Get to Production Sooner

Complete projects 21% Faster with JRebel on WebLogic

Table Of Contents:

1. Comparison Chart: Turnaround-Reducing Options:

	Usual Redeploy (Dynamic Classloaders)	**JVM HotSwap**	**FastSwap**	**JRebel**
Time spent per reload	2.5 mins average	< 1 sec	< 1 sec	< 1 sec
Issues	Loses application state, navigate from scratch, rewarm caches, potential for class loader leaks	Needs debugger session, slows application	WLS 10.3.0+; only changes to class files in exploded directories are supported; Reflection API issues(1)	PermGen heap should be set to at least 128m
Class reloading	Full	Method bodies only	Enhanced with some documented limitations	Everything except Static Hierarchy Changes
Skipping builds	Limited	None	Full	Full
Java EE changes	Full	None	JSP, Servlet, EJB support classes	~90% (2)
Framework Configuration and Component changes	Full	None	None	~75% (3)
Average mandatory redeploys/day	~30	~24	~5	~1
Time Spent Redeploying Annually (40-hour weeks)	7.5	6	1 .25	0.25

(1) Added methods and fields are not visible through the Reflection API, synthetic methods and fields necessary to run with FastSwap are visible, which can cause behaviour changes and issues with frameworks like Spring.

(2) Supports JSP, JSF, Servlet, EJB, JPA and CDI changes

(3) For supported frameworks: Spring, Hibernate, Stripes, Guice, Struts, Tapestry4, Velocity, Wicket, etc. SDK for third-party plugins is available. For full list see www.zeroturnaround.com/jrebel/comparison

Get to Production sooner. Complete projects 21% Faster with JRebel on WebLogic.

2. Brief Overview

This whitepaper looks into the challenge of "Turnaround" – the time spent redeploying and restarting when a developer wants to see the effects of changes they have coded. It looks into the challenges imposed by the Java language, identifies the technologies currently available to remedy the situation, and shows how to combine the functionality and feature set of WebLogic Server with the development speed and productivity that JRebel enables. Together, they empower teams to complete enterprise projects faster, make development more enjoyable for developers, and deliver high-quality applications to production sooner.

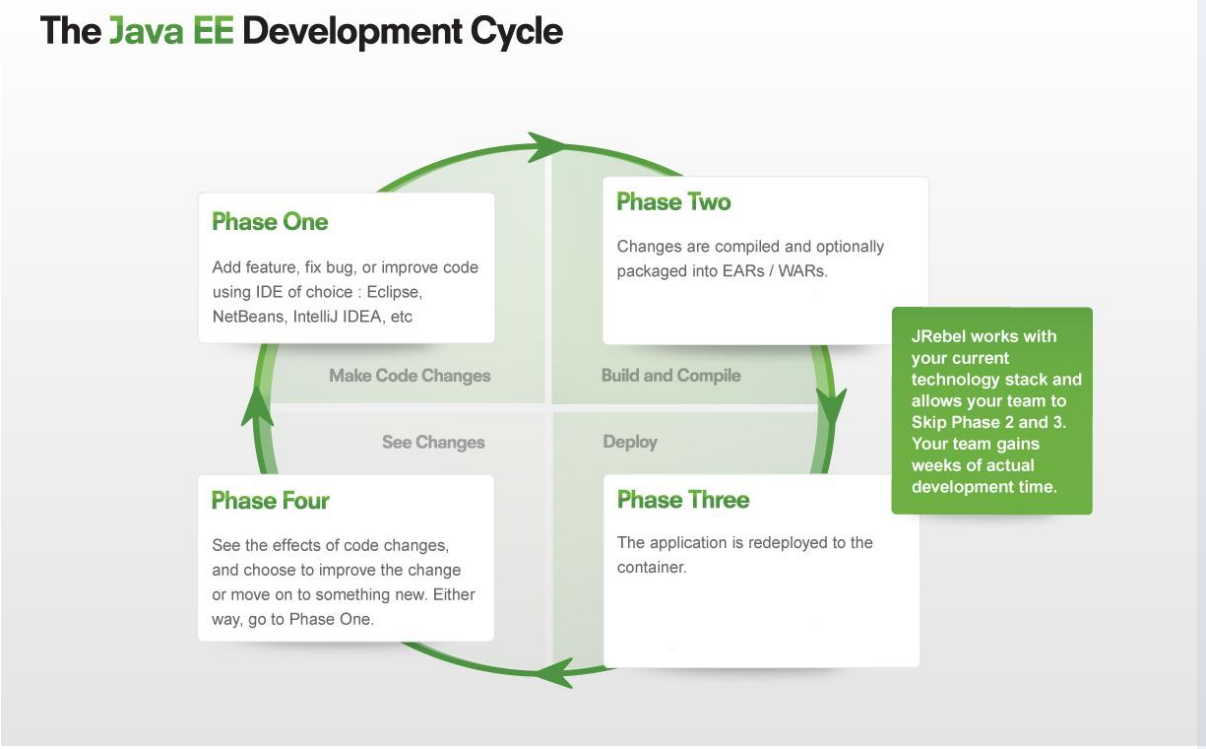
"Every time I want to see code changes in action, I have to redeploy my app server. I understand that there are technical challenges, but I believe that development should be an incremental and iterative process, where I see changes immediately - instead of watching the logs roll by, losing focus, or wasting my time."

