蚂蚁金服
ANT FINANCIAL

# Build a Java Debugger for the Cloud

***Practices in the Ant Financial Cloud***

San Hong Li

Jonathan Lu

# About us

- San Hong Li
  - Senior Expert,  working on OpenJDK in **Alibaba**
- Jonathan Lu
  - JDK Engineer from **Alibaba** Infrastructure Service Team

# What to Expect

- By the end of this session, you should be able to:
  - Understand why we need build a “cloud” debugger
  - Gain insight into the implementation of Alibaba cloud debugger(named as **ZDebugger**)
  - Discover the features of **ZDebugger** for debugging Java applications in the cloud

# Agenda

1. **Background**: why we need build a ‘cloud ’ debugger?
2. **Show you code**: How we implement the **ZDebugger** & changes made to OpenJDK
3. **Performance**: How fast do we get data at a “breakpoint”?
4. **Demo**: What does it look like?
5. **Wrap-up**: Summary, and Next steps

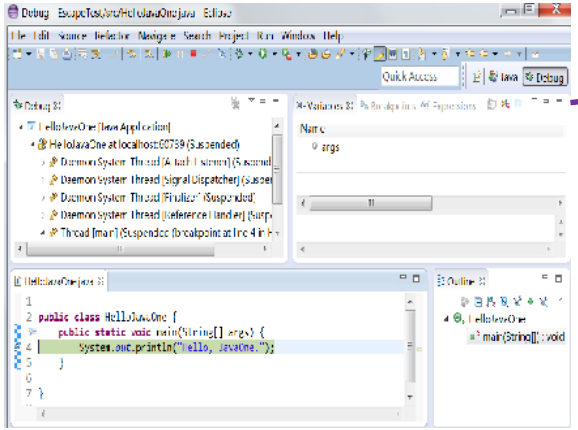
# Ant Financial Cloud Introduction

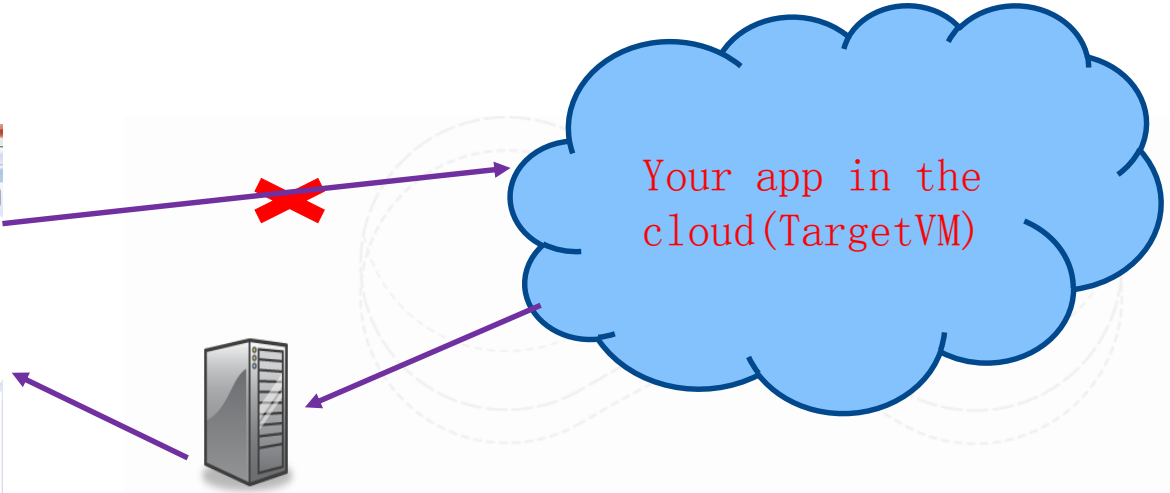
- **Enable financial business agility:** make the users to be able to focus on the business requirements but not the IT complexities and the common financial concerns.
- **Compliance with financial regulations and supervisions:** provide the common financial components and best practices according to the financial regulations and supervisions.
- **Runtime support for java stack:** middleware, message queue and monitoring
- **Continuous integration/delivery:** provides source control, build and package, deployment, test, continues integration and delivery.


