J2SE™ 1.5 “Tiger” Feature List

Abstract

This is a draft of the J2SE 1.5 feature list (codenamed “Tiger”)

This specification will define the target feature and API set for J2SE 1.5 The goal is that the final release will include all the high and medium priority features from this target list, but depending on implementation and API design schedules some items may be deferred to a later release if they are unable to make the release schedule. The final specification will reflect the final J2SE 1.5 deliverables.

Version 0.47(draft)

Sun Microsystems
December 16, 2003

1 Introduction

This specification will describe the feature and API set for the next feature release of Java™ 2 Standard Edition (J2SE) version 1.5, code named “Tiger”, targeted to ship summer 2004. Tiger is one of a series of feature releases to J2SE.

The intention is to ship update releases on a regular 12-18 month cycle, with each release including a combination of quality improvements and a relatively small amount of new features. In assessing features for Tiger it is necessary to balance the schedule goals for Tiger against the desirability of individual features.

1.1 Themes

It is expected that most of the effort in the Tiger release will be around product quality (bug fixing) whilst maintaining full compatibility with previous J2SE releases. The themes are therefore focused on parts of the platform that will benefit the most from a small amount of feature work including any new APIs. The Tiger release is targeted at the following major themes:

- Reliability, Availability, Serviceability
  - Monitoring and Manageability
- Performance and Scalability
- Ease of Development
- Desktop Client

1.1.1 Reliability, Availability, Serviceability - Monitoring and Manageability

The aim of this theme is to meet the needs of the growing installed base of business critical services deployed on the J2SE and J2EE platforms. One component is to enable the Java platform to integrate with industry standard RAS tools.

1.1.2 Performance and Scalability

Improve both the scalability of server based applications that use the Java platform and the performance of client and server based applications.

1.1.3 Ease of Development

The Java language and platform have been designed with ease of development in mind, this role of this theme is to drive further enhancements in this area for individual developers and developers of tools

1.1.4 Desktop Client

Java desktop users have different needs and challenges than server based developers. This theme aims to address some of those needs.

1.2 Granularity

This specification is only intended to cover "substantial" features that have significant API impact. It is not intended to cover individual bug fixes or minor RFEs. Implementation features that do not require an API change are not part of this JSR.

This specification will not itself define any new APIs, rather it will reference APIs defined in other JCP Specifications or through the JCP maintenance process. See Section 1.4 below for more details on component JSRs included in Tiger.

1.3 Prioritizing Features

Target features for Tiger will be prioritized into three buckets: Release Drivers, Group Drivers, and Targets of Opportunity.

1.3.1 Release Drivers

Release Drivers are those features that are considered essential to the success of the release. They will be given the highest priority during the development phase, and if necessary the release may be delayed in order to accommodate delays in developing a Release Driver.

Because Release Drivers can potentially delay the release, we would like to be very selective in designating features as Release Drivers.

1.3.2 Group Drivers

Group Drivers are features that will be treated as priority items within engineering teams. These features will take second place to Release Drivers, but will be tracked carefully at both the individual team level and the overall release management level.

The intention is that all Group Drivers will be included in the final release, but the release will not be delayed in order to accommodate a delayed Group Driver. Thus some Group Drivers may be dropped from the release.

We need to be careful to manage the number of Group Drivers so that engineering teams have time to address their open bug lists and their performance items as well as all their Group Drivers.

1.3.3 Targets of Opportunity (also known as "General Features")

Targets of Opportunity are lower priority features than Release Drivers and Group Drivers.

The intention is that after engineering teams have resolved their bug backlogs and have implemented their Release Drivers and Group Drivers they will move on to implementing Targets of Opportunity.

It is expected that in practice a large percentage of the Targets of Opportunity will be accommodated in the release, but this will vary from area to area, depending on the teams' progress in addressing their other work. The release will not be delayed to accommodate Targets of Opportunity.

1.4 Component JSRs

The Tiger JSR is an “umbrella” JSR. It does not by itself define any APIs. Instead it lists APIs that will be defined elsewhere in the Java Community Process. Small API updates to existing APIs are being handled through the JCP maintenance process. New APIs or major updates to existing APIs are being handled through separate JSRs. The following table lists the component JSRs that are being included in Tiger.

The last stage of JSR approval in the JCP is the “final” ballot. Before Tiger itself can go final, all its components must already have gone final. Thus each of the component JSRs must have gone through all the steps of the Java Community Process, including the “final” ballot, before this Tiger umbrella JSR can go final.

Some JSRs are initially being delivered independently from Tiger, with a separate Reference Implementation and TCK. These JSRs can go final long before Tiger goes final. However some other JSRs (e.g. JSR-163) are being delivered for the first time as part of Tiger, and their Reference Implementation and TCK is being delivered as part of the Reference Implementation and TCK for Tiger. Thus, these JSRs can only go final when there is a suitably frozen version of the Tiger Reference and TCK, which will be shortly before Tiger itself goes final.

In order to allow concurrent development, the individual JSRs are not required to advance in lock step with the Umbrella JSR. The intention of the Umbrella JSR is to permit high level review of the Tiger release contents. For the component JSRs the initial JSR requests typically provide enough information for this assessment. Thus the individual Tiger components need not be as far along in the JCP as the Tiger JSR itself. However, all component JSRs must be fully approved before they can ship in Tiger. If a component JSR goes awry (either because of schedule difficulties, or because it gets rejected by a JCP ballot) it will be removed from Tiger.

JSR	JSR Name	Tiger Features	JSR Status
003	Java(tm) Management Extensions (JMX(tm)) Specification	4639350	Maintenance Review Complete
013	Decimal Arithmetic Enhancement	4609098	Proposed Final Draft
014	Add Generic Types To The Java(tm) Programming Language	4286955	Public Review
028	Java SASL Specification	4634892	Maintenance Review
114	JDBC Rowset Implementations	4639395	Public Review
133	Revise Java memory model	4639373	Passed Community Ballot
160	Java Management Extensions(JMX(tm)) Remote API	4876725	Final Release Complete
163	Java(tm) Platform Profiling Architecture	4639363	Public Review

JSR	JSR Name	Tiger Features	JSR Status
166	Concurrency Utilities	4486658	Public Review
174	Monitoring and Management specification for the Java Virtual Machine	4530538	Public Review
175	A Metadata Facility for the Java Programming Language	4636466	Public Review
200	Network Transfer Format for Java Archives	4666040	Passed Community Ballot
201	Extending the Java Programming Language with Enumeration, Autoboxing, Extended for loops and Static import	4401321 4609038 4280390 4639378	Passed Community Ballot
204	Unicode Supplementary Character Support	4533872	Community Review
206	Java API for XML Processing (JAXP) 1.3	4614947	Public Review

Note: This is a draft JSR list

2 Index by Priority

Release Drivers

4064105 Compile-time type safety with generics (JSR-014)
4530538 JVM monitoring and management API
4533872 Unicode supplementary character support (JSR-204)
4593133 API to generate Java stack traces for all threads
4636466 Java Language Metadata (Annotation)
4639350 Add JMX support into J2SE
4639363 Java Platform Profiling Architecture (JSR-163)
4640853 Support latest Unicode version

Group Drivers

4261803 Need an unsynchronized StringBuffer
4280390 Extended for loops
4287596 JPDA pluggable connections and transports
4313885 Scanning and formatting [scanning]
4401321 Add type-safe enums to Java
4421040 JPDA: Add Generics support
4449394 JDI: provide a read-only subset of JDI; add can...(); add exceptions
4486658 add concurrency library into Java core (JSR-166)
4495742 Add non-blocking SSL/TLS functionality, usable with any I/O abstraction
4609038 Request auto boxing of primitives in Java
4609098 Add Decimal Arithmetic Enhancements into J2SE (JSR-013)
4614947 JAXP support for current XML standards, XML/Namespace 1.1, SAX 2.0.1 (JSR 206)
4632193 Swing Skins Look and Feel
4639069 Make javac produce CLDC-style verification tables
4639373 Revise Java Memory Model (JSR-133)
4639391 update classfile specification (JSR 202)
4639395 support disconnected Rowsets (JSR-114)
4666040 Support pack/crunch compression for Java downloads (JSR-200)
4674944 On-the wire interoperability
4686178 Accessibility bugs for tiger
4700777 HTTP client: Connect and read timeouts
4728816 JPDA: Add support for enums
4748085 Support DOM L3
4856541 add varargs support
4876725 Add JMX JSR-160 to J2SE
4944151 (was 4164450) document com.sun.tools.javac.Main for use in a program.