Oracle WebLogic Server is world's best application server for building and running enterprise applications and services. It fully implements the latest Java EE standards and offers choice in development frameworks and tooling. Comprehensive and accessible management capabilities enable administration of sophisticated systems via a well-designed graphical console and/or automation. All users benefit from Oracle WebLogic Server's reliability and performance, which has been tested over years of enterprise-grade production use in demanding customer environments the world over.

JRebel is a JVM plugin that works seamlessly with WebLogic Server to eliminate the build and redeploy phases from the development cycle. Using this approach, on some applications, we estimate that JRebel can save up to 21% of the time that developers spend coding, eliminate a frustrating java-specific process, and speed up the completion of projects. JRebel supports Java as far back as Java SE 1.4, directly integrates into Eclipse, IntelliJ, and NetBeans, supports JSP, EJB, JSF, JPA, and CDI changes, as well as changes to frameworks such as Spring, Hibernate, Seam, Guice, Stripes, Struts, Tapestry4, Velocity, and Wicket.

3. Turnaround Explained

"Turnaround" is defined as the duration from coding a change to seeing the effects of that change in an application. When using Java, Turnaround has become something of an elephant in the room: everyone knows that it consumes a significant percentage of development time (and is frustrating to developers), but since solutions tend to be expensive or incomplete, most teams simply ignore the problem and take it as part of the standard Java EE Development Cycle.



Eleven-hundred Java EE Developers recently participated in a survey that led to ZeroTurnaround's WebLogic Server Redeploy and Restart Report. Based on the survey inputs, it was calculated that developers using WebLogic Server spend an average of 12.9 minutes per coding hour redeploying their code. This covers many different types of applications in terms of size and complexity, though it can be noted that firms using WebLogic Server tend to build large and complex enterprise-level projects, which take longer to build and redeploy, and rely on the dependability, scalability, and security features that WebLogic Server provides.

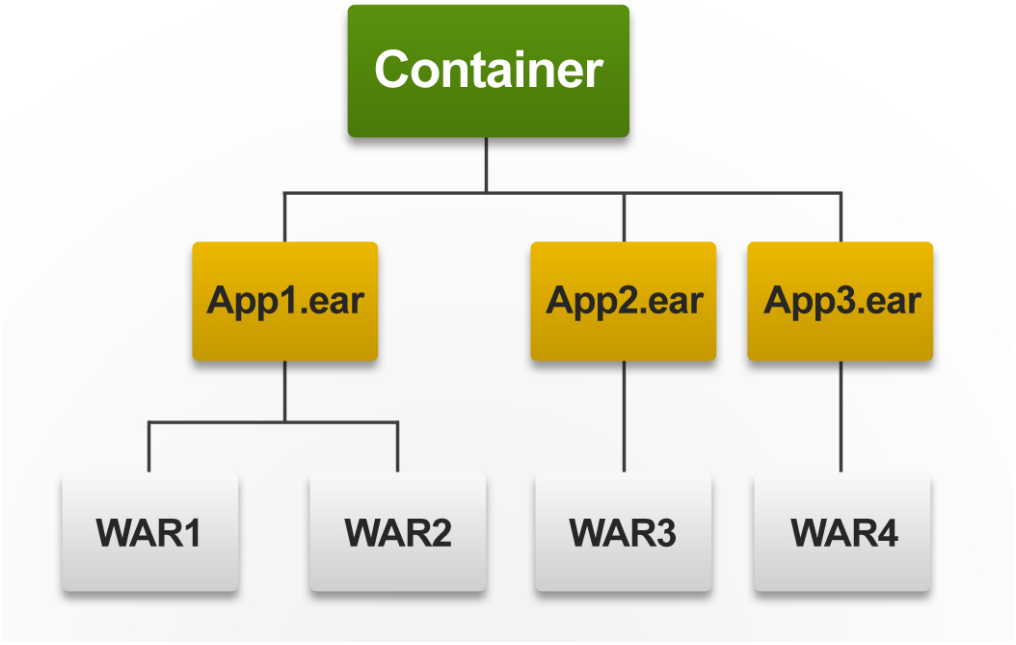
Why does the development cycle look like this?

