MWP DSL

meili-inc Yujie-Liu

outline

- Work of DSL
- Dev status quo in meli-inc for client and server
- Industry Status
- Target of DSL
- Challenge of DSL
- Practice of DSL

Work of Dsl

- Solution of mobile client and server separation based on data
- combine the code of the control layer of server and view layer of client
- for the assembly、stitching and conversion of business data for wireless terminal
- embedded in the MWP of meili-inc

dev status of mobile client

- server response the control layer, one activity dependent on a big response of a interface (major)
- mobile client combine the view layer itself, one activity dependent on several interface and callbacks nested

dev status of server

- process a larger amount of trivial business logic related of view layer(major)
- focus on the module layer itself with little adaptation

Industry status

- fackbook GraphQL focus on the new way of query data for client based on data itself
- ali use nodejs as middle layer
- meli-inc costa for pc template rendering

Engineering Target of DSL

- Force mobile client mvc , avoid callbacks nested
- h5, android,IOS code reuse
- mobile client and server separation, reduce dev cost, mobile dev focus on the control and view layer while server dev focus on the module itself
- limited DSL for easy learning and easy maintain and safe
- the control layer logic could be modified by DSL without dependent on release version of mobile client

Tech Target of DSL

- Hot update of DSL
- bagpipe to improve performance such as lazy load or better fault tolerance
- interface asynchronous and parallel of DSL to improve the performance
- support the components, and the landing way of micro-service

