Mybatis高级

T A H N K Y O U F O R W A T C H I N G

主讲老师Lison：525765982

课程咨询依娜老师：2470523467

目 录

CONTENTS

源码分析概述

mybatis架构分析

包分析

设计模式的原则

日志模块分析

适配器模式

代理模式

日志模块分析

数据源模块分析

工厂模式

数据源模块分析

数据库连接池源码分析

缓存模块分析

装饰器模式

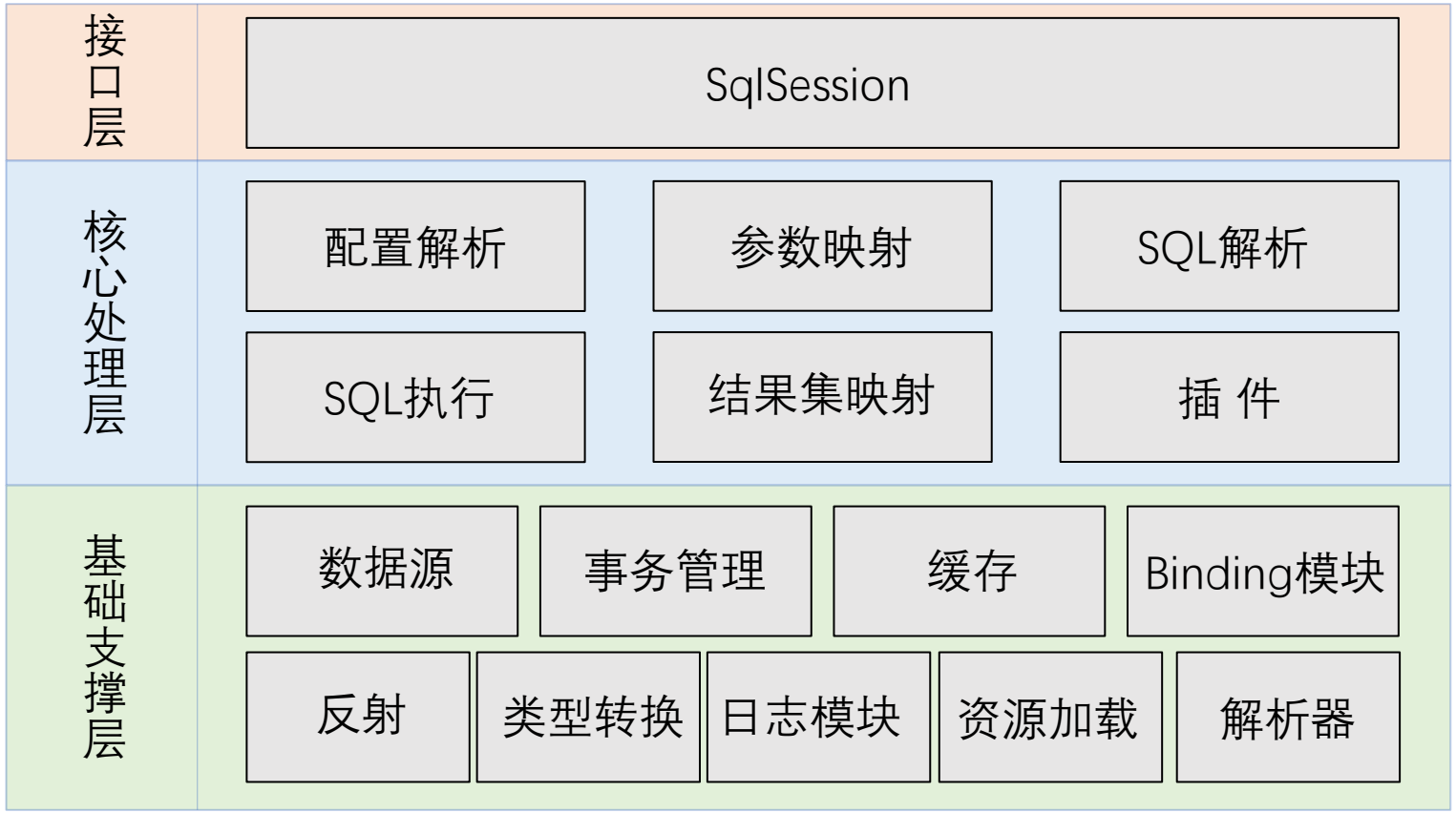
缓存模块分析

源码包分析

MyBatis 源码下载地址：

MyBatis源码结构.x

mybatis整体架构



基础支撑层源码分析


接口层
SqlSession
核心处理层
配置解析
参数映射
SQL解析
SQL执行
结果集映射
插 件
基础支撑层
数据源
事务管理
缓存
Binding模块
反射
类型转换
日志模块
资源加载
解析器

谈谈设计模式的几个原则

- 单一职责原则：一个类或者一个接口只负责唯一项职责，尽量设计出功能单一的接口；
- 高层模块不应该依赖低层模块具体实现，解耦高层与低层。既面向接口编程，当实现发生变化时，只需提供新的实现类，不需要修改高层模块代码;
- 开放-封闭原则：程序对外扩展开放，对修改关闭；换句话说，当需求发生变化时，我们可以通过添加新模块来满足新需求，而不是通过修改原来的实现代码来满足新需求；

目 录

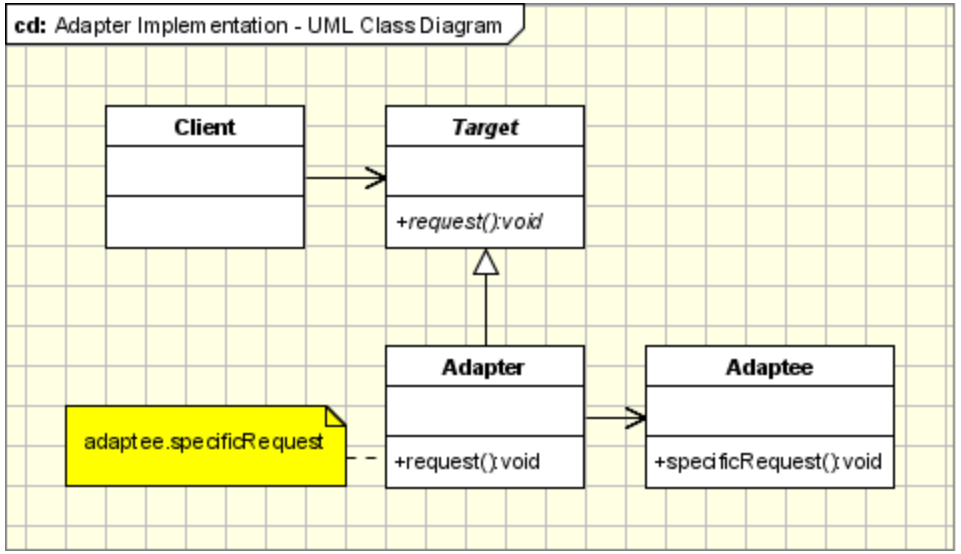
CONTENTS

基础支撑层源码分析 日志模块需求

- MyBatis没有提供日志的实现类，需要接入第三方的日志组件，但第三方日志组件都有各自的Log级别，且各不相同，二MyBatis统一提供了trace、debug、warn、error四个级别；
- 自动扫描日志实现，并且第三方日志插件加载优先级如下：slf4J → commonsLoging → Log4J2 → Log4J → JdkLog;
- 日志的使用要优雅的嵌入到主体功能中；

适配器模式

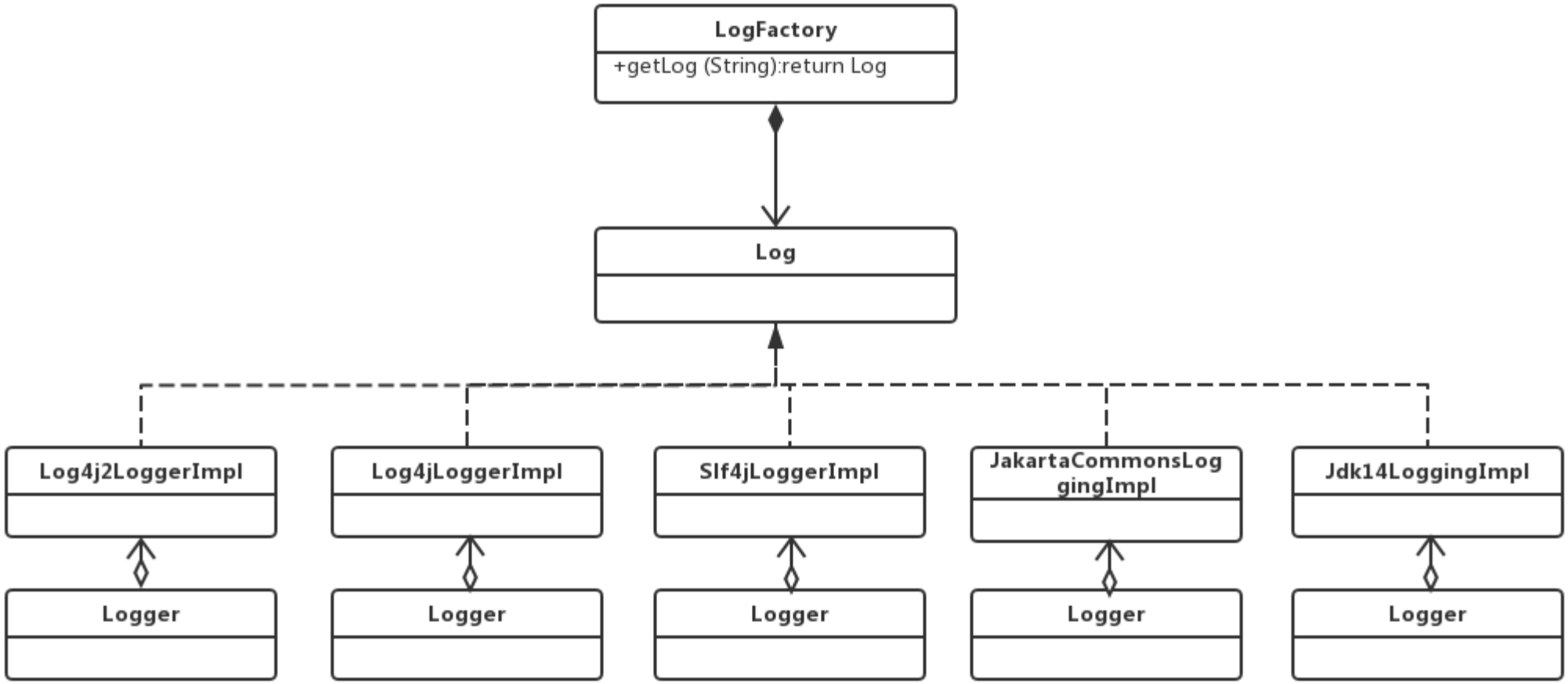
- 适配器模式（Adapter Pattern）是作为两个不兼容的接口之间的桥梁，将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作；



- ✓ Target：目标角色,期待得到的接口.
- ✓ Adaptee：适配者角色,被适配的接口.
- ✓ Adapter：适配器角色,将源接口转换成目标接口.