Targets of Opportunity

4173528 Add a method to generate a UUID (Universal Unique Identifier)
4495754 Add library support for common bit manipulation operations
4507539 support using dynamic proxies as RMI stubs
4533021 Container needs API to change Z-ordering of children
4623511 Documented api to call javadoc in process
4629589 Add Object Reference Template to ORB
4632213 Swing printing support
4633024 Augment Java math libraries with methods from C libm and IEEE 754
4633227 JDI spec: usage of methods implementing HotSwap feature needs clarification
4634457 Support for standard LDAP controls
4634892 Support for Java SASL API (JSR 28)
4635056 Support on-line certificate status checking protocol (OCSP)
4635454 Full pluggability for JSSE
4635618 LDAP Name Manipulation
4639378 support for importing constants
4639861 API to test reachability of a host
4667645 Improve Security Access and Control
4667658 JNLP API Enhancements
4696506 HTTP client: Improve cookie support
4696512 HTTP client: Improve proxy server configuration and selection
4702695 Add new AccessibleRelations, AccessibleRoles, AccessibleState (constants) (TP)
4702697 Introduce the AccessibleStreamable API method to javax.accessible (TP)
4813046 JVMTI spec requirements for debugging
4923484 Add RSA-OAEP parameters for XML Encryption

3 Reliability, Availability and Serviceability - Monitoring and Manageability

---

Compile-time type safety with generics (JSR-014)

ID: 4064105 Release Driver
Also in theme: Ease of development

Summary

Adding generics support to the Java language adds additional compile-time type safety checking and removes the needs for explicit casts which reduces the possibility of runtime type errors

Description

Request inclusion of JSR 014: Add Generic Types To The Java Programming Language into J2SE 1.5

The JSR proposal is to add generic types and methods to the Java programming language. The main benefit of adding genericity to the Java programming language lies in the added expressiveness and compile-time type safety that stems from making type parameters explicit and making type casts implicit. This is crucial for using libraries such as collections in a flexible, yet safe way. The proposed extension is designed to be fully backwards compatible with the current language, making the transition from non-generic to generic programming very easy. In particular, one can retrofit existing library classes with generic interfaces without changing their code.

This is an updated description the previous description is in the comments field

---

JVM monitoring and management API

ID: 4530538 Release Driver

Summary

Monitoring and managing the JVM are key components of JVM serviceability. The JVM should be able to hook into existing management consoles as well as provide opportunities for new tools

Description

The API will provide support for both monitoring and managing the core Java Runtime resources (especially for the JVM).

This will include:
- heap and memory usage of an application
- thread activity

- garbage collection and gc pauses in deployments
- GC control settings
- CPU usage
- system thresholds and resource limits (quotas)
- list number of file descriptors (and similar resources)
- info on hardware resources (CPUs, memory, etc)

This API will provide a mechanism to track when a potential low memory condition could exist [part of 4593108].

This RFE implements the subset of JSR-163 which specifies the Java monitoring and management API. It also implements the requirements specified by JSR-174 which has no API specification.

API to generate Java stack traces for all threads

ID: 4593133 Release Driver

Summary

The ability to generate a thread dump programmatically using an API is need by both J2SE client and server applications. The current mechanism doesn't work if you don't have a console on windows and retrieval on other operating systems can be improved for remote or large jvm installations

Description

Provide an API in Java to dump stack traces from Java programmatically.

The stack trace output should be in a standard well-defined format so it can be parsed by third party tools.

An array of stack trace elements can be returned for a specific thread or a map of strace trace array elements are returned for all threads.

Add JMX support into J2SE

ID: 4639350 Release Driver

Summary

JMX provides an established mechanism to support monitoring and manageability

Description

This project is intended to provide a standard Management API framework for use within the Java Platform. This will support external customers

who want to add manageability to their applications, but it will also be the base for adding more manageability support to the JRE itself.

There are two main components to this project:

Add support for JMX MBeans to the J2SE core.

Add remote access to these MBeans through JSR 160.

Java Platform Profiling Architecture (JSR-163)

ID: 4639363 Release Driver
Also in theme: Ease of development

Summary

The addition of a new profiling api is to deliver additional profiling features and improved profiling support that was previously available using jvmpi. ISVs who are currently shipping tools based on jvmpi have driven this change and will adopt the new api

Description

The specification will be for APIs to extract profiling information from a running Java[TM] virtual machine. Both time and memory profiling will be supported. Both sampling and exact mechanisms will be supported. The APIs will be designed to allow implementations which minimally perturb the profile. The APIs will allow inter-operability of profiling and advanced garbage collection technologies. The APIs will allow reliable implementation on the widest range of virtual machines, part of which will be achieved by grouping functionality into optional sets.

Queries for which optional capabilities are supported will be provided. An API will be provided by which containers may bill work to a component. The APIs will be targeted to provide a Java programming language model of execution, however, some aspects of the virtual machine, native and operating system models may be directly provided or provided via an extension mechanism. The APIs will be intended to supersede the current experimental interface - the Java Virtual Machine Profiling Interface (JVMPI) - and thus must provide roughly comparable functionality.

The APIs will accommodate implementations which can dynamically enable and disable profiling; and thus will allow implementations which have negligible performance impact when profiling is disabled. While profiling in the application development phase will be the primary goal of this specification, the design objectives for low performance overhead and data perturbation will also support profiling in the deployment phase.

This work is being developed through the JCP as JSR-163.

JPDA: Add Generics support
ID: 4421040 Group Driver

Summary

Adding Generics (see 4064105) to the Java platform includes the use of additional attributes that are not known by the current debug api. This feature adds that additional support

Description

but first - Impact of debugging on Generics

Current generics proposal and implementation includes class-file attributes for generic class signature and generic member signature but do not provide generic type info for local variables. Thus one additional attribute will need to be added.

Impact on JVMDI

JVMDI will need to be extended to provide access to the three new attributes needed for generics. An alternative is adding functionality to JVMDI to extract arbitrary attribute information, but this has two problems: it is extremely complex and a random VM may not keep this information or may not keep it in a compatible form.

Impact on JVMDI implementation (aka HotSpot)

The above changes to JVMDI would need to be added to our VM.

Impact on JDWP

JDWP would need to be extended to transport this information. Either it would need to be packaged out of context or the it could be added consistently but this would break compatibility and the JDWP version would need to be reved.

Impact on back-end and front-end

They would need to be extended to traffic in this information.

Impact on JDI

The impact of class use vs. class reference (see below) is significant on the JDI because types are transparent.

Design will take some head scratching and maybe more discussions. My first impression is that it will require several new interfaces and also functionality replicated for generics.

Two new sub-interfaces of Type will need to be added. Their exactly structure will need to be carefully considered so that existing debuggers "fail" as gently as possible. Several other methods will need to be added to Method, ReferenceType, Type, ...

Examples of functionality replication, in Method argumentTypes() and returnType() are defined; genericArgumentTypes() and genericReturnType() would need to be added. This could be rather ugly.

Impact on JDB tool

JDB should be reved to use the above new JDI functionality, this should be minor. More problematic is the impact on the expression evaluator (parser/interpretor). The parser would need to be overhauled. The interpreter currently cheats wrt type information, were it to be made correct this would add significant work. This was not our plan; our plan was to move expression evaluation down to the back-end. Generics would complicate this.

Class use vs. class definition

Where class definition is for example (of Collection):

```
public interface Collection { ... }
```

And class use is for example (of Collection in Map):

```
public Collection values();
```

Both JDI and the Doclet API use class objects (aka class definition objects) to represent class uses. For example, method objects in both cases have a method:

```
Type returnType()
```

Where one subclass of Type is a class object. This works great in Java as we know it. In generic Java, the above definitions might look something like (forgive the simplification):

```
public interface Collection<A> { ... }

public interface Map<A,B> { ...
      public Collection<B> values();
}
```

Here the use "Collection<B>" is not the same as the definition "Collection<A>".

Add JMX JSR-160 to J2SE
ID: 4876725 Group Driver

Summary

JMX is the framework used to provide the monitoring and management support in J2SE 1.5. JMX 160 provides a network connector to access this information remotely in a management console or similar tool

Description

Add all required components of JSR-160 to J2SE.

Specifically, what should be added are these two packages from JSR 160:
javax.management.remote
javax.management.remote.rmi
JSR 160 defines three further packages which are optional and which will not be included in J2SE. These packages define a custom protocol which is relatively unproven, so we do not want to freeze it into J2SE yet.

The intent is that JSR 160 can be used to access the JMX instrumentation of the JVM defined by JSRs 163 and 174. Depending on how the JVM is started, it can create a JMX MBean server containing this instrumentation, and create a JSR 160 connector making it available remotely. The default behaviour will be not to do this but new command-line options will cause it to happen. It will also be possible for users to make their own instrumentation available through JMX and JSR 160.

The package structure of JSR 160 means that the client side will also be included, even though for the uses above it is not strictly necessary.

JPDA pluggable connections and transports
ID: 4287596 Group Driver

Summary

The ability to add custom 3rd party transport mechanisms to jpda will improve debugging to devices and allows additional transports other that plain sockets or shared memory.

Description

This feature involves the ability to create a custom transport between the back and front ends of the multi-layered JPDA technology. Such a transport could be infrared, a serial cable, or other types of connections. This feature involves only putting APIs to allow custom transports, and not developing any custom transports.

JDI: provide a read-only subset of JDI; add can...(); add exceptions

ID: 4449394 Group Driver

Summary

The Java Debug Interface(JDI) is the high level debug API that interfaces to the underlying Java debug wire protocol (JDWP). For some applications read only access is allowed, this feature defines that subset

Description

Some JDI access will be read-only. Annotate the JDI specification with which methods are usable when in read-only mode and which aren't. Specify the exception that is thrown. Add something like a canModify() capability.

---

add concurrency library into Java core (JSR-166)

ID: 4486658 Group Driver
Also in theme: Ease of development ,Performance and Scalability

Summary

The concurrency utility library provides additional apis to make it easier to write threadsafe multi-threaded code. The library provides popular utilities like semaphores, atomic locks, read write locks and others.

Description

A great number of bugs occur simply due to developers need to create their own concurrency constructs on top of Java's very low-level non-oo concurrency constructs or using the Java's primitive constructs directly (necessitating a high-degree of domain knowledge in regard to concurrency issues in both cases).

By creating a standard concurrency library, the comparatively few concurrency experts can provide the typical developer a higher-level, debugged, object-oriented aid in the development of concurrent applications. The design patterns and documentation that would accompany this library would also help to increase the awareness and base level knowledge that the typical developer would have regarding concurrent programming.

JSR-166 is developing such a library through the JCP.

Make javac produce CLDC-style verification tables
ID: 4639069 Group Driver

Summary

To achieve the optimal performance from the new J2SE verifier requires the class files to contain a new verification table. javac is being extended to produce this new table.

Description

javac should have an option to produce CLDC-style attributes to support the "split verifier".

The "split verifier" is an important uograde to the classfile specification which will be added to the J2SE spec as part of Tiger feature 4639391.

On-the wire interoperability
ID: 4674944 Group Driver

Summary

RMI/IIOP interoperability is the main goal of this feature in allowing CORBA systems to work together. The reference ORB needs to track current interoperability standards

Description

This feature will include any resolutions from OMG that affect on-the wire interoperability for the J2EE Application Servers. Since CORBA 2.3.1, several interoperability issues have been submitted and resolved thru OMG. Since RMI/IIOP interoperability is one of the main goals of J2EE, Sun's ORB needs to incorporate resolutions/updates of any interop issues since CORBA 2.3.1.

Current plan is to identify major interoperability issues that have been resolved to date since CORBA 2.3.1, and incorporate resolutions in Sun's J2SE ORB. Some of the resolutions require changes to the existing APIs and implementation.

JPDA: Add support for enums
ID: 4728816 Group Driver

Summary

The JPDA APIs must expose the elements of the Java language so that they can be accessed by debuggers.

Description

This RFE
4401321 Add type-safe enums to Java

will add a new language feature to Java that must be supported in the JPDA APIs.

This RFE doesnt cover changing the JDI spec to use enums.

JDI spec: usage of methods implementing HotSwap feature needs clarification
ID: 4633227 Target of Opportunity

Summary

This feature clarifies the hotswap feature with regards to the JDI spec in order for debug tools to correctly identify changed classes

Description

Specification of the JDI methods implementing HotSwap feature, i.e.
4287595 JPDA: "HotSwap" Class File Replacement (Redefinition)
needs some clarification.

This is current specification for Method.isObsolete() in JDI part of Java SDK 1.4:

public boolean isObsolete()
Determine if this method is obsolete.
Returns:
true if this method has been replaced by
a non-equivalent method using
VirtualMachine.redefineClasses(java.util.Map).

This specification is not enough clear in two aspects.

1. The meaning of the term "equivalent method" is not defined

neither here nor in VirtualMachine.redefineClasses(Map classToBytes).
This specification should have at least reference to method
equivalence in the specification for the JVMDI function Redefine
Classes
(http://java.sun.com/j2se/1.4/docs/guide/jpda/jvmdi-spec.html#RedefineClasses):

> An original and a redefined method should be considered
> equivalent if:
> their bytecodes are the same except for indicies into
> the constant pool and
> the referenced constants are equal.

2. The statement "true if this method has been replaced by a non-equivalent
method ..." should specify that this is restricted only to those methods
which have active stack frames.

Below is a piece from evaluation of 4514956 bug which clarifies correct
usage of isObsolete() method.

It is correct for isObsolete() to return false in this use case.

In the javadoc for VirtualMachine.redefineClasses() we have the
following:

> All classes given are redefined according to the definitions
> supplied. If any redefined methods have active stack frames,
> those active frames continue to run the bytecodes of the
> previous method. The redefined methods will be used on new
> invokes.

The key phrase is:

> "If any redefined methods have active stack frames,
> those active frames continue to run the bytecodes of the
> previous method."

Only in the case of those active stack frames will a call
to thread.frame(0).location().method().isObsolete()
return true. Thus you must be holding such a StackFrame
before or during the VirtualMachine.redefineClasses() call.
Any StackFrame or any Method lookup obtained after the
redefineClasses() will discover the redefined information,
and hence isObsolete() must correctly return false.

But the spec does not say that isObsolete() is supposed to be
used mainly for the methods which have active stack frames.

There is the only statement mentioning Method.isObsolete() in the spec
for VirtualMachine.redefineClasses():
 If resetting these frames is desired, use
 ThreadReference.popFrames(StackFrame) with Method.isObsolete().

This statement looks obscure. It would be quite fruitful to give here
some clues for the right usage of Method.isObsolete() method.

JVMTI spec requirements for debugging
ID: 4813046 Target of Opportunity

Summary

The addition of these changes will make it possible to for debuggers to cleanly catch vm startup and allow a clean shutdown.

Description

This feature collects the 4 original RFEs requesting changes to JPDA.

4232338 JDWP: Need new thread status for not-yet-started threads
4199411 JVMTI spec: add start thread to VM_INIT event.
4195445 JDWP, JDI: Add return value to Method Exit Event
4195444 JVMDI spec: Need exit call

We've only fixed the following for tiger:
4195444 JVMDI spec: Need exit call
4199411 JVMTI spec: add start thread to VM_INIT event.

The rest will be done in a post tiger release.

---

4 Scalability and Performance

Add non-blocking SSL/TLS functionality, usable with any I/O abstraction

ID: 4495742 Group Driver

Summary

Customers will be able to use the non-blocking sockets from the New IO package in 1.4 with Secure Socket Layer(SSL) and the later IETF revision Transport Layer Security (TLS) sockets.

Description

The current JSSE javax.net.ssl.SSLSocket API is based on the java.net.Socket model, which by nature uses a blocking API model. The introduction of java.nio.channels API (JSR-51) brought a selectable, non-blocking I/O model to J2SE, but was not based on Sockets. Developers have requested a non-blocking SSL/TLS implementation that could be applied to such diverse I/O abstractions, such as:

Socket-Input/OutputStream
non-blocking I/O (polling) (e.g. SocketChannel-nonblocking)
selectable non-blocking I/O, (e.g. SocketChannel-Selectors)
other asynchronous I/O models (e.g. the proposed JSR 203)
local ByteBuffers/byte arrays

A new abstraction called SSLEngine is introduced which separates the SSL/TLS functionality from the I/O. The SSLEngine operates simply on inbound and outbound byte streams, and it is the responsibility of the SSLEngine user to arrange for reliable I/O transport to the peer. By separating the SSL/TLS abstraction from the I/O transport mechanism, the SSLEngine can be used with a wide variety of I/O types.

Need an unsynchronized StringBuffer

ID: 4261803 Group Driver

Summary

Significant performance improvement for common existing coding patterns.

Description

Every modification to StringBuffer is synchronized, to make it thread safe. If a large string is being built from many small pieces, this can be a significant CPU drain. I have yet to

encounter a situation where I am modifying a StringBuffer from multiple threads, so I would very much like a class just like StringBuffer that didn't synchronize.

It would be nice if such a class was included in the java.lang package.

