# Guice

guice是一个轻量级的IOC框架

<https://github.com/google/guice>

有以下几种binding：

linked bindings

直接绑定比较简单如下所示

public class BillingModule extends AbstractModule {

@Override

protected void configure() {

bind(TransactionLog.class).to(DatabaseTransactionLog.class);

}

}

Binding Annotations

注解绑定通常可以用在同一个接口有不同的子类，通过注解标识注入哪一个具体的子类

创建一个注解

@BindingAnnotation @Target({ FIELD, PARAMETER, METHOD }) @Retention(RUNTIME)

public @interface PayPal {}

然后

bind(CreditCardProcessor.class)

.annotatedWith(PayPal.class)

.to(PayPalCreditCardProcessor.class);

具体注入的时候加上绑定的注解

public class RealBillingService implements BillingService {

@Inject

public RealBillingService(@PayPal CreditCardProcessor processor,

TransactionLog transactionLog) {

...

}

guice有自带的注解可以实现如上功能，避免创建自定义注解

bind(CreditCardProcessor.class)

.annotatedWith(Names.named("Checkout"))

.to(CheckoutCreditCardProcessor.class);

//

public class RealBillingService implements BillingService {

@Inject

public RealBillingService(@Named("Checkout") CreditCardProcessor processor,

TransactionLog transactionLog) {

...

}

instance bindings

把一个类具体绑定到一个具体的实例上

bind(String.class)

.annotatedWith(Names.named("JDBC URL"))

.toInstance("jdbc:mysql://localhost/pizza");

bind(Integer.class)

.annotatedWith(Names.named("login timeout seconds"))

.toInstance(10);

@Provides methods

该注解可以创建一个实例类似spring的@bean注解

public class BillingModule extends AbstractModule {

@Override

protected void configure() {

...

}

@Provides

TransactionLog provideTransactionLog() {

DatabaseTransactionLog transactionLog = new DatabaseTransactionLog();

transactionLog.setJdbcUrl("jdbc:mysql://localhost/pizza");

transactionLog.setThreadPoolSize(30);

return transactionLog;

}

}

如果在创建实例的时候遇到@Named或自定义guice注解，会注入对于的实例作为参数传入

@Provides @PayPal

CreditCardProcessor providePayPalCreditCardProcessor(

@Named("PayPal API key") String apiKey) {

PayPalCreditCardProcessor processor = new PayPalCreditCardProcessor();

processor.setApiKey(apiKey);

return processor;

}

provider bindings

当@Provides方法开始变得复杂时，您可以考虑将其移至它们自己的类中。 provider类实现Guice的Provider接口，这是一个用于提供值的简单通用接口，类似spring中的@Component，@Service等。

public class DatabaseTransactionLogProvider implements Provider<TransactionLog> {

private final Connection connection;

@Inject

public DatabaseTransactionLogProvider(Connection connection) {

this.connection = connection;

}

public TransactionLog get() {

DatabaseTransactionLog transactionLog = new DatabaseTransactionLog();

transactionLog.setConnection(connection);

return transactionLog;

}

}

public class BillingModule extends AbstractModule {

@Override

protected void configure() {

bind(TransactionLog.class)

.toProvider(DatabaseTransactionLogProvider.class);

}

<https://github.com/google/guice/wiki/InjectingProviders>

constructor bindings

构造器绑定，guice通过反色获取一个class的构造器注入一个新的对象

public class BillingModule extends AbstractModule {

@Override

protected void configure() {

try {

bind(TransactionLog.class).toConstructor(

DatabaseTransactionLog.class.getConstructor(DatabaseConnection.class));

} catch (NoSuchMethodException e) {

addError(e);

}

}

}

上面的例子中DatabaseTransactionLog必须有只有一个DatabaseConnection入参的构造器，否则会失败。每个toConstructor（）绑定的作用域都是独立的，创建的实例是多例的。

untargetted bindings.

无目标绑定，仅仅注入一个对象，通常没有to，如

bind(MyConcreteClass.class);

bind(AnotherConcreteClass.class).in(Singleton.class);

如果需要用到@Named注解的话，to仍然不能丢弃

bind(MyConcreteClass.class)

.annotatedWith(Names.named("foo"))

.to(MyConcreteClass.class);

bind(AnotherConcreteClass.class)

.annotatedWith(Names.named("foo"))

.to(AnotherConcreteClass.class)

.in(Singleton.class);

Just-in-time Bindings

在modules显示绑定的类型，injector随时随地可以使用。如果一个类型没有显示绑定的话，injector就会创建一个jit绑定。有以下几种方式：

Eligible Constructors

该类型的类的构造器必须是public的，或者无参数的，或者是带有@Named注解的。

public class PayPalCreditCardProcessor implements CreditCardProcessor {

private final String apiKey;

@Inject

public PayPalCreditCardProcessor(@Named("PayPal API key") String apiKey) {

this.apiKey = apiKey;

}

@ImplementedBy

@ImplementedBy(PayPalCreditCardProcessor.class)

public interface CreditCardProcessor {

ChargeResult charge(String amount, CreditCard creditCard)

throws UnreachableException;

}

等效bind(CreditCardProcessor.class).to(PayPalCreditCardProcessor.class);

@ProvidedBy(DatabaseTransactionLogProvider.class)

public interface TransactionLog {

void logConnectException(UnreachableException e);

void logChargeResult(ChargeResult result);

}

等效bind(TransactionLog.class)

.toProvider(DatabaseTransactionLogProvider.class);