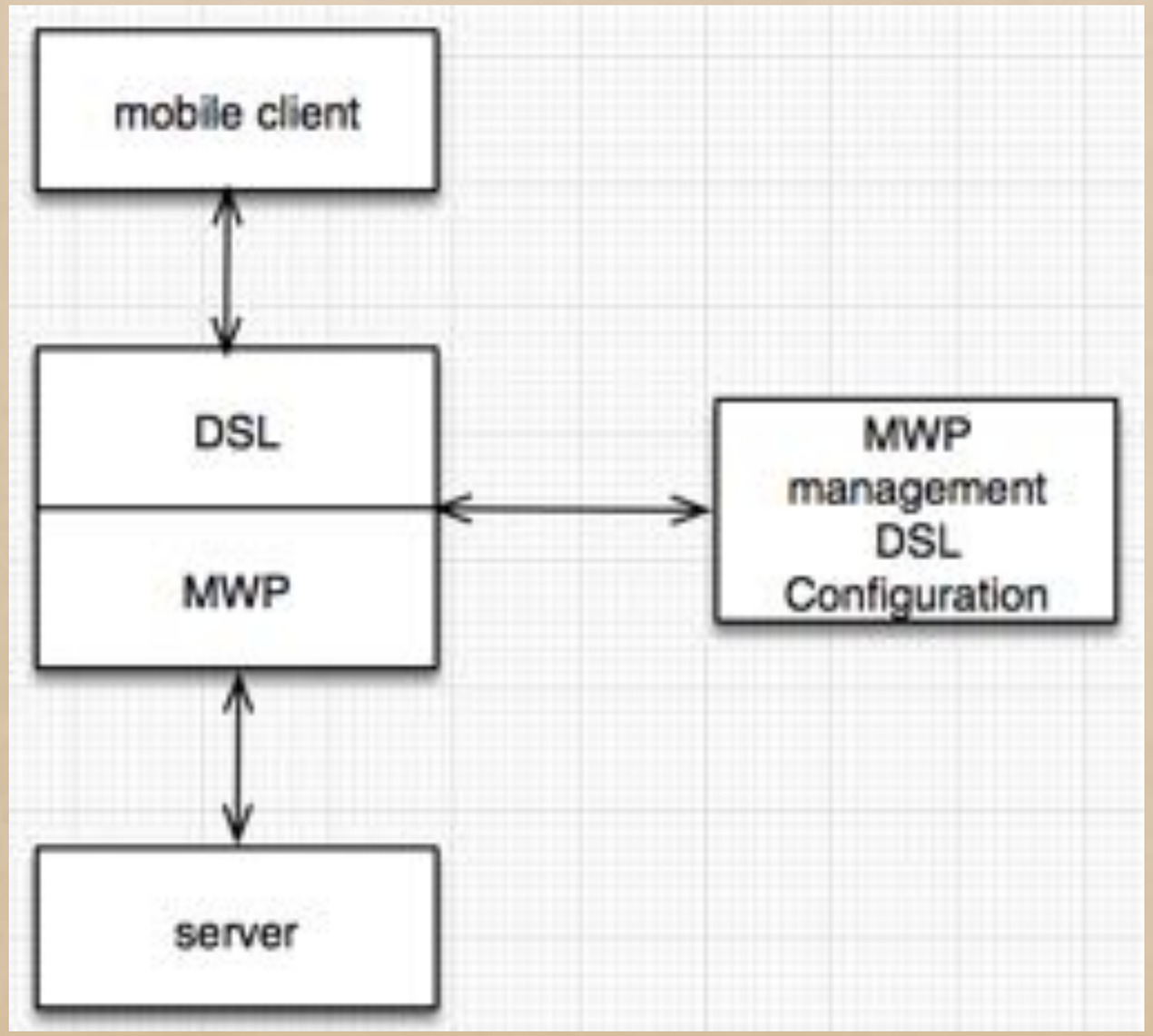
Challenge of DSL-Business

- Complex business scenarios, one dal interface has N map interface with any random combine and callback situation
- limited but enough ability of DSL
- easy use of DSL

Challenge of DSL-performace&stability

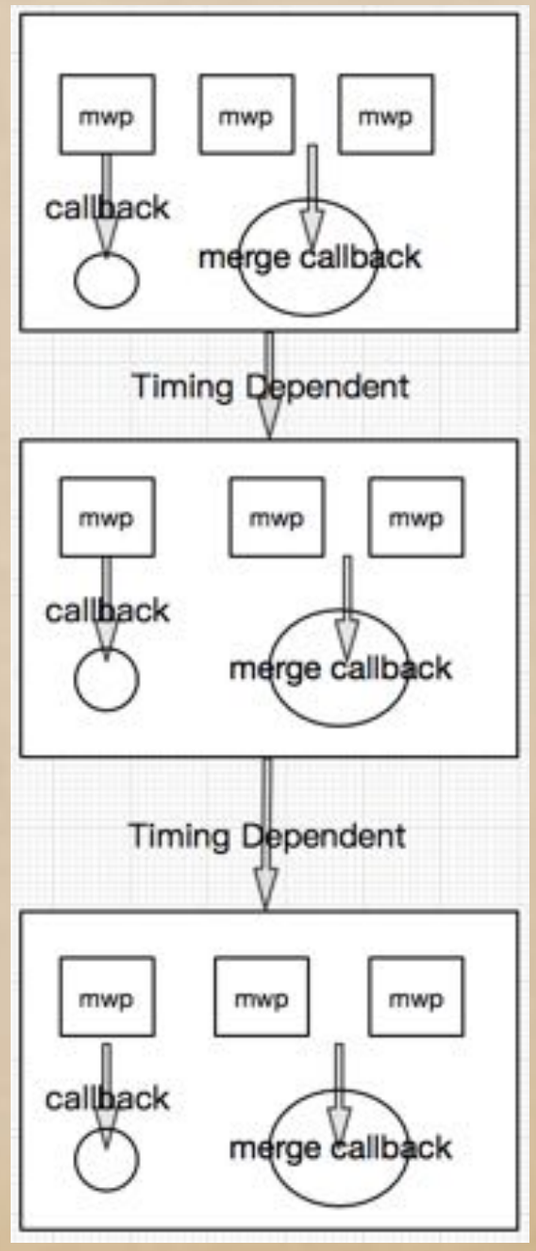
- Load pressure from lightweight map forwarding to multiple map combine and compute
- Complex of Coding、debug、thread model and dispatch with Full asynchronous for non-blockin
- Feature of Mwp(high stability, high qps, low rt, performance problems will be magnified)

