Velocity模板引擎的优化 实践

君山
GreenTeaJUG
2013-2-23

关于我

花名：君山
真名：许令波
博客：http://xulingbo.net
微博：@淘宝君山
邮箱：xulingbo0201@163.com
简介：2009年毕业加入淘宝，一直在做淘宝商品详情系统的团队做开发和性能优化方面的工作，开发过一个Sketch模板引擎、给developerworks 投稿获得过最佳作者，著有《深入分析Java Web技术内幕》一书。

DETAIL系统的优化历程

- 前端优化
  - assets合并
  - 整合页面中inline的js\css
  - combo合并css、js文件
  - BigRender（使用textarea，控制浏览器渲染节奏）
- 服务端优化
  - 将iframe改为jsonp调用
  - 建立异步系统 （JS触发加载）
  - Velocity模板优化（sketch框架）
  - 逐步去除DB依赖（各个C走Tair缓存）
- 网络优化
  - TCP初始拥塞窗口优化
  - 去除空格、TAB压缩字符串，减少页面大小
- 静态化
  - CDN化
- 今年要做的事
  - 交易链路优化
  - 欢迎加入我们的团队

摘要

1. 当前存在的问题
2. 解决问题的思路
3. 优化的原则和目标
4. 传统办法解决思路
5. 激进的解决问题的思路、具体实例介绍和成果
6. 遇到的一些问题介绍
7. Q&A

淘宝前台系统模板问题

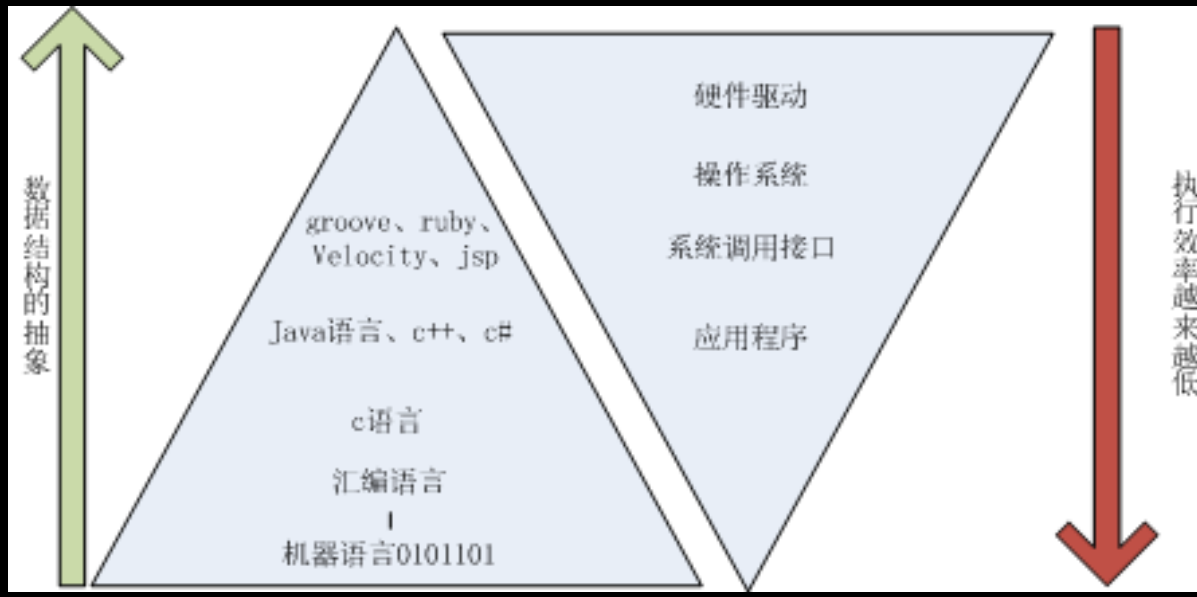
- 发现模板方面的问题
  1. Velocity是动态解释性语言，执行效率较差
  2. 页面复杂，反射调用非常多
  3. CPU压力较大，压力测试时CPU基本都达到80%左右，通过检测工具可以发现模板渲染占用了60%以上的CPU时间。
  4. 模版渲染占去大部分响应时间是前台系统的一个瓶颈，模板渲染时产生很多临时对象，对JVM的GC影响很大，导致系统频繁GC。
  5. 整个页面输出比较大，平均在80KB左右，大部分时间都在out.print。页面模板中空白字符比较多，浪费网络传输量。