Basically, we can look at step 2 and 3 in the development cycle as compiling classes, packaging the application as a .WAR or .EAR archive, and deploying that package to the app server. This makes sense in production, as it provides a standardized and consistent way to assemble and deploy the application. It enables WebLogic Server to manage its deployment across a wide variety of topologies, from single domains to large multi-server clusters. Unfortunately, this level of enterprise support can sometimes hinder the development process, making developers wait for every small change to be deployed.

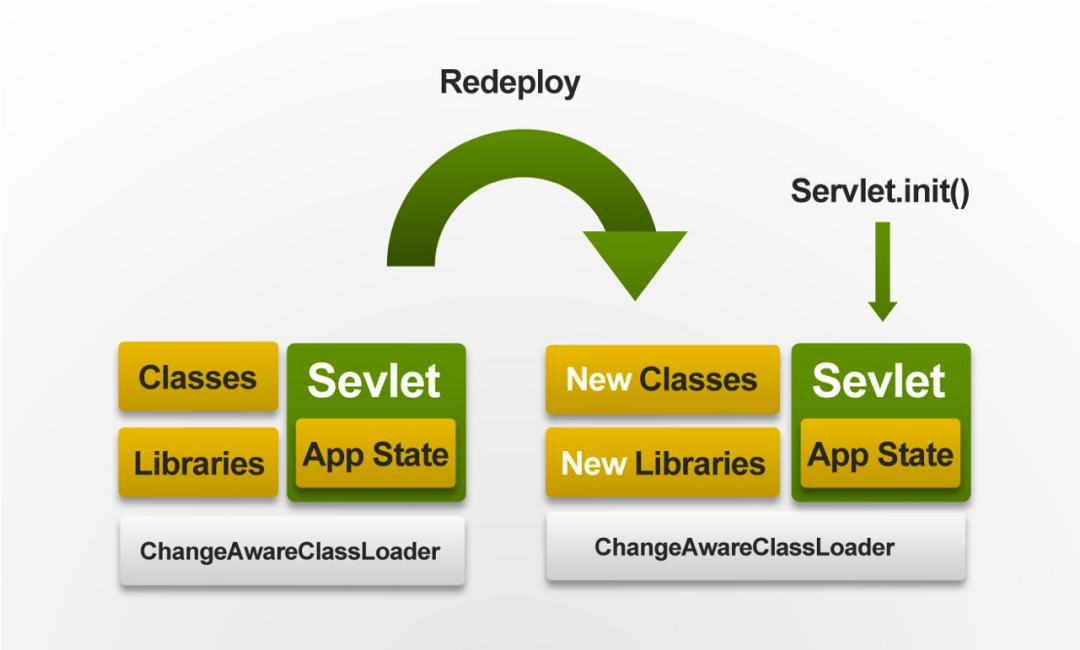
How does WebLogic Server handle redeployment using dynamic classloaders?

For rapid incremental development the server uses dynamic classloaders, like this:

1. Dynamic classloaders are created by the server. In the default configuration, WebLogic Server creates one classloader for each application and web module. If custom classloaders are configured, then every EJB module may also have a separate class loader. The classloaders form a hierarchy as illustrated:



2. Classloaders are reloaded - In WebLogic Server, each web application is managed by an instance of the ChangeAwareClassLoader which loads the web application classes. When a user presses "redeploy" in the WebLogic Console the following will happen:
    - The previous ChangeAwareClassLoader instance is replaced with a new one
    - All reference to servlets are dropped
    - New servlets are created according to the web.xml
    - Servlet.init() is called on each servlet instance



Calling Servlet.init() recreates the "initialized" application state with the updated classes loaded using the new classloader instance.

