链滴

你用过依赖注入框架 Google Guice 吗?

作者：Cosolar
原文链接：https://ld246.com/article/1639316349334
来源网站：链滴

Spring框架的依赖注入是家喻户晓的，但是在实际的开发中我们想使用便捷的依赖注入功能，但是又想引入Spring框架的复杂性，该怎么办呢？

有了Google Guice，这个问题便简单了，首先在你的maven项目里引入

```
<dependency>
    <groupId>com.google.inject</groupId>
    <artifactId>guice</artifactId>
    <version>4.0</version>
</dependency>
```

官方文档里给出的例子又臭又长，我不使用官方的例子，下面我们来写个最简单的HelloWorld

```
import javax.inject.Singleton;

import com.google.inject.Guice;
import com.google.inject.Inject;
import com.google.inject.Injector;

@Singleton
class HelloPrinter {

	public void print() {
		System.out.println("Hello, World");
	}

}

@Singleton
public class Sample {

	@Inject
	private HelloPrinter printer;

	public void hello() {
		printer.print();
	}

	public static void main(String[] args) {
		Injector injector = Guice.createInjector();
		Sample sample = injector.getInstance(Sample.class);
		sample.hello();
	}

}
```

原文链接：你用过依赖注入框架 Google Guice 吗?

我们使用Guice创建了一个注射器Injector，然后从Injector拿到你想要的对象就可以了，Guice会自装配依赖树。Guice的启动速度是很快的，在一个大型应用中，Guice装配所有的模块决不会超过1s。uice是一个非常干净的依赖注入框架，框架除了依赖注入功能之外，没有任何其它非相关模块功能。

Guice里最常用的两个注解就是@Singleton和@Inject，Singleton表示构建的对象是单例的，Inject示被标注的字段将使用Guice自动注入。在一般的项目中这两个注解一般可以完成90%以上的装配工。

Guice需要实例化对象，请确保相应被实例化的对象有默认构造器。

当某个接口有多个实现时，我们使用@ImplementedBy注解在接口定义上，指定接口的具体实现类

```
@ImplementedBy(SimpleHelloPrinter.class)
interface IHelloPrinter {
        void print();
}

@Singleton
class SimpleHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Simple World");
        }

}

@Singleton
class ComplexHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Complex World");
        }

}

@Singleton
public class Sample {

        @Inject
        private IHelloPrinter printer;

        public void hello() {
                printer.print();
        }

        public static void main(String[] args) {
                Injector injector = Guice.createInjector();
                Sample sample = injector.getInstance(Sample.class);
                sample.hello();
        }

}
```

如果我们不用Singleton标注，每次获取实例时，Guice会重新构造一个，这个会有反射构造器的性能耗，在高性能场景下，请谨慎。

原文链接：你用过依赖注入框架 Google Guice 吗?

```
class HelloPrinter {

        private static int counter;

        private int myCounter;

        public HelloPrinter() {
                myCounter = counter++;
        }

        public void print() {
                System.out.printf("Hello, World %d\n", myCounter);
        }

}

public class Sample {

        @Inject
        private HelloPrinter printer;

        public void hello() {
                printer.print();
        }

        public static void main(String[] args) {
                Injector injector = Guice.createInjector();
                Sample sample = injector.getInstance(Sample.class);
                sample.hello();
                sample = injector.getInstance(Sample.class);
                sample.hello();
                sample = injector.getInstance(Sample.class);
                sample.hello();
                sample = injector.getInstance(Sample.class);
                sample.hello();
        }

}
```

我们可以不使用@ImplementedBy注解，因为这样不优雅，谁会在定义接口的时候就能预知实现类名称呢。我们可以使用Guice Module定义装配规则，它相当于Spring的XML文件，只不过它的装配则都是使用代码定义的。你可能会辩解说代码定义怎么能比得上XML定义呢，其实Guice Module在个大型项目中也是非常的简洁，一般只会占用几十行代码，Module里面配置的仅仅是特殊的专配规。能规则的可读性而言，代码要比XML舒服的多。

```
interface IHelloPrinter {
        void print();
}

@Singleton
class SimpleHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Simple World");
        }

}

@Singleton
class ComplexHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Complex World");
        }

}

class SampleModule extends AbstractModule {

        @Override
        protected void configure() {
                bind(IHelloPrinter.class).to(SimpleHelloPrinter.class);
        }

}

@Singleton
public class Sample {

        @Inject
        private IHelloPrinter printer;

        public void hello() {
                printer.print();
        }

        public static void main(String[] args) {
                Injector injector = Guice.createInjector(new SampleModule());
                Sample sample = injector.getInstance(Sample.class);
                sample.hello();
        }

}
```