- 适用场景：当调用双方都不太容易修改的时候，为了复用现有组件可以使用适配器模式；在系统中接入第三方组件的时候经常被使用到；
- 注意：如果系统中存在过多的适配器，会增加系统的复杂性，设计人员应考虑对系统进行重构；

日志模块类图


LogFactory
+getLog (String):return Log
Log
Log4j2LoggerImpl
Log4jLoggerImpl
Slf4jLoggerImpl
JakartaCommonsLoggingImpl
Jdk14LoggingImpl
Logger
Logger
Logger
Logger
Logger

代理模式那些事

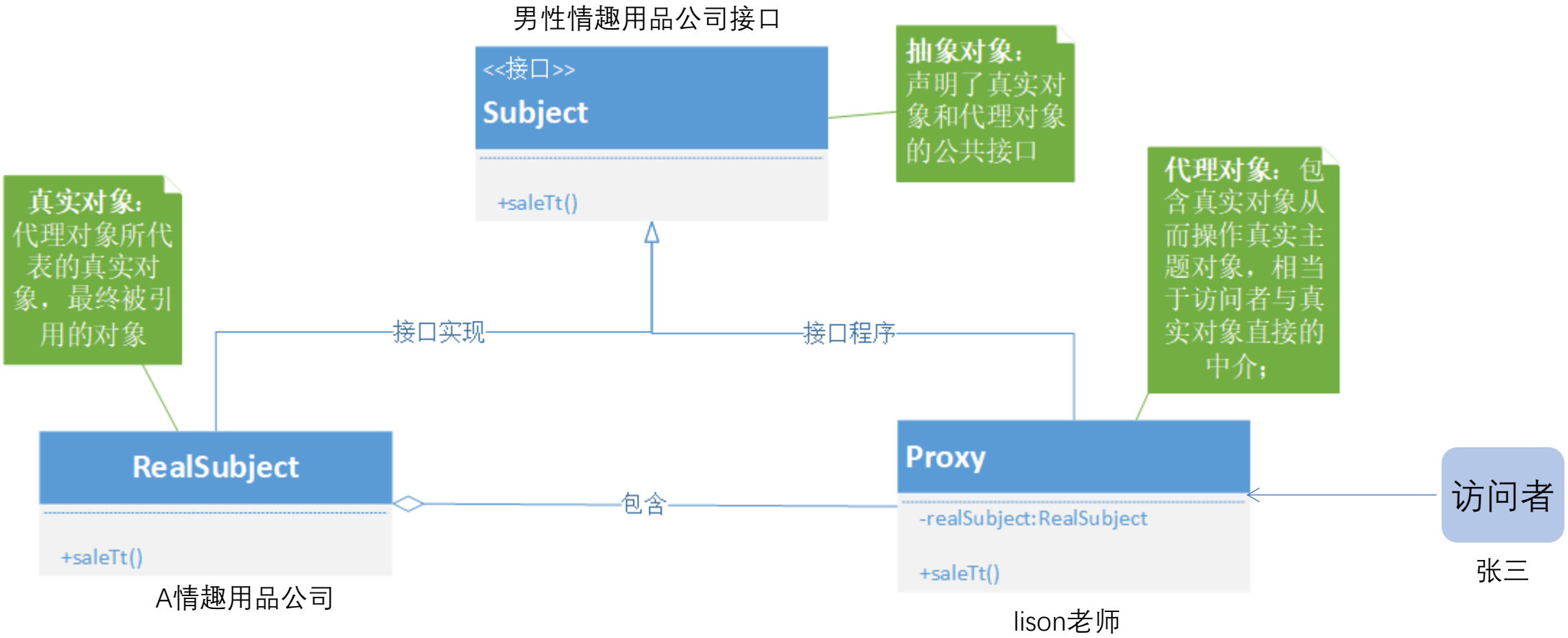
- 定义：给目标对象提供一个代理对象，并由代理对象控制对目标对象的引用；
- 目的：（1）通过引入代理对象的方式来间接访问目标对象，防止直接访问目标对象给系统带来的不必要复杂性；（2）通过代理对象对原有的业务增强；

张三

lison代购

A情趣用品公司

代理模式类图


男性情趣用品公司接口
<<接口>>
Subject
+saleTt()
抽象对象：
声明了真实对象和代理对象的公共接口
真实对象：
代理对象所代表的真实对象，最终被引用的对象
代理对象：包含真实对象从而操作真实主题对象，相当于访问者与真实对象直接的中介；
接口实现
接口程序
RealSubject
+saleTt()
包含
Proxy
-realSubject:RealSubject
+saleTt()
访问者
A情趣用品公司
lison老师
张三

why proxy?

- **代理模式给我们带来的便利：**

✓ 作为中介解耦客户端和真实对象，保护真实对象安全；（房屋中介）

✓ 防止直接访问目标对象给系统带来的不必要复杂性；（海外代购,SSH）

✓ 对业务进行增强，增强点多样化如：前入、后入、异常；（AOP）

动态代理那些事

张三

张三老婆

.....