JNLP API Enhancements

ID: 4667658 Target of Opportunity
Also in theme: Desktop Client

Summary

This feature is based on feedback to the JNLP JSR (56) to improve performance, extensibility and add additional services. JNLP is the transport mechanism used by Java Web Start

Description

Customers have requested enhancements to the JNLP API (JSR-56). These are:

PrintService (4496952)
Extension management enhancements to support optional packages (4801527, 4802593)
Single Instance Service (4390904)
ability to associate a JNLP application with a file extension (4756982)
JNLP Shortcut/Menu creation (4667651)
Selectable file service (4436034, 4467214)

Support for standard LDAP controls

ID: 4634457 Target of Opportunity

Summary

This feature provides a standard implementation of certain ldap directory routines that enables better management of large search results from LDAP queries

Description

Add support for the following two LDAP controls to the LDAP service provider in JNDI:

- Paged Results (RFC 2696)
- Server-Side Sorting (RFC 2891)

The features are already supported by classes in the com.sun.jndi.ldap.ctl package. This project migrates those classes to the javax.naming.ldap package in J2SE.

5 Ease of Development

Java Language Metadata (Annotation)

ID: 4636466 Release Driver

Summary

This feature provides an annotation mechanism to improve ease of development, especially for J2EE style web/enterprise applications by moving boilerplate type code generation to developer tools

Description

Within the Java platform there has been a growing trend towards marking fields, or methods, or classes as having particular characteristics that indicate they should be processed in special ways by development tools, or deployment tools, or run-time libraries.

For example, the JavaBeans architecture introduced various stylistic naming patterns (such as getFoo/setFoo method names) that could be used to indicate that particular methods were used for accessing properties, for registering event handlers, etc. Similarly the Enterprise JavaBeans architecture introduced various stylistic patterns that allow methods to be marked as remote methods, home methods, etc. In addition, the EJB architecture defined significant extra information in its deployment descriptors that is used to provide information on things like the persistence relationships of fields, or the transaction properties of methods, etc. Many other areas could benefit from this work, including component architectures and testing platforms.

In general, the desire to provide various kinds of auxiliary information for Java elements appears to be growing. While the existing mechanisms have been adequate for simple uses, they are becoming increasingly awkward for more complicated uses.

Since there seems to be a recurrent need to be able to provide auxiliary information on Java language elements, it appears to be appropriate to define an explicit way of doing this in the Java language, to allow arbitrary attribute information to be associated with particular classes/interfaces/methods/fields. We refer to this mechanism as "Java language metadata".

We believe there are several elements needed as part of this work:

Definition of a Java language extension that allows medatadata information to be supplied for (at least) classes, interfaces, methods, and fields. This language extension will allow metadata to be recognized by development tools. It appears likely that it will be useful to allow attribute values to be associated with given metadata attributes.
The exact syntax will need to be determined by the expert group. There appear to be a number of possibilities, including (but not limited to!) using a

Definition of a runtime delivery format for metadata and of runtime APIs so that tools and libraries can accesss metadata information at deployment time and at runtime.

Definition of rules for the attribute namespace so as to avoid accidental collisions over the same attribute name. Details will be determined by the expert group, but it seems that a mechanism similar to the Java class naming conventions might be useful.

Add type-safe enums to Java
ID: 4401321 Group Driver

Summary

Enums are the 2nd most requested viable language feature.

The typesafe enum feature significantly simplifies a commonly occurring coding pattern, adding clarity, type safety and robustness, and reducing the likelihood of error. It is a feature that would be useful to nearly every Java programmer. The proposed facility combines ease of use, power, and performance.

Description

Enumeration types similar to the c++ "enum" are not available in Java. I searched the bug reports and could not find a match, this may be from a dilution of other issues related to the terms Enumeration, Enumerated. Along with the issues and workarounds outlined in

http://www.firstsql.com/java/gotchas/lfside3.htm

there is no way to use a type checked enumeration in the java "switch" statement with any of the cumbersome
workarounds. I find I use a lot of enumerations with my coding style, it seems this would also be useful for others programming in Java. I could go on here but the point I am trying to make is simple.

(was 4164450) document com.sun.tools.javac.Main for use in a program.
ID: 4944151 Group Driver

Summary

Technologies such as java server pages that compile java code "on the fly" need to perform a large number of 'javac' operations. Executing a separate process for each run is inefficient so the compiler is called in process. However there is no standardized javac method that can be called across JVMs. This feature addresses this missing need

Description

Please document com.sun.tools.javac.Main for use within a Java program.

Extended for loops
ID: 4280390 Group Driver

Summary

Ease of development - significantly reduces verbosity of a very common construct. Reduces the likelihood of error. Interacts well with generics.

Description

I believe that having foreach functionality in the Java Language would be an excellent addition. This is easy to explain and understand, so I will be brief.

Here is some sample code of mine:

```
Iterator paneIter= contentLists.keySet().iterator();

while(paneIter.hasNext()) {

  String paneName= (String) paneIter.next();
  //... loop body, maybe just one line
}
```

I would like for 'each' functionality so that this can reduce to:

```
for(String paneName, contentLists.keySet()) {
  //... loop body (no braces necessary if just one line)
}
```

Basically, what the compiler does is understand that the second parameter to the for() construct is a Collection. Knowing what a Collection is, it gets the iterator, and can produce in a very straight forward way, the bytecode for the loop for the code I first wrote.

Scanning and formatting [scanning]
ID: 4313885 Group Driver

Summary

The scanning and formatting improvements will help developers port applications to Java and make it simpler to generate complex formatted output.

Description

An API for text scanning (based upon regular expressions) and formatting (in the spirit of C's printf procedure). This API will bring regular expressions and a compact notation for formatted output to the Java platform.

Request auto boxing of primitives in Java
ID: 4609038 Group Driver

Summary

Converting primitive types to Java objects, for example with the collections API, requires extra work by the developer to convert the type to a Java reference. This feature will convert primitive types to Java types for assignments and when passing as method parameters

Description

Provide a mechanism to auto box primitive types in Java. This is specified in JSR 201

---

support disconnected Rowsets (JSR-114)
ID: 4639395 Group Driver

Summary

From JSR 114

The current JDBC API provides an environment for creating and manipulating tabular data associated with tabular data stores. Implementations of the Rowset interface extend this model to allow tabular data to be passed between tiers and components. This ability to "disconnect" tabular data from its source increases the scalabilty of applications and the flexibility of the programming model.

Description

Provide richer support for JDBC Rowsets in the J2SE platform, especially for disconnected rowsets.

This has been a significant missing feature for some time and needs to be rectified.

---

add varargs support
ID: 4856541 Group Driver

Summary

This feature allows java methods to accept variable arguments. This allows simpler implementation of features like printf style formatting.

Description

Adding varargs to Java

Abstract

We propose to add variable argument list methods to Java. Existing methods (such as java.text.MessageFormat.format) could be retofitted to accept variable argument lists without affecting existing clients, while enabling new clients to use an improved invocation syntax. The overload resolution algorithm is modified to support variable argument lists, boxing, and unboxing while retaining backward compatibility both at compile-time and runtime. The implementation resides entirely in the compiler; no support in the VM is necessary.

Motivation

Java has two different kinds of interfaces for composing text output. The first kind consists of chained or sequential method calls such as those from java.io.PrintStream or composition using java.lang.StringBuffer (or the moral equivalent using String concatenation). These methods are convenient and statically typesafe but these techniques do not internationalize well because the order of "snippets" in a message are fixed by the order of calls in the source. The second kind consists of "formatting" classes such as java.text.MessageFormat and related classes that support internationalization. These interfaces are awkward to use and are not statically typesafe, but internationalization is easily supported because the format argument that specifies the order and content of the assembly of the resulting message can be replaced at runtime with one appropriate to the user's native tongue. Think "resource files".

The ideal would be formatting classes that are easy to use, internationalizable, and statically typesafe. These three goals cannot be achieved without nontrivial innovation in the language. Instead we propose a simple language extension that is easy to use and internationalizable, and that supports dynamic (runtime) type safety. No VM modifications are necessary.

