Welcome

Java User Group Switzerland

Increasing productivity with JRebel

Claude Gex

13.10.2011

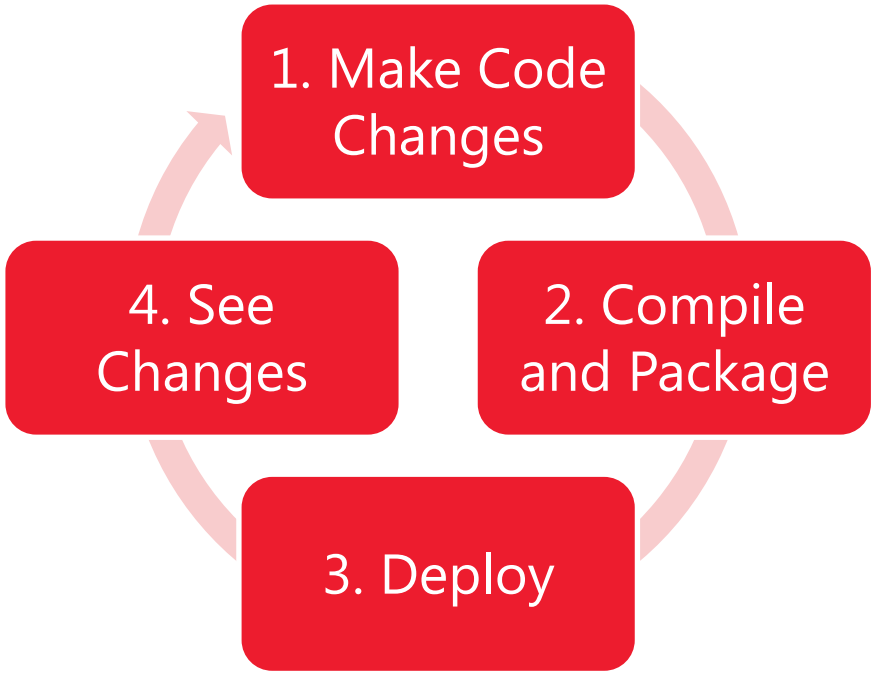
Redeploy again? It’s such a small change! #☠@💣 !

What to expect

1. Introduction
2. Fundamental techniques for “replacing” classes
3. JRebel
4. Demo
5. JRebel Plugin
6. Other Tools / Outlook

Development phases

- Turnaround time
  - Total time between phase 1 and 4
  - Needs to be reduced to the max!
- We're only interested in phase 1 and 4
- Time is money!



Changes we do

- Java Code
- JSP, Facelets
- Configurations (XML, Annotations)
  - JSF, Struts
  - Spring
  - JPA / Hibernate
  - Log4j, Logback …
- Resource Bundles
- Simple resource such as
  - Static HTML, CSS, JavaScript, Images…

Reducing turnaround time little by little

1. Configure your environment
2. Configure exploded directory deployment
3. Check what features your application server has
4. Check if your framework has reloading capabilities
5. Does this not yet meet your needs? Dig deeper and get some additional tools

1. Configure your environment

- Obvious to do, but nevertheless often disregarded…
- IDE needs to be configured
  - Activate “Build automatically”
  - Suitable server plugin is essential!
- If you use Eclipse and Maven
  - I recommend to use M2Eclipse
  - Do not forget to install “Maven Integration for WTP” (m2e-extras)
- In my opinion its best to start the server in the IDE

2. Exploded Directory Deployment

- Removes the need of creating deployment archives (WAR, EAR)
- Supported by many servers (not all verified)
  - Tomcat
  - JBoss AS
  - Oracle Weblogic
  - Glassfish
  - Websphere
- Only changed files needs to be copied, so called “incremental publish”
- Static resources gets published and do not require redeployment

3. Application Server capabilities

- Is automatic reloading of modules supported? (should be the case for all servers nowadays)
- Is it possible to configure when to reload a module? Some resources like CSS should only be copied…
- Does the application server provide more features?

Oracle Weblogic FastSwap, since version 10.3

- Configurable in WLS deployment descriptors
- New methods, fields, constructors
- Added methods not available through reflection API, no support for EJB or other frameworks

