Rich feed 改造

Yubaofu xiaohanghu

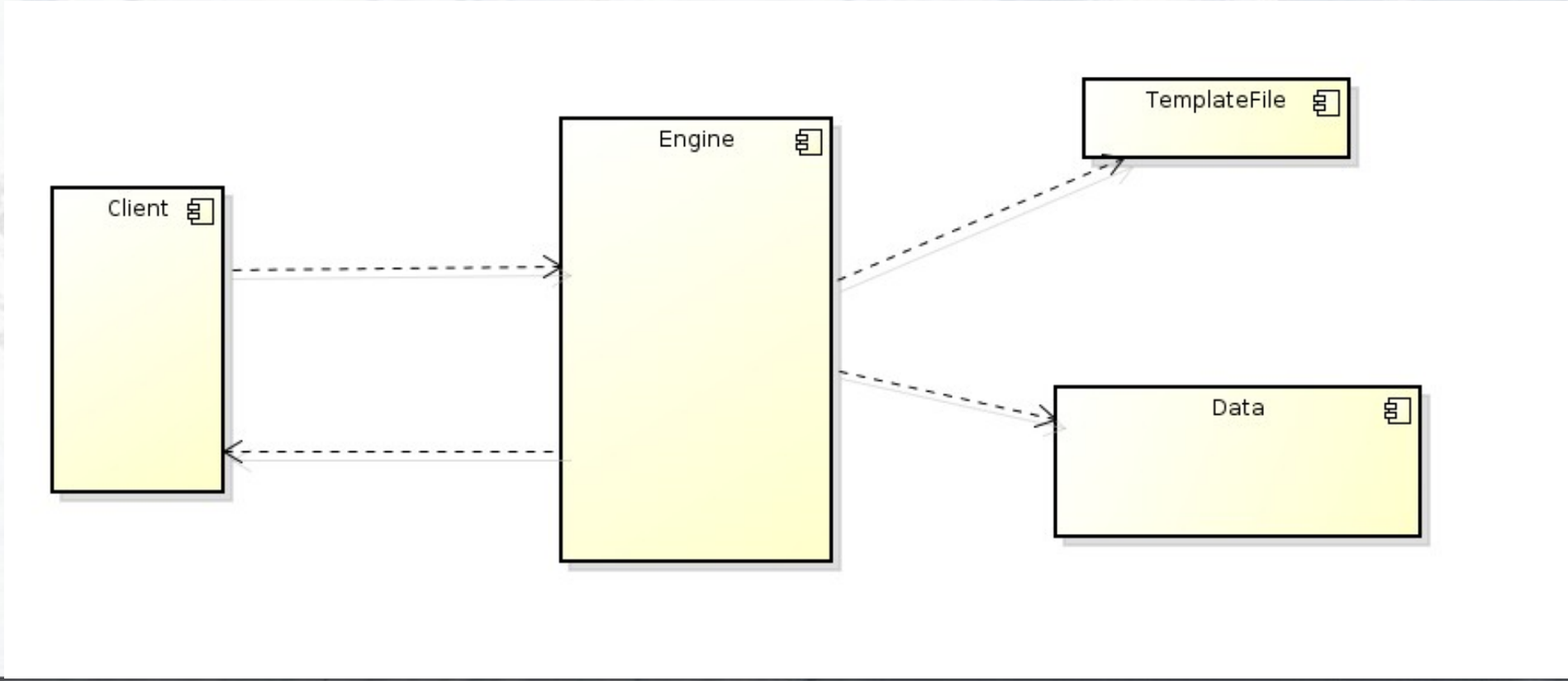
议题

- 改造总的目标
- 模板引擎设计
- 简化版的模板引擎介绍
- 初步目标
- 接下的工作

Rich feeds 改造总目标

- 渲染组件与业务解耦，可扩展、易维护
- 页面模板分解、细化。支持批量模板请求处理
- 丰富模板语法，支持 include 、 if 等语法。逐步优化渲染性能
- 独立的渲染服务集群
- 模板统一存储、管理。逐步支持模板在线更新、可视化编辑等功能
- 热加载模板、 前端 js 模板兼容