解决问题的思路

- 针对性解决问题
  1. 将Velocity模板直接转成Java类去执行，将Velocity语法转成Java语法
  2. 将方法的反射调用转成直接Java原生方法调用
  3. 减少页面大小，删除空行等无效字符输出
  4. 将页面中的字符转成字节输出减少编码转换

优化的原则

- 三角结构
- 减少翻译的代价、一步到位
- 将变的转化为不变的
- 对不变的做预处理



优化目标

1. 试图减少代码量，从而减少代码的执行时间
2. 减少临时对象，降低内存开销
3. Velocity静态化，类型确定
4. 动态编译，类似JIT技术

在VELOCITY上解决问题

减少树的总节点数量

```
#set($one=-1)
#set($pageId = "$!page.pageId")
#set($pages = $tbStringUtil.getInt("$pageId")+$one)
#set($offsets=$pages*($count+$rightCount))
```

优化成

```
#set($offsets=($!page.pageId - 1)*($count + $rightCount))
```

这样可以减少很多语法节点。

在VELOCITY上解决问题

减少树的总节点数量

```
$parent.getChildren().getSon().getName()
$parent.getChildren().getSon().getAge()
```

优化成

```
#set($son=  $parent.getChildren().getSon())
$son.getName()
$son.getAge()
```

这样可以减少方法的反射调用。

在VELOCITY上解决问题

1. 减少宏的调用，因为每次调用都要都要reload,而 Velocity针对macros的自动reload，采用了同步排他锁进行控制,比较影响性能
2. 减少vm模板的检查频率，设置**modificationCheckInterval**
3. **页面Cache减少生成JJTree的频率**
4. MapGetExecutor改造，用clazz.isAssignableFrom(Map.class)替代

```
Class [] interfaces = clazz.getInterfaces();
for (int i = 0 ; i < interfaces.length; i++)
{
    if (interfaces[i].equals(Map.class))
```

终极武器—SKETCH替代VELOCITY

1. vm模板如何被转成java类
2. 方法调用的无反射优化
3. 字符输出改成字节输出

要解决的问题

1. 如何将Velocity的语法转成Java语法
2. 如何构建Java的代码结构
3. 如何生成Java类
4. 如何编译Java类
5. 如何执行Java类
6. 出错处理
7. 如何和其他框架结合起来