4. Frameworks with reloading capabilities (by design)

- Tapestry 5
- Grails
- JBoss Seam
- Play! Framework



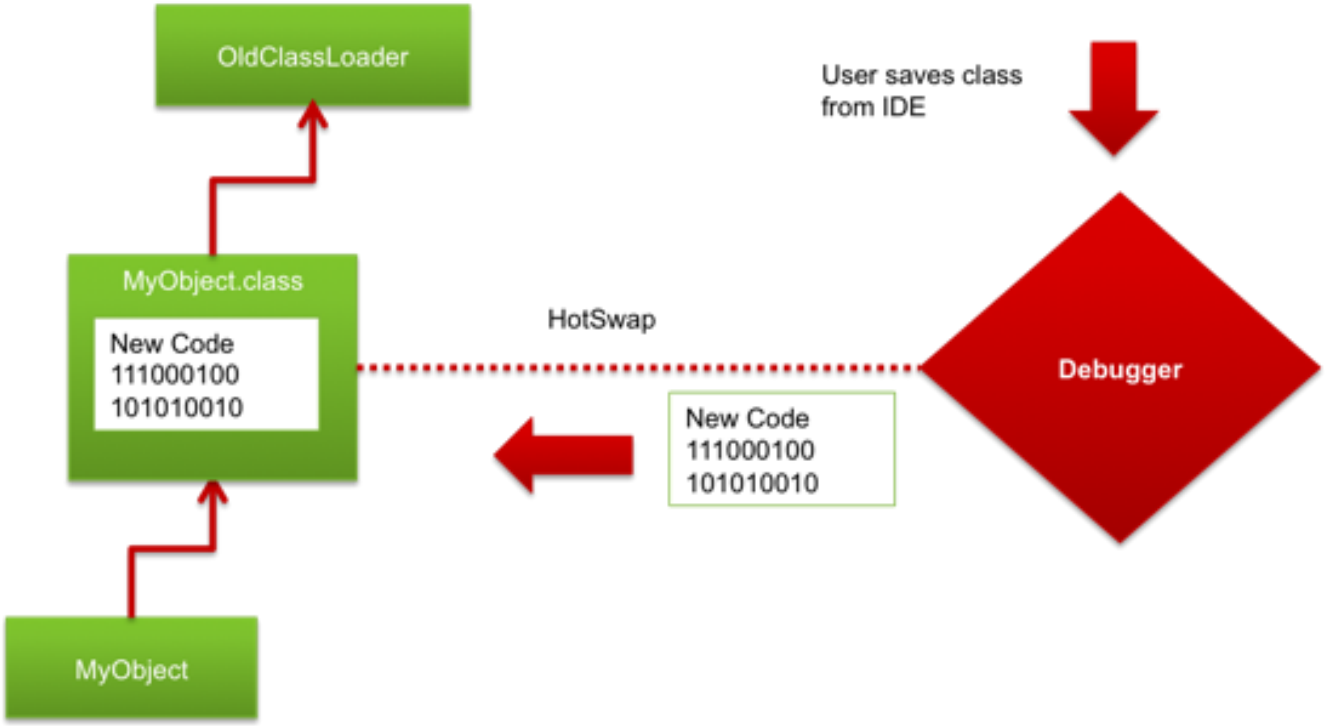

5. Not yet satisfied? Additional tools needed…

Fundamental techniques for “replacing” classes

- HotSwap (or in general: features the JVM offers out of the box)
- Throwaway classloaders
- Bytecode modifications