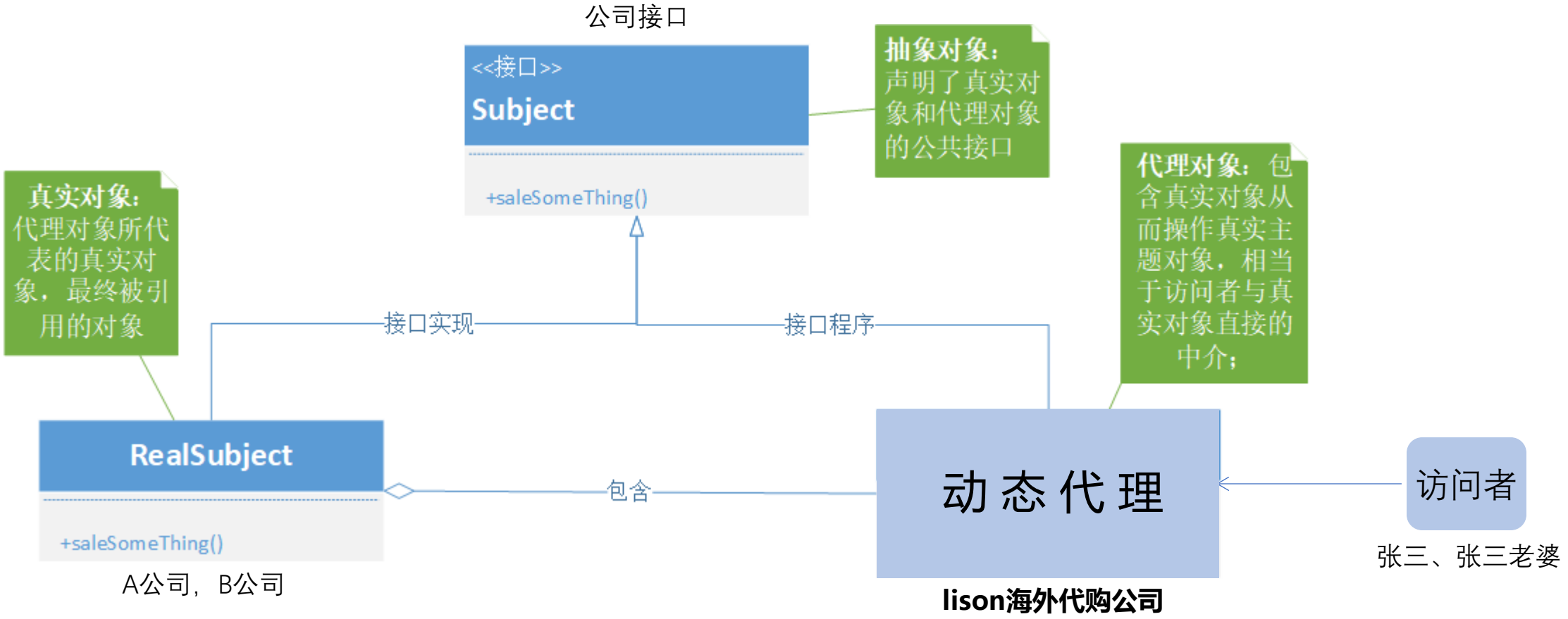
lison海外代购公司

A情趣用品公司

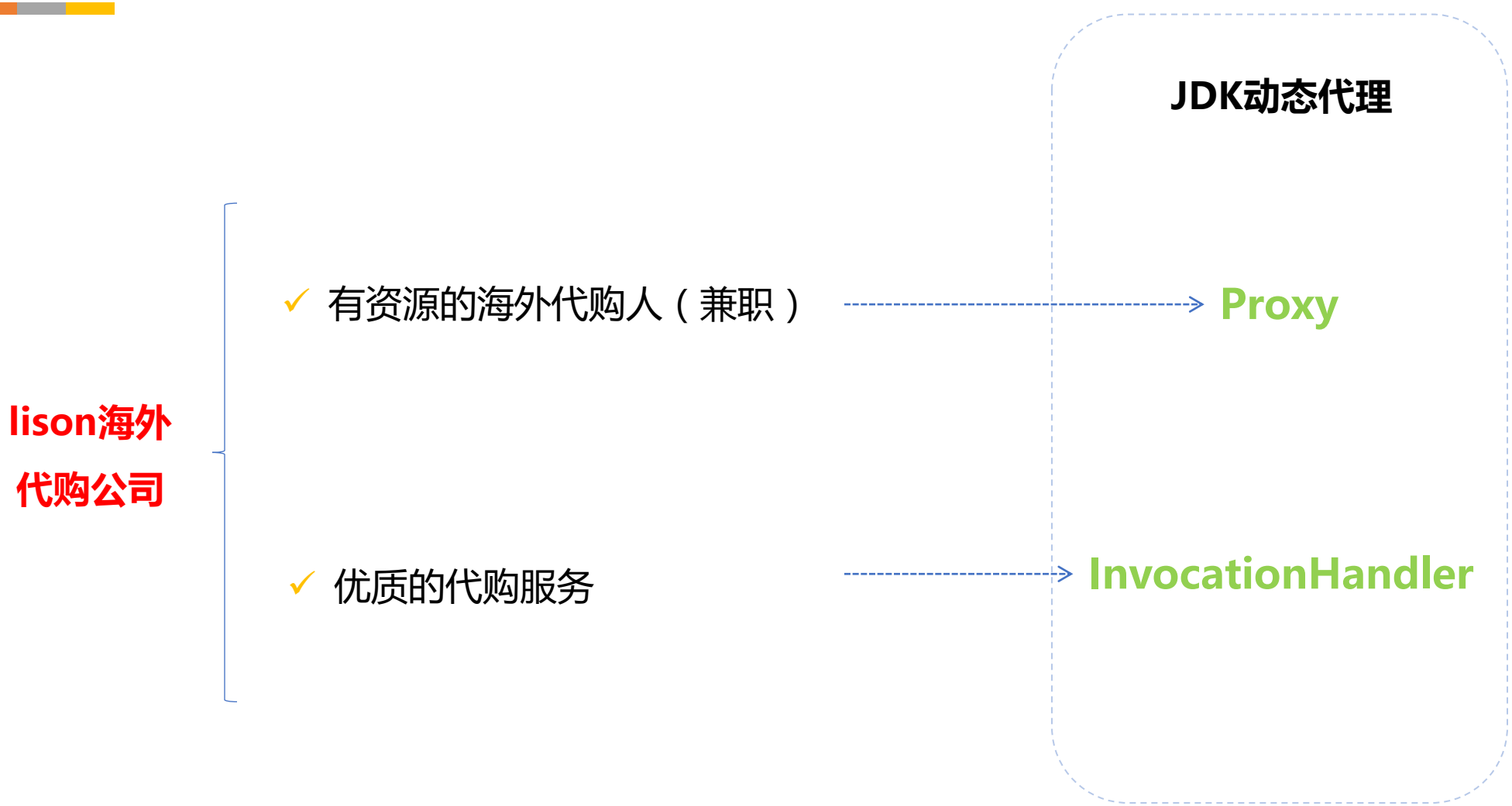
B情趣用品公司

.....

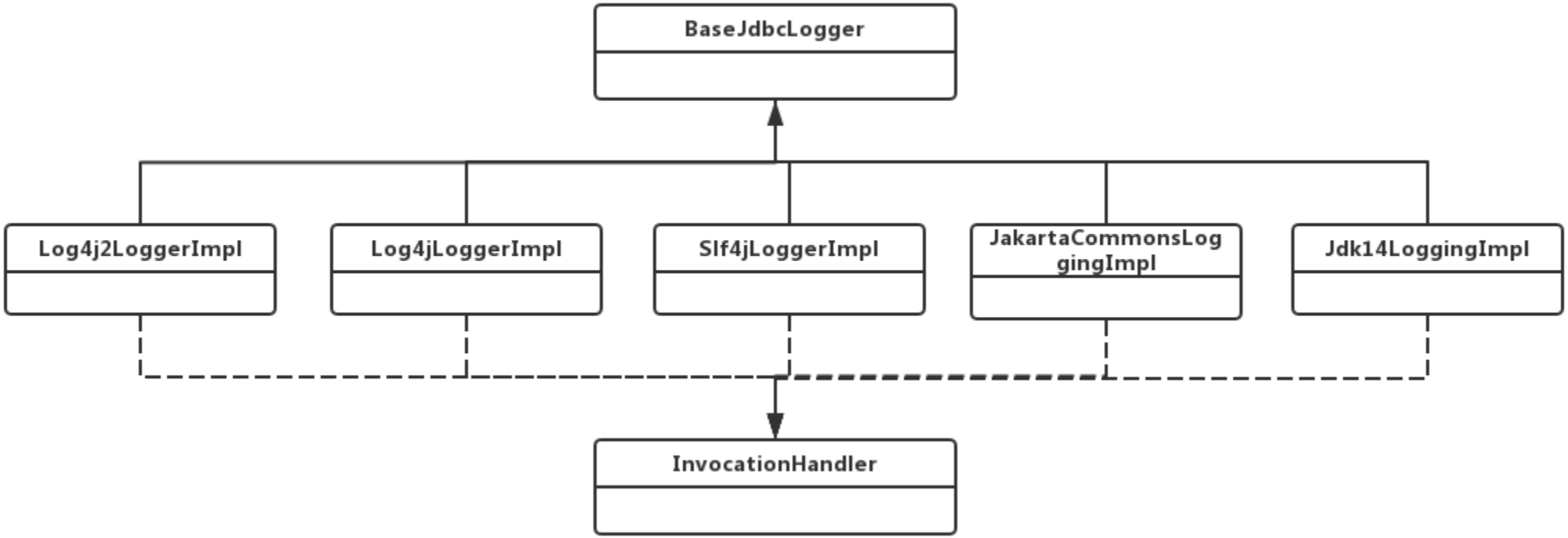
代理模式类图


公司接口
<<接口>>
Subject
+saleSomeThing()
抽象对象：
声明了真实对象和代理对象的公共接口
真实对象：
代理对象所代表的真实对象，最终被引用的对象
代理对象：包含真实对象从而操作真实主题对象，相当于访问者与真实对象直接的中介；
接口实现
接口程序
RealSubject
+saleSomeThing()
包含
动 态 代 理
访问者
A公司，B公司
张三、张三老婆
lison海外代购公司

解密lison海外代购公司



日志模块JDBC包类图



目 录

CONTENTS

源码分析概述

mybatis架构分析

包分析

设计模式的原则

日志模块分析

适配器模式

代理模式

日志模块分析

数据源模块分析

工厂模式

数据源模块分析

数据库连接池源码分析

缓存模块分析

装饰器模式

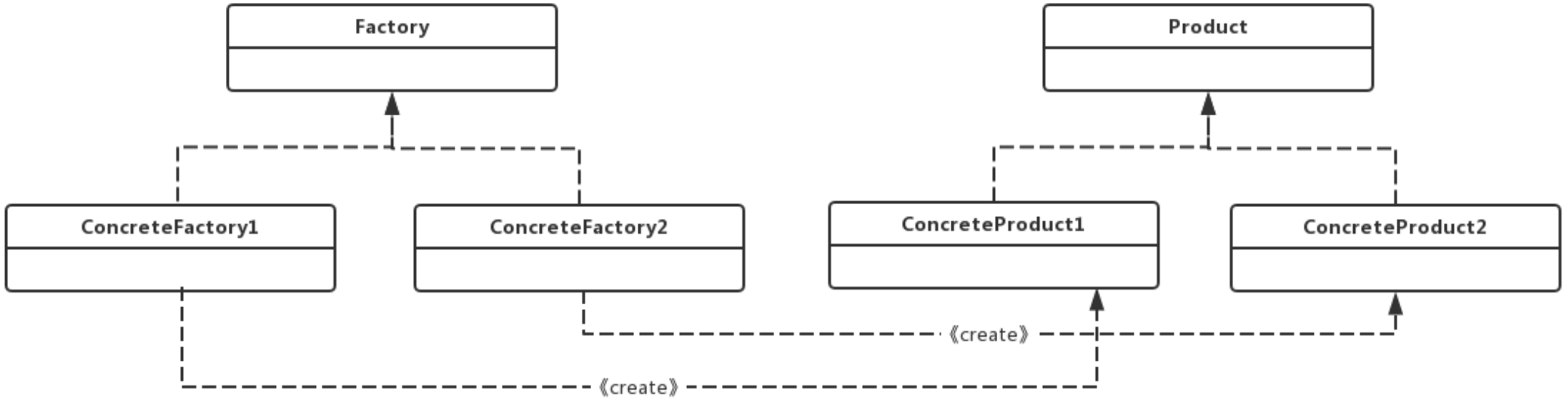
缓存模块分析

基础支撑层源码分析 数据源模块需求

- 常见的数据源组件都实现了javax.sql.DataSource接口；
- MyBatis不但要能集成第三方的数据源组件，自身也提供了数据源的实现；
- 一般情况下，数据源的初始化过程参数较多，比较复杂；

工厂模式uml类图

- 工厂模式（Factory Pattern）属于创建型模式，它提供了一种创建对象的最佳方式。定义一个创建对象的接口，让其子类自己决定实例化哪一个工厂类，工厂模式使其创建过程延迟到子类进行