我们还可以使用@Named名称指令来指定依赖注入实现

```
@Singleton
class SimpleHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Simple World");
        }

}

@Singleton
class ComplexHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Complex World");
        }

}

class SampleModule extends AbstractModule {

        @Override
        protected void configure() {
                bind(IHelloPrinter.class).annotatedWith(Names.named("simple")).to(SimpleHelloPrinter.clas
                bind(IHelloPrinter.class).annotatedWith(Names.named("complex")).to(ComplexHelloPrinter.cl
        }

}

@Singleton
public class Sample {

        @Inject
        @Named("simple")
        private IHelloPrinter simplePrinter;

        @Inject
        @Named("complex")
        private IHelloPrinter complexPrinter;

        public void hello() {
                simplePrinter.print();
                complexPrinter.print();
        }

        public static void main(String[] args) {
                Injector injector = Guice.createInjector(new SampleModule());
                Sample sample = injector.getInstance(Sample.class);
                sample.hello();
        }

}
```

我们不使用字段注入，改用构造器注入，如果我们需要在构造器里做一些特别的初始化工作

```
@Singleton
public class Sample {

        @Named("simple")
        private IHelloPrinter simplePrinter;

        @Named("complex")
        private IHelloPrinter complexPrinter;

        @Inject
        public Sample(@Named("simple") IHelloPrinter simplePrinter, @Named("complex") IHelloPrinter complexPrinter) {
                this.simplePrinter = simplePrinter;
                this.complexPrinter = complexPrinter;
        }

        public void hello() {
                simplePrinter.print();
                complexPrinter.print();
        }

        public static void main(String[] args) {
                Injector injector = Guice.createInjector(new SampleModule());
                Sample sample = injector.getInstance(Sample.class);
                sample.hello();
        }

}
```

原文链接：你用过依赖注入框架 Google Guice 吗?

还可以自动注入Set，Map容器，但是得首先加上扩展库

```
<dependency>
    <groupId>com.google.inject.extensions</groupId>
    <artifactId>guice-multibindings</artifactId>
    <version>4.0</version>
</dependency>
```

注入Set

```
@Singleton
class SimpleHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Simple World");
        }

}

@Singleton
class ComplexHelloPrinter implements IHelloPrinter {

        public void print() {
                System.out.println("Hello, Complex World");
        }

}

class SampleModule extends AbstractModule {

        @Override
        protected void configure() {
                Multibinder<IHelloPrinter> printers = Multibinder.newSetBinder(binder(), IHelloPrinter.class);
                printers.addBinding().to(SimpleHelloPrinter.class);
                printers.addBinding().to(ComplexHelloPrinter.class);
        }

}

@Singleton
public class Sample {

        @Inject
        private Set<IHelloPrinter> printers;

        public void hello() {
                for (IHelloPrinter printer : printers) {
                        printer.print();
                }
        }

        public static void main(String[] args) {
                Injector injector = Guice.createInjector(new SampleModule());
                Sample sample = injector.getInstance(Sample.class);
                sample.hello();
        }

}
```

注入Map

```java
@Singleton
class SimpleHelloPrinter implements IHelloPrinter {

	public void print() {
		System.out.println("Hello, Simple World");
	}

}

@Singleton
class ComplexHelloPrinter implements IHelloPrinter {

	public void print() {
		System.out.println("Hello, Complex World");
	}

}

class SampleModule extends AbstractModule {

	@Override
	protected void configure() {
		MapBinder<String, IHelloPrinter> printers = MapBinder.newMapBinder(binder(), String.class, IHelloPrinte
		printers.addBinding("simple").to(SimpleHelloPrinter.class);
		printers.addBinding("complex").to(ComplexHelloPrinter.class);
	}

}

@Singleton
public class Sample {

	@Inject
	private Map<String, IHelloPrinter> printers;

	public void hello() {
		for (String name : printers.keySet()) {
			printers.get(name).print();
		}
	}

	public static void main(String[] args) {
		Injector injector = Guice.createInjector(new SampleModule());
		Sample sample = injector.getInstance(Sample.class);
		sample.hello();
	}

}
```

在全世界都沉迷于复杂的Spring框架时，Guice无疑是一股清流，在炎热的夏天，它就像一杯冰爽的汁，让人畅快不已。

本文转自 https://www.jianshu.com/p/9ac108d14608，如有侵权，请联系删除。

原文链接：你用过依赖注入框架 Google Guice 吗?