环境搭建

`App.config`的配置

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Context.Support.ContextHandler, 
Spring.Core"/>
      <section name="objects" 
type="Spring.Context.Support.DefaultSectionHandler, Spring.Core"/>
    </sectionGroup>
  </configSections>
  <spring>
    <!--Spring对象容器的配置-->
    <context>
      <resource uri="~/Conf/Context.xml"/><!--主配置文件路径-->
    </context>
  </spring>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
  <startup>
    <supportedRuntime version="v2.0.50727"/>
  </startup>
</configuration>
```

引用依赖的动态库文件

- 注意版本,不能混着不同版本号引用,否则会报错

`Common.Logging.dll`

`Spring.Core.dll`

`Spring.Aop.dll`

新建接口`ICommand.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SpringDotnetDemo.AOPDemo
{
    public interface ICommand
    {
        object Execute(object context);
    }
}
```

新建实现类`ServiceCommand.cs`2

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SpringDotnetDemo.AOPDemo
{
    class ServiceCommand : ICommand
    {
        public object Execute(object context)
        {
            Console.Out.WriteLine($"Service implementation : [{context}]");
            return null;
        }
    }
}
```

新建环绕通知类

- `ConsoleLoggingAroundAdvice.cs` 实现 `AOP` 联盟的接口

```
using AopAlliance.Intercept;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SpringDotnetDemo.AOPDemo
{
    public class ConsoleLoggingAroundAdvice : IMethodInterceptor
    {
        public object Invoke(IMethodInvocation invocation)
        {
            Console.Out.WriteLine("Advice executing; calling the advised 
method...");
```

```
            object returnValue = invocation.Proceed();
            Console.Out.WriteLine($"Advice executed; advised method returned
{returnValue}");
            return returnValue;
        }
    }
}
```

基于代理的配置

`Conf/Context.xml`文件的配置

```
<?xml version="1.0" encoding="utf-8" ?>
<!--objects:配置容器里对象-->
<objects xmlns="http://www.springframework.net">
  <object type="SpringDotnetDemo.UserInfoDal, SpringDotnetDemo">
  </object>
  <object id="consoleLoggingAroundAdvice"
 type="SpringDotnetDemo.AOPDemo.ConsoleLoggingAroundAdvice,SpringDotnetDemo"/>
</objects>
```

主方法调用示例

```
using Spring.Aop.Framework;
using Spring.Context;
using Spring.Context.Support;
using SpringDotnetDemo.AOPDemo;
using System;

namespace SpringDotnetDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            IApplicationContext ctx = ContextRegistry.GetContext();
            ProxyFactory factory = new ProxyFactory(new ServiceCommand());
            factory.AddAdvice(new ConsoleLoggingAroundAdvice());
            ICommand command = (ICommand)factory.GetProxy();
            command.Execute("This is the argument");
            Console.ReadKey();
        }
    }
}
```

测试结果

```
Advice executing; calling the advised method...
Service implementation : [This is the argument]
Advice executed; advised method returned
```

基于声明式的配置

基于`Spring.Aop.Framework.ProxyFactoryObject`类的代理

`Conf/Context.xml`文件的配置

- 该代理是对目标类的所有方法进行增强

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!--objects:配置容器里对象-->
<objects xmlns="http://www.springframework.net">
  <object type="SpringDotnetDemo.UserInfoDal, SpringDotnetDemo">
  </object>
  <object id="consoleLoggingAroundAdvice"
 type="SpringDotnetDemo.AOPDemo.ConsoleLoggingAroundAdvice,SpringDotnetDemo"/>
  <object id="myServiceObject"
type="Spring.Aop.Framework.ProxyFactoryObject,Spring.Aop">
    <property name="Target"><!--确定目标类-->
      <object id="myServiceObjectTarget"
 type="SpringDotnetDemo.AOPDemo.ServiceCommand,SpringDotnetDemo"/>
    </property>
    <property name="InterceptorNames"><!--确定切面类-->
      <list>
        <value>consoleLoggingAroundAdvice</value><!--可配置多个增强类，但是要注意顺序-
->
      </list>
    </property>
  </object>
</objects>
```