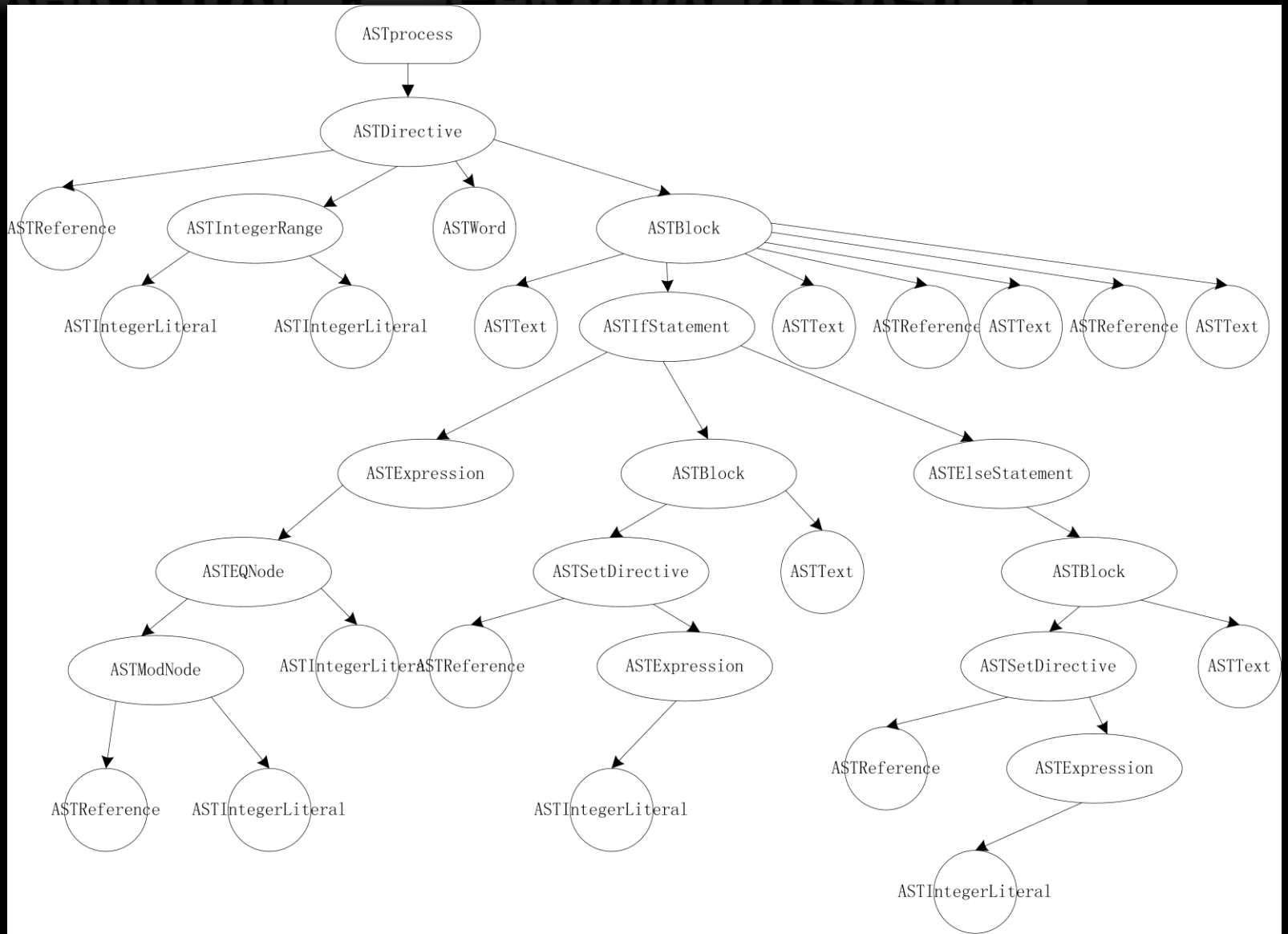
VM模板如何被转成JAVA类

仍然沿用Velocity中将一个vm模板解释成一棵AST语法树，但是重新修改这棵树的渲染规则，我们将重新定义每个语法节点生成对应的Java语法，而不是渲染出结果。

在SimpleNode类中重新定义一个generate方法，如下：

```
public Object generater(Object data, Writer writer) throws IOException,
    ParseException {
      SketchCompilationContext context = (SketchCompilationContext) data;
      int i, k = jjtGetNumChildren();
      for (i = 0; i < k; i++) {
          StringWriter spWriter = new StringWriter();
          jjtGetChild(i).generater(data, spWriter);
          writer.write(spWriter.toString());
      }
      return data;
  }
```

Velocity语法转成对应的Java语法



Velocity语法转成对应的Java语法

```
private Object _foreach_3414368_43072917262352(final PageContext pageContext, final I _I) throws Exception {
    final ContextAdapter context = pageContext.getContext();
    final PageWriter out = pageContext.getOut();
    Iterator _it = _COLLE((_I.exampleDO == null ? null : ((Mode) _I.exampleDO).getItemList()));
    int _VelocityCount = 1;
    while (_it.hasNext()) {
        Object _i = _it.next();
        pageContext.addForVarsDef("_i", _i);
        pageContext.addForVarsDef("_VelocityCount", _VelocityCount);
        out.write(_S0);
        if (EPUT.is(EPUT.eq(EPUT.mod(_i, 2), 0))) {
            context.put("str", "偶数");
            out.write(_S0);
        } else {
            context.put("str", "奇数");
            out.write(_S0);
        }
        out.write(_S0);
        out.write(_EVTCK(context, "$i", _i));
        out.write(_S1);
        out.write(_EVTCK(context, "$str", context.get("str")));
        out.write(_S2);
        _VelocityCount++;
    }
    return Boolean.TRUE;
}
```