模板引擎功能



模板引擎组件

- 引擎
- 渲染
- 模板文件
- 语法

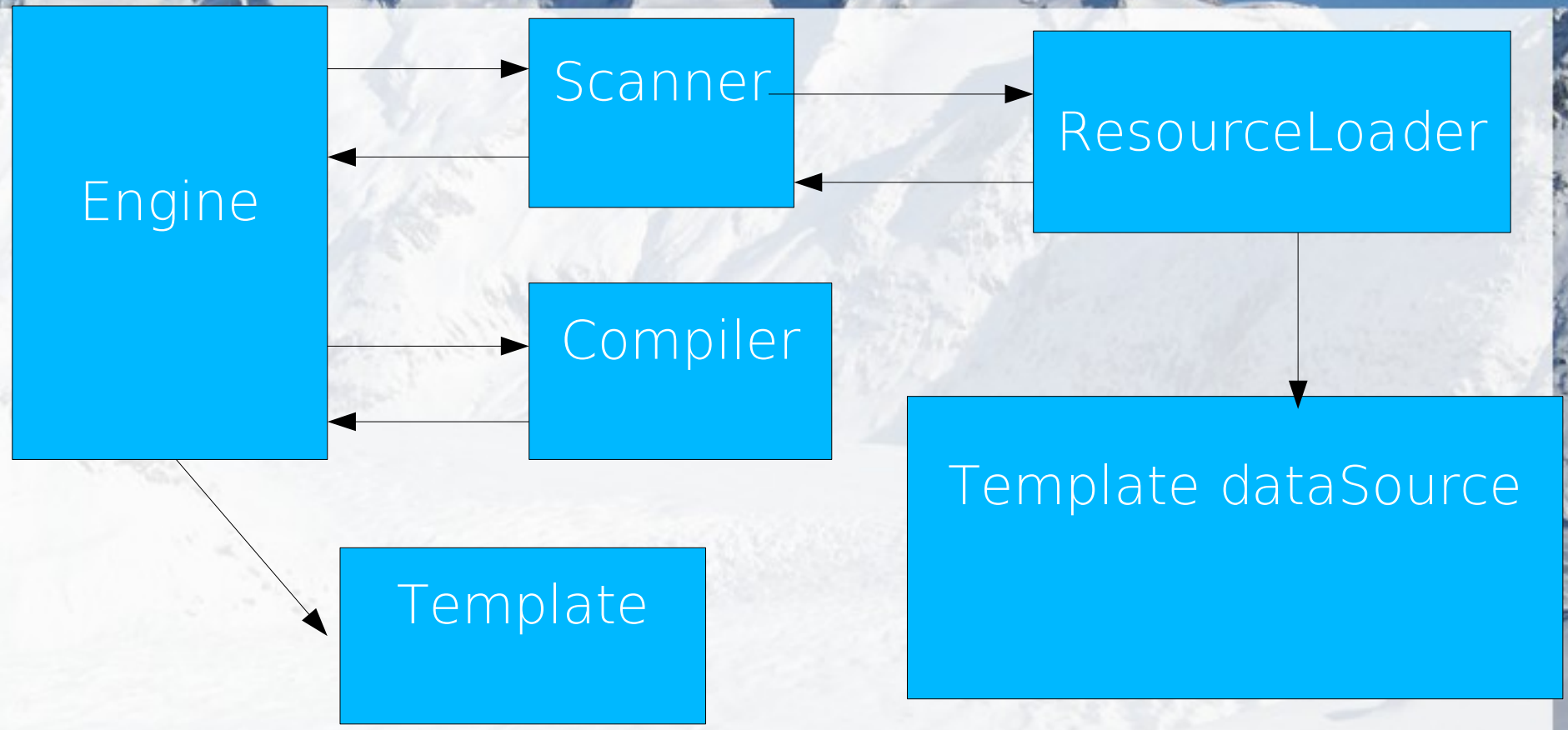
引擎实现

- ResourLoader 资源加载器
- Scanner 解析模板文件
- 生成 java 源代码
- 编译成 .class
- 缓存模板文件对应的 Template 对象

渲染实现

- 从引擎里取出对应的 Template
- render 生成 string

组件间关系


Engine
Scanner
ResourceLoader
Compiler
Template dataSource
Template

Engin 功能

- 生成并编译模板对应的 class 文件（逻辑上）
- 容器

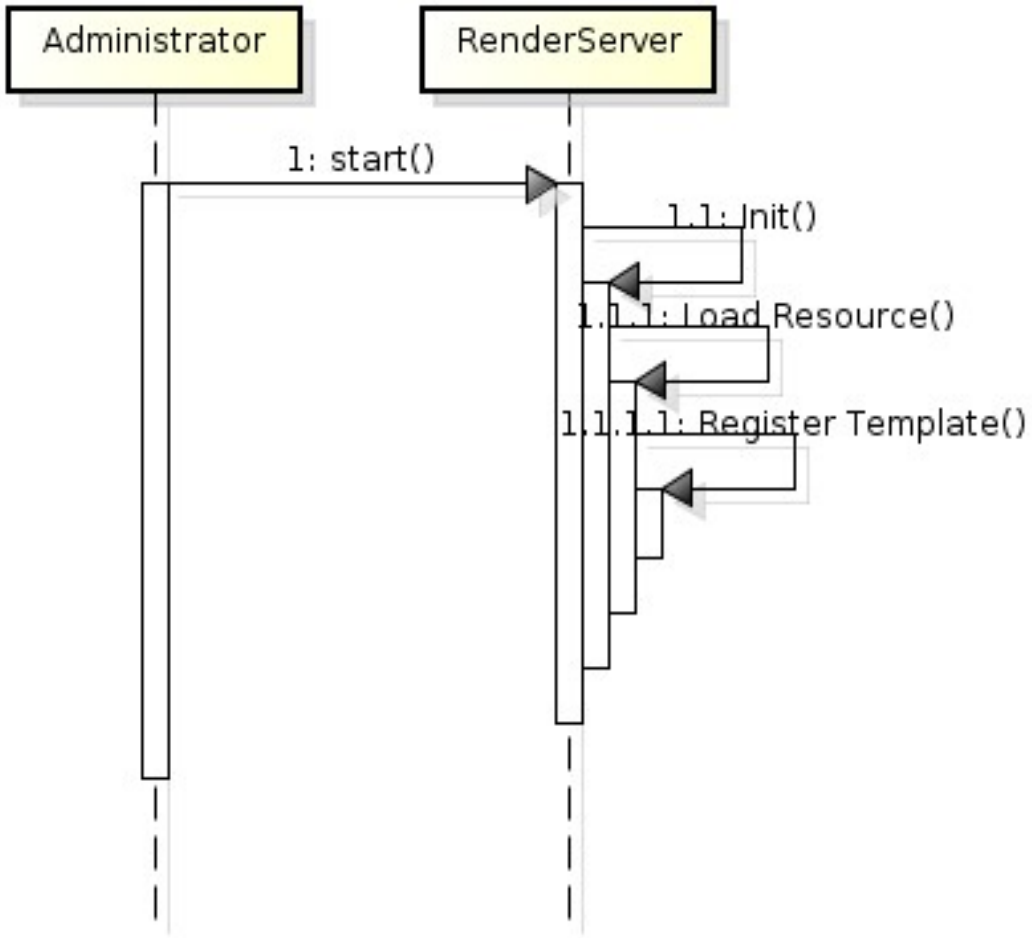
ResourceLoader 功能

- 提供统一的加载资源方法
- 可以从文件系统或 url 加载资源 (数据库)

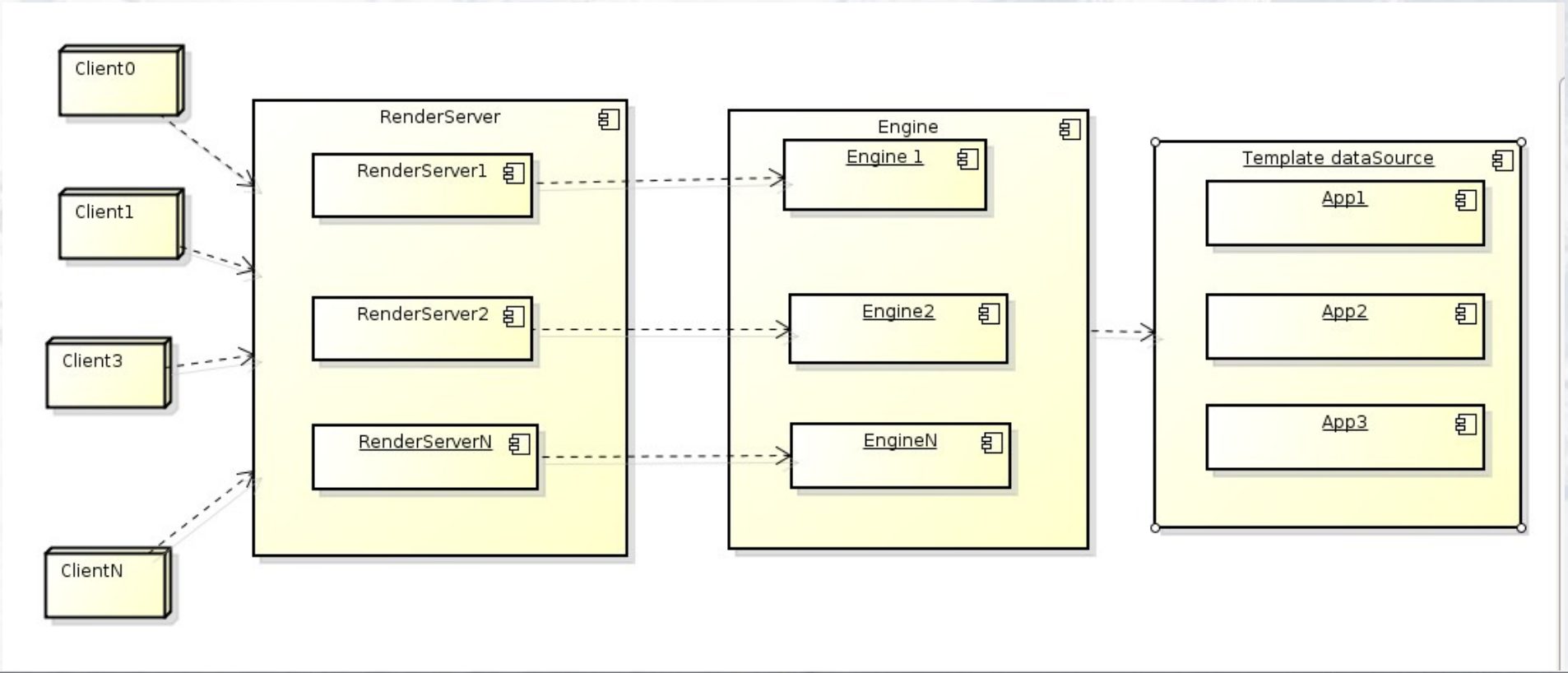
scanner 功能

- 解析模板文件
- 语法检查
- 标签解析器
- 生成 java source code(逻辑上)