If an application is deployed as an .EAR archive, WebLogic allows redeployment of each application module separately, when it is updated. This saves time spent waiting for non-updated modules to reinitialize after redeployment.

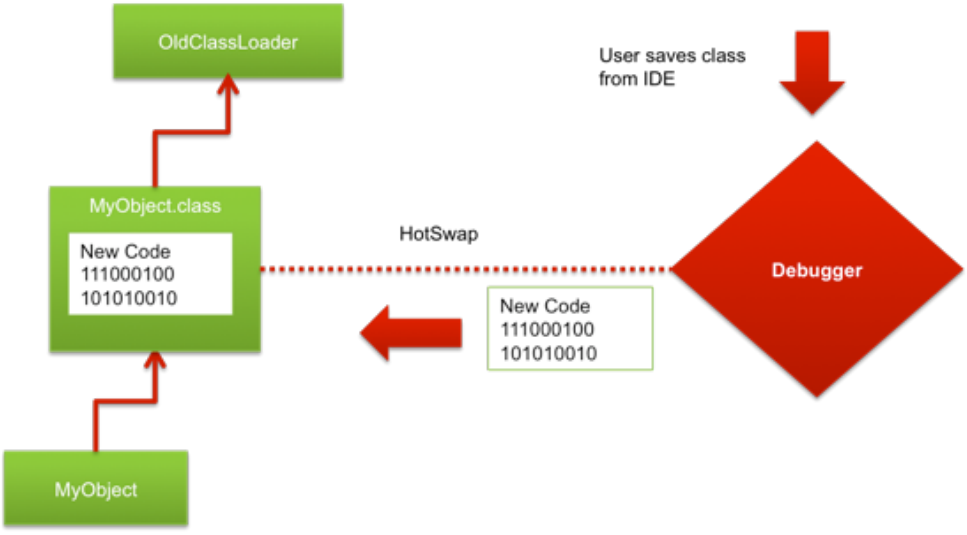
It is also possible to avoid the packaging step for some applications by deploying them to WLS in exploded directory form. Unfortunately, this doesn't eliminate the deployment step, but it does help to reduce Turnaround.

Unfortunately, although dynamic classloaders are a good step forward, they often cause other problems, and don't go far enough to eliminate Turnaround. Some issues include:

- To recreate the "initialized" state, initialization is run from scratch, which usually includes loading and processing metadata/configuration, warming up caches, and running a variety of checks (e.g. reinitializing the Spring ApplicationContext). In a small application this might take a few seconds and is fast enough to seem instant, but in a larger application this can take many minutes.
- Java EE applications on all application containers often cause the classloader of a previous application version to leak memory on redeployment. After a few redeploys an OutOfMemoryError will occur. This is an unfortunate limitation of the class loader model in the current Java ecosystem.
- According to the Redeploy Report survey results, developers reported that the average redeploy time using dynamic classloaders was 2.5 minutes, and developers redeploy an average of 5 times per hour.

4. The HotSwap Conundrum

In 2002, Sun introduced a new experimental technology into the Java SE 1.4 JVM, called HotSwap. It was incorporated within the Debugger API, and allowed debuggers to update class bytecode in place, using the same class identity. This meant that all objects could refer to an updated class and execute new code when their methods were called, preventing the need to reload a container whenever class bytecode was changed. All modern IDEs (including Eclipse, IDEA and NetBeans) support it. As of Java 5 this functionality is also available directly to Java applications, through the Instrumentation API.



Unfortunately, this redefinition is limited only to changing method bodies — it cannot either add methods or fields or otherwise change anything else, except for the method bodies. This limits the usefulness of HotSwap, and it also suffers from other problems:

- The Java compiler will often create synthetic methods or fields even if you have just changed a method body (e.g. when you add a class literal, anonymous and inner classes, etc).
- Running in debug mode will often slow the application down or introduce other problems

This causes HotSwap to be used less than, perhaps, it should be.

5. Improving Iterative Deployment with FastSwap

WebLogic Server (10.3.0 and later) provides a dynamic class swapping feature called FastSwap, in which some steps have been taken to address the iterative redeployment concerns that are imposed through the use of dynamic classloaders and the limitations inherited with HotSwap.

