语言增强功能

Java Programming Language Enhancements

Enhancements in Java SE 7

- Binary Literals - In Java SE 7, the integral types (`byte`, `short`, `int`, and `long`) can also be expressed using the binary number system. To specify a binary literal, add the prefix `0b` or `0B` to the number.
- Underscores in Numeric Literals - Any number of underscore characters (_) can appear anywhere between digits in a numerical literal. This feature enables you, for example, to separate groups of digits in numeric literals, which can improve the readability of your code.
- Strings in switch Statements - You can use the `String` class in the expression of a `switch` statement.
- Type Inference for Generic Instance Creation - You can replace the type arguments required to invoke the constructor of a generic class with an empty set of type parameters (<>) as long as the compiler can infer the type arguments from the context. This pair of angle brackets is informally called the *diamond*.
- Improved Compiler Warnings and Errors When Using Non-Reifiable Formal Parameters with Varargs Methods - The Java SE 7 complier generates a warning at the declaration site of a varargs method or constructor with a non-reifiable varargs formal parameter. Java SE 7 introduces the compiler option `-Xlint:varargs` and the annotations `@SafeVarargs` and `@SuppressWarnings({"unchecked", "varargs"})` to supress these warnings.
- The try-with-resources Statement - The `try`-with-resources statement is a `try` statement that declares one or more resources. A *resource* is an object that must be closed after the program is finished with it. The `try`-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements the new `java.lang.AutoCloseable` interface or the `java.io.Closeable` interface can be used as a resource. The classes `java.io.InputStream`, `OutputStream`, `Reader`, `Writer`, `java.sql.Connection`, `Statement`, and `ResultSet` have been retrofitted to implement the `AutoCloseable` interface and can all be used as resources in a `try`-with-resources statement.
- Catching Multiple Exception Types and Rethrowing Exceptions with Improved Type Checking - A single `catch` block can handle more than one type of exception. In addition, the compiler performs more precise analysis of rethrown exceptions than earlier releases of Java SE. This enables you to specify more specific exception types in the `throws` clause of a method declaration.

Enhancements in Java SE 6

No language changes were introduced in Java SE 6.

Enhancements in Java SE 5.0

- **Generics** - This long-awaited enhancement to the type system allows a type or method to operate on objects of various types while providing compile-time type safety. It adds compile-time type safety to the Collections Framework and eliminates the drudgery of casting. See the Generics lesson in the Java Tutorials. (JSR 14)
- **Enhanced `for` Loop** - This new language construct eliminates the drudgery and error-proneness of iterators and index variables when iterating over collections and arrays. (JSR 201)
- **Autoboxing/Unboxing** - This facility eliminates the drudgery of manual conversion between primitive types (such as int) and wrapper types (such as Integer). (JSR 201)

- **Typesafe Enums** - This flexible object-oriented enumerated type facility allows you to create enumerated types with arbitrary methods and fields. It provides all the benefits of the Typesafe Enum pattern (*Effective Java*, Item 21) without the verbosity and the error-proneness. (JSR 201)
- **Varargs** - This facility eliminates the need for manually boxing up argument lists into an array when invoking methods that accept variable-length argument lists.
- **Static Import** - This facility lets you avoid qualifying static members with class names without the shortcomings of the "Constant Interface antipattern." (JSR 201)
- **Annotations** (Metadata) - This language feature lets you avoid writing boilerplate code under many circumstances by enabling tools to generate it from annotations in the source code. This leads to a "declarative" programming style where the programmer says what should be done and tools emit the code to do it. Also it eliminates the need for maintaining "side files" that must be kept up to date with changes in source files. Instead the information can be maintained *in* the source file. (JSR 175)
  NOTE: The `@Deprecated` annotation provides a way to deprecate program elements. See How and When To Deprecate APIs.

Generics papers 泛型白皮书

- JSR14: Adding Generic Types to the Java Programming Language
  - Generics lesson in the Java Tutorials
  - JSR 14: Add Generic Types To The Java Programming Language
- Making the Future Safe for the Past: Adding Genericity to the Java Programming Language (PDF)
  Bracha, Odersky, Stoutamire, and Wadler. OOPSLA 98, Vancouver, October 1998. (other formats)
- GJ: Extending the Java Programming Language with Type Parameters (PDF)
  Bracha, Odersky, Stoutamire, and Wadler. A tutorial on GJ. August 1998. (other formats)
- Adding Generics to the Java Programming Language (PDF)
  Bracha. Slides from JavaOne 2003 presentation.
- Adding Wildcards to the Java Programming Language (PDF)
  Torgersen, Hansen, Ernst, Ahe, Bracha and Gafter. An ACM paper, 2004.

Enhancements in J2SE 1.4

- Assertion Facility - Assertions are boolean expressions that the programmer believes to be true concerning the state of a computer program. For example, after sorting a list, the programmer might assert that the list is in ascending order. Evaluating assertions at runtime to confirm their validity is one of the most powerful tools for improving code quality, as it quickly uncovers the programmer's misconceptions concerning a program's behavior.