Position of DSL



Business essence of DSL

- N interface random combination
- M flush to the client (M > 1 is bigpipe support in the future)
- T isolate callback (include error and exception)
- 3 basic atomic composition (isolate, merge, timing dependent)



DSL Schema


```
{
  "dslApi": "dsl.test.1",
  "dslV": "1",

  "stepList":[
    {
      "mergeList":[
        {
          "mwpList":[
            {
              "mwpApi": "mwp.PetStore.helloWorld",
              "mwpV": "1"
            }
          ],
          "isFlush":true,
          "mergeCallback": "import com.alibaba.fastjson.JSONObject;\nimport
        }
      ]
    }
  ]
}
```

DSL client frame

- multiple flush key process their own callback
- flush key and bagpipe decoupling
- flushkey isolation , better isolation exception process


```
"payload": {
  "ret": "SUCCESS",
  "api": "dsl.test.2",
  "v": "1",
  "data": {
    "flushkey4": {
      "ret": "SUCCESS",
      "data": {
        "height": 4.0
      }
    },
    "flushkey2": {
      "ret": "SUCCESS",
      "data": {
        "height": 4.0
      }
    },
    "flushkey1": {
      "ret": "SUCCESS",
      "data": {
        "height": 4.0
      }
    }
  }
},
```

performance Optimization

- full synchronized and thread dispatch (rxjava, net eventloop, single thread to process multiple callback to avoid concurrent and thread copy)

- hige performance groovy integrated (optimization of gc, permgen, code cache and execute efficiency)

groovy integrated

- reason for using groovy as the target language

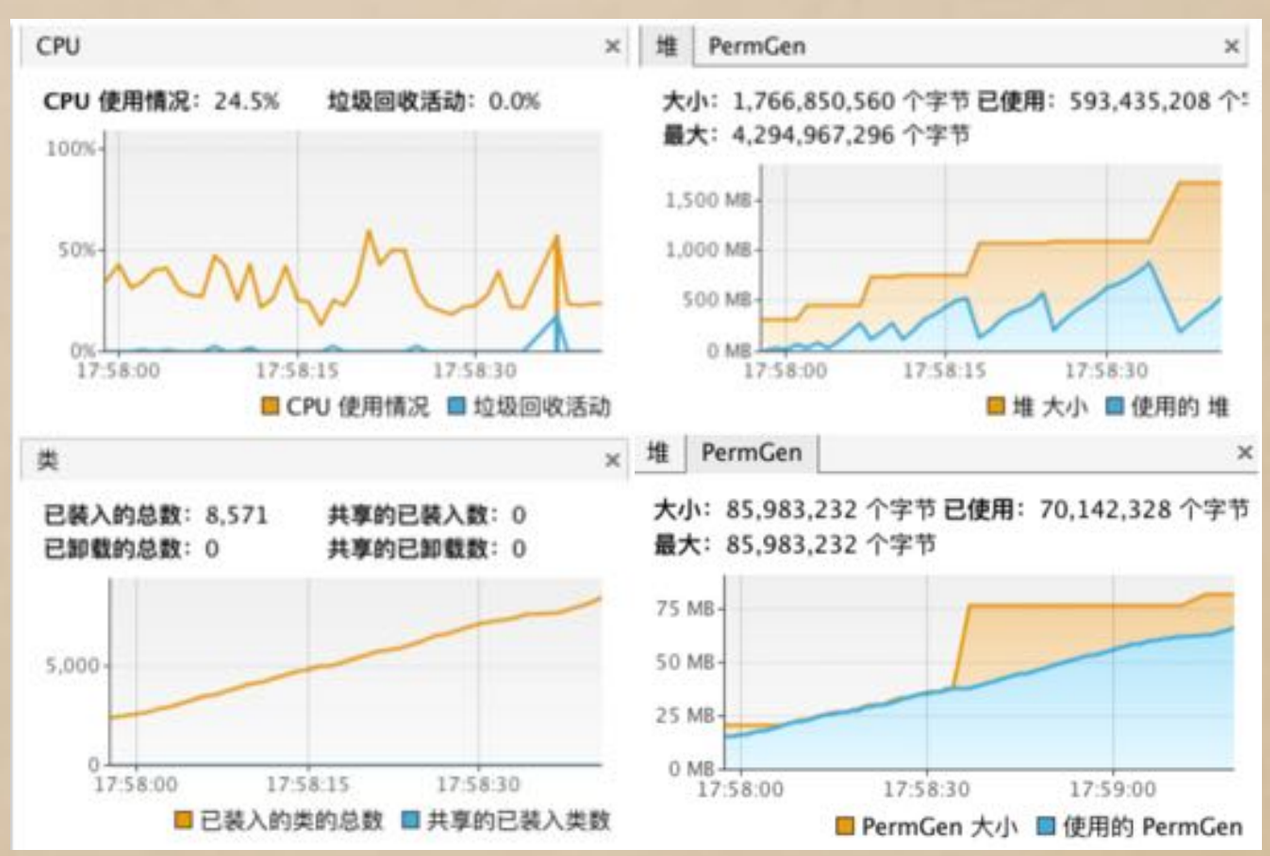
1. friendly for integration with java
2. **easy** deployment and maintain (running on jvm)
3. **performance** approaching to java and without gc problem after optimization
4. easy learning
5. mature solution for using groovy to build dal