http://cloud.alipay.com/

# Challenges to debugging in cloud

## Local Debugger






- Need have exactly the same code version in Eclipse

# We NEED a ‘cloud’ debugger

**Security consideration:** doesn’t permit an incoming debug connection from debug clients which are behind firewall

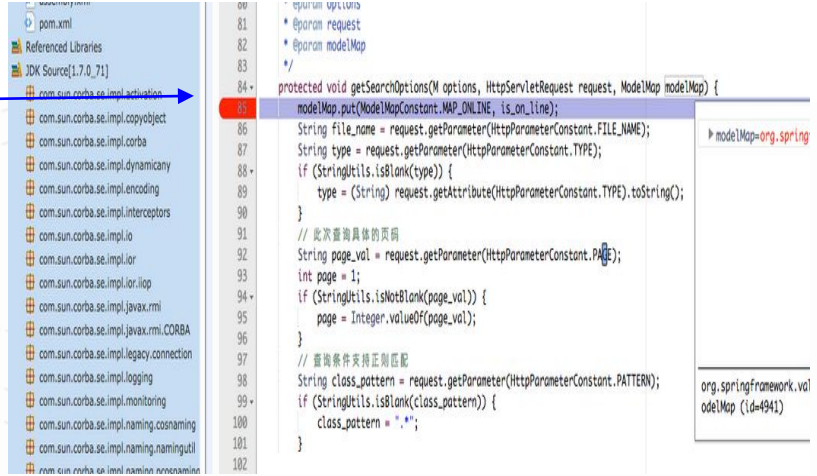
**Performance requirement:** doesn’t allow the application to be suspended for a long time during debugging

**Business requirement:** need the ability to allow the user to enable the debugging capacity of running target JVM on the fly