- ✓ **工厂接口（Factory）**：工厂接口是工厂方法模式的核心接口，调用者会直接和工厂接口交互用于获取具体的产品实现类；
- ✓ **具体工厂类（ConcreteFactory）**:是工厂接口的实现类，用于实例化产品对象，不同的具体工厂类会根据需求实例化不同的产品实现类；
- ✓ **产品接口（Product）：**产品接口用于定义产品类的功能，具体工厂类产生的所有产品都必须实现这个接口。调用者与产品接口直接交互，这是调用者最关心的接口；
- ✓ **具体产品类（ConcreteProduct）：**实现产品接口的实现类，具体产品类中定义了具体的业务逻辑；

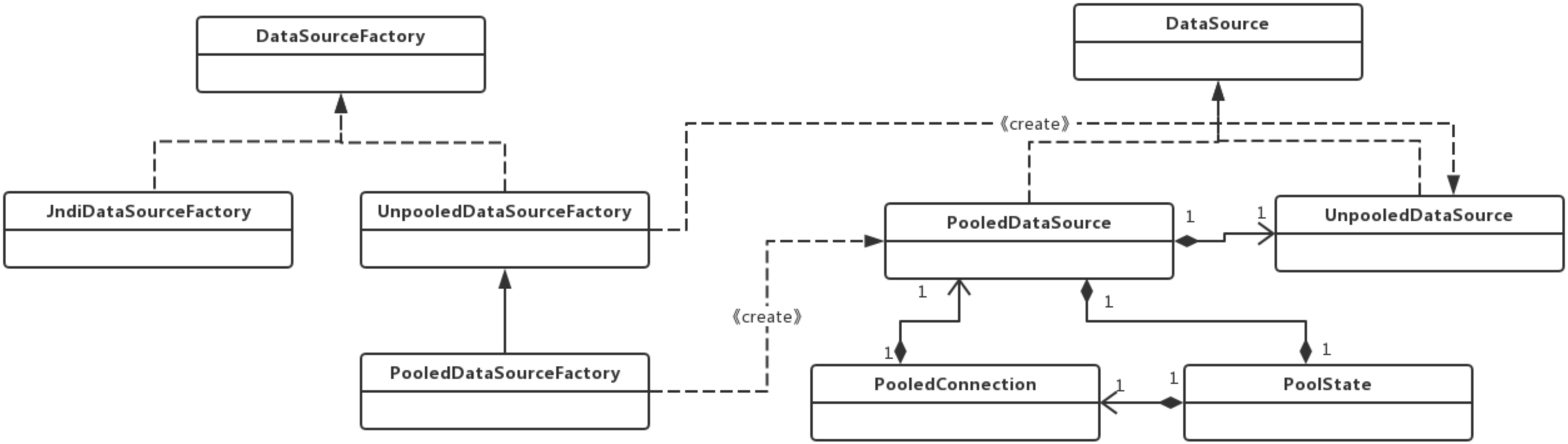
为什么要使用工厂模式？

- 创建对象的方式：
  - ✓ 使用new关键字直接创建对象；
  - ✓ 通过反射机制创建对象；
  - ✓ 通过工厂类创建对象；

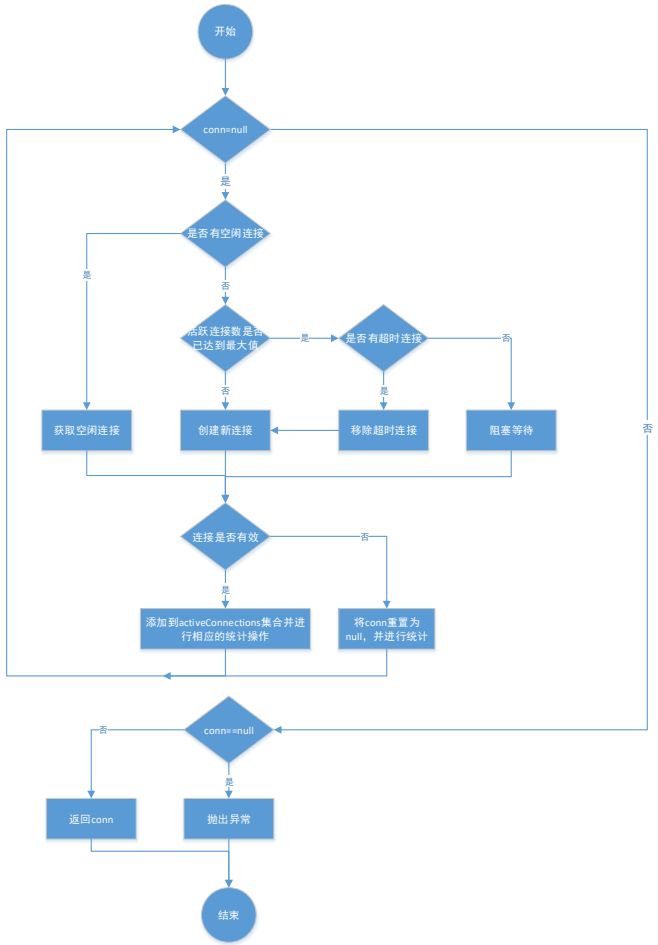
缺点：
- ✓ 对象创建和对象使用使用的职责耦合在一起，违反单一原则；
- ✓ 当业务扩展时，必须修改代业务代码，违反了开闭原则；

优点：
- ✓ 把对象的创建和使用的过程分开，对象创建和对象使用使用的职责解耦；
- ✓ 如果创建对象的过程很复杂，创建过程统一到工厂里管理，既减少了重复代码，也方便以后对创建过程的修改维护；
- ✓ 当业务扩展时，只需要增加工厂子类，符合开闭原则；

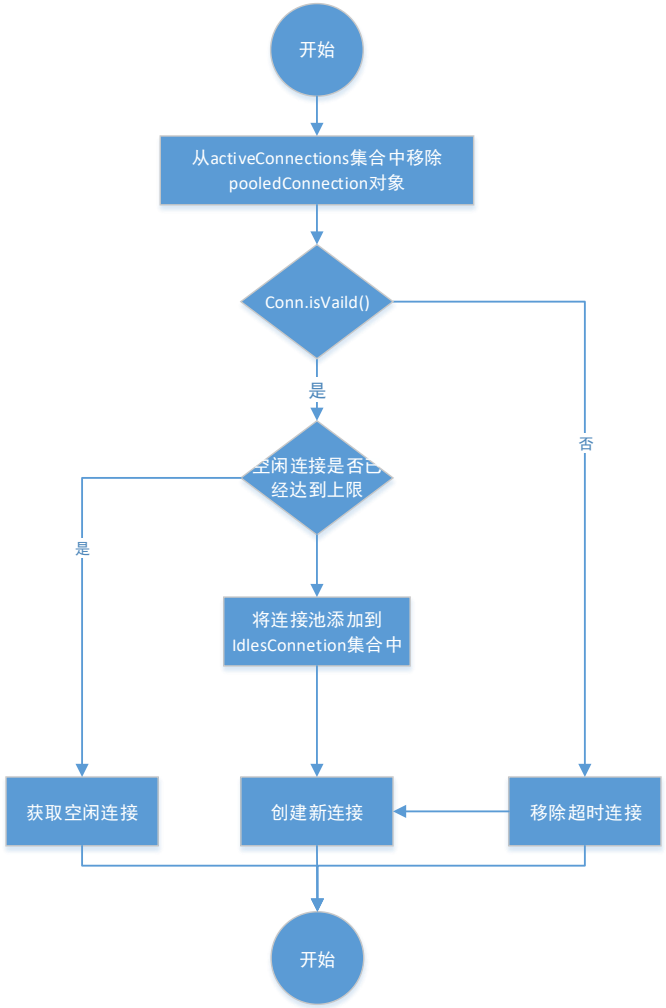
数据源模块类图


DataSourceFactory
DataSource
《create》
JndiDataSourceFactory
UnpooledDataSourceFactory
PooledDataSource
1
1
UnpooledDataSource
1
1
《create》
1
1
PooledDataSourceFactory
1
1
PooledConnection
1
1
PoolState

PooledDataSource 获取和归还连接过程



getConnection()



pushConnection()