groovy integrated

- three methods

GroovyShell

```
GroovyShell shell = new GroovyShell();
long start = System.currentTimeMillis();
for(int i = 0; i < 100000; i++){
    String t = "println 'Hello Groovy !'";
    Object obj = shell.evaluate(t);
}
long end = System.currentTimeMillis();
System.out.println(end-start);
```



groovy integrated

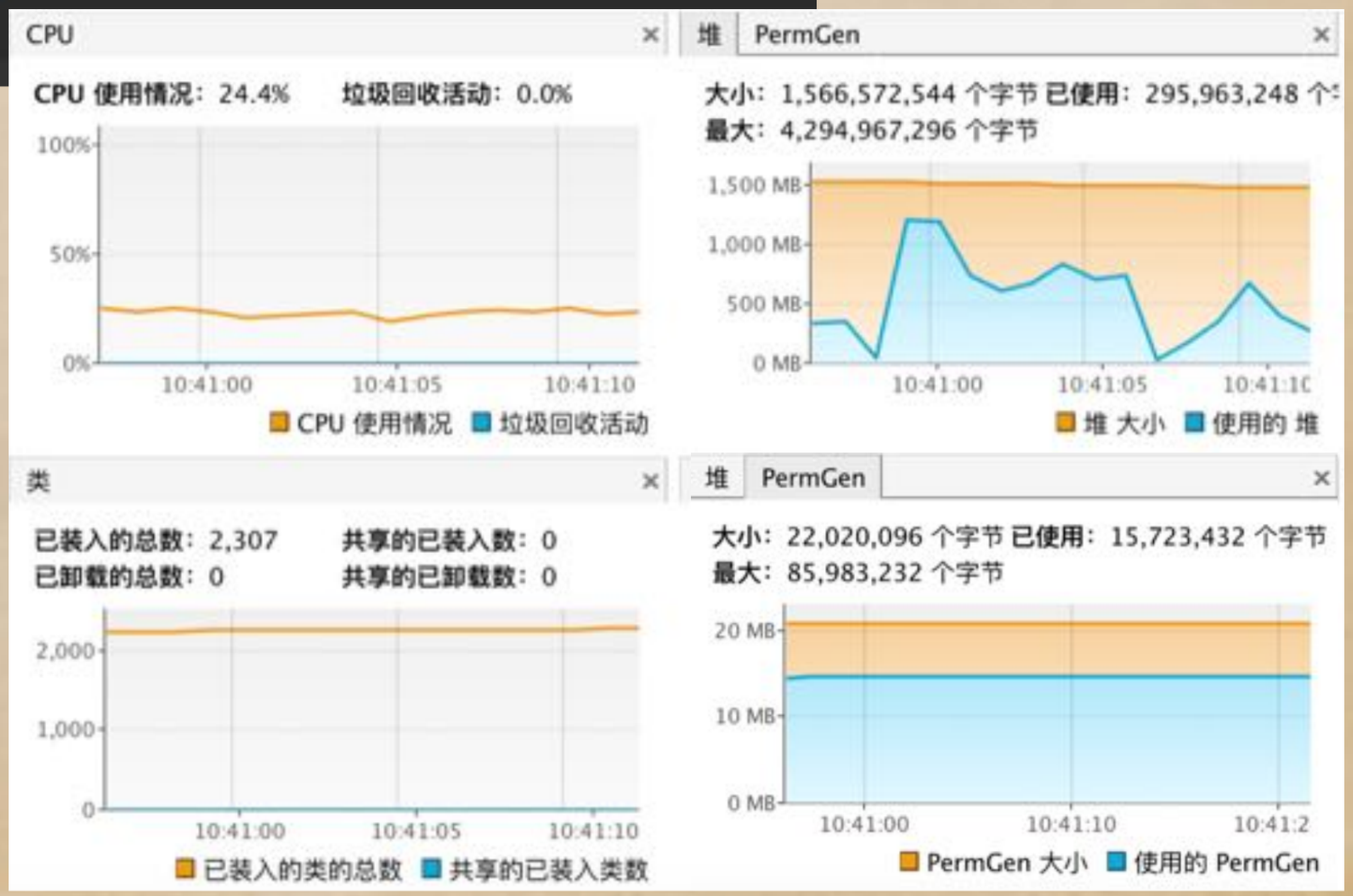
GroovyClassLoader

```
GroovyClassLoader loader =  new   GroovyClassLoader(Thread.currentThread().getContextClassLoader(), config);
//config.getOptimizationOptions().put("indy", true);

Class<?> clazz = loader.parseClass( new File( "/USERS/salah/Downloads/workspace/hello.groovy" ));
GroovyObject clazzObj = (GroovyObject)clazz.newInstance();
long start = System.currentTimeMillis();
for ( int   i= 0 ;i< 10000 ;i++){
    clazzObj.invokeMethod( "test1" ,  null );

}
long end = System.currentTimeMillis();
System.out.println(end-start);
```

```
def test1(){
    def j = 0
    def i = 0
    while(i < 100000){
        j = j + i
        i++
    }
    print j
}
```



