Code Optimization

The trouble with doing something right the first time is that nobody appreciates how difficult it was.
—Fortune

PACE
Patni Academy for Competency Enhancement

patni

Why is the Application Slow ?

- because it has not been performance-tuned. Fortunately, performance tuning is usually easier than debugging. When debugging, you have to fix bugs throughout the code; in performance tuning , you can focus your effort on the few parts of the application that are the bottlenecks.
- Performance is not the only consideration in developing applications. Issues like code quality, maintainability, and readability are of equal or greater importance.

PACE
Patni Academy for Competency Enhancement

patni

System Limitations

- Three resource s limit all applications:
  - CPU speed and availability
  - System memory
  - Disk (and network) input/output (I/O)
- What to Tune
  - the first step is to determine which of these is causing your application to run too slowly.

PACE
Patni Academy for Competency Enhancement

patni

Tuning Strategy

- Identify the main bottlenecks (look for about the top five bottlenecks, but go higher or lower if you prefer).
- Choose the quickest and easiest one to fix, and address it (except for distributed applications where the top bottleneck is usually the one to attack).

PACE
Patni Academy for Competency Enhancement

patni

Code Optimization

- Classes
- Methods
- Loops, Switches
- Object Creation
- String Manipulations
- Exceptions
- Sorting
- Servlets, JSP
- EJB

PACE
Patni Academy for Competency Enhancement

patni

Ex: Classes & Objects

- *Default Constructors & Constructor Hierarchies*
  a. public class cls_default {
     public static void main(String args[]) {
     cls_default x = new cls_default();}
     }

  b. class A {…}
     class B extends A {…}
     class C extends B {…}

- *Class and Instance variable Initialization*
  - Class variables – initialized only once – at program invocation
  - Instance variables initialized every time an instance is created

- *Recycling Class Instances (Objects)*
  - Creation and Garbage collection – overhead
  - Use Pooling - most container objects (e.g., Vectors, Hashtables) can be reused

- *Early and Late Initialization*
  - Early – large number of objects to be created, CPU time is critical
  - Late – few objects to be created – created just before use / never created if not required

PACE
Patni Academy for Competency Enhancement

patni

Ex: Methods

- Inlining
  - expanding the inlined method's code in the code that calls the method
- Final Methods
  - Help compilers inline methods by declaring methods / classes as final
- Synchronized Methods
  - Synchronized methods are slower than non-synchronized ones, because of the overhead associated with obtaining a lock on the method's object
- Inner Classes

```
public class meth_inner {
  private void f() {}
  class A { A() { f(); } }
  public meth_inner() { A a = new A(); }
  public static void main(String args[]) { meth_inner x = new
meth_inner();}
}
```

PACE
Patni Academy for Competency Enhancement

patni

Ex: Loops, Switches

- Move Code Out of the Loop
- Use Temporary Variables
- Don't Terminate Loops with Method Calls
- Use int for Index Variables
- Use System.arraycopy( )
- Use Efficient Comparisons
- Put the Most Common Case First
- Speedup using exception-driven loop termination

PACE
Patni Academy for Competency Enhancement

patni

Ex: String Manipulation

- Strings are Immutable
  - Following code: String str = "testing"; str = str + "abc";
  - Is Equivalent to: String str = "testing"; StringBuffer tmp = new StringBuffer(str); tmp.append("abc"); str = tmp.toString();
  - Use StringBuffer objects explicitly if you're building up a string
- Accumulating Strings Using char[] Arrays
  - Create an output char[] array, add characters to it, and then convert the result to a string
- Using == and String.equals() to Compare Strings
  - The == operator compares the references themselves and not the referenced objects
  - if (s1 == s2 || s1.equals(s2)) will short-circuit the equals() method call If the references are identical
- Interning Strings
  - str = str.intern(); adds the string to the internal pool if not already there, and returns a reference to the interned string. Interned strings can be compared with each other using ==
- Obtaining the Length of a String
  - for (int i = 0; i < s.length(); i++) VS for (int i = 0, len = s.length(); i < len; i++)
- Using charAt() VS toCharArray() and ==
- Converting Strings to Numbers – number from string 15 times faster than Double from string