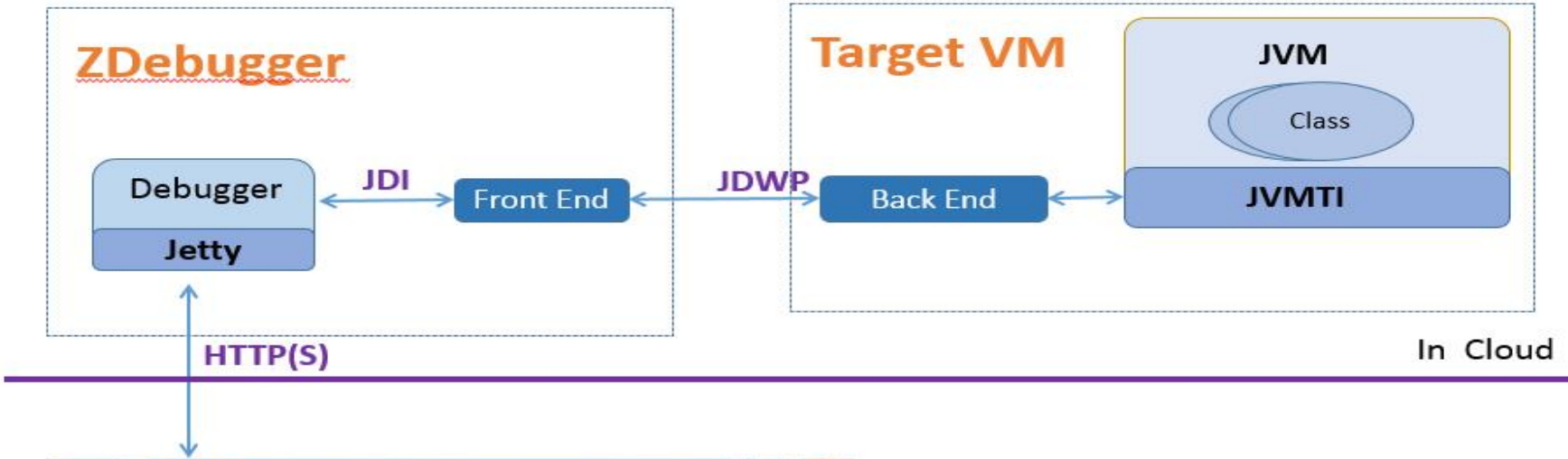

*The debugger in IDE can not help us in cloud environment!*

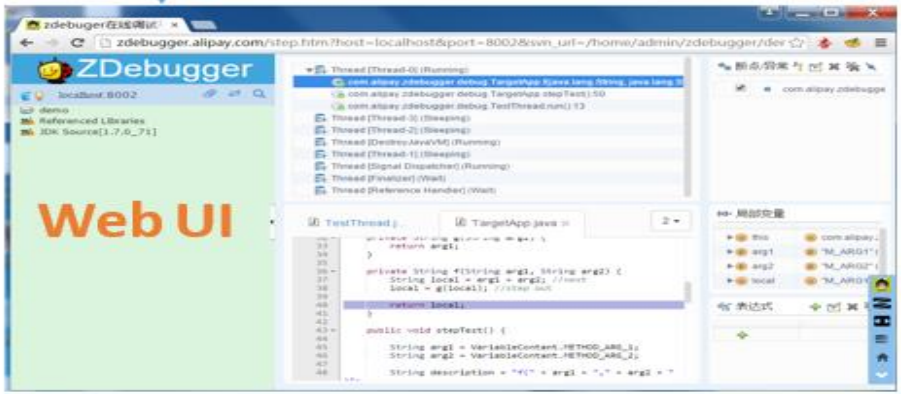
# Introduction to ZDebugger

- Cloud debugger
  - Deployed in cloud, use your **browser** to debug the java applications on any machine
- Bound to cloud source repository
  - Check out the code you are debugging from repository automatically
- Watch Point
  - Task a fast snapshot at Breakpoint (FSAB) but without slowing down the service
- Debug-On-Late-Attach(DOLA)
  - Don't require to re-boot JVM when you want to debug your app.




# ZDebugger Structure



# Skeleton implementation of ZDebugger

Implemented as jetty based web server

- serves the debugging requests from browser
- converts to according JDI calls into target JVM over JDWP,
- listens on JDWP event queue (by one background thread)
  - handles JDWP event from queue and
  - updates message to the UI if appropriate(via websocket)

# Set breakpoint as a example

(1) The user sets the breakpoint in browser: [class_name, line_number]

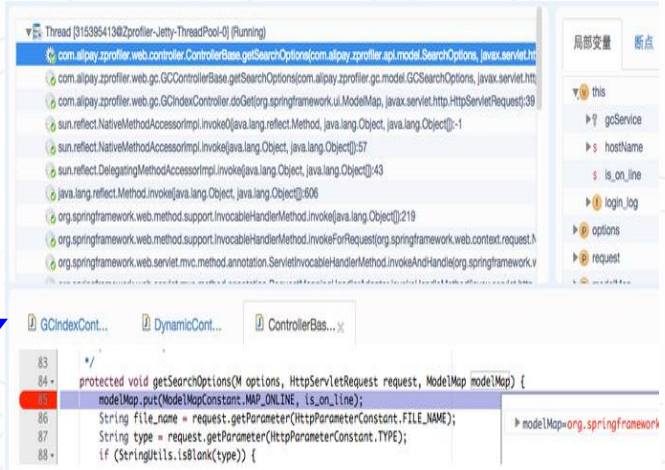
(2) Servlet receives request, converts to JNI call and creates a breakpoint request to target VM

```
56              EventRequestManager em = refType.virtualMachine().eventRequestManager();
57              bp = em.createBreakpointRequest(location);
58              bp.setSuspendPolicy(getSuspendPolicy());
59              bp.enable();
```

(3) One background thread is listening on event queue

```
59      public void run() throws InterruptedException {
60          EventQueue queue = virtualMachine.eventQueue();
61          while (connected) {
62              //get the event from queue
63              EventSet eventSet = queue.remove();
64              EventIterator it = eventSet.eventIterator();
65              if (eventSet.size() == 1) {
66                  handleEvent(it.next());
67              }
68              //...
69          }
70      }
71
72      public void handleEvent(Event event) {
73          if (event instanceof BreakpointEvent) {
74              //handle the breakpoint event
75          } else if (event instanceof StepEvent) {
76              //handle step event
77          }
78      }
```