groovy integrated

GroovyScriptEngine

```
String[] roots = new String[]{"/Users/salah/Downloads/workspace"};
GroovyScriptEngine engine = new GroovyScriptEngine(roots);
long start = System.currentTimeMillis();
for(int i = 0; i < 10000; i++) {
    Class scriptClass = engine.loadScriptByName("hello.groovy");
    Object scriptInstance = scriptClass.newInstance();
    scriptClass.getDeclaredMethod("test1", new Class[]{}).invoke(scriptInstance, new Object[]{});
}
long end = System.currentTimeMillis();
System.out.println(end-start);
```

conclusion of 3 methods

- GroovyShell is not good enough, it will compile and load the groovy class every time when execute the groovy function, which will lead to the gc ` permgem problem as well as low performance, which better not use.

- GroovyClassLoader has optimization in the new version which will cache the class, but you should do the reload by yourself

- GroovyScriptEngine add the function of dependency management and reload based on GroovyClassLoader, and it will check whether need reload overtime, so the performance is not as good as GroovyClassLoader itself.

groovy integrated

- Target of optimization , native java (50 times)
- both compile as class and load to codecache,but why?

```
public class hello2 extends Script {
    public hello2() { CallSite[] var1 = $getCallSiteArray(); }

    public hello2(Binding context) {
        CallSite[] var2 = $getCallSiteArray();
        super(context);
    }

    public static void main(String... args) {
        CallSite[] var1 = $getCallSiteArray();
        var1[0].call(InvokerHelper.class, hello2.class, args);
    }

    public Object run() {
        CallSite[] var1 = $getCallSiteArray();
        Object j = Integer.valueOf(0);

        for(Object i = Integer.valueOf(0); ScriptBytecodeAdapter.compareLessThan(i, Integer.valueOf(100000)); i = var1[2].call(i)) {
            Object var4 = var1[1].call(j, i);
            j = var4;
        }

        return var1[3].callCurrent(this, j);
    }
}
```

```
public class InnerJava {
    public InnerJava() {
    }

    public int test() {
        System.out.print(11);
        return 22;
    }

    public void test1() {
        int var1 = 0;

        for(int var2 = 0; var2 < 100000; ++var2) {
            var1 += var2;
        }

        System.out.println(var1);
    }
}
```