PACE
Patni Academy for Competency Enhancement

patni

Ex: Exceptions

- The Cost of try-catch Blocks Without an Exception
  - try-catch blocks generally use no extra time if no exception is thrown
- The Cost of try-catch Blocks with an Exception
  - Exception should be thrown only when the condition is truly exceptional.
  - Throwing an exception and executing the catch block has a significant overhead.
  - For example, an end-of-file condition is not an exceptional condition
- Using Exceptions Without the Stack Trace Overhead
- Conditional Error Checking

Ex: Sorting

- Avoiding Unnecessary Sorting Overhead
  - Use available sort methods in java.util.Arrays & java.util.Collections
  - Optimize sort by re-implementing standard Sort as a method in the class being sorted.
- An Efficient Sorting Framework
  - If a varied and flexible sorting capability is needed, build a good sorting framework which allows generic changing of sorting algorithm and comparison-ordering methods

PACE
Patni Academy for Competency Enhancement

patni

Servlets/JSP

Avoid String Concatenation "+="

- String is immutable object
- String concatenation creates multiple, intermediate representations
- Use StringBuffer, e.g.
  - *String badStr = new String();*
  - *StringBuffer goodBuff = new StringBuffer(1000);*
  - *for (int i=0; i<1000; i++) {*
  - *badStr += myArray[i]; //crreates new Strings*
  - *goodBuff.addpend(myArray[i]); //same buffer*
  - *String goodStr = new String(goodBuff);*

PACE
Patni Academy for Competency Enhancement

patni

Avoid Overhead of Acquiring & Closing JBDC Connection

- Avoid acquiring connections directly from JDBC driver
- Use JDBC connection pooling in JDBC2.0
  - DataSource.getConnection()

e.g.

```
private DataSource ds = null;
public void doGet(…)
{…
Connection conn=ds.getConnection("db2admin",
"db2admin");
PrepareStatement pStmt = conn.prepareStatement("…");
ResultSet rs = pStmt.executeQuery(); …
```

PACE
Patni Academy for Competency Enhancement

patni

Avoid Overhead of Acquiring & Closing JBDC Connection

- Failing to close JDBC connections can cause other users to wait for connections
- Close JDBC statements when you are through with them
- Ensure your code to close all JDBC resources even in exception and error conditions

PACE
Patni Academy for Competency Enhancement

patni

Reuse DataSource for JDBC connections

- Avoid overhead of acquiring a javax.sql.DataSource for each SQL access
- Acquire javax.sql.DataSource in init() method and maintain it in a common location for reuse
- *//Caching DataSource which is obtained from init()*

```
private DataSource ds = null;
public void init(…)
{…
Context ctx = null;
try {
…
ctx = new InitialContext(…)
ds = (DataSource)ctx.lookup("jdbc/SAMPLE");
ctx.close();
} catch …
}
```

PACE
Patni Academy for Competency Enhancement

patni

Minimize Use of System.out.println

- System.out.println synchronize processing for the duration of disk I/O
  - Can significantly slow throughput
- Avoid using indiscriminate System.out.println
- Use a "final boolean" to control error and debugging situations

PACE
Patni Academy for Competency Enhancement

patni

Release HttpSessions When Finished

- Only a certain number of HttpSession can be maintained in memory
- Use HttpSession.invalidate() to release session resource

Use HttpServlet.init() to Perform Expensive Operations

- Servlet.init() only invoked once when servlet is first loaded
- Use HttpServlet.init() to perform expensive operations that need only be done once
  - init() is thread-safe
  - init() can be cached safely in servlet instance variables

PACE
Patni Academy for Competency Enhancement

patni

Don't Create HttpSessions in JSPs by Default

- By default JSP creates HttpSession
  - HttpSession is an implicit object
- If do not need HttpSession in JSP, turn it to false
  - Save some performance overhead
- <%@page session="false"%>

PACE
Patni Academy for Competency Enhancement

patni

EJBs

PACE
Patni Academy for Competency Enhancement

patni

Common tuning practices for EJBs

EJB 2 - Use Local interfaces if both EJB Client and EJB are in the same EJB Server.
EJB 1.1 - Use Vendor specific pass-by-reference implementation to make remote EJBs as Local EJBs if both EJB Client and EJB are in the same EJB Server.