(4) Update to UI



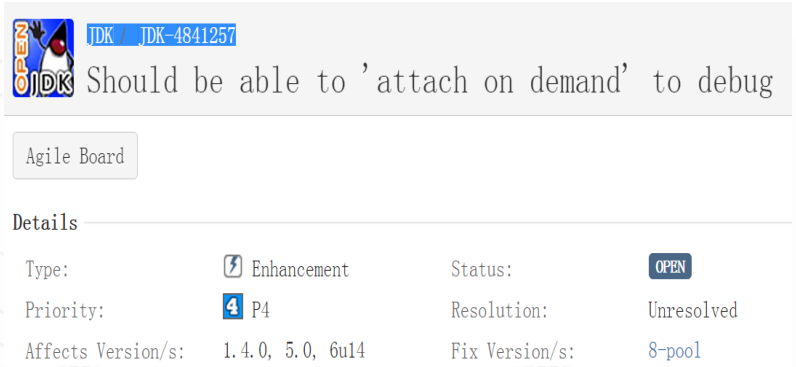
# Challenges we encounter...

- How to set a breakpoint If the class is not loaded by target JVM yet
- How to set a breakpoint for a  inner class or the case that different java classes are located in same java source file
- Integrate with code repository in cloud
- …

# On demand debugging

The current implementation of HotSpot doesn't support the late attach for jdwp agent:

https://bugs.openjdk.java.net/browse/JDK-4841257




Why we need **DOLA(Debug-on-Late-Attach)**?

- In our production environment, for performance consideration, we don't start JVM with **-agentlib:jdwp** option
- It is hard to reproduce the problem if restarting the JVM after the problem occurred

# Debug on Late Attach

How to use it:

- **java** -XX:+DebugOnLateAttach HelloWorld
- **jcmd** [*pid*] AgentLib.load agent.name=jdwp agent.options=transport=dt_socket,server=y,suspend=n,address=2012

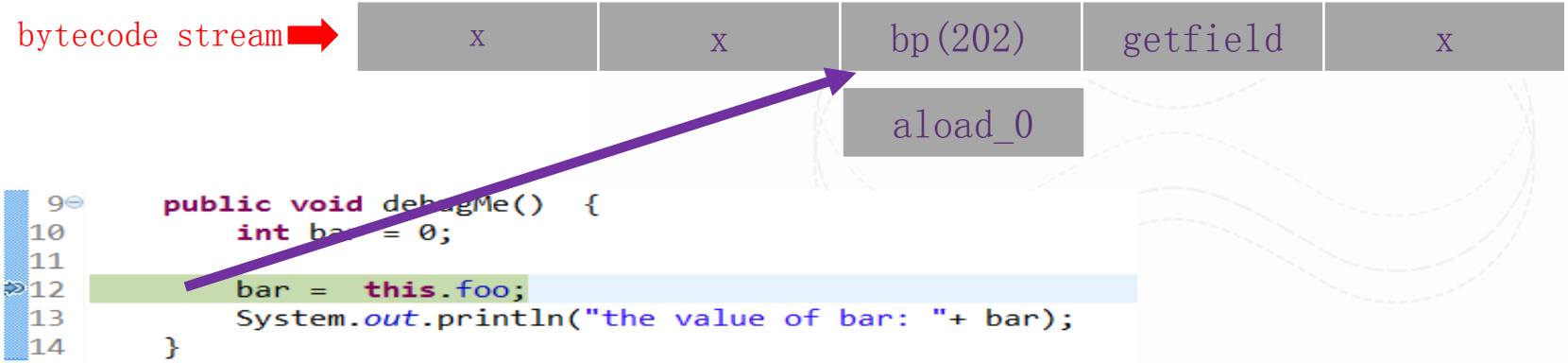
**DOLA** only supports **breakpoint set** and **variable inspections**, but they are enough for our use!