主方法调用示例

```csharp
using Spring.Aop.Framework;
using Spring.Context;
using Spring.Context.Support;
using SpringDotnetDemo.AOPDemo;
using System;

namespace SpringDotnetDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            IApplicationContext ctx = ContextRegistry.GetContext();
            ICommand command = (ICommand)ctx["myServiceObject"];
            command.Execute("This is the argument");
            Console.ReadKey();
        }
    }
}
```

测试结果

```
Advice executing; calling the advised method...
Service implementation : [This is the argument]
Advice executed; advised method returned
```

基于`Spring.Aop.Framework.AutoProxy.ObjectNameAutoProxyCreator`类的代理

新增`HelloWorldSpeaker.cs`文件

```
using AopAlliance.Intercept;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SpringDotnetDemo.AOPDemo
{
    public enum Language
    {
        English = 1,
        Portuguese = 2,
        Italian = 3
    }

    public interface IHelloWorldSpeaker
    {
        void SayHello();
    }

    public class HelloWorldSpeaker : IHelloWorldSpeaker
    {
        private Language language;

        public Language Language
        {
            set { language = value; }
            get { return language; }
        }

        public void SayHello()
        {
            switch (language)
            {
                case Language.English:
                    Console.WriteLine("Hello World!");
                    break;
                case Language.Portuguese:
                    Console.WriteLine("Oi Mundo!");
                    break;
                case Language.Italian:
                    Console.WriteLine("Ciao Mondo!");
                    break;
            }
        }
    }

    public class DebugInterceptor : IMethodInterceptor
    {
        public object Invoke(IMethodInvocation invocation)
        {
            Console.WriteLine("Before: " + invocation.Method.ToString());
            object rval = invocation.Proceed();
            Console.WriteLine("After:  " + invocation.Method.ToString());
            return rval;
        }
    }

}
```

`Conf/Context.xml`文件的配置

以下示例只会对`EnglishSpeakerOne`、`EnglishSpeakerTwo`及`PortugeseSpeaker`对应的示例进行增强，不会对`ItalianSpeakerOne`的示例进行增强

```
<?xml version="1.0" encoding="utf-8" ?>
<!--objects:配置容器里对象-->
<objects xmlns="http://www.springframework.net>
  <object id="ProxyCreator"
type="Spring.Aop.Framework.AutoProxy.ObjectNameAutoProxyCreator, Spring.Aop">
    <property name="ObjectNames">
      <list>
        <value>English*</value><!--object id的引用:代表已English开头的object id-->
        <value>PortugeseSpeaker</value><!--object id的引用:完全匹配名字为
PortugeseSpeaker的object id-->
      </list>
    </property>
    <property name="InterceptorNames">
      <list>
        <value>debugInterceptor</value>
      </list>
    </property>
  </object>

  <object id="debugInterceptor" type="SpringDotnetDemo.AOPDemo.DebugInterceptor,
SpringDotnetDemo"/>

  <object id="EnglishSpeakerOne"
type="SpringDotnetDemo.AOPDemo.HelloWorldSpeaker, SpringDotnetDemo">
    <property name="Language" value="English"/>
  </object>

  <object id="EnglishSpeakerTwo"
type="SpringDotnetDemo.AOPDemo.HelloWorldSpeaker, SpringDotnetDemo">
    <property name="Language" value="English"/>
  </object>

  <object id="PortugeseSpeaker"
type="SpringDotnetDemo.AOPDemo.HelloWorldSpeaker, SpringDotnetDemo">
```

```
    <property name="Language" value="Portuguese"/>
  </object>

  <object id="ItalianSpeakerOne"
type="SpringDotnetDemo.AOPDemo.HelloWorldSpeaker, SpringDotnetDemo">
    <property name="Language" value="Italian"/>
  </object>
</objects>
```

测试示例

```
using Spring.Aop.Framework;
using Spring.Context;
using Spring.Context.Support;
using SpringDotnetDemo.AOPDemo;
using System;
using System.Collections;

namespace SpringDotnetDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            IApplicationContext ctx = ContextRegistry.GetContext();
            IDictionary speakerDictionary =
ctx.GetObjectsOfType(typeof(IHelloWorldSpeaker));
            foreach (DictionaryEntry entry in speakerDictionary)
            {
                string name = (string)entry.Key;
                IHelloWorldSpeaker worldSpeaker =
(IHelloWorldSpeaker)entry.Value;
                Console.Write(name + " says; ");
                worldSpeaker.SayHello();
            }
            Console.ReadKey();
        }
    }
}
```