目 录

CONTENTS

源码分析概述

mybatis架构分析

包分析

设计模式的原则

日志模块分析

适配器模式

代理模式

日志模块分析

数据源模块分析

工厂模式

数据源模块分析

数据库连接池源码分析

缓存模块分析

装饰器模式

缓存模块分析

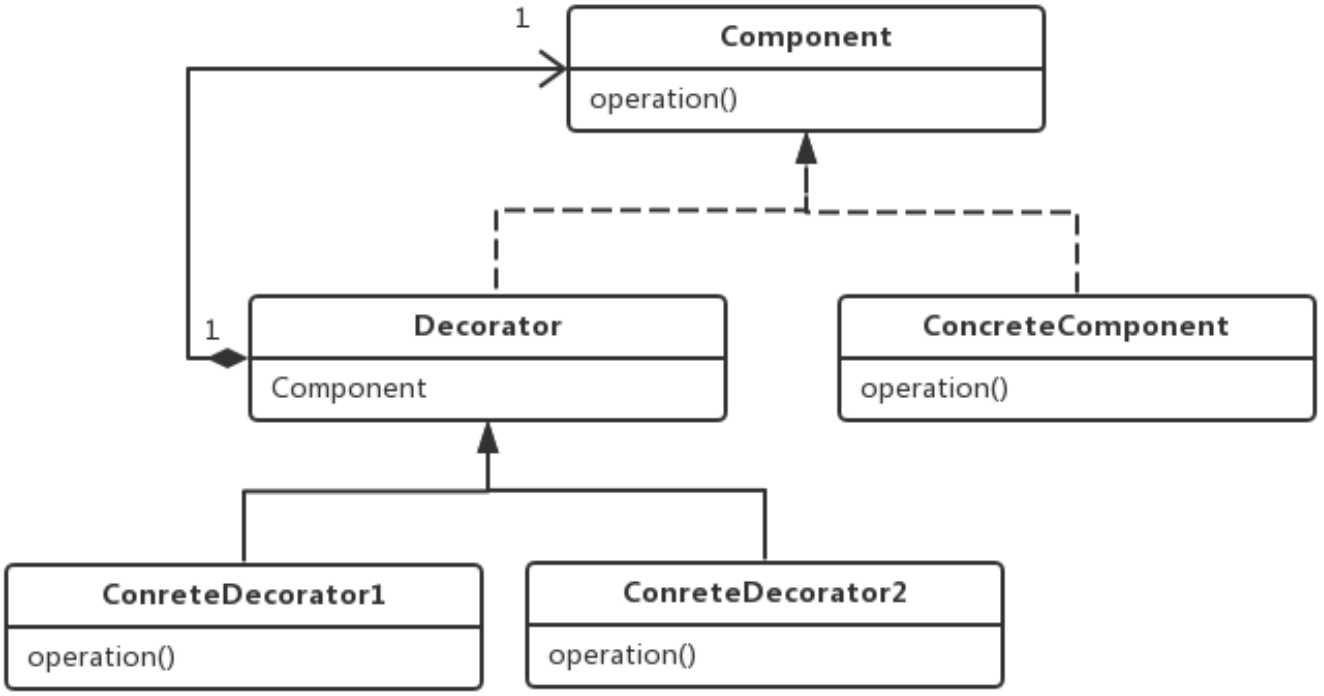
基础支撑层源码分析 缓存模块需求

- Mybatis缓存的实现是基于Map的，从缓存里面读写数据是缓存模块的核心基础功能；
- 除核心功能之外，有很多额外的附加功能，如：防止缓存击穿，添加缓存清空策略（fifo、lru）、序列化功能、日志能力、定时清空能力等；
- 附加功能可以以任意的组合附加到核心基础功能之上；

怎么样优雅的为核心功能添加附加能力？

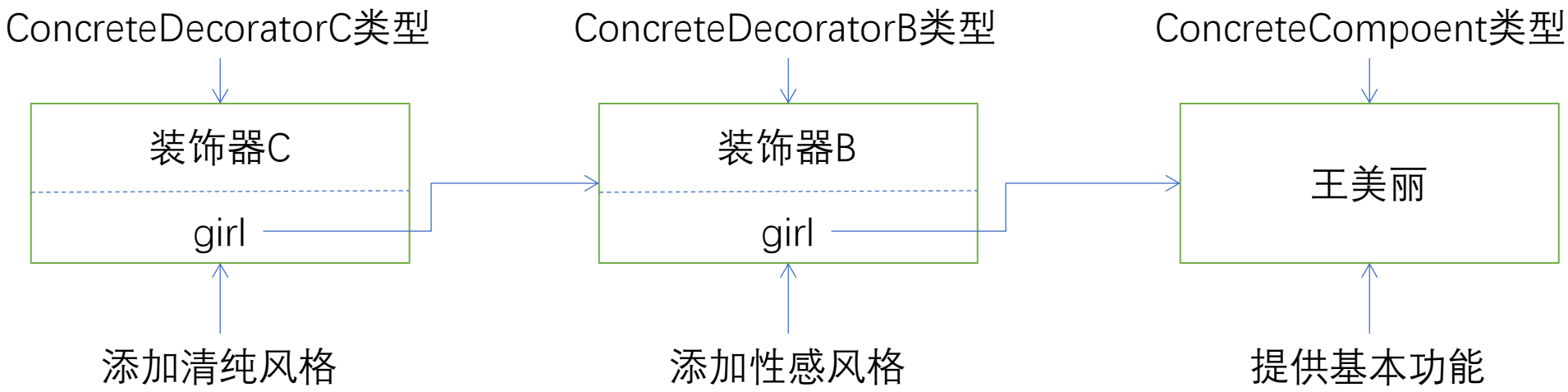
装饰器模式uml类图

- 装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，是一种用于代替继承的技术，无需通过继承增加子类就能扩展对象的新功能。使用对象的关联关系代替继承关系，更加灵活，同时避免类型体系的快速膨胀；



- ✓ **组件（Component）**：组件接口定义了全部组件类和装饰器实现的行为；
- ✓ **组件实现类（ConcreteComponent）**:实现Component接口，组件实现类就是被装饰器装饰的原始对象，新功能或者附加功能都是通过装饰器添加到该类的对象上的；
- ✓ **装饰器抽象类（Decorator）**：实现Component接口的抽象类，在其中封装了一个Component 对象，也就是被装饰的对象；
- ✓ **具体装饰器类（ConcreteDecorator）**：该实现类要向被装饰的对象添加某些功能；

装饰器模式使用图示



- 优点

  相对于继承，装饰器模式灵活性更强，扩展性更强；

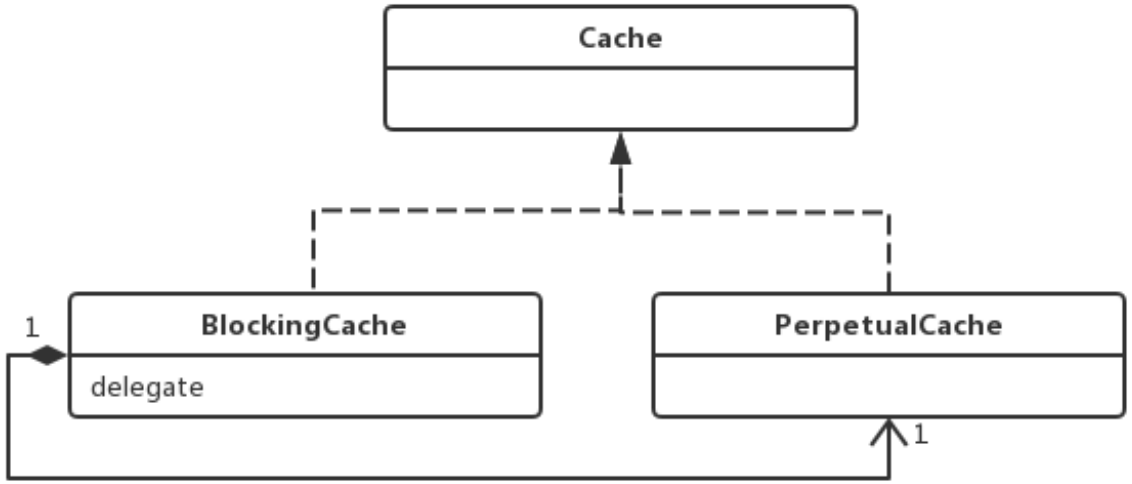
- ✓ 灵活性：装饰器模式将功能切分成一个个独立的装饰器，在运行期可以根据需要动态的添加功能，甚至对添加的新功能进行自由的组合；
- ✓ 扩展性：当有新功能要添加的时候，只需要添加新的装饰器实现类，然后通过组合方式添加这个新装饰器，无需修改已有代码，符合开闭原则；

装饰器模式使用举例

- ✓ IO中输入流和输出流的设计

```
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(new FileInputStream("c://a.txt")));
```

- ✓ Servlet API中提供了一个request对象的Decorator设计模式的默认实现类HttpServletRequestWrapper，HttpServletRequestWrapper类增强了request对象的功能。
- ✓ Mybatis的缓存组件