Examples

I'll introduce the language feature by way of an example.

```
package java.text;

class MessageFormat {
   public static String format(String pattern, Object[] arguments...)
   {
      // body omitted
   }
}
```

This shows one possible way to modify an existing class to take advantage of the new language facility. This existing method has been modified by adding the new ellipsis token "..." to the declaration. The resulting method appears identical from the point of view of the VM, but the compiler allows a new invocation syntax:

```
import java.text.MessageFormat;
import java.io.PrintStream;
class Test {
   public static void test(PrintStream out, String[] args) {
      // existing invocation syntax
      out.println(MessageFormat.format("Args are {0} {1} {2}",
                                 args));
      // new invocation syntax
      out.println(MessageFormat.format("Names are {0} {1} {2}",
                                    "Neal", "Josh", "Mark"));
   }
}
```

This may not appear to be much of an advantage, but in real applications that have been internationalized a fair bit of scaffolding typically exists to simplify what would have been required before this extension. Typical clients simulate varargs by declaring a series of overloaded methods, with one, two, three, etc additional arguments. As an example of the scale of the simplifications that will be possible with the new invocation syntax, see the class com.sun.tools.javac.util.Log in the implementation of javac.

Synopsis of the Specification

JLS 8: Formal Parameters

The syntax for method declarations (JLS 8.4) and constructor declarations (JLS 8.8) are modified to support an ellipsis before the closing paren. A method declared with an ellipsis is required to have an array type as its last formal parameter.

JLS 8: Overriding

We would like to require (JLS 8.4.6.1 and 8.4.6.4) that a method that overrides a varargs method is itself declared with an ellipsis. We cannot do that for backward comatibility because retrofitting an existing method with an ellipsis would break its overriders. Instead, in -source 1.5 it would be a warning only; in -source 1.6 (or some later release) it would be enforced as an error.

JLS 13: Binary Compatibility

The binary compatibility chapter (JLS 13) is modified to require the new JVM "Varargs" attribute on those methods that were declared with an ellipsis an on no others. Or perhaps this belongs in the JVM specification.

JLS 15: Overload Resolution

Background: Overload resolution (JLS 15.12) must be modified to support boxing and unboxing conversions. Those changes require using a two-pass overload resolution algorithm. The first pass is for compatibility, and excludes boxing conversions and unboxing conversions. This ensures that existing methods and method invoations are unchanged in their semantic interpretation. Only when the first pass finds no applicable methods does the second pass take place, which considers boxing and unboxing conversions as well.

We further modify this new overload resolution algorithm for varargs. The first pass ignores ellipses in the methods under consideration, even when the methods have been retrofittted for varargs, ensuring backward compatibility. The second pass allows a sequence of arguments to match the trailing array for-

mal parameter of a method declared with an elipsis when the values can be converted to the element type of the array.

As now, overload resolution selects among the candidates by finding the most specific method. The meta rule is that one method is more specific than another if all arguments that could be accepted by the one could be accepted by the other. These new rules allow the possibility that two (or more) methods are more specific than each other; this is considered an ambiguity and results in a compile-time error.

JLS 15: Runtime evalutation of method invocation

Argument evaluation (JLS 15.12.4.2) must be modified to specify allocating an array and initializing its elements from the relevant arguments if this is necessary to match the invoked method's signature.
Examples

```
class U {
   static void f(String s, int a, short b) {
System.out.println("a");
   }
   static void f(String s, int a, int b) {
System.out.println("b");
   }
   static void f(String s, Integer[] args ...) {
System.out.println("c");
   }
   static void f(String s, Number[] args ...) {
System.out.println("d");
   }
   static void f(String s, Object[] args ...) {
System.out.println("e");
   }
   public static void main(String[] args) {
f("x", 12, (short)13);// a
f("x", 12, 13);// b
f("x", 12, 13, 14);// c
f("x", 12, 13.5);// d
f("x", 12, true);// e
   }
}
```

---

support for importing constants

ID: 4639378 Target of Opportunity

Summary

The current workarounds used by developers to create static constants requires them to implement an interface or class. This feature will allow the use of a simple import statement to achieve the same result.

Description

Many classes and interfaces defines constants as static final values such as RED, BLUE, GREEN, etc.

Programmers often want to have easy access to these constants so they can name then as simply RED, BLUE, etc.

At the moment there is no direct support for doing this in the Java language. Unfortunately a rather dangerous idiom has emerged to work around this limitation. Programmers define the constants in interfaces and then those classes that want easy access to the constants say that they implement the interface. This pulls the constants into scope.

Unfortunately this idiom doesn't simply affect the implementation of the class. It also affects its API. So for implementation convenience programmers are tempted to add implementation features into their class API.

It seems better to solve this at the Java language level by adding an explicit mechanism to import statics from a class or interface.

This might take the general form:

```
import static x.y.Z.*;
```

which would mean import all static constants from the class x.y.Z.

6 Desktop Client

Swing Skins Look and Feel
ID: 4632193 Group Driver

Summary

Skins are easier to develop and maintain than to create a new L&F programatically with the additional benefit that a large number of skins already exist

For 'themed' environments Java needs to meet the accessibility requirement as mandated by Section 508 of the Federal Rehability Code.
See http://www.access-board.gov/sec508/guide/1194.21.htm#(g) for the specific requirement in the 508 regulations: "Applications shall not override user selected contrast and color selections and other individual display attributes."

Description

Add a new plaf to Swing that will allow people to use "skins", such as those that come with GTK. This will allow developers and graphic artists alike the ability to make their own customized look, and to some extent feel, without resorting to APIs.

Support pack/crunch compression for Java downloads (JSR-200)
ID: 4666040 Group Driver

Summary

This technology has already been used to reduce JRE download sizes. This feature is aimed towards providing a technology to reduce download size for all applications improves the end user experience, adoption rates and saves bandwidth cost.

Description

Support pack/crunch compression for user based Java downloads to reduce download size.

From JSR 200

This JSR will define a dense download format for JavaTM classfiles. It is expected that this format can achieve considerable size savings over compressed JAR files.

In recent years, we have had great advancement of processor speeds with comparatively poor improvements in network bandwidth, leaving us with high performing, low cost systems operating on slow networks. With the growing popularity of the JavaTM Programming Language, the overall size and volume

of JavaTM applications have multiplied and it is desirable to reduce the download size for large web deployed JavaTM applications. Currently the JavaTM platform uses JavaTM Archive (JAR) encapsulation of these applications and their classes. The JavaTM archive format can compress these classes at the byte level only, leading to a meager reduction factor of about two. We need to compress the classes much more efficiently, thus making network transfers faster and therefore more reliable.

Accessibility bugs for tiger
ID: 4686178 Group Driver

Summary

This feature details the accesibility work as determined by the working committee for Section 508 (federal USA) rehability Code.

Java needs to meet the accessibility requirement as mandated by Section 508 of the Federal Rehability Code.
http://www.access-board.gov/sec508/guide/1194.21.htm

Description

This is a blanket list of the accessibility bugs
that need to be fixed by the Swing team in Tiger.

4634626 Implement context popup menus for components
4504068 Inconsistency in showing value
4495286 JTable needs an accessible method to select rows/cols when cell selection is false
4422535 AccessibleValue implementation only accepts Integers
4422362 AccessibleValue wrong maximum value with BoundedRangeModel components
4170173 JTextComponent.AccessibleJTextComponent.getAfterIndex works incorrectly
4303294 Implement discontiguous selection from the keyboard for list-like components

New additions
4104452 JRadioButton - Arrow keys don't work
4804344 JTree should support + and - to expand and collapse nodes.
4733624 AccessibleIcon returns null on Accessible Children of JList

---

Add new AccessibleRelations, AccessibleRoles, AccessibleState (constants) (TP)
ID: 4702695 Target of Opportunity

Summary

Several new AccessibleRelations and AccessibleRoles that are used in other desktop applications do not have equivalent support in Java. This feature adds those that are missing to the Java Accessibility API

Description

Add new AccessibleRelations, AccessibleRoles, and AccessibleState: (constants)

New relations:
"Flows_to"
"Flows_from"
"Subwindow_of"

New roles:
"Header"
"Footer"
"Paragraph"
"Ruler"

New state:
"Manages_Descendents"

Improve Security Access and Control
ID: 4667645 Target of Opportunity

Summary

Many Java Web Start customers are require higher levels of access control than currently supported by JWS. This feature would make this configuration easier for deployment and development use.

Description

We've rec'd requests from both developers and deployers to provide more control over Java deployment's security settings. This includes:

Security level support
Enterprise Security/ "Security Zones"
Accepted certificate management
Maintain Authentication Sign-ons (4371730)

Introduce the AccessibleStreamable API method to javax.accessible (TP)
ID: 4702697 Target of Opportunity

Summary

This is an accessibility requirement as mandated by Section 508 of the Federal Rehability Code.

Description

In several situations it is highly desirable for assistive technologies to parse and present the raw stream behind a component on the screen (e.g. HTML, bitmap images, MathML). This new interface provides a standard way to get at and use that stream. This interface is already in the GNOME Accessibility API.