With FastSwap, Java classes are redefined in-place without reloading the classloader, thereby having the decided advantage of fast turnaround times. This means that you do not have to wait for an application to redeploy and then navigate back to wherever you were in the Web page flow. Instead, you can make your changes, auto compile, and see the effects immediately.

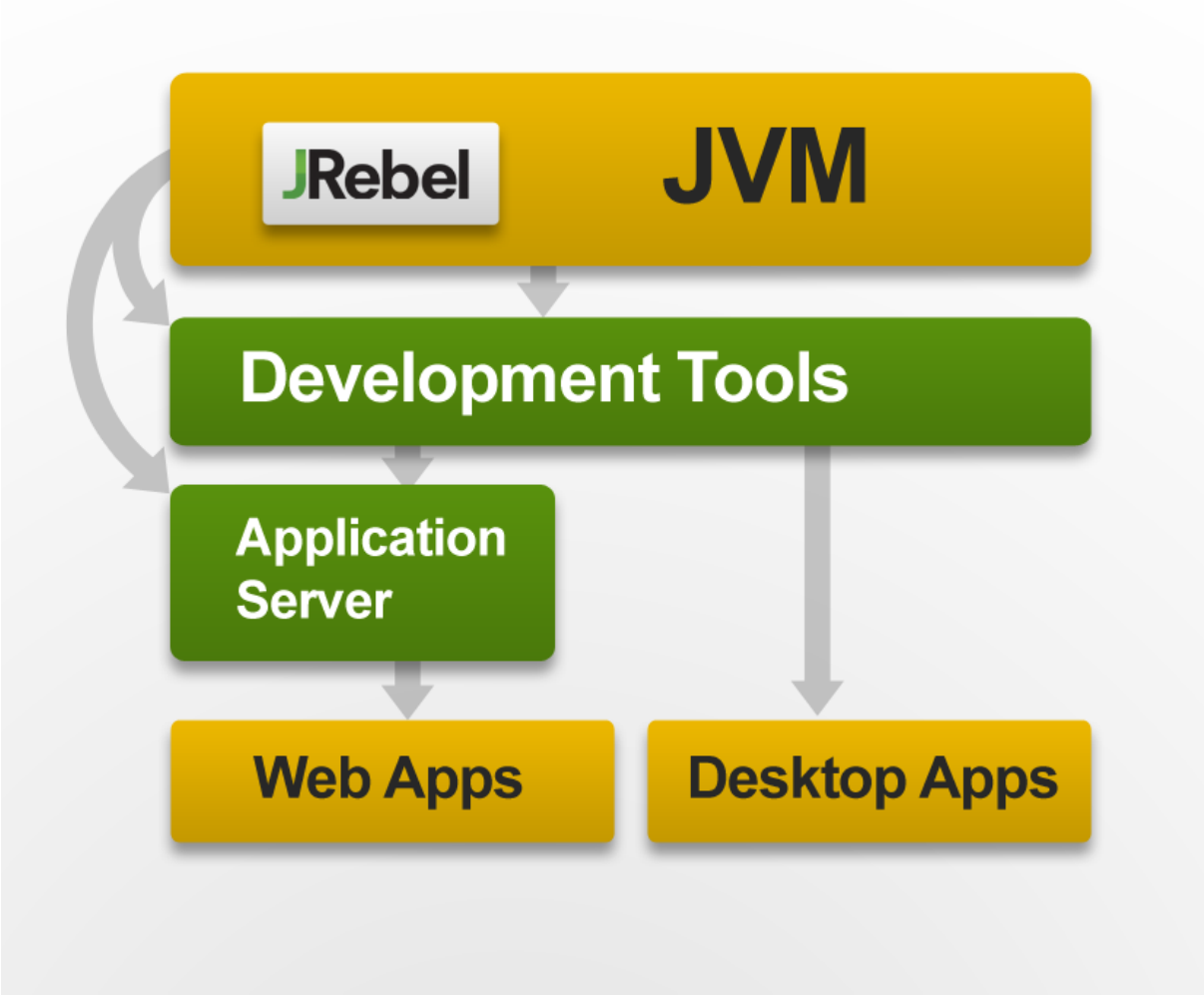
FastSwap extends the HotSwap model by providing support for dynamic redefinition of classes with new shapes. With FastSwap, changes to classes such as the addition and removal of methods and fields, adding and removing constructors, numerous static level class changes and some EJB 3.0 specific enhancements are all supported.