# Debug on Late Attach(DOLA)

*:Under the Hood*

# Implementation of DOLA

- **Modify** the interpreter of HotSpot to support breakpoint post in bytecode patch path




- **De-optimize** all alive nmethods when the breakpoint is set
  - For the case if the compiler doesn’t record all dependencies from startup
- **Add** the Agent_OnAttach functionality to jdwp agent
- **Extend** the jcmd to support agent loading (usability improvements)

# The Problem of Breakpoint...

The developer uses the breakpoint in traditional debugger, e.g. in eclipse likes this:

- Set the breakpoint on the line of code
- When the thread gets suspended once the breakpoint is hit
  - Inspect the stack frames, variables, etc.

**Problem:**

- The suspended period is **NOT PREDICTED** in the online production environment, might cause the timeout of service serving!

# Watchpoint for Our Needs

**Watchpoint** is a “customized” breakpoint:

- Set just likes a normal breakpoint
- Once a watchpoint is hit:
  - Make a snapshot of:
    - ✓ local variables
    - ✓ Parameters
    - ✓ instance variables
    - ✓ stack frame
  - send back to user’ UI for inspection
- This watchpoint will be disabled automatically after hit unless you enable it explicitly again

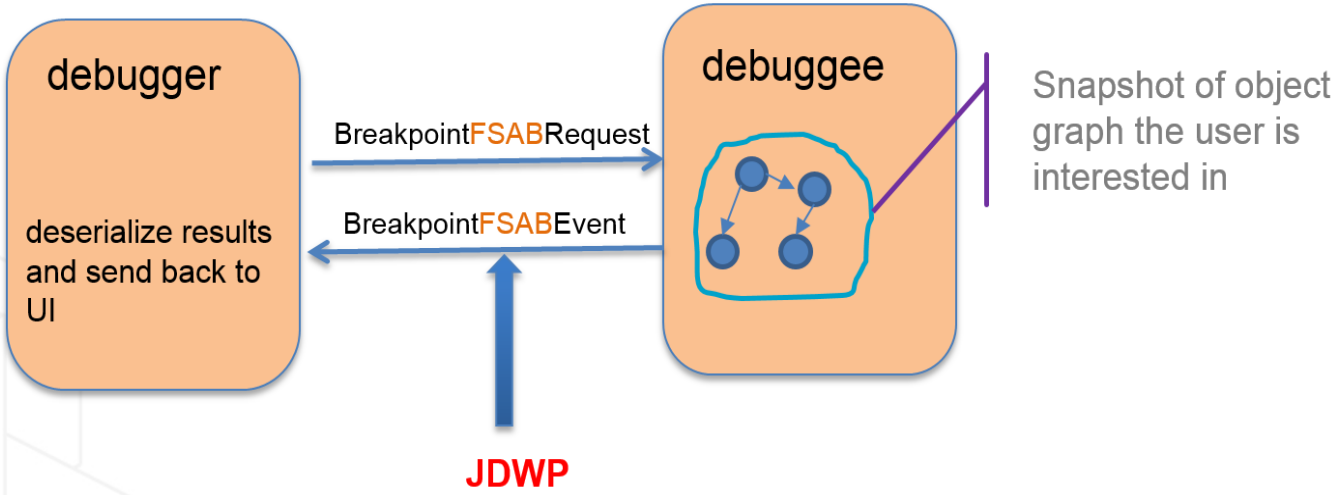
*Forget the BTrace, as the watchpoint in ZDebugger is more convenient for you ☺*

# Fast Snapsot at Breakpoint(FSAB)

*:dissecting the watchpoint*

# Implementation of FSAB

In order to support watchpoint, AJDK implements a FSAB mechanism:




Highlights:

- **Extend** the JDWP to add new command for FSAB
- **Make** the snapshot of object graph inside JVM, not using JVMTi
- **Use** Google protobuf for data interchange format

AJDK : *customized OpenJDK for Alibaba*

# Iterating over OOP

In order to speed up the object scanning, serialize the object graph by manipulating oop directly, instead of using standard JVMTi/JNI, similar mechanism used by heap dumper, example code from heapDumper.cpp

```
00848:   for (FieldStream fld(ikh, false, false); !fld.eos(); fld.next()) {
00849:     if (!fld.access_flags().is_static()) {
00850:       Symbol* sig = fld.signature();
00851:       address addr = (address)o + fld.offset();
00852:
00853:       dump_field_value(writer, sig->byte_at(0), addr);
00854:     }
00855:   }
00856: }
```