Further, we would like to see this interface implemented on HTMLEditorKit for HTML text, and on ImageIcon (and the Swing classes that embed ImageIcons like JButton).

---

7 Miscellaneous

Unicode supplementary character support (JSR-204)

ID: 4533872 Release Driver

Summary

Supplementary characters are needed for support of additional Chinese and Japanese characters introduced since Unicode 3.0. Unicode 3.0 was based on 16bit unicode characters with a maximum limit of 65,536 characters, Unicode 3.1 and 3.2 exceed that limit with needs to handle 94,000 and 95,000 characters respectively

Description

Support Unicode "supplementary characters" (as defined in http://www.unicode.org/glossary/) throughout the J2SE platform. This affects APIs and implementations for character properties, I/O, font rendering, and other areas. Supplementary characters cannot be handled by the existing APIs accepting or returning single 16-bit char values, so a new approach is necessary.

Once support for supplementary characters is defined, the Java platform can be upgraded to support the latest Unicode version (RFE 4640853).

Support latest Unicode version

ID: 4640853 Release Driver

Summary

The supported version of Unicode in Java is 3.0. Upgrading to 4.0 will enable customers to develop up-to-date applications to meet international business and government requirements.

Description

The Java programming language and APIs use the Unicode standard as the foundation of their character representation. Unicode is an evolving standard, and the Java platform needs to be updated to support the latest stable Unicode version available at the time of code freeze.

Goal is Unicode 4.0 as the supported spec in Tiger release.

Revise Java Memory Model (JSR-133)
ID: 4639373 Group Driver

Summary

The current specification has been found to be hard to understand and has subtle, often unintended, implications. Certain synchronization idioms sometimes recommended in books and articles are invalid according to the existing specification. Subtle, unintended implications of the existing specification prohibit common compiler optimizations done by many existing Java virtual machine implementations.

Only the specifications will be updated in each case.

Description

The proposed specification describes the semantics of threads, locks, volatile variables and data races. This includes what has been referred to as the Java memory model.

The specification is expected to revise substantially Chapter 17 of "The Java Language Specification" and Chapter 8 of "The Java Virtual Machine Specification". It is not expected to result in changes to existing APIs, but clarify the semantics of some existing methods in java.lang.Thread and java.lang.Object (java.lang.Object defines the wait and join methods).

This work is being done in the JCP as JSR-133.

Add Decimal Arithmetic Enhancements into J2SE (JSR-013)
ID: 4609098 Group Driver

Summary

Big decimal support is used by financial applications however operators used for example currency calculations or mortgage calculations is missing and this feature fills those needs

Description

Add the functionality of JSR-013 Improved Decimal Arithmetic
into the J2SE core.

From JSR-013
The proposed enhancements to the BigDecimal class primarily add floating point arithmetic to the existing class, allowing the use of decimal numbers for general-purpose arithmetic (especially financial and

user-centric applications) without the overheads and potential errors resulting from conversions to and from another type. In addition, safe conversion methods are added to protect programmers from inadvertent data loss.

JAXP support for current XML standards, XML/Namespace 1.1, SAX 2.0.1 (JSR 206)
ID: 4614947 Group Driver

Summary

JAXP is a key part of Java web services support . This feature updates the current XML delivered technologies to XML 1.1 and Namespace 1.1 and Sax 2.0.1

Description

Tiger should support the latest version of JAXP.

This should include (for example) XML Schema support.

The XML current XML specifications have been revived. JAXP will be update to conform to the new specifications:
XML 1.1
Namespaces 1.1
SAX 2.0.1

HTTP client: Connect and read timeouts
ID: 4700777 Group Driver

Summary

Client Java applications cannot set timeouts on server connections at an application level. This feature will enable application level timeouts so that client applications can return promptly if remote servers are not available.

Description

The current HTTP API provides no way of requesting timeouts on connect and read operations. (Actually there is a way to request timeouts, but it involves system properties and it only sets timeouts on a global basis.)

This feature will be of obvious benefit to any HTTP client application that must behave robustly in the face of server failure.

Support DOM L3
ID: 4748085 Group Driver

Summary

JAXP is an important part of Java web services support, this feature increments the supplied technologies to support Document Object Model Level 3

Description

Dom Level 3 will be out in the J2SE 1.5 time frame. It will be advantageous for JAXP being included in J2SE 1.5 to support.

Container needs API to change Z-ordering of children
ID: 4533021 Target of Opportunity

Summary

Currently the only way to change the order of how components are overlapped is to remove the components and add them back in a different order. This feature will provide an api to do this programmatically which can then be used when navigating though components via a keyboard

Description

When an application that uses partial or completely overlapping stacked canvases is implemented using Java, it usually
requires creating a lightweight panel for each stacked canvas and adding all those panels as children of some top level lightweight window. Each panel can in turn contain a number of components.

During application execution, those canvases need to be raised and lowered programmatically. A common case is that the user uses the keyboard to navigate to a text component on a canvas which is currently obscured by another canvas, and therefore needs that canvas to come to the top of the Z-order.

The current implementation of Z-ordering in a container is static in the sense that Z-order of each child relative to the others is decided once at the time the child is added to the container, and never altered. This means the only way we can alter the Z-order of existing components in a container is by removing some set of components and adding them again to the same container at with

different indices. This is not ideal, because the removal process causes handlers to be called, resources to be destroyed, and focus to shift, which in turn requires a workarounds to get everything back to the state where it was before the Z-order modification.

We propose that an API be added to the Container class to allow Z-order to be changed dynamically. _Raising_ a child component in the Z-order of its parent container using this API should not generate focus events or cause any other unnecessary handlers to be called, though lowering a child could potentially have more ramifications, and perhaps it might make sense to add new listeners and events to track movement in the Z direction.

This API could mirror "add(component, index)" on Container and perhaps be called "setZOrder(component, index)".

Add Object Reference Template to ORB
ID: 4629589 Target of Opportunity

Summary

This feature is being evaluated to add basic RAS features such as load balancing support in the J2SE ORB

Description

Work has been proceeding to define an object reference template standard through the OMG. This is an extension to portable interceptors that supports portable server activation frameworks, load balancing, fault tolerance, dynamic bridging, and other advanced features. We plan to add this to the J2SE ORB.

API to test reachability of a host
ID: 4639861 Target of Opportunity

Summary

Raw socket support has been discussed for a number of releases, the ability to ping from within Java has been seen as missing and been a popular feature request. A number of third party & open source APIs are available but there is no standard API that network management applications and tools can rely on.

Description

This RFE is submitted to track the requirement to test the reachability of a host (or more likely an InetAddress). This RFE stems from the

feedback to 4093850 "ICMP protocol support a.k.a. PING applets" where it clear that many developers have a basic requirement to test if a host was reachable (something akin to the ping utility).

In its simplist form it we could test if a host is reachable within a specified timeout. Another variant could be to test if a host is reachable from a specified java.net.NetworkInterface.

Implementation-wise there needs to be flexibility to choose how reachability is determined. If we can create an ICMP socket then we can send/receive an echo request/reply ala classic ping. Alternatively if we can't create an ICMP socket then a UDP or TCP approach might be used (for example we could send a UDP packet to the echo port and handle timeout/port unreachable).

---

support using dynamic proxies as RMI stubs
ID: 4507539 Target of Opportunity

Summary

This feature would benefit the customer by simplifying the development and deployment of applications that use RMI. One build-time step (use of the rmic tool) would be eliminated, and a major potential source of errors (using out-of-date stub classes, incorrectly installed stub classes, etc.) would be eliminated as well.

Description

Currently, exporting a remote object from J2SE RMI requires the presence of a pre-generated stub class whose name is a function of the remote implementation class being exported. This stub class typically must be generated as a separate compile-time step using the "rmic" tool, and it must be made available to all clients of the remote object through various possible means (HTTP server, file system, etc.).

It would be very straightforward for RMI to instead use a dynamic proxy class (obtained using the java.lang.reflect.Proxy API) for the remote stub of a remote object being exported-- in fact, java.lang.reflect.Proxy was designed with this application in mind, and its generated dynamic proxy classes are very similar to the stub classes generated by "rmic" with the "-v1.2" option used. This would require one new public class in the java.rmi APIs: a public InvocationHandler class whose instances contain a java.rmi.server.RemoteRef instance.

Supporting the use of dynamic proxies as RMI stubs would simplify the development and deployment of RMI applications by eliminating the "rmic" step from the build process and by eliminating the need to provide for the distribution of remote stub classes.

It would also facilitate exporting dynamic proxy instances as remote objects

themselves (which is currently only prevented by the need for named stub classes), which has also been a highly requested feature-- which would allow for more convenient logging, checking, and other server-side pre- and post-processing handling of remote method invocations.