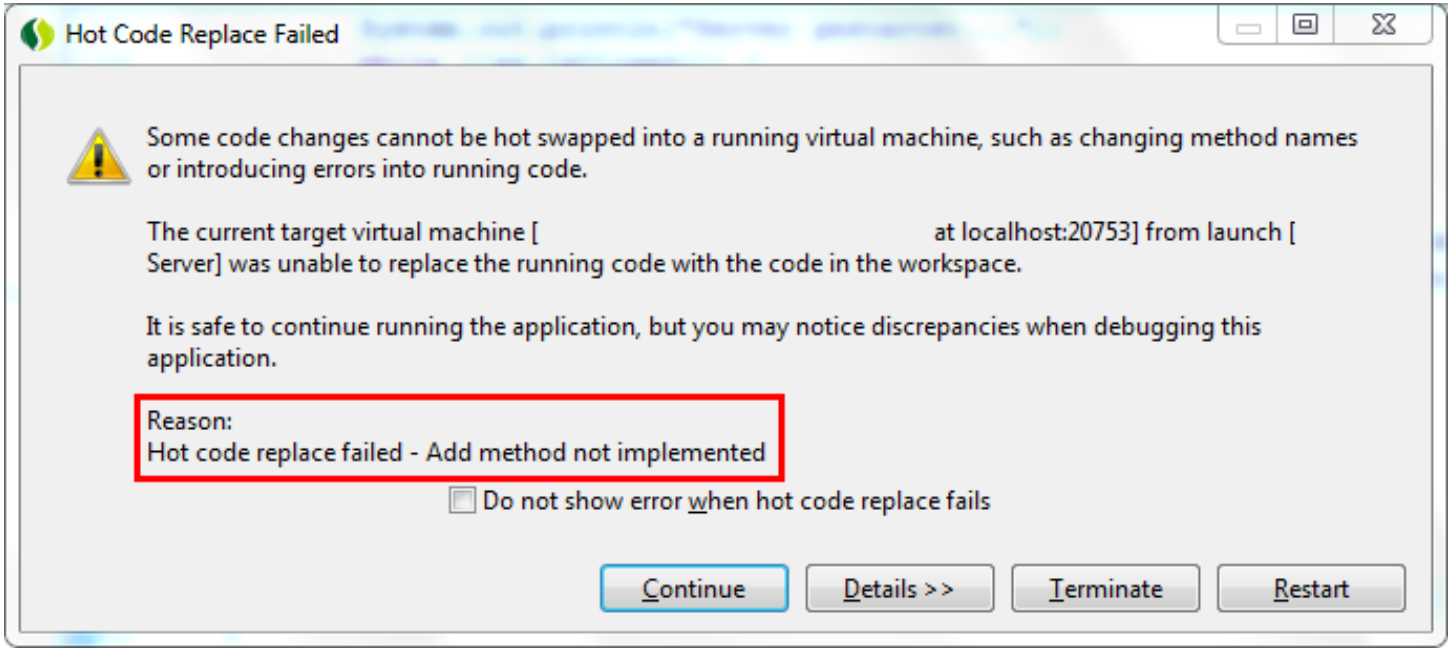
HotSwap

- Code changes are transferred by the Java Platform Debugger Architecture (JPDA) to the JVM



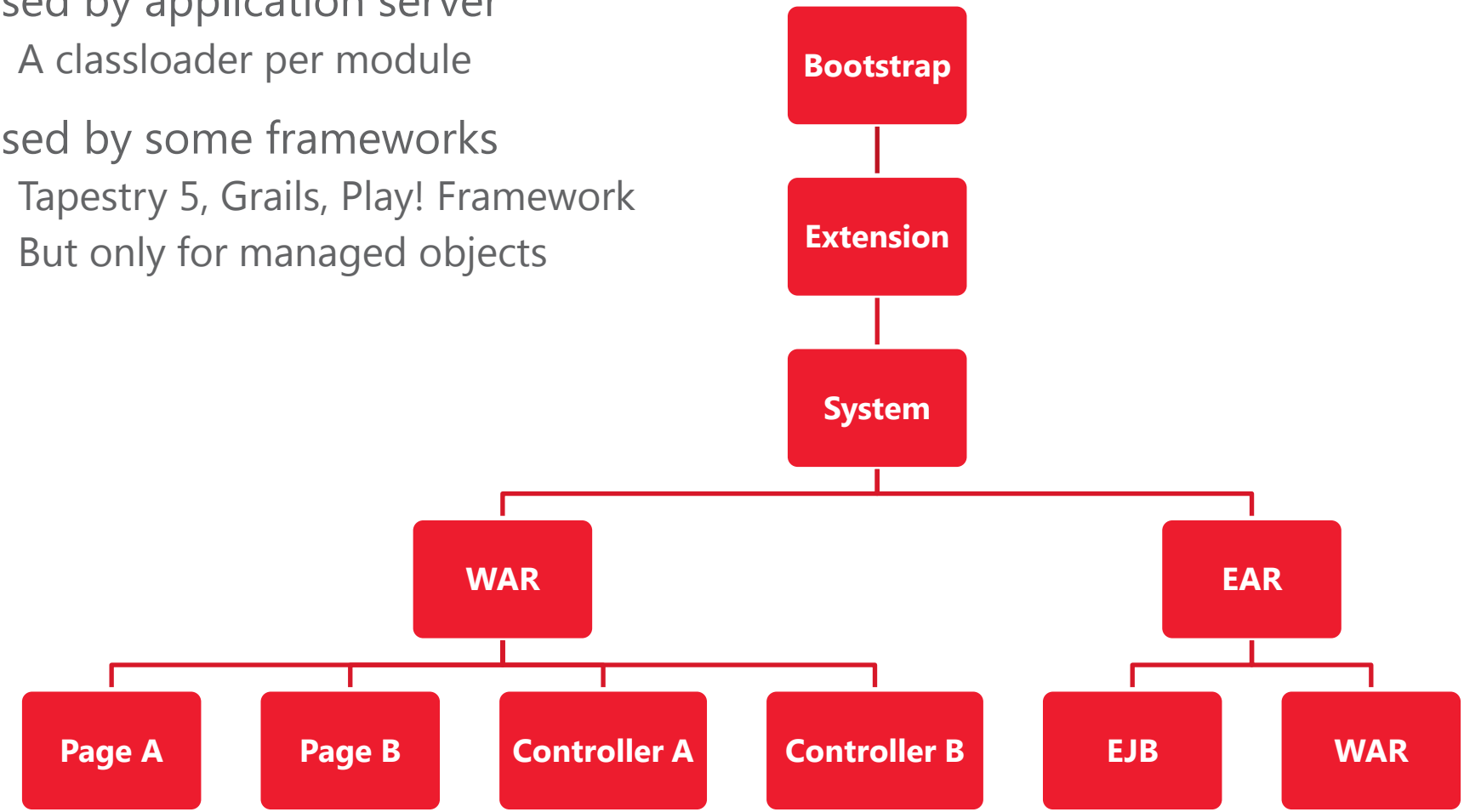
HotSwap - limitations

- Application needs to be started in debug mode
- Only changes to method bodies are possible



