Chapter3 Support for DIVINE software visualization in YaJDB

3.1 An introduction of YaJDB

We develop one tool named YaJDB to support the software visualization and animation metaphor. YaJDB is one kind of a debugger extending JDB. YaJDB is used to catch the information during program execution. The content of caught information depends on what YaJDB supports. While YaJDB supports software visualization, the information is concerned with the structure and value of the variable. While YaJDB supports animation metaphor, the information is concerned with the execution state information including location of a program, what kind of statements, variable's type, variable's value.

3.2 An overview of DIVINE

DIVINE[5, 6, 16] , **D**ebugging **I**nformation **VI**sualizatio**N** in 3D **E**nvironment, is a debugging information visualization system that visualizes the debugging information received from YaJDB. A software developer can understand the structure of variables and observe the change of variables' values through DIVINE.

The information of variables contains the values of variables on DIVINE. However, whether a variable holds a value depends on type of a variable. The types

of the Java programming language are divided into two categories: primitive types and reference types. Table 3-1 shows the detail of type classification. The primitive types are the boolean type and the numeric types. The numeric types are the integral types (byte, short, int, long, and char) and the floating-point types (float and double). The reference types are class types, interface types, and array types. An object is a dynamically created instance of a class type or a dynamically created array. A variable of a primitive type always holds a value of that exact type, but a variable of type object can hold a null reference or a reference to any object, whether class, interface or array. A reference to a unique object holds one unique identifier. Supposing that variables of type object hold the same reference indicates these objects reference the same memory location. DIVINE can utilize this characteristic to present the relationship between variables.

Table 3-1 : the classification of variable type

Primitive type		
Type declared in JAVA program as:		
boolean	byte	char
double	float	int
long	short	void
Reference type		
Type declared in JAVA program as	For example	Comment
a class	Date	class type
an interface	Runnable	interface type
an array	int[]	array type whose component type is integer type
an array	Date[]	array type whose component type is Class type
an array	Runnable[]	array type whose component type is Interface type

DIVINE provides more then one visualization metaphor. Users can choose suitable metaphor for a variable on DIVINE or DIVINE visualizes a variable with the suitable metaphor according to its data structure. Hence, YaJDB has to provide the related information such as the type of a variable. If the type of a variable belongs to reference type, DIVINE has to know the composition of reference type to judge what kind of data structure (such as graph, linked-list and so on) a variable is.

The values of variables may be different at different execution state. Therefore, YaJDB has to send an announcement to DIVIEN when execution state changes. Once DIVIEN receives the announcement, DIVIEN requests YaJDB to update the information related to variables.

According to mentioned support by YaJDB, we design three basic instructions for DIVINE to gain the variable detail and one instruction to announce the change of execution state to DIVINE. Because we want to make it possible for DIVINE to visualize all kind of data type and data structure, we only provide basic instructions. Besides, DIVINE can utilize them more flexibly. Table 3-2 shows the instructions detail.

Table 3-2 : the detail instructions detail.

Instruction	Output format
Ask [variable name] DIVINE ask YaJDB for type of one variable.	<VARNAME>@name<PRITYPE>@type <VARNAME>@name<REFTYPE>@type<ID>@id <VARNAME>@name<ARRTYPE>@comtype<ARRDIM>@dimension @length<ID>@id

	<VARNAME> @name <NULL> NULL
Struct [class name] DIVINE ask YaJDB for structure of one class.	<CLASSNAME>@name<ATTQUN>@number @detail @number: the number of attribute @detail : <ARRTYPE>@comtype<ARRDIM>@dimension @length<ID>@id<ATTNAME>@attribute name <REFTYPE>@type<ATTNAME>@attribute name <PRITYPE>@type<ATTNAME>@attribute name
Show [variable name] DIVINE ask YaJDB for value of one variable.	<NAME>@name<VALUE>@value <NAME>@name <VALUE> NULL
Update YaJDB announce execution state change to DIVEN	No output

@name: variable name or class name; @type: type name; @comtype: component type of an array; @id: identifier number; @dimension: dimension of an array; @length: length of an array;

3.3 The interaction between DIVINE and YaJDB

We provide dual connection between DIVINE and YaJDB to accomplish that a user can watch visualization with whether DIVINE or YaJDB. There are two way to start visualization. One starts with DIVINE; another with YaJDB. Figure 3-1 and Figure 3-2 show the interaction between DIVINE and YaJDB. The following two subsections describe the detail.