Documented api to call javadoc in process
ID: 4623511 Target of Opportunity

Summary

Addiing an api to call javadoc in process will make it easier for IDE tools to read javadoc information without having to start a separate javadoc program. This improves performance on desktop clients

Description

Documented api to call javadoc in process with a JVM rather than as a separate process. This makes it easier for IDE tools to use the output without having to re-read the information from a file

---

Swing printing support
ID: 4632213 Target of Opportunity

Summary

This feature resolves some the anomalies experienced when printing certain Swing components

Description

Swing currently doesn't extend the default printing behavior. The result is often unexpected behavior when printing a Swing component. For example, if you print a text page, Swing will print the Scrollbar to paper.

Swing should extend the default printing behavior to paint the basic contents of the component in a manner consistent with the output device.

Initially will be targeting the JTable Component.

This is an umbrella printing bug, the JTable change is tracked as 4791650

Augment Java math libraries with methods from C libm and IEEE 754
ID: 4633024 Target of Opportunity

Summary

This feature is to complete the support of Java mathematic functions for those developers who want to migrate from C to Java.

Description

The current Java math libraries (java.lang.Math and java.lang.StrictMath) lack many methods found in the C math librariy (e.g. hyperbolic transcendental functions) as well as functions recommended by the IEEE 754/854 floating-point standards. Java's math library should be augmented to include these methods.

The list of major math library methods added are

log10
cbrt
hypot
sinh, cosh, tanh
log1p
expm1

Additionally, ulp and signum were added.

Full pluggability for JSSE
ID: 4635454 Target of Opportunity

Summary

The level of encryption and its restrictions differs widely, this feature will allow 3rd parties to add new cryptographic providers to the Java Secure Socket Extension without having to release their own JRE

Description

JSSE in J2SE 1.4 allows certain pluggabilities (i.e., crypto providers), but it has several limits. We should provide full pluggability for JSSE.

LDAP Name Manipulation
ID: 4635618 Target of Opportunity

Summary

Developers are currently forced to manipulate LDAP-style names at the character level. The rules are complex enough that it is difficult to get this completely right. This feature will provide a few methods that provide this support and improve ldap deployments

Description

The JNDI API provides a generic way to access names across multiple naming systems. Due in large part to this generality, there is little in the way of support for performing syntactic manipulations that depend on idiosyncrasies of the underlying naming systems. The manipulation of LDAP-style names is particularly common, and is particularly difficult without such support. Methods will be added to the javax.naming.ldap package to assist developers with the escaping of name components while composing LDAP names, and the unescaping of name components while decomposing them.

HTTP client: Improve cookie support
ID: 4696506 Target of Opportunity

Summary

There is no standard way for applications to parse the Set-Cookie value to manage cookies as a Java http client. This feature will add this functionality

Description

Improve cookie support. Currently cookie support is limited to header set/get and the HTTP API lacks a framework for managing cookies.

There are no standard Java classes to support cookies on the client side, and only limited cookie support for servlets. At best, clients must directly examine the "Set-Cookie:" and "Set-Cookie2:" HTTP headers themselves, and parse the values.

On the server side, support for cookies is limited to the servlet "Cookie" class APIs, which are basically a set of instance variable accessor methods only, plus the ability to retrieve cookies from and add them to a client's HttpServletRequest object. They do not, however, aid servlets which also wish to act as HTTP clients in their own right.

There is no support for collections of cookies, managed as a single object instance, for either clients or servers.

There is very limited support for handling URL redirections for HTTP clients (the only option is for the client to deal with them itself), and no support for managing cookies during URL redirections.

Add a method to generate a UUID (Universal Unique Identifier)
ID: 4173528 Target of Opportunity

Summary

This feature requests the addition of a Universal Unique Indentifier in Java, this value can be generated using the host operating system and is useful for persistant Java objects

Description

This feature is to add UUID functionality to the J2SE platform by the addition of a UUID class for manipulating Leach-Salz variants.

The uuid (Universal Unique Identifier) is an XOPEN standard for creating a globally unique id.

Quoting below from:
http://www.opengroup.org/onlinepubs/9629399/apdxa.htm

"A UUID is an identifier that is unique across both space and time[note 1], with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

The generation of UUIDs does not require a registration authority for each single identifier. Instead, it requires a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address, which is usually already applied to network-connected systems. This 48-bit address can be assigned based on an address block obtained through the IEEE registration

authority. This UUID specification assumes the availability of an IEEE 802 address. "(end quote)

Note 1:
http://www.opengroup.org/onlinepubs/9629399/apdxa.htm#tag_foot_1

We use a UUID as an object id for each business object. These objects could be created in different countries (i.e different web servers) and are replicated to many databases. The best solution to this is the uuid. We are writing an all Java likeness to the uuid, but since there is no 100% pure Java way to access the 48-bit IEEE 802 address (network card id) then we cannot faithfully reproduce a true uuid.

On some systems, generating a time-space based UUID requires privileged user credentials and is not currently implemented, time, random,name and DCE are available

---

Add library support for common bit manipulation operations
ID: 4495754 Target of Opportunity

Summary

This feature requests the addition of additional operations on data bits. While programmers don't need these every day, they're difficult to get right and extremely difficult to get fast. We already have fast, correct implementations hidden inside of BigInteger (private methods). Further, once we've made these into public mehtods (presumably in Integer and Long), VMs will be free to intrinsify them, at which time they'll be blindingly fast.

Description

Java adds the arithmetic shift operator, but roll and count leading zeros would be very nice intrinsic routines (bytecodes anyone). These are very important for bit-twidling, and have fast processor-specific equivalents on most processors.

Support for Java SASL API (JSR 28)
ID: 4634892 Target of Opportunity

Summary

This feature provides a standard API allows customers to supply their own SASL mechanisms in an open and supported way.

The Java SASL API (and the underlying supported SASL mechanisms) may be used by

other standards that use SASL, such as IMAPv4.

Description

The LDAP service provider supports SASL authentication by using a preview of the Java SASL API (JSR 28). In the Tiger timeframe, JSR 28 will be final. The API should be integrated into Tiger and the LDAP service provider should be modified to use it.

Also migrate SASL mechanisms (digest-md5, GSSAPI, and External) to use JSR 28.

Support on-line certificate status checking protocol (OCSP)
ID: 4635056 Target of Opportunity

Summary

This feature will allow customers to be able to check the status of a certificate on-line which is important in ensuring security.

Description

Provide client side support for OCSPv1 (RFC 2560) as an alternative revocation check mechanism to CRLs. This is to be integrated into the CertPath API and the Sun CertPath provider.

HTTP client: Improve proxy server configuration and selection
ID: 4696512 Target of Opportunity

Summary

This feature is to add management of HTTP proxy servers by using a HTTP client API

Description

Improve proxy server configuration and selection. Currently proxy server configuration is static, global, and configured only by system properties. This needs improvements so that proxy server configuration is dynamic, controllable by web containers and some applications, and failure of proxy servers needs to be handled. This item will require small-scale API additions.

Add RSA-OAEP parameters for XML Encryption
ID: 4923484 Target of Opportunity

Description

In section 5.4.2 of the W3C XML Encryption recommendation (http://www.w3.org/TR/xmlenc-core) it specifies the RSA-OAEP algorithm for key transport.

The JCE does not allow an application to specify the OAEPParams element as input to the RSA/ECB/OAEPWith<digest>And<mgf>Padding transformation.

---

8 Open Issues

Note: Open Issues are to be resolved before final release of this draft.

Issue with dropped Target of Opportunity 4629583 "Bring J2SE ORB up to date with recent OMG specification" The current level of CORBA compliance is 2.3.1 with modifications and with interoperability GIOP 1.2.

Description of 4629593:

The current Sun ORB shipping as part of J2SE 1.4 is CORBA 2.3.1 compliant.
We need to update the ORB in J2SE 1.5 for compliance to CORBA 2.5.
in the areas of Java to IDL/IDL to Java and POA

Also request of inclusion of the following Java exceptions
org.omg.CORBA.REBIND
org.omg.CORBA.TIMEOUT
org.omg.CORBA.TRANSACTION_UNAVAILABLE
org.omg.CORBA.TRANSACTION_MODE
org.omg.CORBA.BAD_QOS
org.omg.CORBA.ACTIVITY_COMPLETED
org.omg.CORBA.ACTIVITY_REQUIRED
org.omg.CORBA.INVALID_ACTIVITY

In addition,the inclusion or exclusion of this feature should not prevent an implementor's ability to supply a later CORBA orb through an approved mechanism. Any issues preventing this are to be resolved as part of this specification.