The use of FastSwap can help facilitiate a more effective and efficient iterative development model, particulaly when paired with the use of Oracle Enterprise Package for Eclipse that provides configurable support for using FastSwap with development projects. However
FastSwap does have some limitations in the extent of dynamic class changes it can support and the limited dynamic reloading support it offers for 3rd party frameworks.

6. Eliminating WebLogic Turnaround with JRebel

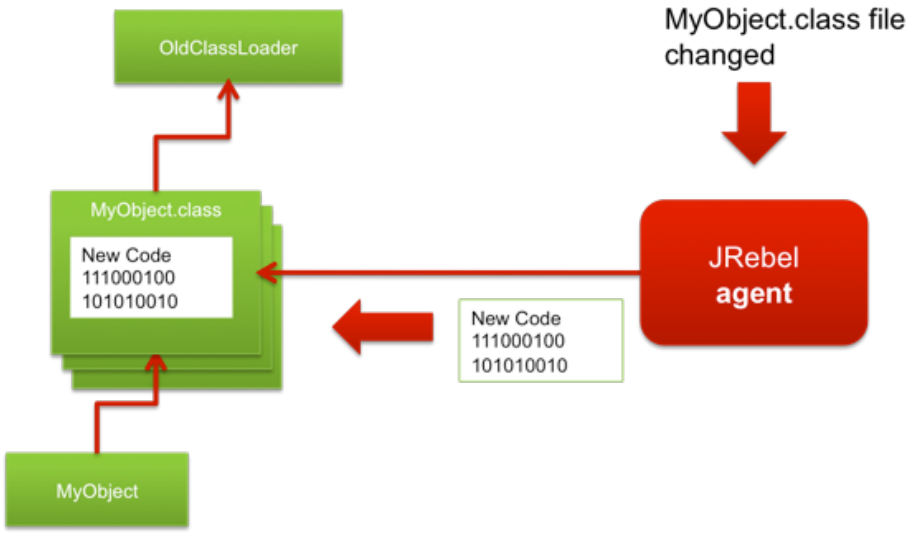
What is JRebel?

JRebel is an award-winning productivity tool for Java EE development. It is a JVM-plugin that maps your project workspace directly to your running application, so that when you change any class, configuration, or resource in your IDE, you can immediately see it in your application, thereby skipping the build and redeploy phases in the development cycle. JRebel supports Java SE 1.4 onwards, Java EE 5, and Java EE 6. It directly integrates into Eclipse, IntelliJ, and NetBeans, and supports changes to JSPs, EJBs, JSF, JPA, and CDI, as well as changes to frameworks such as Spring, Hibernate, Seam, Guice, Stripes, log4j, Struts, Tapestry4, Velocity, and Wicket.



What can JRebel handle?

The feature JRebel is best know for is picking up changes in Java classes on-the-fly. When a class is loaded JRebel will try to find a corresponding .class file for it. It will search your workspace as specified in the rebel.xml configuration file. If it finds a .class file, JRebel instruments the loaded class and associates it with the found .class file. The .class file timestamp is checked for changes when the class is used and updates are propagated through the extended class loader, to the application.