方法调用的无反射优化

1. 大部分情况下$exampleDO.getItemList()方法调用这种调用类型都是确定的
2. 通过在模板渲染时跟踪变量$exampleDO的java类型就可以将$exampleDO.getItemList()的反射调用转成原生的java方法调用

如通过_TRACE()方法跟踪exampleDO的执行

```
Iterator _it = _COLLE(_TRACE("", "_I.exampleDO", "-209571699",
    context.get("exampleDO"),"getItemList", new Object[]{}));
```

最终直接转成

```
Iterator _it = _COLLE(( ExampleDO )(I.exampleDO).getItemList()));
```

渐进式方法调用的无反射优化

由于一个模板中一次执行并不能执行到所有的方法，所以一次执行并不能将所有的方法调用转变成反射方式。这种情况下就会多次生成模板对应的Java类及多次编译。

字符输出改成字节输出

1. 将模板中的静态字符串直接是out.write(_S0)输出，这里的_S0是一个字节数组，而vm模板中是字符串，将字符串转成字节数组是在这个模板类初始化时完成的。你可能有疑问，为何将字符串转成字节数组来输出，如果看过Java中文编码一章你就会知道，字符的编码是非常耗时的，如果我们将静态字符串提前编码好，那么在最终写Socket流时就会省去这个编码时间，从而提高执行效率。从实际的测试来看，这对提升性能很有帮助。
2. 删除空格、tab等无用字符串，减少java系统往外突出的字节数

优化的效果

页面大小	优化前 QPS	优化前 RT(ms)	优化后的 QPS	优化后 RT(ms)	提升%
47355	319.05	109.7	455.87	76.776	43%
48581	306.85	114.061	445.39	78.582	45%
55735	296.65	117.983	437.46	80.007	47%
63484	193.69	180.698	302.55	115.684	56%
83152	180.88	193.498	236	148.305	30%
92890	170.68	205.064	214.27	163.342	26%
99732	103.64	337.707	161.46	216.77	56%
144292	108.76	321.81	148.18	236.199	36%
67144	148.49	235.714	268.07	130.565	81%
79703	124.51	281.1	243.64	143.657	96%
92537	123.85	282.595	190.8	183.44	54%
127047	117.52	297.829	164.1	213.284	40%
129479	105.36	332.197	155	225.8	47%

麻烦的节点

1. #foreach难题---重复定义变量

```
#foreach($i in $list)
    #foreach($i in $list)
      $i
    #end
#end
```

2. #define难题---变量中的变量

```
#define( $hello )
        Hello $who
#end
#set( $who = "World!") $hello
```

3. #break、#stop等

遇到的一些问题

1. #foreach难题---重复定义变量

```
#foreach($i in $list)
    #foreach($i in $list)
      $i
    #end
#end
```