测试结果（`ItalianSpeakerOne`的方法不增强）

```
ItalianSpeakerOne says; Ciao Mondo!
EnglishSpeakerTwo says; Before: Void SayHello()
Hello World!
After:  Void SayHello()
EnglishSpeakerOne says; Before: Void SayHello()
Hello World!
After:  Void SayHello()
PortugeseSpeaker says; Before: Void SayHello()
Oi Mundo!
After:  Void SayHello()
```

基于`Spring.Aop.Framework.AutoProxy.DefaultAdvisorAutoProxyCreator`

`Conf/Context.xml`文件的配置

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!--objects:配置容器里对象-->
<objects xmlns="http://www.springframework.net">
  <object id="ProxyCreator"
type="Spring.Aop.Framework.AutoProxy.DefaultAdvisorAutoProxyCreator,
Spring.Aop"/>
  <object id="debugInterceptor" type="SpringDotnetDemo.AOPDemo.DebugInterceptor,
SpringDotnetDemo"/>
  <object id="EnglishSpeakerTwo"
type="SpringDotnetDemo.AOPDemo.HelloWorldSpeaker, SpringDotnetDemo">
    <property name="Language" value="English"/>
  </object>
  <object id="SpeachAdvisor"
type="Spring.Aop.Support.RegularExpressionMethodPointcutAdvisor, Spring.Aop">
    <property name="advice" ref="debugInterceptor"/>
    <property name="patterns">
      <list>
        <value>.*Say.*</value>
      </list>
    </property>
  </object>
</objects>
```

pattern的匹配规则

以`SpringDotnetDemo.AOPDemo`命名空间下`HelloWorldSpeaker`类的`SayHello`方法为例

以方法的**完全限定名**根据以下正则表达式判断

```csharp
Match match = Regex.Match("SpringDotnetDemo.AOPDemo.HelloWorldSpeaker.SayHello",
@".*Say.*");
bool success = match.Success;
```

(常用)基于`Spring.Aop.Support.SdkRegularExpressionMethodPointcut`

- `xml`配置增加命名空间`xmlns:aop="http://www.springframework.net/aop"`

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!--objects:配置容器里对象-->
<objects xmlns="http://www.springframework.net"
xmlns:aop="http://www.springframework.net/aop">
  <object id="advisor"
type="Spring.Aop.Support.SdkRegularExpressionMethodPointcut, Spring.Aop">
    <property name="patterns">
```

```
        <list>
          <value>*.Say.*</value>
          <value>.*.Say.*</value>
        </list>
      </property>
    </object>
    <aop:config>
      <aop:advisor pointcut-ref="advisor" advice-ref="debugInterceptor"/>
    </aop:config>

    <object id="debugInterceptor" type="SpringDotnetDemo.AOPDemo.DebugInterceptor,
SpringDotnetDemo"/>

    <object id="EnglishSpeakerTwo"
type="SpringDotnetDemo.AOPDemo.HelloWorldSpeaker, SpringDotnetDemo">
      <property name="Language" value="English"/>
    </object>
</objects>
```

数据库访问

- 引用`Spring.Data.dll`

新增`DbProvider.xml`

```
<?xml version="1.0" encoding="utf-8" ?>
<objects xmlns='http://www.springframework.net'
         xmlns:db="http://www.springframework.net/database">
  <db:provider id="DbProvider" provider="OracleClient-2.0"
connectionString="Data Source=YITIHUATEST;User
Id=hit_app;Password=hit;Pooling=True"/>
</objects>
```

provider常用的数据库驱动：

- `SqlServer-2.0（使用System.Data.SqlClient.dll连接SQL Server）`
- `OracleClient-2.0（使用System.Data.OracleClient.dll连接Oracle）`
- `OracleODP-2.0（使用System.DataAccess.Client.dll连接Oracle）`