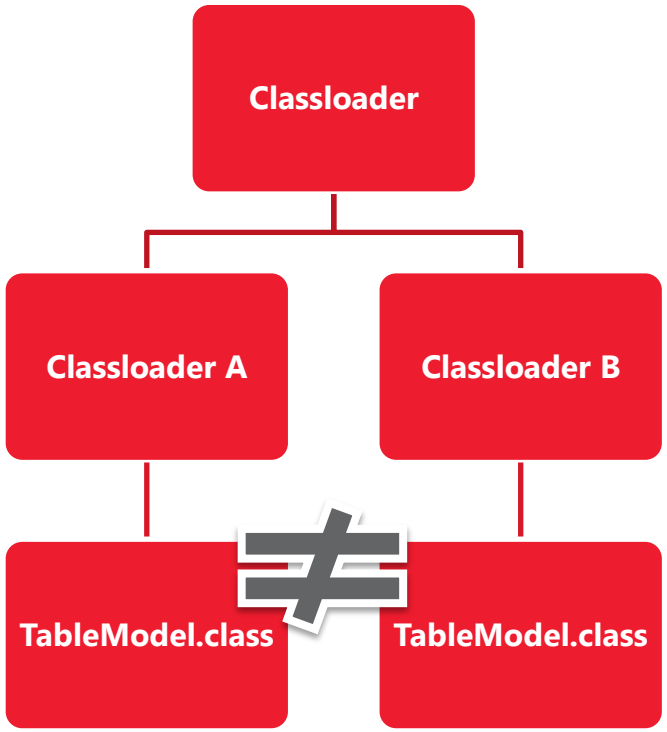
Throwaway classloaders

- Used by application server
  - A classloader per module
- Used by some frameworks
  - Tapestry 5, Grails, Play! Framework
  - But only for managed objects



Throwaway classloaders – limitations

- Classes are tightly bound to their classloaders at runtime.
- Same class loaded by 2 different classloaders is not the same!
- Existing instances always point to the original class definition
- Thus along with discarding the classloader and the class definitions all instances of it needs to be thrown away
- State copy isn't trivial (for recreation of objects)

