


Username: Pralay Patoria **Book:** Pro .NET Performance. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Reflection

Reflection has a bad reputation for being a performance hog in many complex applications. Some of this reputation is justified: there are extremely expensive operations that you can perform using Reflection, such as invoking a function by its name using `Type.InvokeMember` or creating an object instance using a late-bound parameter list using `Activator.CreateInstance`. The main costs when invoking methods or setting field values through Reflection stem from the work that has to happen in the background—instead of strongly-typed code that can be compiled by the JIT to machine instructions, code that uses Reflection is effectively interpreted at runtime through a series of costly method calls.

For example, invoking a method using `Type.InvokeMember` requires determining which method to invoke using metadata and overload resolution, making sure the specified arguments match the method's parameters, performing type coercions if necessary, verifying any security concerns, and finally executing the method call. Because Reflection is heavily based on object parameters and return values, boxing and unboxing may add an additional extra cost.

 **Note** For more performance tips surrounding .NET Reflection APIs from an internal perspective, consider Joel Pobar's MSDN Magazine article, "Dodge Common Performance Pitfalls to Craft Speedy Applications", available online at <http://msdn.microsoft.com/en-us/magazine/cc163759.aspx>.

Often enough, Reflection can be eliminated from performance-critical scenarios by using some form of *code generation*—instead of reflecting over unknown types and invoking methods/properties dynamically, you can generate code (for each type) that will do so in a strongly-typed fashion.