`新增IEmployeeInfoDao类接口`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SpringDotnetDemo.Ado.net.Dao
{
    public interface IEmployeeInfoDao
    {
        object ExecuteScalar(string sqlText);
    }
}
```

`新增EmployeeInfoDao类继承Spring.Data.Core.AdoDaoSupport及实现IEmployeeInfoDao接口`

```
using Spring.Data.Core;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SpringDotnetDemo.Ado.net.Dao
{
    public class EmployeeInfoDao : AdoDaoSupport, IEmployeeInfoDao
    {
        public object ExecuteScalar(string sqlText)
        {
            return
this.AdoTemplate.ExecuteScalar(System.Data.CommandType.Text,sqlText);
        }
    }
}
```

新增`Dao.xml`

```
<?xml version="1.0" encoding="utf-8" ?>
<objects xmlns="http://www.springframework.net" default-autowire="byType">
  <object id="EmployeeInfoDao"
type="SpringDotnetDemo.Ado.net.Dao.EmployeeInfoDao,SpringDotnetDemo"></object>
</objects>
```

`objects`节点配置`default-autowire="byType"`时,配置在这个节点的类型从容器中去出来时会自动注入父类的属性，所以从容器中取出`EmployeeInfoDao`时容器会帮我们注入`Spring.Data.Core.AdoDaoSupport`类中的`DbProvider`属性,而`DbProvider`的set方法会帮我们创建`AdoTemplate`的实例,所以在`EmployeeInfoDao`中就可以直接使用`AdoTemplate`

`App.config`的配置

- 增加引用配置文件的路径

  ```
  <resource uri="~/Conf/DbProvider.xml"/>
  <resource uri="~/Conf/Dao.xml"/>
  ```

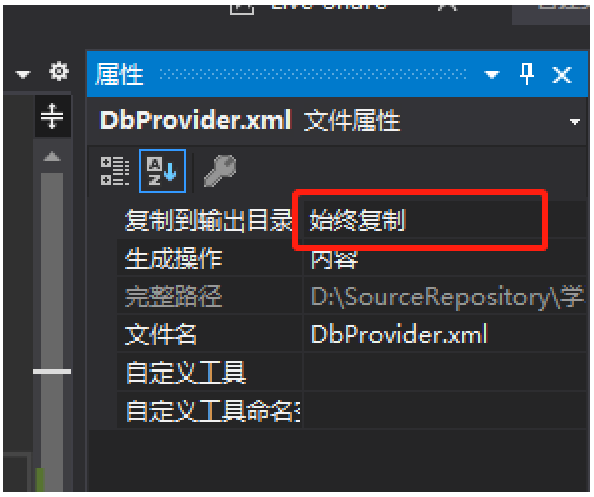
完整配置

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Context.Support.ContextHandler,
Spring.Core"/>
      <section name="objects"
type="Spring.Context.Support.DefaultSectionHandler, Spring.Core"/>
    </sectionGroup>
  </configSections>
  <spring>
```

```
    <!--Spring对象容器的配置-->
    <context>
      <resource uri="~/Conf/Context.xml"/><!--主配置文件路径-->
      <resource uri="~/Conf/DbProvider.xml"/>
      <resource uri="~/Conf/Dao.xml"/>
    </context>
  </spring>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
  <startup>
    <supportedRuntime version="v2.0.50727"/>
  </startup>
</configuration>
```

注:

新增的xml要设置成始终复制否则调试会报错


属性

DbProvider.xml 文件属性

复制到输出目录	始终复制
生成操作	内容
完整路径	D:\SourceRepository\学
文件名	DbProvider.xml
自定义工具	
自定义工具命名空	

测试示例

```
using Spring.Aop.Framework;
using Spring.Context;
using Spring.Context.Support;
using SpringDotnetDemo.Ado.net.Dao;
using SpringDotnetDemo.AOPDemo;
using System;
using System.Collections;

namespace SpringDotnetDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            IApplicationContext ctx = ContextRegistry.GetContext();
            IEmployeeInfoDao employeeInfoDao =
(IEmployeeInfoDao)ctx.GetObject("EmployeeInfoDao");
            object v = employeeInfoDao.ExecuteScalar(@"select *from
fin_opr_register t where t.clinic_code='523318'");
            Console.WriteLine(v);
            Console.ReadKey();
        }
    }
}
```