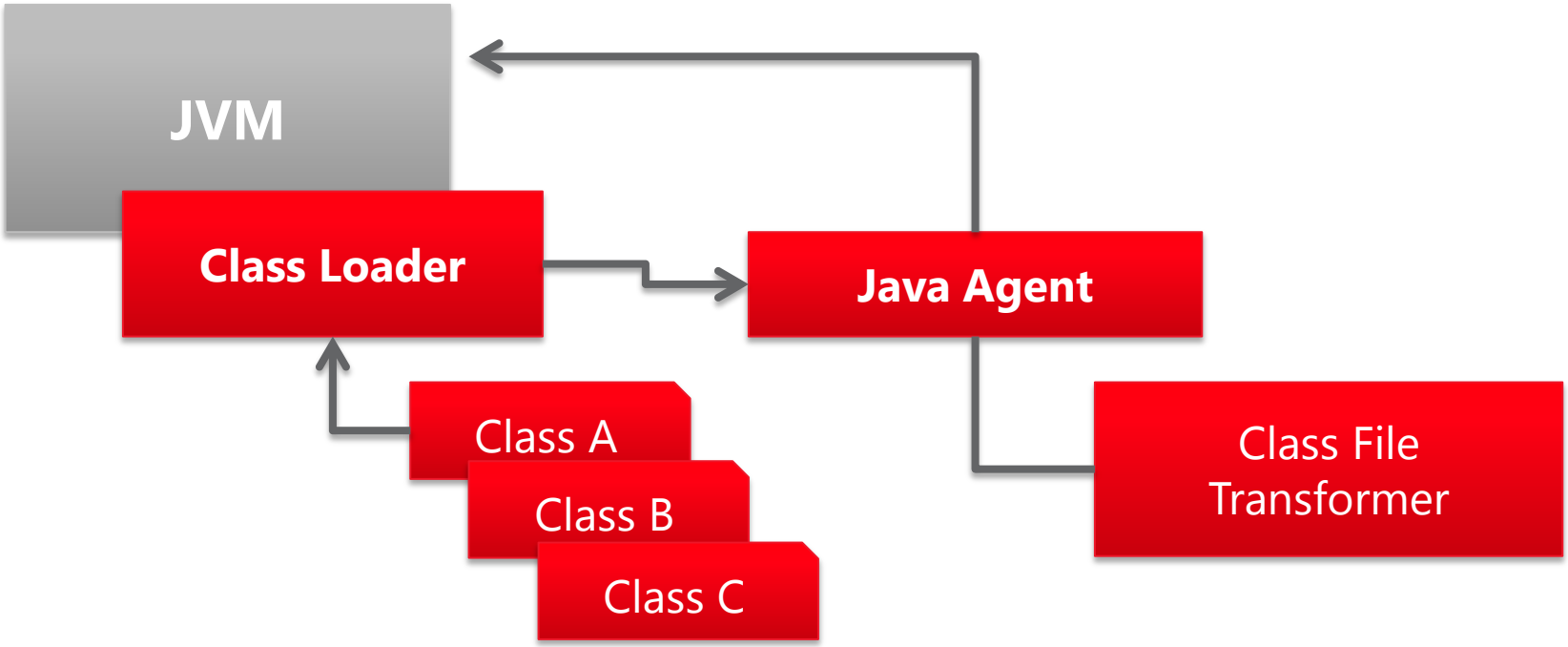


Reloading using bytecode modifications

- Modify monitored class definitions as they are loaded
  - To not mess things up, this includes the whole reflection part!
- Delegate to the latest version of a class as needed
  - Method calls, field access
- This is extremely hard to do (constructors, type checks, reflection…)
- Also consider debugging support
- **BUT:** by now this seems to be the only way to achieve enough flexibility
- And it allows to put reinitialization hooks for configs

Bytecode modifications

- Bootstrapping done by a Java Agent (Instrumentation API since JDK 1.5)
- Class transformation done using Libraries (e.g. Javassist)



JRebel is …

- A award-winning development tool
- A Java-Agent plugin
- A generic solution that works with all kind of applications
- Not limited to Java: Scala and Groovy officially supported
- Commercial tool
- Free licenses for OSS, Scala Developers and for non-commercial projects (JRebel social)

JRebel ZEROTURNAROUND

Rebellious Features

- Reloads code changes instantly …
- … while debugging is still supported
- … and it even hides newly generated class definitions from stacktrace
- Works with exploded directory and packaged deployment
- Reloadable resources are configurable
- Supports a wide range of frameworks while it's still open for extension (Plugin API)
- IDE and Maven Plugins available
  - Eclipse, IntelliJIDEA, NetBeans, Oracle JDeveloper

JRebel supported class structure changes

Features

- Adding / removing
  - Methods
  - Constructors
  - Fields
  - Annotations
  - Enum values
- Changing
  - Static values
  - Interfaces

Limitations

- Removing superclass
- Adding / removing implemented interfaces

JRebel environment and framework support

Java EE Support

- JSP EL & Scriptlet changes
- EJB 1-3 session bean interface changes
- JSF changes (Mojarra)
- JPA changes (Hibernate, EclipseLink, TopLink, OpenJPA)
- CDI changes (Weld)

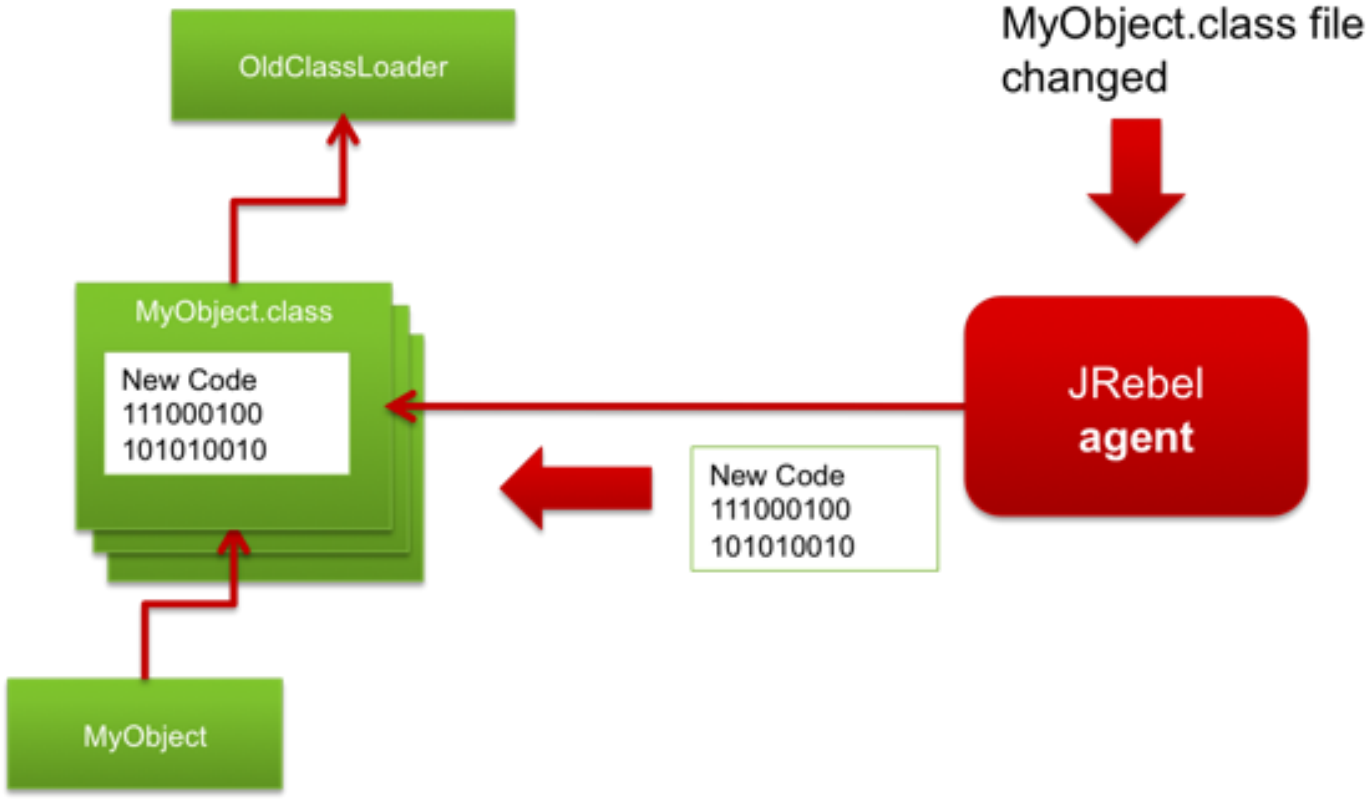
Frameworks (extract)