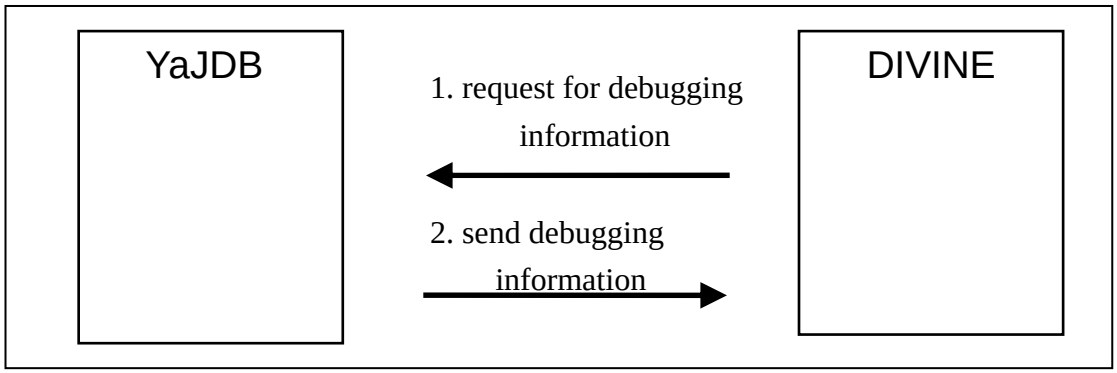

Figure 3-1: visualization starts with DIVINE

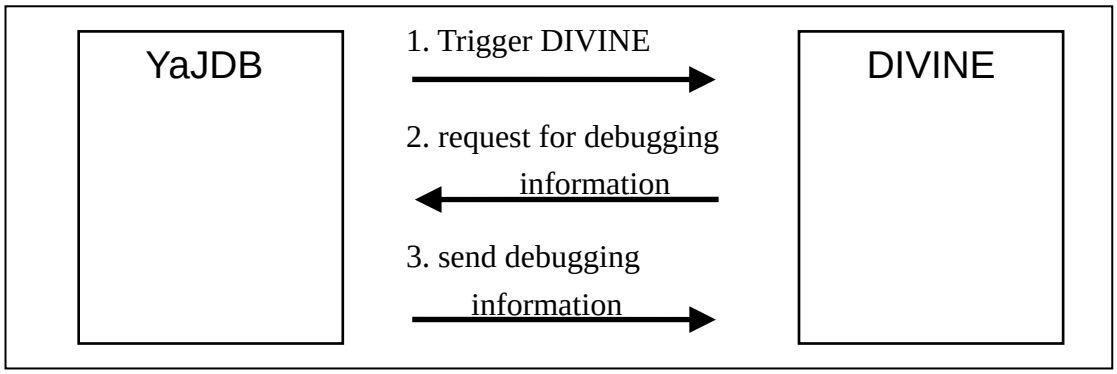

Figure 3-2: visualization starts with YaJDB

3.3.1 Visualization starts with DIVINE

While debugging a program (called a “debuggee”) with YaJDB, a user firstly loads the debuggee into YaJDB. Then, DIVINE connects to YaJDB via TCP/IP and executes remote debugging. Even DIVINE and YaJDB are both on local machine. After establishing the connection, DIVINE sends commands to YaJDB, i.e., get the structure of a class and watching values of variables, and receives information from YaJDB. After receiving the debugging information, DIVINE visualizes the information.