2. #define难题---变量中的变量

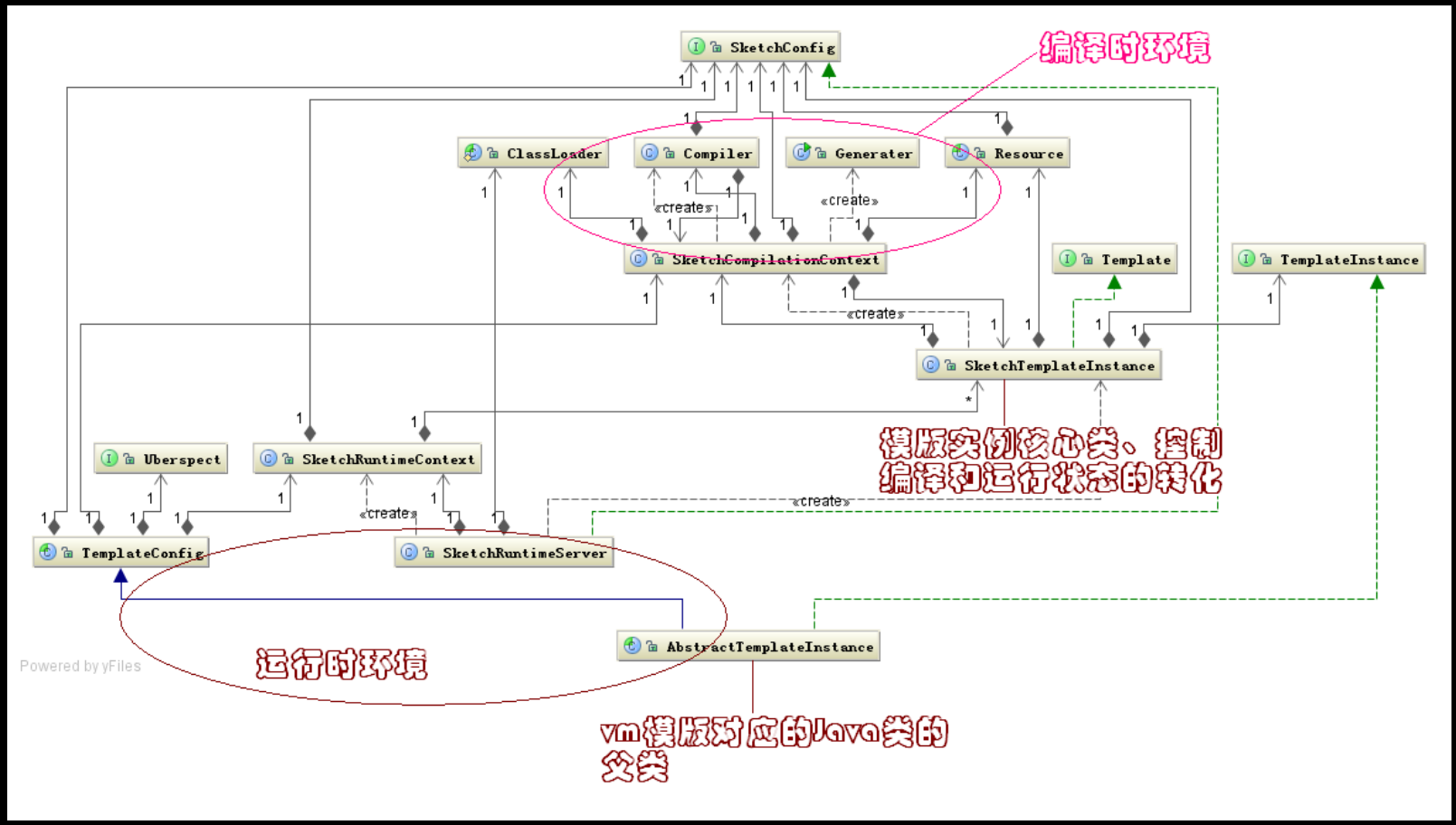
```
#define( $hello )
        Hello $who
#end
#set( $who = "World!") $hello
```

