TOUR OF SCALA

INTRODUCTION

Welcome to the tour

This tour contains bite-sized introductions to the most frequently used features of Scala. It is intended for newcomers to the language.

This is just a brief tour, not a full language tutorial. If you want a more detailed guide, consider obtaining a book or consulting other resources.

What is Scala?

Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It seamlessly integrates features of object-oriented and functional languages.

Scala is object-oriented

Scala is a pure object-oriented language in the sense that every value is an object. Types and behaviors of objects are described by classes and traits. Classes can be extended by subclassing, and by using a flexible mixin-based composition mechanism as a clean replacement for multiple inheritance.

Scala is functional

Scala is also a functional language in the sense that every function is a value. Scala provides a lightweight syntax for defining anonymous functions, it supports higher-order functions, it allows functions to be nested, and it supports currying. Scala's case classes and its built-in support for pattern matching provide the functionality of algebraic types, which are used in many functional languages. Singleton objects provide a convenient way to group functions that aren't members of a class.

Furthermore, Scala's notion of pattern matching naturally extends to the processing of XML data with the help of right-ignoring sequence patterns, by way of general extension via extractor objects. In this context, for comprehensions are useful for formulating queries. These features make Scala ideal for developing applications like web services.

Scala is statically typed

Scala's expressive type system enforces, at compile-time, that abstractions are used in a safe and coherent manner. In particular, the type system supports:

- Generic classes
- Variance annotations
- Upper and lower type bounds
- Inner classes and abstract type members as object members
- Compound types
- Explicitly typed self references
- Implicit parameters and conversions
- Polymorphic methods

- Polymorphic methods

Type inference means the user is not required to annotate code with redundant type information. In combination, these features provide a powerful basis for the safe reuse of programming abstractions and for the type-safe extension of software.

Scala is extensible

In practice, the development of domain-specific applications often requires domain-specific language extensions. Scala provides a unique combination of language mechanisms that make it straightforward to add new language constructs in the form of libraries.

In many cases, this can be done without using meta-programming facilities such as macros. For example:

- Implicit classes allow adding extension methods to existing types.
- String interpolation is user-extensible with custom interpolators.

Scala interoperates

Scala is designed to interoperate well with the popular Java Runtime Environment (JRE). In particular, the interaction with the mainstream object-oriented Java programming language is as seamless as possible. Newer Java features like SAMs, lambdas, annotations, and generics have direct analogues in Scala.

Those Scala features without Java analogues, such as default and named parameters, compile as closely to Java as reasonably possible. Scala has the same compilation model (separate compilation, dynamic class loading) as Java and allows access to thousands of existing high-quality libraries.

Enjoy the tour!

Please continue to the next page to read more.

next →

Contributors to this page: