链滴

通过 Latke-Demo 对 Latke 工作流程的初步分析

作者：ZephyrJung

原文链接：https://ld246.com/article/1479809894659

来源网站：链滴

Latke代码看到IOC包后就彻底懵逼了，从已看过的几段代码来看，都是各种set、get，变量名称含义象，不知所云。由此感到单纯的读代码恐怕无法理解IOC包的实现逻辑了。于是决定运行一下demo来跟踪一下代码。以下是latke-demo项目从请求发起到响应结束时经过的代码段（层级关系可能不正确，但是大致流程就是如此）

虽然还没有完全明白，但是跟踪下来感觉颇有收获，待慢慢补充细节

1. DispatcherServlet.service

```
//以此作为起点。此处是最原始的servlet，配置在web.xml中，再细节的方面，暂时可以当做黑盒了。
//不得不说，jsp+servlet作为基础，当年没有好好学是个错误。（垃圾讲师误人啊……）
protected void service(final HttpServletRequest req, final HttpServletResponse resp) throws Se
rvletException, IOException {
    //HTTPRequestContext，Latke封装，没有定义构造器
    final HTTPRequestContext httpRequestContext = new HTTPRequestContext();
    httpRequestContext.setRequest(req);
    httpRequestContext.setResponse(resp);
    /*
    *  在servlet的init方法中，为SYS_HANDLE:List<Handler>添加了如下Handler
    *  StaticResourceHandler(getServletContext()));
    *  RequestPrepareHandler());
    *  RequestDispatchHandler());
    *  ArgsHandler());
    *  AdviceHandler());
    *  MethodInvokeHandler());
    */
    final HttpControl httpControl = new HttpControl(SYS_HANDLER.iterator(), httpRequestCon
ext);
    try {
        //【1】在此处转而进行上述列表中的handler方法
        //这样看标题2、3的过程就一目了然了
        httpControl.nextHandler();
    } catch (final Exception e) {
        httpRequestContext.setRenderer(new HTTP500Renderer(e));
    }
    result(httpRequestContext);
}
```

- LCV:HTTPRequestContext(未完成)
- LCV:HttpControl(未完成)

2. HttpControl.nextHandler

```
public void nextHandler() {
    if (ihandlerIterable.hasNext()) {
        try {
            ihandlerIterable.next().handle(httpRequestContext, this);
        } catch (final Exception e) {
            LOGGER.log(Level.ERROR, "Request processing failed", e);
        }
    }
}
```

3. handle

3.1 StaticResourceHandler.handle

```
public void handle(final HTTPRequestContext context, final HttpControl httpControl) throws E
ception {
    final HttpServletRequest request = context.getRequest();
    if (StaticResources.isStatic(request)) {
        if (null == requestDispatcher) {
            throw new IllegalStateException(
                    "A RequestDispatcher could not be located for the default servlet [" + this.defaul
ServletName + "]");
        }
        context.setRenderer(new StaticFileRenderer(requestDispatcher));
        return;
    }
    httpControl.nextHandler();
}
```

3.2 RequestPrepareHandler

```
public void handle(final HTTPRequestContext context, final HttpControl httpControl) throws E
ception {
    final HttpServletRequest request = context.getRequest();
    final long startTimeMillis = System.currentTimeMillis();
    request.setAttribute(Keys.HttpRequest.START_TIME_MILLIS, startTimeMillis);
    httpControl.nextHandler();
}
```

3.3 RequestDispatchHandler

```
public void handle(final HTTPRequestContext context, final HttpControl httpControl) throws E
ception {
    final HttpServletRequest request = context.getRequest();
    final String requestURI = getRequestURI(request);
    final String httpMethod = getHTTPMethod(request);
    final MatchResult result = doMatch(requestURI, httpMethod);
    if (result != null) {
        httpControl.data(MATCH_RESULT, result);
        httpControl.nextHandler();
    }
}
```

3.4 ArgsHandler

```
public void handle(final HTTPRequestContext context, final HttpControl httpControl) throws E
ception {
    final MatchResult result = (MatchResult) httpControl.data(RequestDispatchHandler.MATCH
RESULT);
    final Method invokeHolder = result.getProcessorInfo().getInvokeHolder();
    final Map<String, Object> args = new LinkedHashMap<String, Object>();
```

```
    final Class<?>[] parameterTypes = invokeHolder.getParameterTypes();
    final String[] paramterNames = getParamterNames(invokeHolder);
    for (int i = 0; i < parameterTypes.length; i++) {
        doParamter(args, parameterTypes[i], paramterNames[i], context, result, i);
    }
    httpControl.data(PREPARE_ARGS, args);
    httpControl.nextHandler();
}
```

3.5 AdviceHandler

【AskD】这个Handler字面上不太理解

```
public void handle(final HTTPRequestContext context, final HttpControl httpControl) throws E
ception {
    final MatchResult result = (MatchResult) httpControl.data(RequestDispatchHandler.MATCH
RESULT);
    @SuppressWarnings("unchecked")
    final Map<String, Object> args = (Map<String, Object>) httpControl.data(ArgsHandler.PR
PARE_ARGS);
    final Method invokeHolder = result.getProcessorInfo().getInvokeHolder();
    final Class<?> processorClass = invokeHolder.getDeclaringClass();
    final List<AbstractHTTPResponseRenderer> rendererList = result.getRendererList();
    final LatkeBeanManager beanManager = Lifecycle.getBeanManager();
    final List<Class<? extends BeforeRequestProcessAdvice>> beforeAdviceClassList = getBefo
eList(invokeHolder, processorClass);
    try {
        BeforeRequestProcessAdvice binstance = null;
        for (Class<? extends BeforeRequestProcessAdvice> clz : beforeAdviceClassList) {
            binstance = beanManager.getReference(clz);
            binstance.doAdvice(context, args);
        }
    } catch (final RequestReturnAdviceException re) {
        return;
    } catch (final RequestProcessAdviceException e) {
        final JSONObject exception = e.getJsonObject();
        final String msg = exception.optString(Keys.MSG);
        final int statusCode = exception.optInt(Keys.STATUS_CODE, -1);
        if (-1 != statusCode && HttpServletResponse.SC_OK != statusCode) {
            final HttpServletResponse response = context.getResponse();
            response.sendError(statusCode, msg);
        } else {
            final JSONRenderer ret = new JSONRenderer();
            ret.setJSONObject(exception);
            context.setRenderer(ret);
        }
        return;
    }
    for (AbstractHTTPResponseRenderer renderer : rendererList) {
        renderer.preRender(context, args);
    }
    //【2】 => 3.6
    httpControl.nextHandler();
    for (int j = rendererList.size() - 1; j >= 0; j--) {
```

```
            rendererList.get(j).postRender(context, httpControl.data(MethodInvokeHandler.INVOKE_
ESULT));
        }
        final List<Class<? extends AfterRequestProcessAdvice>> afterAdviceClassList = getAfterList
invokeHolder, processorClass);
        AfterRequestProcessAdvice instance;
        for (Class<? extends AfterRequestProcessAdvice> clz : afterAdviceClassList) {
            instance = beanManager.getReference(clz);
            instance.doAdvice(context, httpControl.data(MethodInvokeHandler.INVOKE_RESULT));
        }
    }
```

3.6 MethodInvokeHandler

```
    public void handle(final HTTPRequestContext context, final HttpControl httpControl) throws E
ception {
        final MatchResult result = (MatchResult) httpControl.data(RequestDispatchHandler.MATCH
RESULT);
        final Map<String, Object> args = (Map<String, Object>) httpControl.data(ArgsHandler.PR
PARE_ARGS);
        // get class instance
        final Method invokeHolder = result.getProcessorInfo().getInvokeHolder();
        final LatkeBeanManager beanManager = Lifecycle.getBeanManager();
        final Object classHolder = beanManager.getReference(invokeHolder.getDeclaringClass());
        final Object ret = invokeHolder.invoke(classHolder, args.values().toArray());
        httpControl.data(INVOKE_RESULT, ret);
    }
```

【AskD】：总觉得这种处理流程有点奇怪，为什么不是在标题2的nexthandler中循环调用，而是用种链式引用呢?