9 Feature Index

9.1 Index by Feature ID

9.2 Index by Theme

Reliability, Availability and Serviceability -Monitoring and Manageability

Scalability and Performance

Ease of Development

Desktop Client

Miscellaneous

9.3 Language Specific Changes

4064105 Compile-time type safety with generics (JSR-014) 8
4280390 Extended for loops 24
4401321 Add type-safe enums to Java 23
4609038 Request auto boxing of primitives in Java 25
4636466 Java Language Metadata (Annotation) 22
4639378 support for importing constants 28

9.4 Alphabetical Index

10 Feature List Change History

Changes from 0.1 to 0.2:

Added introduction
Removed 4593108 Detect Low memory conditions. This is now part of the Monitoring and Manageability JSR request
Removed 4533879 Enhance remote debugging/ Moved to implementation document
Removed 4665444 Improve Java libraries startup time. Moved to implementation document
Removed 4665470 improve memory footprint of the J2SE platform. Moved to implementation document
Removed 4607289 Reduce download size of the JRE. Moved to implementation document
Added 4629583 Bring J2SE ORB up to date with recent OMG specifications. Moved from implementation document
Removed 4629608 idlj does not fully support local interfaces. Moved to implementation document

Changes from 0.2 to 0.22:

Clarified 4639391 update classfile specification
Added details of 4593108 to 4530538 JVM monitoring and management API
Moved 4421040 to RAS theme: JPDA: Add Generics support
Clarified 4638307 Event Dispatch On Main Thread
Moved 4615046 to RAS theme: Provide conditional breakpoints in debug api
Updated 4639069: Make javac produce CLDC-style verification tables
Title of 4614947 updated. Now Support for XML Schema in JAXP (Tiger).
Removed 4607361 API for reading class files without loading them.
Moved 4667645 to desktop client theme: Improve Security Access and Control of Java Web Start
Updated 4629589: Add Object Reference Template to ORB
Updated 4639861: Title now API to test reachability of a host
Added 4633024: Augment Java math libraries with methods from C libm and IEEE 754
Clarified 4504839: Java libraries should provide support for unsigned integer arithmetic

Changes from 0.22 to 0.3:

Added Summary to each Feature
Added Index by priority and list of language features
Gathered Features 4195444, 4195445, 4199411 and 4232338 into JVMTI feature 4813046
Moved 4635454 JSSE feature from XML theme to Misc theme
Split feature 4614947 jaxp into 2 parts, new feature created is 4748085 for DOM support
Moved 4666040 pack and crunch into desktop client theme (from misc theme)
Moved 4609098 Big decimal to group driver as it is a JSR being planned
Removed 4615533 (3d changes) was a target of opportunity with no resource
Removed 4630118 (weak reference) was a target of opportunity with no resources.
JSRs 199, 200, 201, 202, 203 and 204 have been launched from existing features

Changes from 0.3 to 0.4:

Dropped Feature 4639391, JSR 202 Updates to the classfile specification
Dropped Feature 4313887, JSR 203 More New I/O APIs for the Java Platform
Dropped Feature 4599433, JSR 121 Application Isolation API Specification
Dropped Feature 4607414, JSR 031 XML Data Binding Specification
Dropped Feature 4607419, JSR 101 Java(tm) APIs for XML RPC
Dropped Feature 4635230, JSR 105 XML Digital Signature APIs
Dropped Feature 4635231, JSR 106 XML Digital Encryption APIs
JSR 206 has been launched from feature 4614947
Removed XML and Web services theme, moved feature to theme Miscellaneous
Dropped Feature 4639069 Make javac produce CLDC style verification tables

Target of opportunity drops
part 1
4052440 Pluggable locale support
4504839 Java Libraries should provide support for unsigned integer arithmetic,
4638307 Event dispatch on Main Thread
part 2
4615046 Provide conditional breakpoints in debug api
4059717 JPDA: Want to be able to set program counter in debugger
4287600 JPDA back-end expression evaluation
4362594 JDWP: Need a way to send output and error streams to debugger
4389187 JDI: Let the user specify the current working directory when starting debugger

Promoted from Target of opportunity to Group Driver
4607272 New IO: Support asynchronous I/O
4609038 Request auto boxing of primitives in Java

New additions
4261803 Need an unsynchronized StringBuffer
4856541 add varargs support
4702674 Finish implementing Accessibility on AWT (native method work, keyboard accessibi
4702695 Add new AccessibleRelations, AccessibleRoles, AccessibleState (constants) (TP)
4702697 Introduce the AccessibleStreamable API method to javax.accessible (TP)

Changes from 0.4 to 0.41

Fixed index by priority to reflect changes in 0.4
removed 4667651, 4756982 and 4779551 added in error from implementation doc
4608895 support CIM/WEBM for monitoring and managing the JVM was dropped

4607272 and 4609038 priority updates were not reflected in document, only in change log

Changes from 0.41 to 0.42

Moved indices to the end to speed regeneration
Re-introduced dropped Feature 4639391, JSR 202 Updates to the classfile specification as a Group Driver instead of release driver for verifier updates
Moved Group Driver Feature 4608529 Complete support of ipv6 in J2SE to implementation document, work scoped to windows updates only, QoS api is not stable
Dropped Group Driver Feature 4607272 New I/O: Support asynchronous I/ due to resource constraints
Re-introduced dropped Group Driver Feature 4639069 Make javac produce CLDC style verification tables for verifier updates
Added Group Driver Feature 4728816 JPDA: Add support for enums
Added Group Driver Feature 4728827 JPDA: Add support for Java Language Metadata
Added Group Driver Feature 4876725: Add JMX JSR-160 to J2SE for remote M&M support

Dropped Target of Opportunity 4050435 Improved interactive console I/O
Dropped Target of Opportunity 4527345 New I/O Add MulticastChannel
Dropped Target of Opportunity 4613021 Provide a high resolution timer in j2se
Dropped Target of Opportunity 4629583 Bring J2SE ORB up to date with recent OMG specification
Dropped Target of Opportunity 4639867 Migrate networking system properties to user preferences
Dropped Target of Opportunity 4640520 Refine sun.misc.Service and promote it to java.util
Dropped Target of Opportunity 4640544 New I/O: Complete socket channel functionality
Dropped Target of Opportunity 4640564 Add support for MIME type parsing and construction
Dropped Target of Opportunity 4696485 HTTP client: Support pipelined requests
Dropped Target of Opportunity 4702674 Finish implementing Accessibility on AWT

Description of 4635060 Changed from Support access info & distribute point extensions to Full support for CRL distribution points extension
Feature 4164450 javac compiler interface doesn't require JSR 199 changes, only a small api addition

Changes from 0.42 to 0.43

Dropped Group JPDA: Add support for Java Language Metadata Driver. Support for metadata will still be available via java.lang.reflect

4813046: JVMTI spec requirements. Description clarified
4495742: Add non-blocking SSL/TLS functionality. Description clarified to match implementation
4666040: Support Pack/Crunch compression. Updated description to match JSR.
4686178: Accessibility bugs for tiger. Updated with additional items from IBM
4609098: Add decimal arithmetic enhancements, description updated from JSR
4748085: Support DOM L3. Tidied description
4632213: Swing printing support. Reduced scope to Jtable component

4633024: Augment Java math libraries. Updated list of math functions
4173528: Add a method to generate a UUID. Minor clarifications to the description

Dropped Target of Opportunity 4635060: Full support for CRL distribution points extension

Changes from 0.43 to 0.44

Dropped Group Driver 4639391, JSR 202 Updates to the classfile specification. Minor attribute changes required for language changes will need to be document as part of the maintenance review
Also dropped related Group Driver Feature 4639069 Make javac produce CLDC style verification tables for verifier updates

4615460: Enhancements to Swing/JFileChooser for IDE tools dropped from the feature list. However components of the feature, (1e) repeat keyboard navigation and (2) file renaming, are fixed in 1.5 by work done by bugs 4654916 and 4887433

4813046: JVMTI spec requirements for debugging, minor clarification to jvmti debug bugs

Feature 4164450, originally JSR 199 and rescoped in version 0.42 is now replaced with 4944151 document com.sun.tools.javac.Main for use in a program.

Addition of Target of Opportunity 4923484: Add RSA-OAEP parameters for XML Encryption

Changes from 0.44 to 0.45

Included Open Issues page
Update JSR status at front

Changes from 0.45 to 0.47

Clarified resolution time for open issues.
Included additional note for open issues regarding ability of licensees to ship a later CORBA version

---