Complete one of variable visualization by a series of command. If a user

watches one variable with DIVINE, DIVINE first sends one command to YaJDB to get the type of a variable and the rest commands DIVINE sends to YaJDB depends on the type of variables. If type of a variable is primitive type, DIVINE sends command “show” to get the value of a variable. If type of a variable is class type, DIVINE sends command “struct” to get the structure of a class and then sends rest commands to get the detail of a variable of a class type recursively. If type of a variable is array type, DIVINE sends command to get the components of an array according to the component type of an array type. If component type is primitive type, DIVINE sends command “show” to get the values of components. But, if component type is reference type, DIVINE will send commands recursively to get the detail according to what kind of reference type.

Let’s take the following program segment for instance:

```
1 public class node {
2       int element;
3       node next ;
4   node(int a, node tail){
5     element = a;
6     next = tail;
7 }}
8 public class list{
9 public static void main(String[] args) {
10        node head;
11        head = new node (10,null);
12        head = new node(20,head);
13        ……
14    }}
```

A user debugs this program and set one breakpoint at line 12. Suppose a user wants to visualize the variable “head” while the program stops at line 12. The communication process is shown in Figure 3.3 and return message detail is shown in Table 3-3.

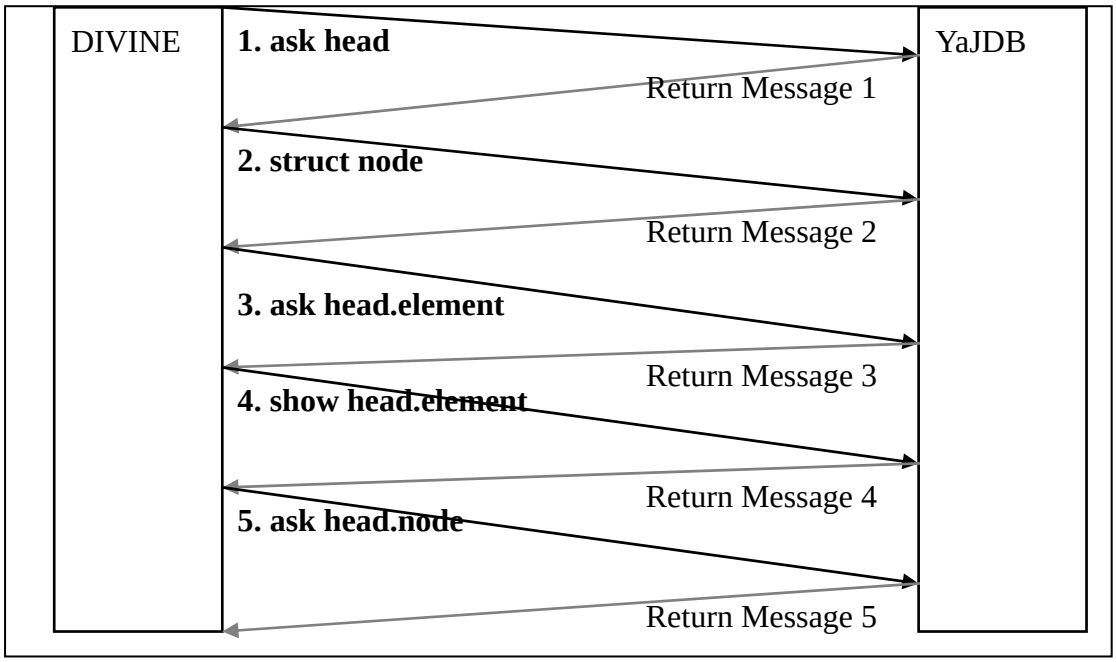

Figure 3-3: communication process

Table 3-3 : return message

Return Message	MESSAGE CONTENT
1	<varname>head<reftype>node<id>100
2	<structure>node<attnum>2<pritype>int<attname>element<reftype>node<attname>next
3	<varname>head.element<pritype>int
4	<varname>head.element<value>10
5	<varname>head.node<Null>null

3.3.2 Visualization starts with YaJDB

We provide another choice to execute DIVINE so that a user can type one

instruction with YaJDB to trigger DIVINE. After triggering DIVINE, the rest interaction is the same with the above section. The instruction we design for triggering DIVINE is “visual [variable name]”. After typing this instruction at YaJDB, YaJDB builds connection with DIVINE, and then visualization starts.