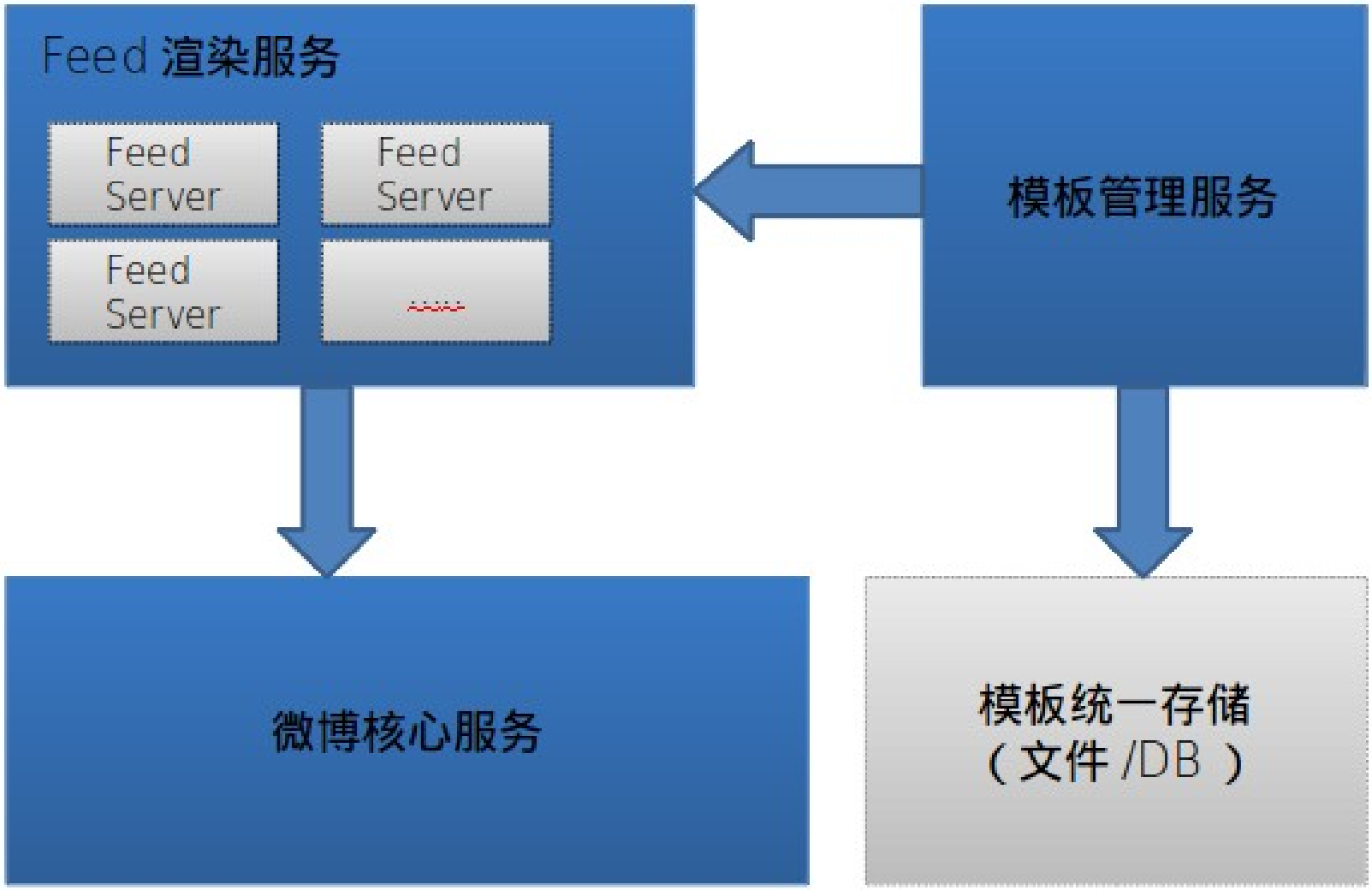
模板引引擎启动


Administrator
RenderServer
1: start()
1.1: Init()
1.1.1: Load Resource()
1.1.1.1: Register Template()

工作示意图


Client0
Client1
Client3
ClientN
RenderServer
RenderServer1
RenderServer2
RenderServerN
Engine
Engine 1
Engine2
EngineN
Template dataSource
App1
App2
App3

部署图


Feed 渲染服务
Feed Server
Feed Server
Feed Server
.....
模板管理服务
微博核心服务
模板统一存储
（文件 /DB ）

简化版引擎实现

- 已有一个初步实现上述的功能的模板引擎
- 从实现上来说，是上面描述的一个简化版
- 从功能上来，这个简化版可以满足咱们的要求
- 简化版的引擎没有生成 java 并生成 class
  而是在内存里生成要相应的数据结构
- 详见 模板引擎设计实现 .pdf

简化引擎 Demo

```
 */
public class EnginTest {

    public static void main(String[] args) {

        // 获取模板引擎
        TemplateEngine templateEngine = TemplateEngine.getDefaultTemplateEngine();

        // 获取模板输入流
        InputStream in = EnginTest.class.getClassLoader().getResourceAsStream("template/demo.html");
        InputStream include = EnginTest.class.getClassLoader().getResourceAsStream("template/include.html");

        // 通过模板输入流，构造模板
        Template template = templateEngine.createTemplate(in, "GBK");
        Template includeTemplate = templateEngine.createTemplate(include, "GBK");

        // 向templateManager注册模板
        TemplateManager templateManager = new ConcurrentTemplateManager();
        templateManager.addTemplate("demo", template);
        templateManager.addTemplate("test", includeTemplate);

        // 构造模板所需要的参数
        Map<String, Object> map = new HashMap<~>();
        map.put("msgid", 123654);
        map.put("msg", 147852);
        map.put("sohucms_summary", "sohucms_summary");
        map.put("pic_temp", "pic_temp");
        map.put("at_temp", "at_temp");
        map.put("from", "from");
        map.put("geo", "geo");
        map.put("num_reply", "num_reply");

        // 模板渲染，输出到StringWrite
        StringWriter sb = new StringWriter();
        template.render(map, sb);

        System.out.println(sb.toString());

    }
```

初期目标

- 渲染组件与业务解耦，可扩展、易维护
- 页面模板分解、细化。支持批量模板请求处理
- 模板语法，支持 include 、 if 等语法。逐步优化渲染性能
- 稳定、高效、简单

app 如何应用

- 查看文档，找出可以重用的模板及对应的 (数据结构变量)
- 开发应用自己的模板文件
- 提交，并由我们注册到模板引擎
- 测试　上线

模板文档

- ${var }　如果没有这个变量，引擎如何处理
- If 表达式 如果没有这个表达式，引擎如何处理
- Include 语句　　如果没有这个值，引擎如何处理
- 必填参数　可选参数 (默认值)

接下来工作

- 现有模板梳理
- 模板粒度拆分
- 渲染服务实现
- 与前端 (js 模板) 沟通，制定模板
- 与前端 (css) 沟通

Thanks!