JVMTi/JNI Example

```
00551:         jobject value = JNI_FUNC_PTR(env, GetObjectField)(env, object, field);
00552:         if (value == NULL) {
00553:             cJSON_AddStringToObject(fields, VAL_STR, "null");
00554:         } else {
00555:             jclass clazz             = NULL;
00556:             char * specificSignature = NULL;
00557:             clazz = (*env)->GetObjectClass(env, value);
00558:             if (clazz != NULL) {
00559:                 error = (*jvmti)->GetClassSignature(jvmti, clazz, &specificSignature, NULL)
00560:                 if (error == JVMTI_ERROR_NONE) {
00561:                     JDI_ASSERT(specificSignature != NULL);
00562:                 }
00563:             }
```

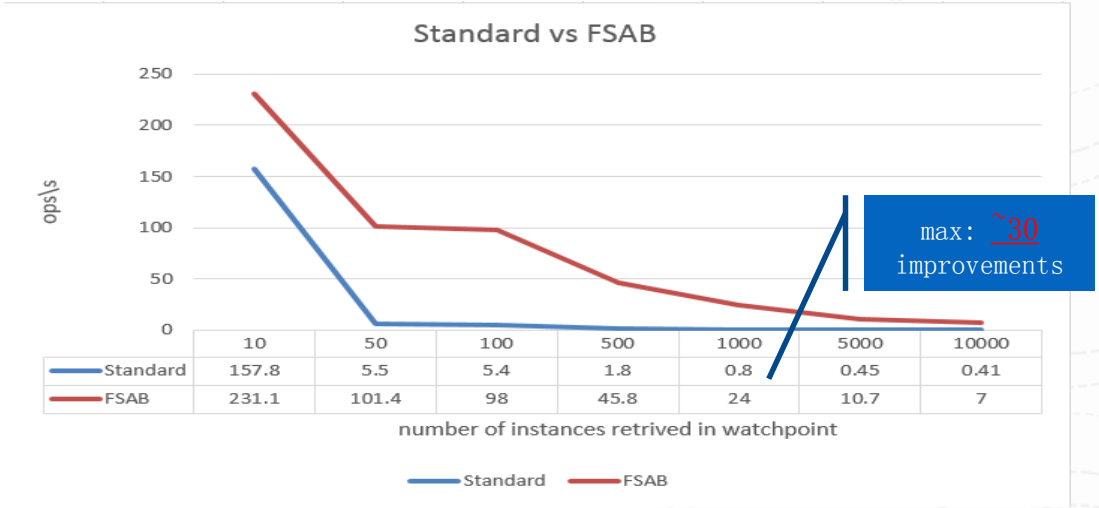
- HotSpot: **Scan** the object graph
- JDWP: **Serialize** to ProtoBuf message
- JDI: **Deserialize** vi java interfaces generated by protobuf on debugger side

oop: pointer into the GC-managed heap
http://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html

# Performance of FSAB

Evaluate the micro performance of FSAB made by JMH:

- **Run netty-based application as a target app and inspect the watchpoint via:**
  - ✓ **(1) FSAB**
  - ✓ **(2) Standard**




## The contributors:

- ❖ JDWP optimization(reducing the jmx cost)
- ❖ oop-based object graph scanning(vs JVMTi/JNI)
- ❖ protobuf(vs cjson)

jmh: Java harness for building, running, and analysing benchmarks written in Java
http://openjdk.java.net/projects/code-tools/jmh/

# What does it look like?

Demo

# Next Step

- Project roadmap
  - Deploy the ZDebugger as a shared service on the Ant Financial Cloud and support multi-tenant debugging mode
- Contribution back to OpenJDK
  - As for the DOLA&FSAB changes we made to OpenJDK, we would talk with community and ask if there is any interest / if we should create a JEP for that…

# Summary

Now that we have completed this session:

- Describe the requirements why we need ZDebugger for cloud debugging
- Go through how we implement ZDebugger
- Talk about the changes we made to OpenJDK: DOLA&FSAB
- Feature demo of ZDebugger

# Q & A
# Thank you!