3. #break、#stop等

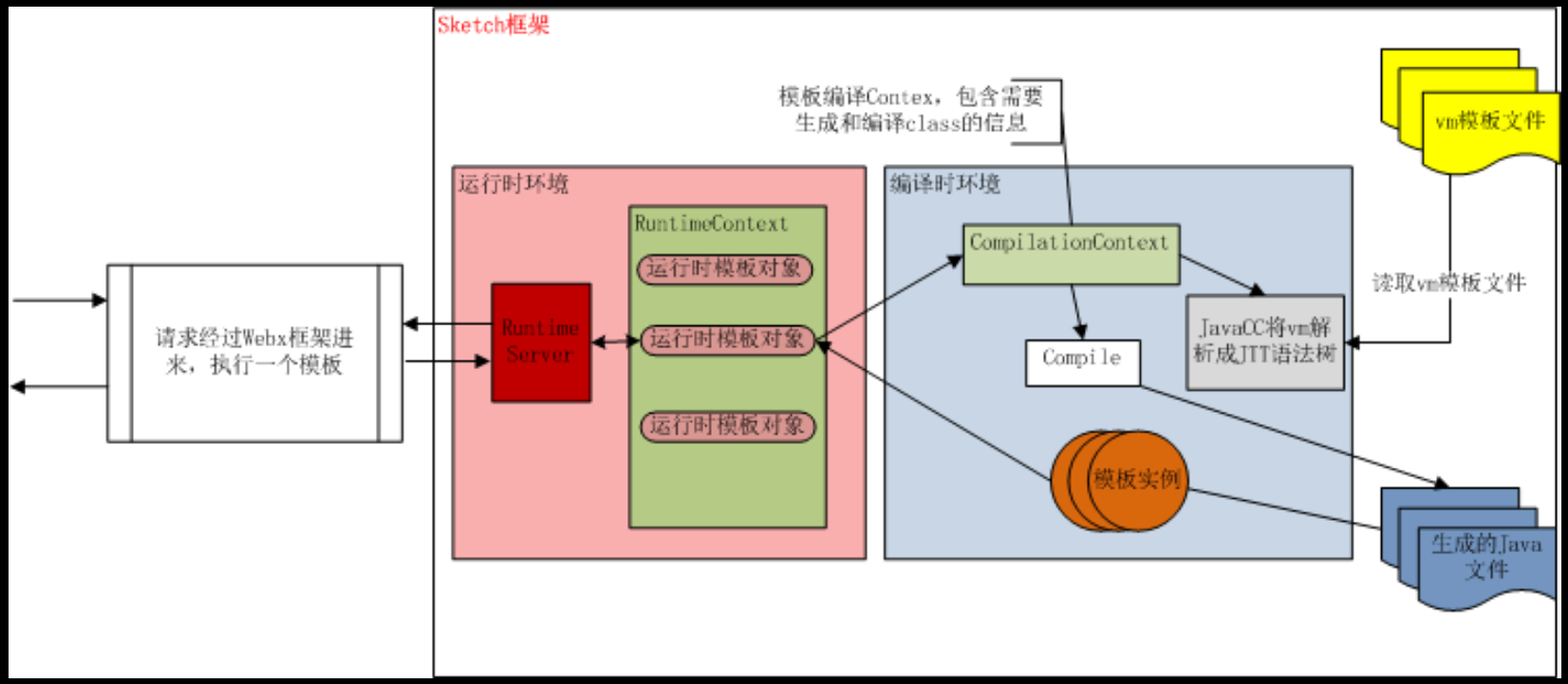
遇到的一些问题

1. 必须能够重复能够渲染语法树
2. 确定方法中的参数变量类型
3. 方法反射调用改成正常的方法调用：如$foo.var改成foo.getVar()/foo.get(“var”)
4. 构建新的classloader，重新编译的模版类重复加载到jvm中
5. JavaC编译器的限制