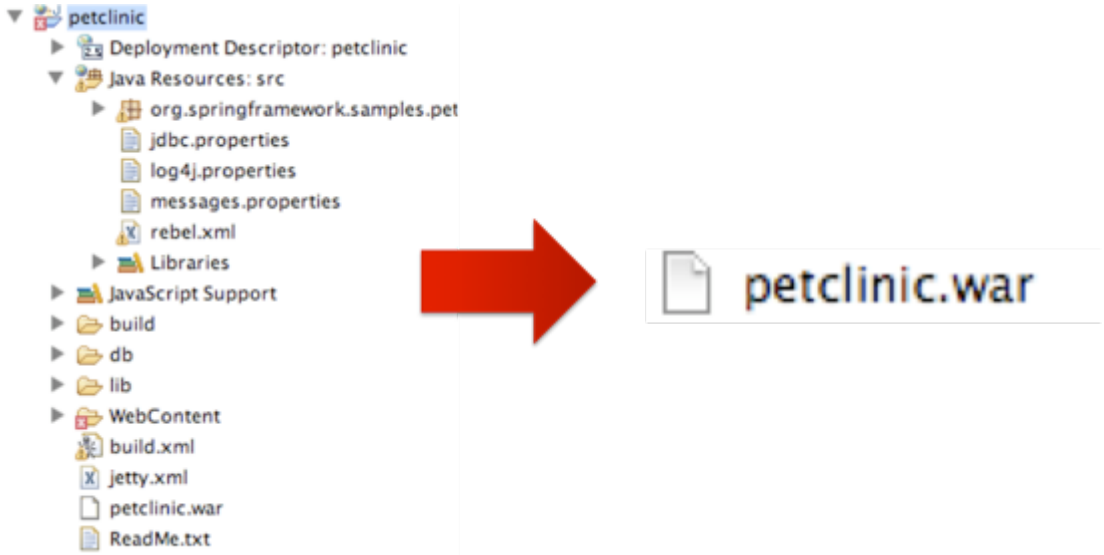


Importantly, when loading updates to a class, JRebel preserves all of the existing instances of that class. This allows the application to continue working, but means that when adding a new instance field it will not be initialized into the existing instances, since the constructor will not be rerun. State is preserved!

Unlike updating the class in the debugging session (know as HotSwap) JRebel is not limited to updates to method bodies. It can also add methods, constructors and fields, change signatures and modifiers, change values of static fields and enums, and even propagate changes to annotations. All changes are also reflected in the Reflection API. The only limitation is that changes to the class hierarchy are not supported -- that is you cannot change the “extends” relation or add an implemented interface without having to redeploy the application.

Skipping builds

In addition to handling class changes JRebel also helps to skip the build or assembly phase. According to the Build Tool Report[5], a survey of 600 Java developers, the average length of a build is 1.9 minutes, with a standard deviation of 2.9 minutes – greatly depending on the build tool used. To eliminate this phase, instead of creating a WAR or EAR for development, the rebel.xml configuration file maps a deployed application back to the IDE workspace. JRebel integrates with the application server, and when a class or resource is updated it is read from the workspace instead of the archive.



This allows for instant updates of not just classes, but any kind of resources like HTML, XML, JSP, CSS, .properties and so on. For Maven users, the JRebel Maven plugin will automatically generate the necessary rebel.xml file.

Java EE, Frameworks, IDEs and beyond

Reloading classes and skipping builds gives an important boost to developer productivity, but JRebel does even more. Using the open-source SDK a number of plugins have been created for the Java EE standards and popular frameworks that handle metadata and configuration changes in the container. Some examples include:

- Adding EJB 1.x-3.x session bean methods
- Changing JPA entities
- Changing JSF beans
- Adding Spring dependencies
- Adding Struts 1.x or 2.x Actions
- Changing Hibernate entities
- Changing ResourceBundle properties
- Adding Guice dependencies

To make configuring and using JRebel as smooth an experience as possible, IDE plugins for Eclipse, NetBeans and IntelliJ IDEA improve the debugger behaviour and provide simple options for configuring JRebel.

Configuring JRebel on WebLogic Server

To configure JRebel with WebLogic you just need to follow the Configuration Wizard launched automatically by the installer. But the three steps (on Windows) are:

1. Create a startWebLogic-jrebel.cmd in the same directory where the regular startWebLogic.cmd script resides:
   ```
   @echo off
   set JAVA_OPTIONS=-noverify -javaagent:%REBEL_HOME%\jrebel.jar %JAVA_OPTIONS%
   ```

```
call %~dp0\startWeblogic.cmd %*
```

Where REBEL_HOME should point to the directory where JRebel was installed. This step is only necessary if you run a standalone server, otherwise it's enough to enable a checkbox in the Launch configuration in the IDE.

2. Use the IDE or Maven plugin to generate the rebel.xml for your project and put it in the source directory in your workspace (make sure it's copied to the target classes directory too). It's also easy enough to edit by hand if necessary. An example follows:

```
<application>
  <classpath>
     <dir name="${myWorkspace}/petclinic/bin"/>
   </classpath>
  <web>
     <link target="/"><dir name="${myWorkspace}/petclinic/WebContent"/></link>
  </web>
</application>
```

3. Use the startWebLogic-jrebel.cmd to start the server and make sure that you see the JRebel banner in the console log. JRebel will indicate the directories it is monitoring for changes and will issue a "Reloaded class XXX" message every time you use a changed class.

From then on, all changes made in the IDE will be reloaded instantly, greatly speeding up development.