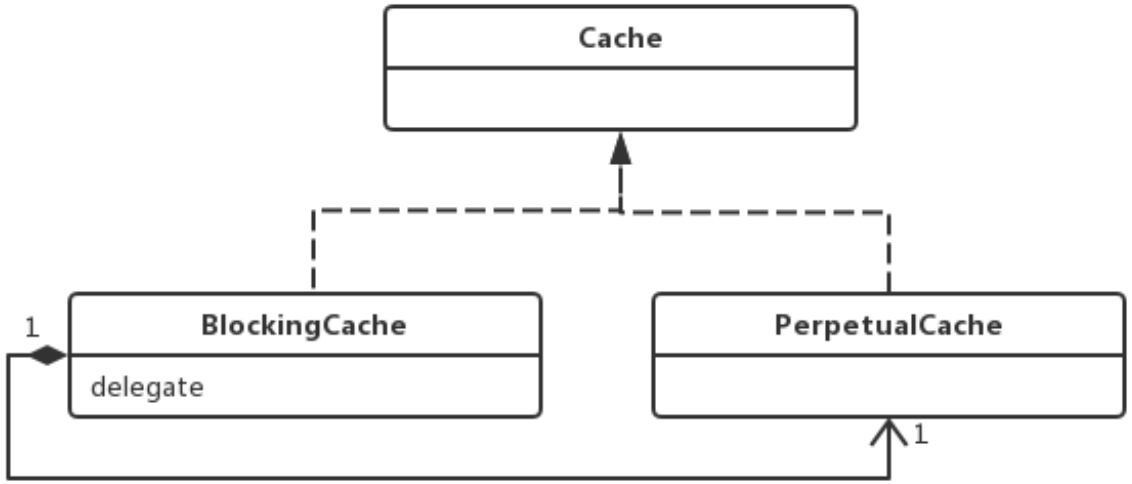


装饰器模式使用举例

- ✓ IO中输入流和输出流的设计

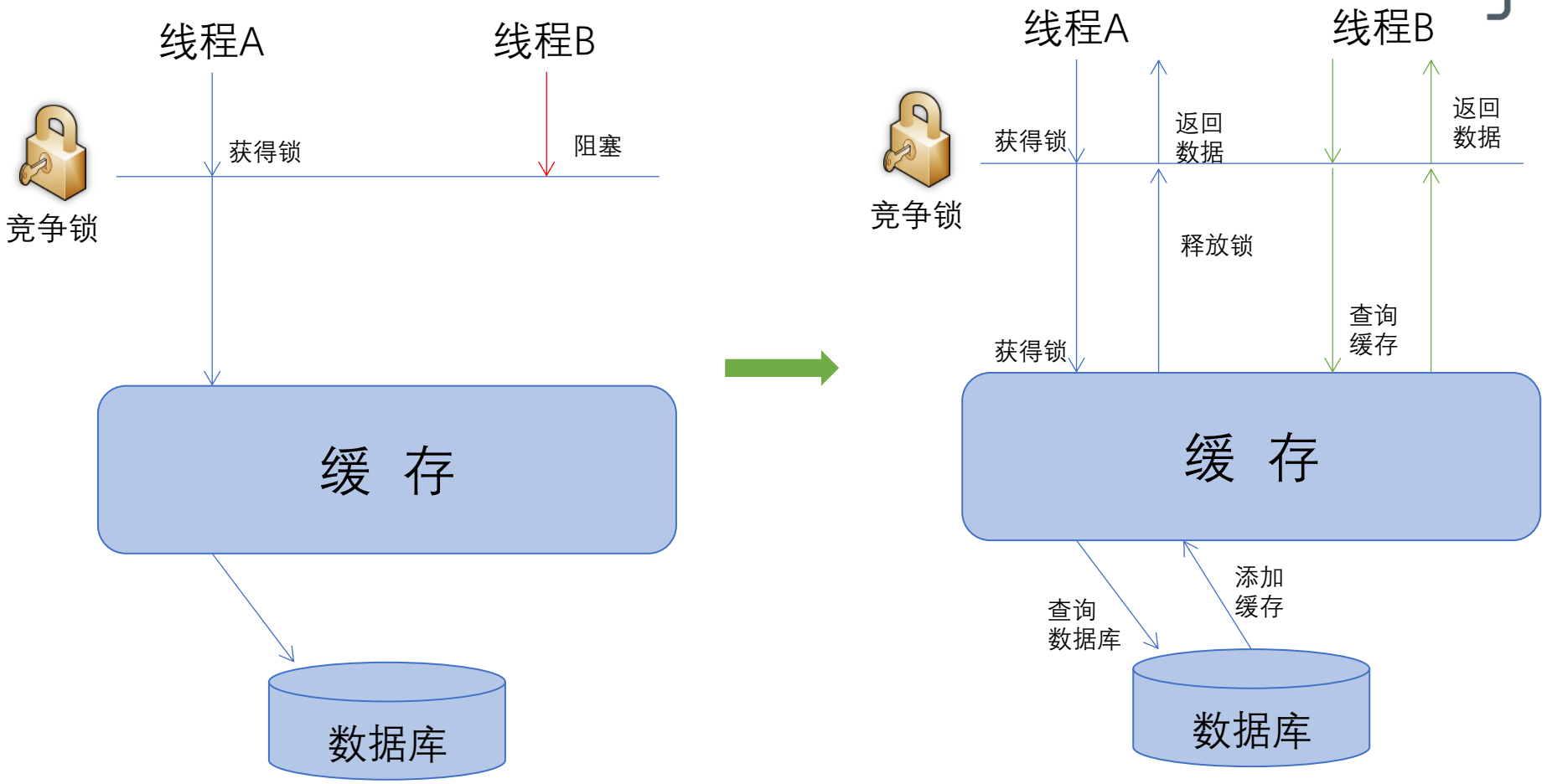
```
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(new FileInputStream("c://a.txt")));
```

- ✓ Servlet API中提供了一个request对象的Decorator设计模式的默认实现类HttpServletRequestWrapper，HttpServletRequestWrapper类增强了request对象的功能。
- ✓ Mybatis的缓存组件



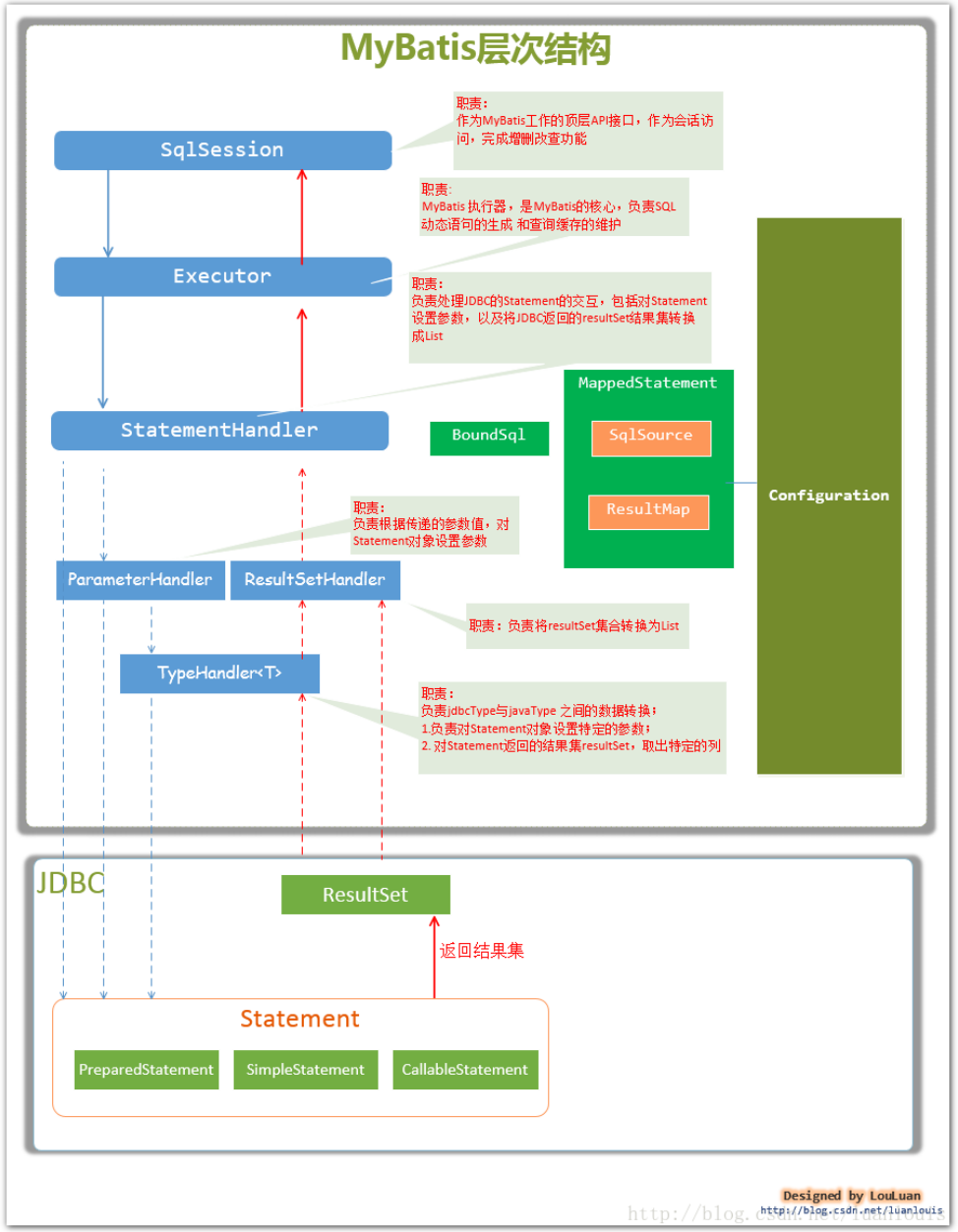
缓存装饰器解读

- ✓ FifoCache
- ✓ LoggingCache
- ✓ ScheduledCache
- ✓ BlockingCache



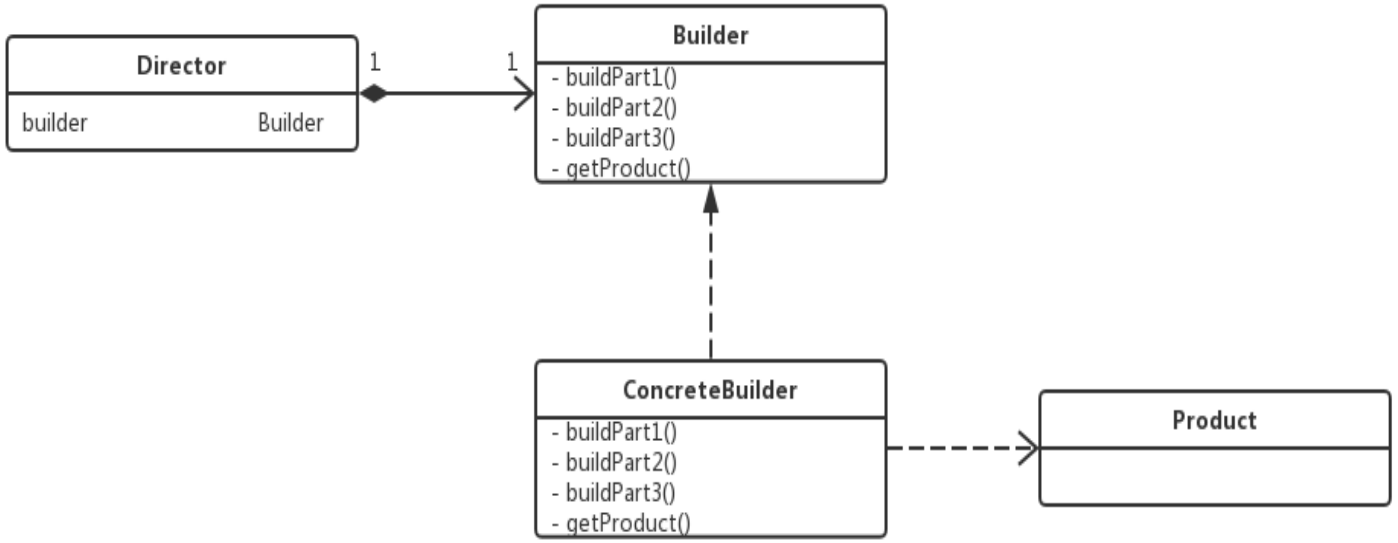
CacheKey解读

mybatis功能流程图



Mybatis的初始化 建造者模式

- 建造者模式（Builder Pattern）使用多个简单的对象一步一步构建成一个复杂的对象。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

