Tag Handling Ideas

# Tag Interpretation Using Reflection

The BrowserMonkey program obviously needs to interpret a large number of tags and do this process in a modular way but most importantly efficiently. You could create a html render by just using a massive if-else statement but this would be hard to debug and horrible to reuse or update (for example to new HTML standards). This is why we have decided to use reflection.

## In Theory

Using an external file (such as a config file) that is easily editable to lookup required information for use while the program is running. Another way of doing this is having the information in the file loaded into the program when is it initially run. This allows for the addition of useful modular features to a program.

“Reflection is the process by which a computer program can observe and modify its own structure and behaviour.” - Wikipedia

Reflection allows the programmer to use multiple different classes and methods in a generic way, perfect for something like html rendering where you have many different processes used to draw the code. Reflection is also very handy for creating a modular piece of software such as a tag interpreter.

Essentially when you instantiate a class using a reflection method you don’t need to know what the class is or what methods it has, the reflection implementation will have methods that will allow you to grab any methods that class has and invoke the one you need.

## In Java

Now I will attempt to give a brief introduction of how these techniques can be implemented in Java.

### External Files

To access external files in java you can use the BufferedReader class. Here is some example code:

Code:

try {

/\*\*Read the specified file\*/

BufferedReader exampleReader = new BufferedReader(new FileReader("C:\\example.txt"));

/\*\*Separate the file by line\*/

String currentLine;

while ((currentLine = exampleReader.readLine()) != null){

//Do stuff with this line

}

} catch (FileNotFoundException ex) {

System.out.println("File not found");

} catch (IOException ex){

System.out.println("IO ERROR");

}

This Code will allow you to read each line in a file and do some useful code to each line.

### Reflection

Java has a built in class for handling reflection: java.lang.reflect

Here is some code for a simple example of reflection. Code:

import java.lang.reflect.\*;

public class DumpMethods {

public static void main(String args[])

{

try {

Class c = Class.forName(args[0]);

Method m[] = c.getDeclaredMethods();

for (int i = 0; i < m.length; i++)

System.out.println(m[i].toString());

}

catch (Throwable e) {

System.err.println(e);

}

}

}

This code will allow you to enter any class name into the command line arguments and then it will output a toString of each method that class contains.

## In BrowserMonkey Browser

To apply the above methods in the BrowserMonkey Browser we should use a Class for each HTML tag that contains code that can be used for painting the component related to the current tag. Perhaps we could create an abstract class or an interface so we can make the whole process a lot more generic.

There will be an external file that we can use to relate the tag we’re looking at to the mini Tag Classes then we will be using reflection to get the necessary paint methods over to the renderer so that it can build the required component based on the tag.

### Pseudo Code

This is how I would see the general idea of the code:

For each documentnode:

look up Tag in external file to retrieve TagClassName

Class currentTagClass = Class.forName(TagClassName)

RenderNode newRenderNode = currentTagClass.paint()

pass the RenderNode where it needs to go