- The reason is groovy is the dynamic language, runtime loading

groovy integrated

- method of performance optimization

InvokeDynamic instruction (keep dynamic , JDK7+)

```
<dependency>
    <groupId>org.codehaus.groovy</groupId>
    <artifactId>groovy-all</artifactId>
    <version>2.4.4</version>
    <classifier>indy</classifier>
</dependency>
```

```
CompilerConfiguration config = new CompilerConfiguration();
config.getOptimizationOptions().put("indy", true);
GroovyClassLoader loader =  new   GroovyClassLoader(Thread.currentThread().getContextClassLoader(), config);


Class<?> clazz = loader.parseClass( new File( "/USERS/salah/Downloads/workspace/hello.groovy" ));
GroovyObject clazzObj = (GroovyObject)clazz.newInstance();
long start = System.currentTimeMillis();
for ( int   i= 0 ;i< 10000 ;i++){
    clazzObj.invokeMethod( "test1" ,  null );

}
long end = System.currentTimeMillis();
System.out.println(end-start);
```

groovy integrated

compilestatic

(without dynamic)

```groovy
import groovy.transform.CompileStatic

@CompileStatic
def test1(){
    def j = 0
    def i = 0
    while(i < 100000){
        j = j + i
        i++
    }
    println j
}
```

```java
import ...

public class hello extends Script {
    public hello() { CallSite[] var1 = $getCallSiteArray(); }

    public hello(Binding context) {
        CallSite[] var2 = $getCallSiteArray();
        super(context);
    }

    public static void main(String... args) {
        CallSite[] var1 = $getCallSiteArray();
        var1[0].call(InvokerHelper.class, hello.class, args);
    }

    public Object run() {
        CallSite[] var1 = $getCallSiteArray();
        return null;
    }

    public Object test1() {
        int j = 0;

        for(int i = 0; i < 100000; ++i) {
            int var3 = j + i;
            j = var3;
        }

        ((hello)this).println(Integer.valueOf(j));
        return null;
    }
}
```

groovy integrated

- other notes

1. GroovyClassLoader should be isolation (new GroovyClassLoader when groovy add or update)

2. jvm tuning (enlarge CodeCache , open classunloading -XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled)

result of optimization

	GroovyScriptEngine	GroovyClassLoader	Java native
no optimization	30983ms	29363ms	608ms
InvokeDynamic	\	11267ms	\
CompileStatic	4020ms	1009ms	\

groovy frequent change-problem

- 300 groovy script，all change every 10s,permgen overflow，useless class without unloading

groovy frequent change-reason

- three conditions of class unloading

1. Instance generated by the class be collected (ok)

类名	实例数 [%]	实例数		大小	
testtest_74		0	(0%)	0	(0%)
testtest_74		0	(0%)	0	(0%)
testtest_74		0	(0%)	0	(0%)
testtest_74		0	(0%)	0	(0%)

2. class's class loader be unloaded (without unloading)

类名	实例数 [%]	实例数		大小	
groovy.lang.**GroovyClassLoader$1**		2,400	(0.4%)	57,600	(0.1%)
groovy.lang.**GroovyClassLoader**		1,200	(0.2%)	213,600	(0.5%)
groovy.lang.**GroovyClassLoader$InnerLoader**		1,200	(0.2%)	232,800	(0.6%)

