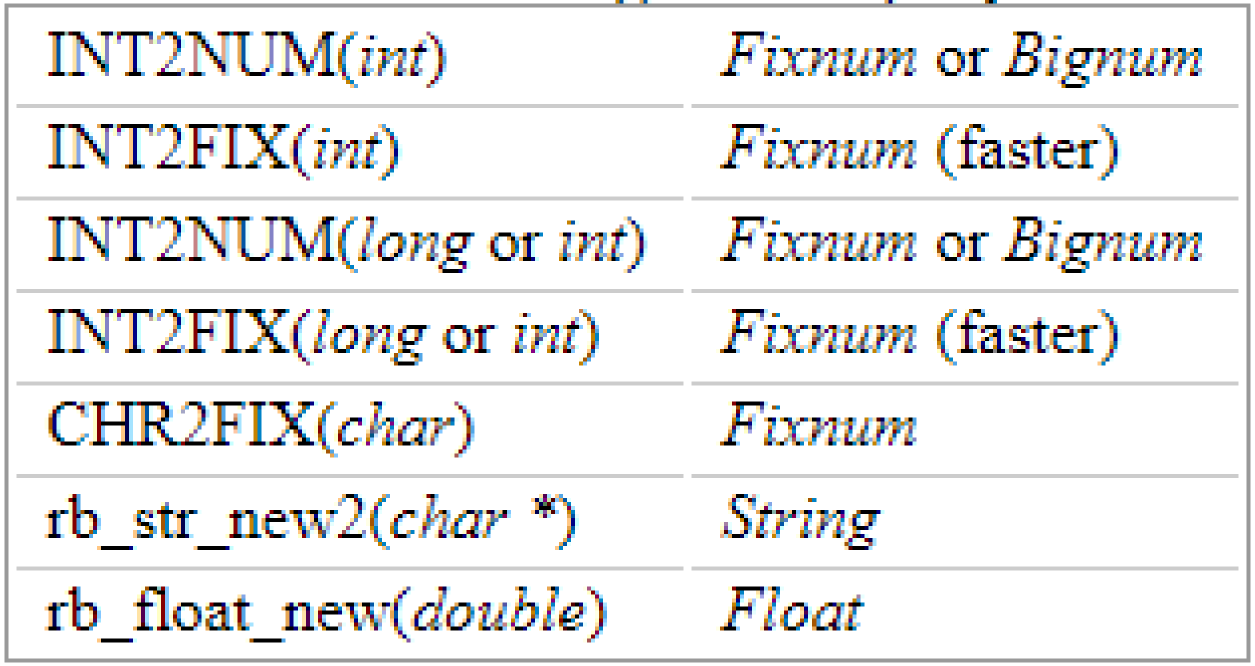
RARRAY(arr)->capa capacity of the Ruby array

RARRAY(arr)->ptr pointer to array storage

VALUE as an Immediate Object

- As we said above, immediate values are not pointers: Fixnum, Symbol, true, false, and nil are stored directly in VALUE.
- Fixnum values are stored as 31-bit numbers *(Or 63-bit on wider CPU architectures.)* that are formed by shifting the original number left 1 bit and then setting the least significant bit (bit 0) to “1.” When VALUE is used as a pointer to a specific Ruby structure, it is guaranteed always to have an LSB of zero; the other immediate values also have LSBs of zero. Thus, a simple bit test can tell you whether or not you have a Fixnum.
- There are several useful conversion macros for numbers as well as other standard datatypes shown in Table 17.1.
- The other immediate values (true, false, and nil) are represented in C as the constants Qtrue, Qfalse, and Qnil, respectively. You can test VALUE variables against these constants directly, or use the conversion macros (which perform the proper casting).

Table 17.1 : C Datatypes to Ruby Objects



INT2NUM(*int*)	*Fixnum* or *Bignum*
INT2FIX(*int*)	*Fixnum* (faster)
INT2NUM(*long* or *int*)	*Fixnum* or *Bignum*
INT2FIX(*long* or *int*)	*Fixnum* (faster)
CHR2FIX(*char*)	*Fixnum*
rb_str_new2(*char* *)	*String*
rb_float_new(*double*)	*Float*