- ResourceBundle
- Spring Framework 2+
- Google Guice
- JBoss Seam 2+
- Struts 1, 2 / Wicket
- Facelets
- Log4j / Logback

Rebellion's configuration - rebel.xml

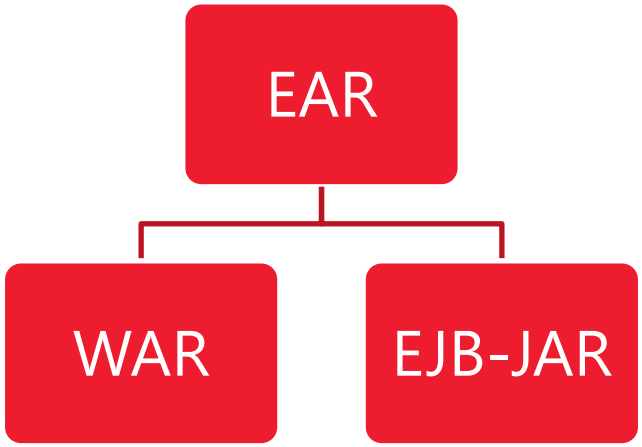
- Configuration per module
- Location
  - WAR: WEB-INF/classes
  - JAR: Root directory
- JRebel checks each module for the existence of rebel.xml
- Generation through IDE or Maven Plugin (I prefer the latter)

JRebel – how reloading looks like



Demonstrating JRebel's features

- Simple application for managing a list of attendees
- Application is entirely monitored by JRebel
- Standard JavaEE stack with
  - JSF
    - Managed Beans
    - Facelets
  - EJB
  - JPA through Hibernate