3. class without any quote

groovy frequent change-reason

引用

字段	类型
this	GroovyClassLoader
delegate	GroovyClassLoader$InnerLoader
<classLoader>	testtest_276
klazz	ClassInfo
value	GroovyClassValuePreJava7$EntryWithValue
[0]	Object[]
[89]	Object[]
table	GroovyClassValuePreJava7$GroovyClassValuePreJava7Segment
[7]	AbstractConcurrentMapBase$Segment[]
segments	GroovyClassValuePreJava7$GroovyClassValuePreJava7Map
map	GroovyClassValuePreJava7
globalClassValue	ClassInfo
[549]	Object[]
elementData	Vector
classes	Launcher$AppClassLoader
scl (sticky class)	ClassLoader

```
/** Approximation of Java 7's {@link java.lang.ClassValue} that works on earlier versions of Java.
 * Note that this implementation isn't as good at Java 7's; it doesn't allow for some GC'ing that Java 7 would allow.
 * But, it's good enough for our use.
 *
 * @param <T>
 */
class GroovyClassValuePreJava7<T> implements GroovyClassValue<T> {
    private static final ReferenceBundle weakBundle = ReferenceBundle.getWeakBundle();
```

```
class GroovyClassValueFactory {
    /**
     * This flag is introduced as a (hopefully) temporary workaround for a JVM bug, that is to say that using
     * ClassValue prevents the classes and classloaders from being unloaded.
     * See https://bugs.openjdk.java.net/browse/JDK-8136353
     * This issue does not exist on IBM Java (J9) so use ClassValue by default on that JVM.
     */
    private final static boolean USE_CLASSVALUE = Boolean.valueOf(System.getProperty("groovy.use.classvalue", "IBM J9 VM".equals(System.getPropert

    private static final Constructor groovyClassValueConstructor;

    static {
        Class groovyClassValueClass;
        if (USE_CLASSVALUE) {
            try {
                Class.forName("java.lang.ClassValue");
                try {
                    groovyClassValueClass = Class.forName("org.codehaus.groovy.reflection.v7.GroovyClassValueJava7");
```

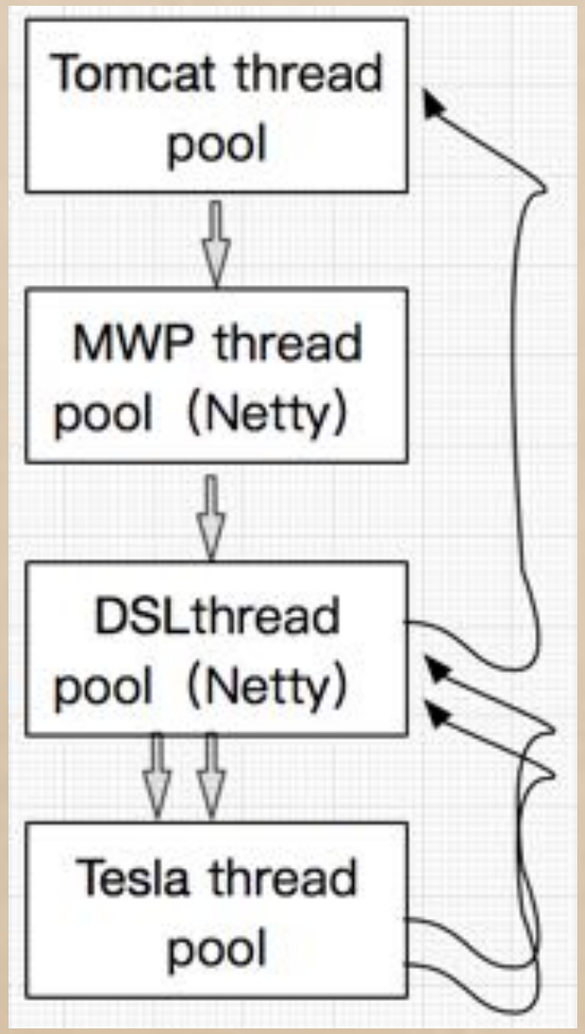
groovy frequent change-solution

1. defect of groovy version itself
   (ClassValue,on JDK7+ using version of 2.3 or 2.4 with ClassValue open ,like -Dgroovy.use.classvalue=true, which is default closed)

Stability

- thread isolation
- DSL interface isolation and fault tolerant
- switch for DSL
- beta release
- beta allocate request

DSL thread model



future

- dsl design and translate
- bigpipe
- graph-ql

Thank you