Reduce network calls

- **Wrap entity beans with session beans / make entity beans as local beans.**
- **Make coarse grained session and entity beans**

Avoid unnecessary data transfer over network

- **Control serialization - modify unnecessary data variables with 'transient' key word.**

Avoid JNDI lookup overhead - Cache EJBHome references.

Avoid transaction overhead for non-transactional methods of session beans

- **Declare 'NotSupported' or 'Never' transaction attributes that avoid further propagation of transactions.**

Avoid large transaction - Set proper transaction age(time-out).
Scalability - Use clustering.
Tune thread count for EJB Server to increase EJB Server capacity.
Choose servlet's HttpSession object rather than Stateful session bean to maintain client state if you don't require component architecture and services of Stateful session bean.
Choose best EJB Server by testing with ECperf tool kit.
Choose normal java object over EJB if you don't want built-in services such as RMI/IIOP, transactions, security, persistence, resource pooling, thread safe, client state etc..

PACE
Patni Academy for Competency Enhancement

patni

Stateless session beans

- Tune the pool size - avoid creation and destruction
- Use setSessionContext() or ejbCreate() method - cache bean resources
- Release acquired resources in ejbRemove() method

PACE
Patni Academy for Competency Enhancement

patni

Stateful session beans

- Tune cache size - avoid overhead of activation and passivation process.
- Set optimal Stateful session bean age(time-out) - avoid resource congestion.
- Use 'transient' key word for unnecessary variables of Stateful session bean - avoid serialization overhead.
- Remove Stateful session beans explicitly from client using remove() method.

PACE
Patni Academy for Competency Enhancement

patni

Entity beans

- Tune the pool size for Entity Beans & Connections - avoid creation and destruction.
- Tune cache size - avoid overhead of activation and passivation process and database calls.
- Use setEntityContext() method to cache bean resources.
- Release acquired resources in unSetEntityContext() method
- Use Lazy loading - avoid unnecessary pre-loading of child data.
- Choose optimal transaction isolation level - avoid blocking of other transactional clients.
- Use proper locking strategy.
- Make read-only entity beans for read only operations.
- Use dirty flag - avoid unchanged buffer data updation.
- Commit the data after transaction completes - reduce database calls between transaction.
- Do bulk updates - reduce database calls.
- Use CMP rather than BMP - utilize built-in performance optimization facilities of CMP.
- Use ejbHome() methods for global operations.
- Use JDBC tuning techniques in BMP.
- Use direct JDBC rather than using entity beans when dealing with huge data such as searching a large database.
- Use business logic that is specific to entity bean data

PACE
Patni Academy for Competency Enhancement

patni

When to Optimize

- When developing an application, it is important to consider performance optimizations and apply them where appropriate in the development cycle. Forgetting these optimizations (or getting them wrong) can be expensive to correct later in development

PACE
Patni Academy for Competency Enhancement

patni

When Not to Optimize

- At the code-writing stage, your emphasis should not be on optimizing: it should be entirely on functionality and producing correct bug-free code. Apart from optimizations (such as canonicalizing objects) that are good design, you should normally ignore performance while writing code. Performance tuning should be done after the code is functionally correct

References

- http://www.glenmccl.com/jperf/index.htm
- http://www.precisejava.com/javaperf/j2ee/EJB.htm

PACE
Patni Academy for Competency Enhancement

patni