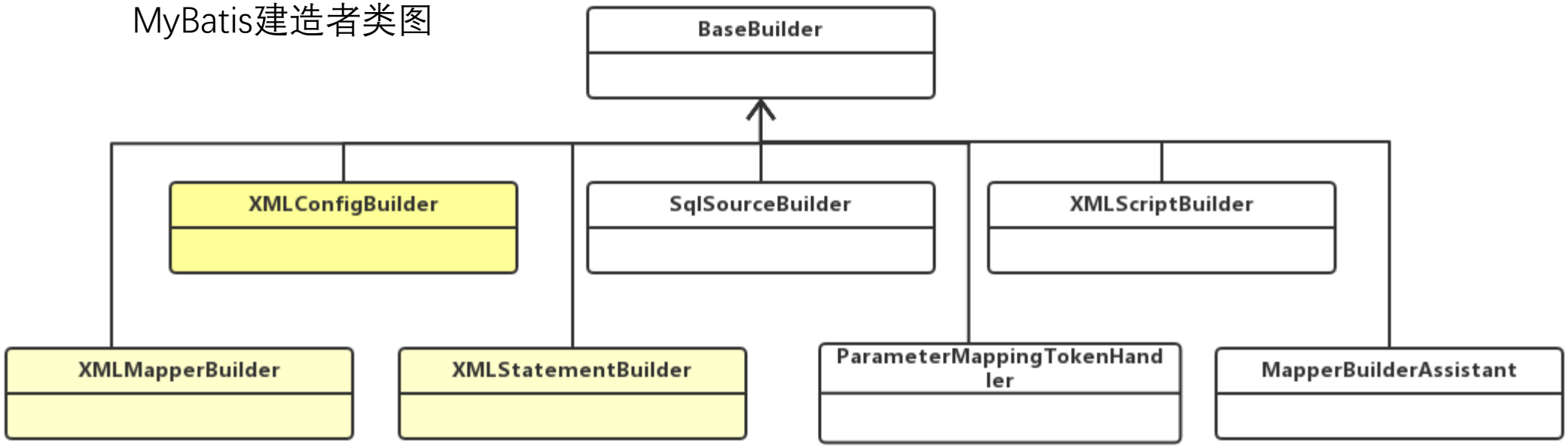


- ✓ **Builder**：给出一个抽象接口，以规范产品对象的各个组成成分的建造。这个接口规定要实现复杂对象的哪些部分的创建，并不涉及具体的对象部件的创建；
- ✓ **ConcreteBuilder**：实现Builder接口，针对不同的商业逻辑，具体化复杂对象的各部分的创建。 在建造过程完成后，提供产品的实例；
- ✓ **Director**：调用具体建造者来创建复杂对象的各个部分，在指导者中不涉及具体产品的信息，只负责保证对象各部分完整创建或按某种顺序创建；
- ✓ **Product**：要创建的复杂对象

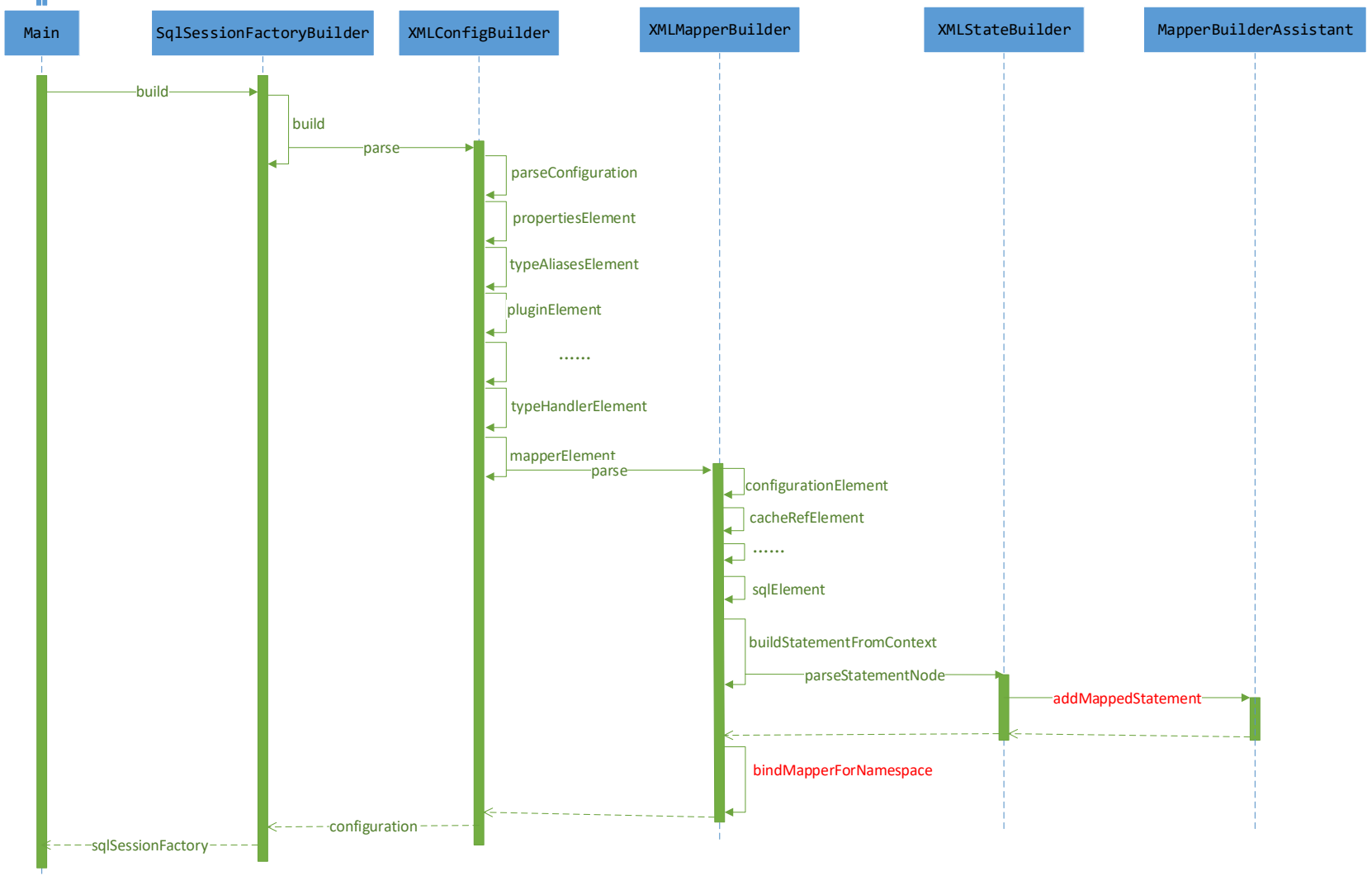
建造者模式 使用场景

- ✓ 需要生成的对象具有复杂的内部结构，实例化对象时要屏蔽掉对象内部的细节，让上层代码与复杂对象的实例化过程解耦，可以使用建造者模式；简而言之，如果“遇到多个构造器参数时要考虑用构建器”；
- ✓ 一个对象的实例化是依赖各个组件的产生以及装配顺序，关注的是一步一步地组装出目标对象，可以使用建造器模式；

MyBatis建造者类图



MyBatis初始化过程



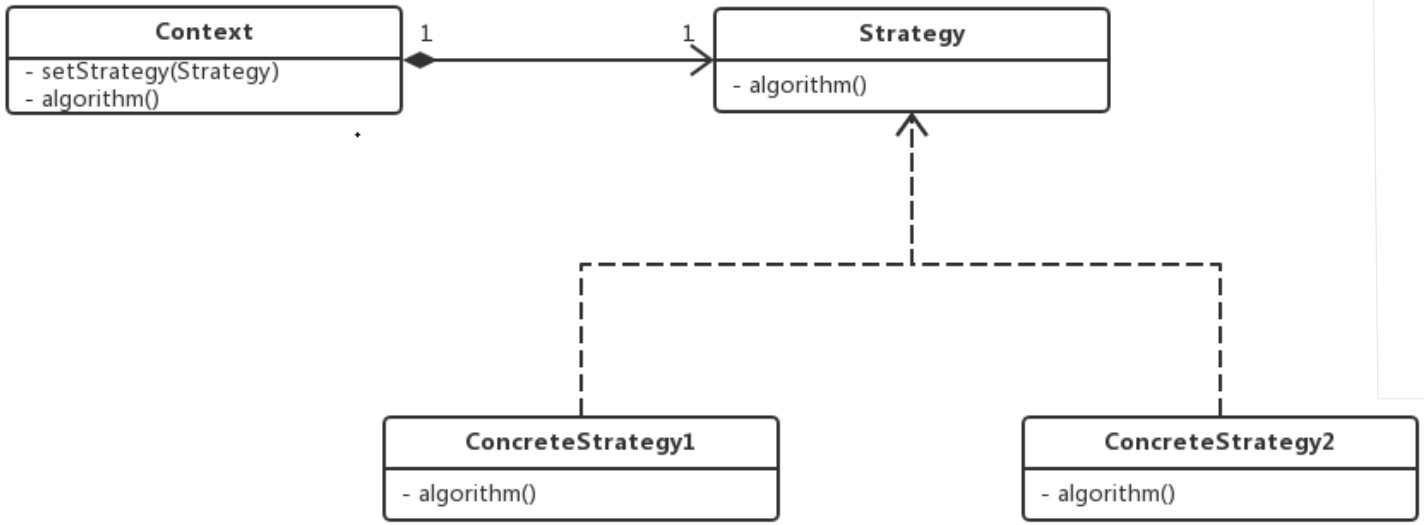
映射器的关键类

待完善：

- ✓ ResultMap
- ✓ MappedStatement
- ✓ SqlSource
- ✓ BoundSql
- ✓ MapperRegistry

创建SqlSession　策略模式

- **策略模式（Builder Pattern）**策略模式定义了一系列的算法，并将每一个算法封装起来，而且使他们可以相互替换，让算法独立于使用它的客户而独立变化。