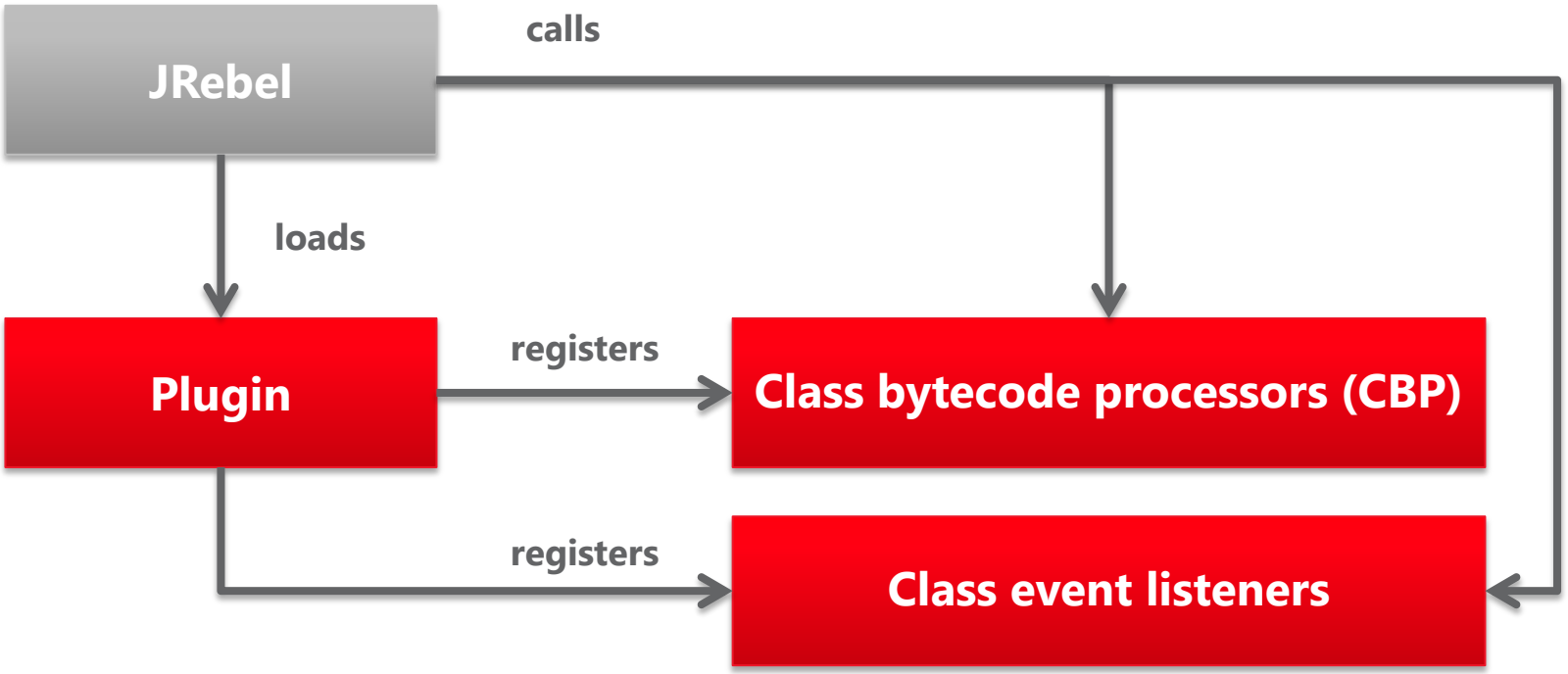
Creating a custom JRebel plugin

- Adding reload capabilities to your application or framework
- JRebel handles class reloading, thus custom plugins are most likely «only» about configuration changes
- Configuration defines how to initializes something
- 2 types of configuration needs to be distinguished:
  - external: files (XML, property files …), database, …
  - internal: annotations, «normal» initialization (initializer blocks, constructors…)
- Implementation complexity varies from easy to hard

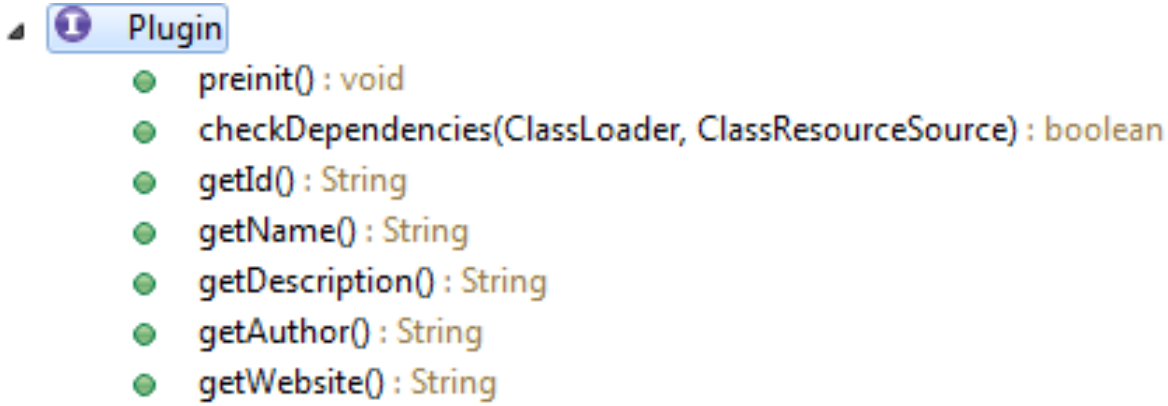
Introducing the JRebel SDK

- The JRebel SDK mainly consists of
  - A public API to JRebel
  - A bundled version of Javassist
- Most important functionality
  - Register class bytecode processors (CBP)
  - Adding own class event listeners (load / reload)
- Rather weak documentation available
  - 1 page on the website
  - official forum (little outdated)
  - JavaDoc