4. JavassistMethodHandler.invoke

执行到这一步，参数method中已经有了相应processor路径，未知是从哪里赋值的，需要再跟

```
    public Object invoke(final Object proxy, final Method method, final Method proceed, final Ob
ect[] params) throws Throwable {
        //反射，获得处理类的类名
        final Class<?> declaringClass = method.getDeclaringClass();
        //获取处理方法名：ClassName.MethodName
        final String invokingMehtodName = declaringClass.getName() + '#' + method.getName();
        // 1. @BeforeMethod handle
        handleInterceptor(invokingMehtodName, params, BeforeMethod.class);
        // 2. Invocation with transaction handle
        //【AskD】这里看起来很突兀，为何会用到userRepository?
        final UserRepository userRepository = beanManager.getReference(UserRepository.class);
        final boolean withTransactionalAnno = method.isAnnotationPresent(Transactional.class);
        final boolean alreadyInTransaction = userRepository.hasTransactionBegun();
        final boolean needHandleTrans = withTransactionalAnno && !alreadyInTransaction;
        // Transaction Propagation: REQUIRED (Support a current transaction, create a new one if
one exists)
        Transaction transaction = null;
        if (needHandleTrans) {
```

```
        transaction = userRepository.beginTransaction();
    }
    Object ret = null;
    try {
        ret = proceed.invoke(proxy, params);
        if (needHandleTrans) {
            transaction.commit();
        }
    } catch (final InvocationTargetException e) {
        if (needHandleTrans) {
            if (null != transaction && transaction.isActive()) {
                transaction.rollback();
            }
        }
        throw e.getTargetException();
    }
    // 3. @AfterMethod handle
    handleInterceptor(invokingMehtodName, params, AfterMethod.class);
    return ret;
}
```

4.1 JavassistMethodHandler.handleInterceptor

```
private void handleInterceptor(final String invokingMehtodName, final Object[] params,
    final Class<? extends Annotation> interceptAnnClass) {
    final Set<Interceptor> interceptors = InterceptorHolder.getInterceptors(invokingMehtodN
me, interceptAnnClass);
    for (final Interceptor interceptor : interceptors) {
        final Method interceptMethod = interceptor.getInterceptMethod();
        final Class<?> interceptMethodClass = interceptMethod.getDeclaringClass();
        try {
            final Object reference = beanManager.getReference(interceptMethodClass);
            interceptMethod.invoke(reference, params);
        } catch (final Exception e) {
            final String errMsg = "Interception[" + interceptor.toString() + "] execute failed";
            LOGGER.log(Level.ERROR, errMsg, e);
            throw new RuntimeException(errMsg);
        }
    }
}
```

4.1.1 InterceptorHolder.getInterceptors

```
static Set<Interceptor> getInterceptors(final String invokingMethodName, final Class<? exte
ds Annotation> interceptAnnClass) {
    if (BeforeMethod.class.equals(interceptAnnClass)) {
        final Set<Interceptor> ret = BEFORE_METHOD_HOLDER.get(invokingMethodName);
        if (null == ret) {
            return Collections.emptySet();
        }
        return ret;
    } else if (AfterMethod.class.equals(interceptAnnClass)) {
        final Set<Interceptor> ret = AFTER_METHOD_HOLDER.get(invokingMethodName);
```

```
        if (null == ret) {
            return Collections.emptySet();
        }
        return ret;
    }
    return Collections.emptySet();
}
```

5. HelloProcessor.index

```
public void index(final HTTPRequestContext context) {
    final AbstractFreeMarkerRenderer render = new FreeMarkerRenderer();
    context.setRenderer(render);
    render.setTemplateName("index.ftl");
    final Map<String, Object> dataModel = render.getDataModel();
    dataModel.put("greeting", "Hello, Latke!");
}
```