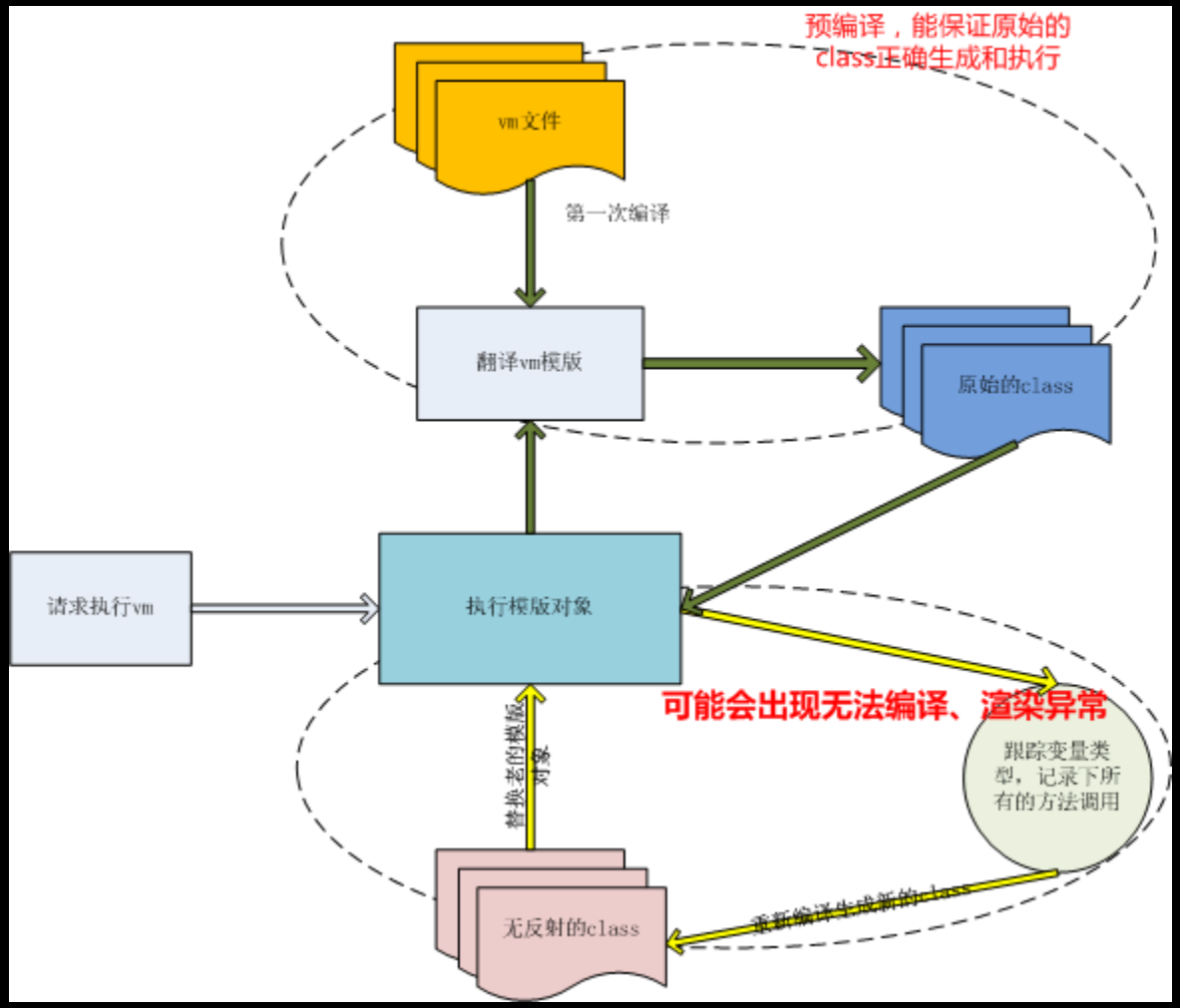
新的模版框架


SketchConfig
编译时环境
ClassLoader
Compiler
Generater
Resource
«create»
«create»
SketchCompilationContext
Template
TemplateInstance
«create»
SketchTemplateInstance
模版实例核心类，控制
编译和运行状态的转化
Uberspect
SketchRuntimeContext
«create»
«create»
TemplateConfig
SketchRuntimeServer
运行时环境
AbstractTemplateInstance
vm模版对应的Java类的
父类
Powered by yFiles

与MVC的整合


Sketch框架
模板编译Contex，包含需要
生成和编译class的信息
vm模板文件
运行时环境
RuntimeContext
运行时模板对象
运行时模板对象
运行时模板对象
编译时环境
CompilationContext
读取vm模板文件
JavaCC将vm解
析成JTT语法树
Compile
模板实例
生成的Java
文件
请求经过Webx框架进
来，执行一个模板
Runtime
Server

出错处理机制


预编译，能保证原始的class正确生成和执行
vm文件
第一次编译
翻译vm模版
原始的class
请求执行vm
执行模版对象
可能会出现无法编译、渲染异常
跟踪变量类型，记录下所有的方法调用
替换老的模版对象
无反射的class

更进一步优化

1. 模版的动静分离
2. 静态模版的CDN化
3. JS渲染orApache渲染

THANKS

一起交流.....