Basic plugin design consists of 3 types



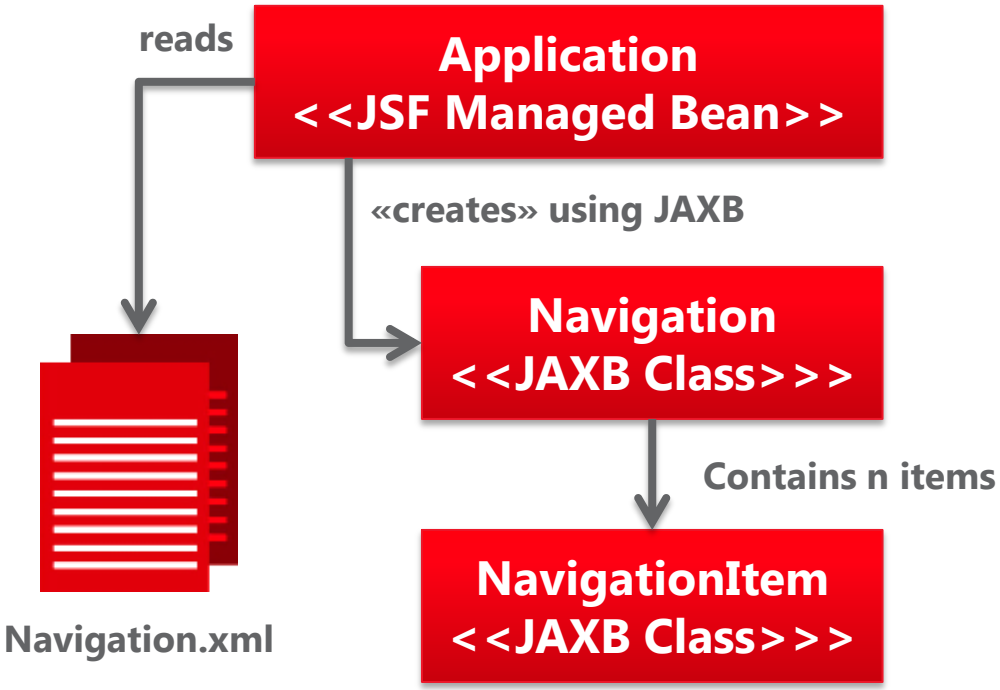
Explaining the plugin interface



- ***preinit*** is called when plugin is loaded
- ***checkDependencies*** checks if plugin should be loaded
- The plugin's ID is used to turn plugin on / off (-Djrebel.plugin-id=true)

Custom JRebel plugin for reloading XML-configuration

- Same demo application as before



Reviewing the previous plugin example

- It reloaded XML if changed
- This could have been done without JRebel, but it nicely demonstrated
  - How plugin development works
  - Non-invasive ways of adding development features
- Other configuration's are made with annotations.
- Annotations can be processed in class event processors

Benefits of using JRebel plugins

- Reloading capabilities for your application / framework
- Even for third party libraries
- It separates the «production» code from «development» code (separation of concerns)
- Enabling / disabling plugin on demand
- Integrates into one plattform (JRebel), this eases setup (only one javaagent to configure)

Summary

- Configuration of IDE and environment is essential
- Check application server's or framework's documentation for reloading features
- We've covered reloading techniques
  - HotSwap, throwaway classloaders and byte code modification
- We've seen that the dynamic reloading features JRebel offers reduces turnaround times (not to mention the time of a "context switch")
- Custom plugin development isn't that hard. But a more complex application / framework will increase plugin complexity

Other Tools / Outlook

Other tools/platforms competing JRebel:

- Fakereplace
- Javeleon (Currently only for Netbeans RCP applications)
- Dynamic Code Evolution Virtual Machine (DCE VM)
- Maybe some of these features will be available in HotSwap JDK 8+
- "Springloaded", included in SpringSource Tool Suite (STS)
  Mainly tested for tc-Server (Tomcat++)

THANK YOU.

Trivadis AG

Claude Gex

Papiermühlestrasse 73
3014 Bern

Mobile +41-76-310 13 44
Tel. +41-31-928 09 60 (Zentrale)
Fax +41-31-928 09 64

claude.gex@trivadis.com
www.trivadis.com

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

trivadis
makes IT easier.

Sources

- HotSwap
  http://download.oracle.com/javase/1.4.2/docs/guide/jpda/enhancements.html#hotswap
- Instrumentation API
  http://download.oracle.com/javase/6/docs/api/java/lang/instrument/package-summary.html
- Zeroturnaround
  http://www.zeroturnaround.com
- JRebel Plugin API JavaDoc
  http://www.zeroturnaround.com/docs/javarebel-sdk/javadoc/org/zeroturnaround/javarebel/Plugin.html

Sources

- Oracle Weblogic JRebel Whitepaper: http://www.oracle.com/technetwork/middleware/weblogic/oraclewls-jrebel.pdf
- Oracle FastSwap http://download.oracle.com/docs/cd/E12839_01/web.1111/e13702/deployunits.htm#i105438
- Javassist http://www.jboss.org/javassist/
- Javeleon http://javeleon.org/?features
- Dynamic Code Evolution VM http://ssw.jku.at/dcevm/