6. MethodInvokeHandler.handle

```
//这里并不是真的返回到了这个方法
//而是栈的弹出过程（几乎用到了上述列举的所有Handler）
//httpControl.nextHandler是一个递归调用的过程？
//或许只是因为调试调用栈的问题看起来像递归，看方法名应该是链表遍历才对...
```

7. DispatcherServlet.result

```
public static void result(final HTTPRequestContext context) throws IOException {
    final HttpServletResponse response = context.getResponse();
    if (response.isCommitted()) { // Response sends redirect or error
        return;
    }
    AbstractHTTPResponseRenderer renderer = context.getRenderer();
    if (null == renderer) {
        renderer = new HTTP404Renderer();
    }
    renderer.render(context);
}
```

8. AbstractFreeMarkerRenderer.render

```
public void render(final HTTPRequestContext context) {
    final HttpServletResponse response = context.getResponse();
    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");
    PrintWriter writer;
    try {
        writer = response.getWriter();
    } catch (final Exception e) {
        try {
            writer = new PrintWriter(response.getOutputStream());
        } catch (final IOException ex) {
```

```
            LOGGER.log(Level.ERROR, "Can not get response writer", ex);
            return;
        }
    }
    if (response.isCommitted()) { // response has been sent redirect
        writer.flush();
        writer.close();

        return;
    }
    final HttpServletRequest request = context.getRequest();
    final Template template = getTemplate((String) request.getAttribute(Keys.TEMAPLTE_DIR_
AME), templateName);
    if (null == template) {
        LOGGER.log(Level.ERROR, "Not found template[{0}]", templateName);
        try {
            response.sendError(HttpServletResponse.SC_NOT_FOUND);
        } catch (final IOException ex) {
            LOGGER.log(Level.ERROR, "Can not send error 404!", ex);
        }
        return;
    }
    try {
        dataModel.put(Keys.REQUEST, request);
        Keys.fillServer(dataModel);
        //这里好像什么都没有
        beforeRender(context);
        final String html = genHTML(context.getRequest(), dataModel, template);
        doRender(html, context.getRequest(), response);
        afterRender(context);
    } catch (final Exception e) {
        LOGGER.log(Level.ERROR, "FreeMarker renders error", e);
        try {
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        } catch (final IOException ex) {
            LOGGER.log(Level.ERROR, "Can not send error 500!", ex);
        }
    }
}
```

8.1 AbstractFreeMarkerRenderer.genHTML

```
protected String genHTML(final HttpServletRequest request, final Map<String, Object> data
odel, final Template template)
        throws Exception {
    final StringWriter stringWriter = new StringWriter();
    template.setOutputEncoding("UTF-8");
    template.process(dataModel, stringWriter);
    final StringBuilder pageContentBuilder = new StringBuilder(stringWriter.toString());
    final long endimeMillis = System.currentTimeMillis();
    final String dateString = DateFormatUtils.format(endimeMillis, "yyyy/MM/dd HH:mm:ss");
    final long startTimeMillis = (Long) request.getAttribute(Keys.HttpRequest.START_TIME_MILL
S);
    final String msg = String.format("<!-- Generated by B3log Latke(%1$d ms), %2$s -->", end
```

```
meMillis - startTimeMillis, dateString);
    pageContentBuilder.append(msg);
    return pageContentBuilder.toString();
}
```

8.2 AbstractFreeMarkerRenderer.doRender

```
protected void doRender(final String html, final HttpServletRequest request, final HttpServlet
esponse response)
        throws Exception {
    PrintWriter writer;
    try {
        writer = response.getWriter();
    } catch (final Exception e) {
        writer = new PrintWriter(response.getOutputStream());
    }
    if (response.isCommitted()) { // response has been sent redirect
        writer.flush();
        writer.close();
        return;
    }
    writer.write(html);
    writer.flush();
    writer.close();
}
```