7. Tracking the effects of JRebel in your environment

The effects of eliminating Turnaround from the development cycle can be quite easily measured and tracked. JRebel itself is able to detect the number of redeploys that it has prevented, and it displays them in the log when WebLogic is first started with JRebel enable, each day.

```
#############################################################

 JRebel 3.1.3-SNAPSHOT (201009021337)
   with Enterprise Add-On! (see http://jrebel.com/enterprise)
 (c) Copyright ZeroTurnaround OU, Estonia, Tartu.

 Over the last 30 days JRebel prevented
 at least 422 redeploys/restarts saving you about 17.1 hours.

 Over the last 338 days JRebel prevented
 at least 15371 redeploys/restarts saving you about 623.4 hours.

 This product is licensed to ZeroTurnaround
 until April 10, 2011
 for up to 10 developer seats on site.

 The following plugins are disabled at the moment:
 * Grails Plugin (set -Drebel.grails_plugin=true to enable)
 * Log4j plugin (set -Drebel.log4j-plugin=true to enable)
Reloads full configuration of log4j
 * RESTEasy Plugin (set -Drebel.resteasy_plugin=true to enable)
Supports adding/changing methods with @Path annotation for servlet-bootstrapped
RESTEasy application.
 * Seam Wicket Plugin (set -Drebel.seam_wicket_plugin=true to enable)
Integration with load time weaving seam annotations to wicket classes
(-javaagent:<path-to-jboss-seam-wicket-jar>)
 * Spring Webflow plugin (set -Drebel.spring_webflow_plugin=true to enable)
Hot-load flow definitions even if in production mode.
 * Stripes plugin 1.0.11 (set -Drebel.stripes_plugin=true to enable)
Adds reloading of Stripes ActionBeans.
 * TopLink Spring Plugin (set -Drebel.toplink_spring_plugin=true to enable)
Reloads SeesionFactory when configuration changes
 * WebObjects Plugin (set -Drebel.webobjects_plugin=true to enable)
WebObjects JRebel Plugin

#############################################################
```

8. Conclusion

Reducing the cost of development projects is a very real and important concern in todays business climate. Enabling developers to work in an agile, iterative development enviroment helps to reduce the overall cost of the development effort. With the use of intelligent technologies such as Oracle WebLogic Server FastSwap and JRebel, changes made to application code and resources as the application is being developed can be instantly detected and reflected. Using these technologies, the amount of time developers spend packaging and redeploying applications in order to test their changes is significantly reduced, enabling applications to move from development to production in much less time and reducing development cost.

9. References, Further Reading

1. **Oracle WebLogic Server:** http://www.oracle.com/us/products/middleware/application-server/weblogic-suite/index.html
2. **JRebel:** http://www.zeroturnaround.com/jrebel/
3. **Reloading Java Classes – 5 part series:** http://www.zeroturnaround.com/blog/reloading-objects-classes-classloaders/
4. **WebLogic Redeploy and Restart Report:** http://www.zeroturnaround.com/weblogic-redeploy-report/
5. **Build Tool Report:** http://www.zeroturnaround.com/blog/the-build-tool-report-turnaround-times-using-ant-maven-eclipse-intellij-and-netbeans/

About Oracle

Oracle provides the world's most complete, open, and integrated business software and hardware systems, with more than 370,000 customers—including 100 of the Fortune 100—representing a

variety of sizes and industries in more than 145 countries around the globe. Oracle's product strategy provides flexibility and choice to our customers across their IT infrastructure. Now, with Sun server, storage, operating-system, and virtualization technology, Oracle is the only vendor able to offer a complete technology stack in which every layer is integrated to work together as a single system. In addition, Oracle's open architecture and multiple operating-system options gives our customers unmatched benefits from industry-leading products, including excellent system availability, scalability, energy efficiency, powerful performance, and low total cost of ownership.

About ZeroTurnaround

ZeroTurnaround builds technologies that make the Java platform more productive: both for development and production. Our Rebel technology integrates directly into the JVMs, application servers, and development tools that most teams use – making them more competitive. JRebel saves Java EE developers between 3-7 weeks of wasted development time (yup; 40-hour workweeks), has been downloaded hundreds of thousands of times, and pays for itself in under a week. Our customers are the who's who of the finance, web application, and technology industries, including many global banks plus American Airlines, Lufthansa, LinkedIn, HP, Siemans, Logica, Kayak, Oracle, IBM, and more. For rolling out or rolling back upgrades to live applications instantly, check out our next product, LiveRebel, in private beta now.