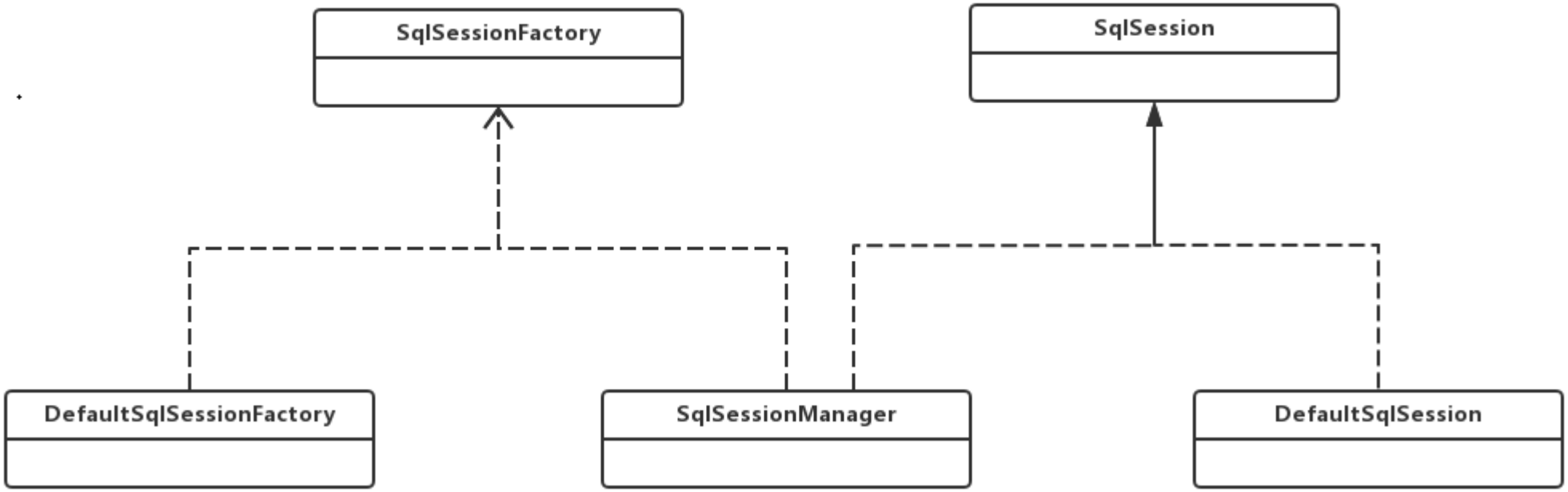
策略模式的使用场景：

- ✓ 针对同一类型问题的多种处理方式，仅仅是具体行为有差别时；
- ✓ 出现同一抽象类有多个子类，而又需要使用 if-else 或者 switch-case 来选择具体子类时。

- ✓ **Context：**算法调用者，使用setStrategy方法灵活的选择策略（strategy）；
- ✓ **Strategy：**算法的统一接口；
- ✓ **ConcreteStrategy：**算法的具体实现；

SqlSession相关类UML

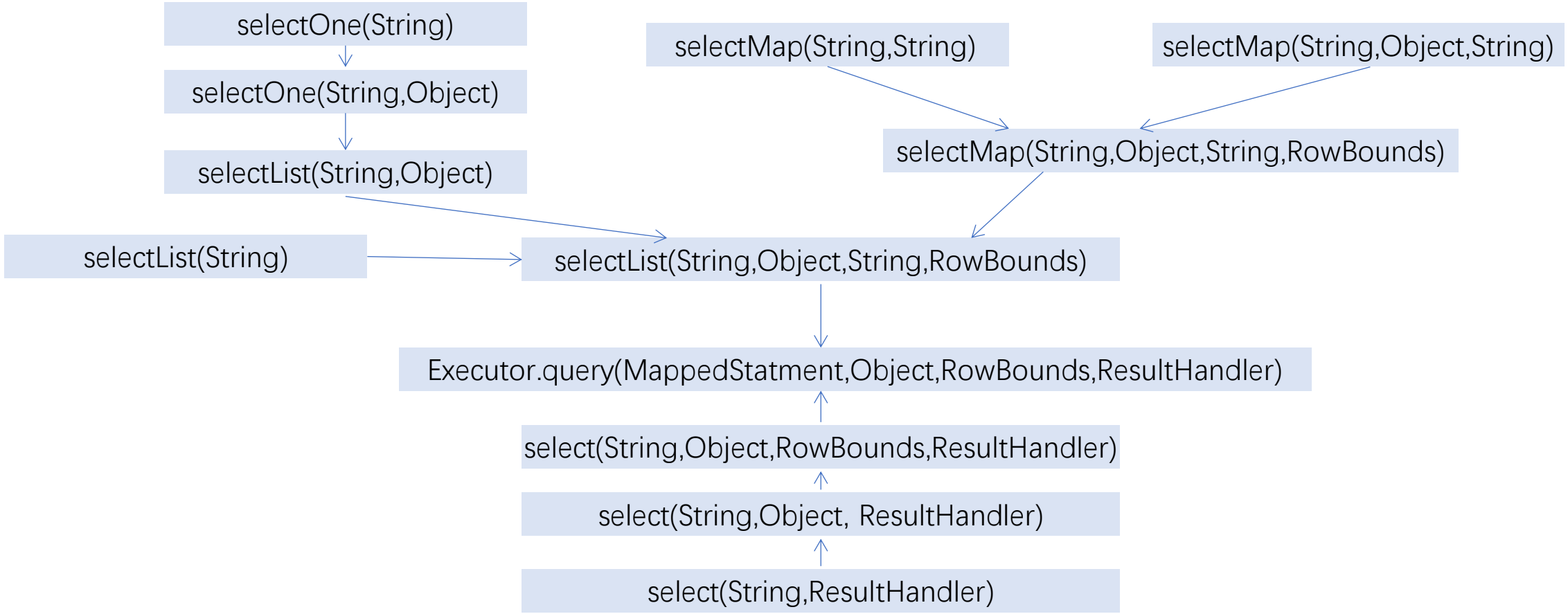
- **SqlSession**是MyBaits对外提供的最关键的核心接口，通过它可以执行数据库读写命令、获取映射器、管理事务等；



SqlSessionManager

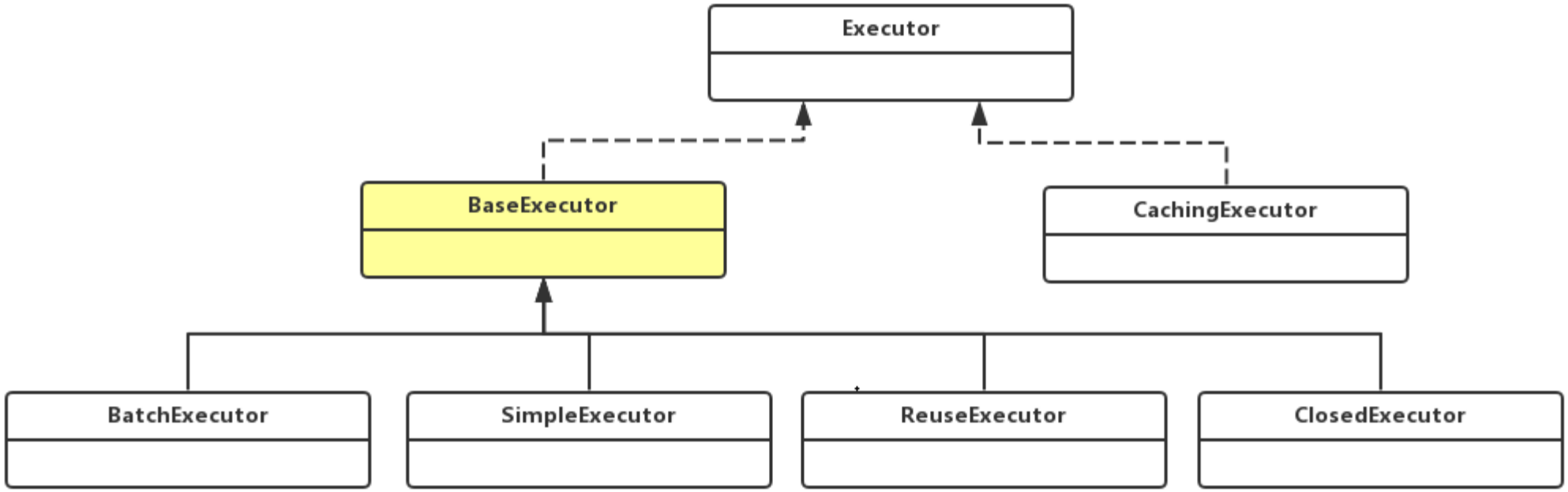
- sqlSessionManager同时继承了SqlSession接口和SqlSessionFactroy接口，提供了创建SqlSession对象和操纵数据库的能力；
- SqlSessionManager有两种获取SqlSession的模式：
  - 第一种模式和SqlSessionFactroy 相同，同一个线程每次访问数据库，每次都可以创建新的SqlSession对象；
  - 第二种模式，同一个线程每次访问数据库，都是使用同一个SqlSession对象,通过localSqlSession实现；

SqlSession查询接口嵌套关系



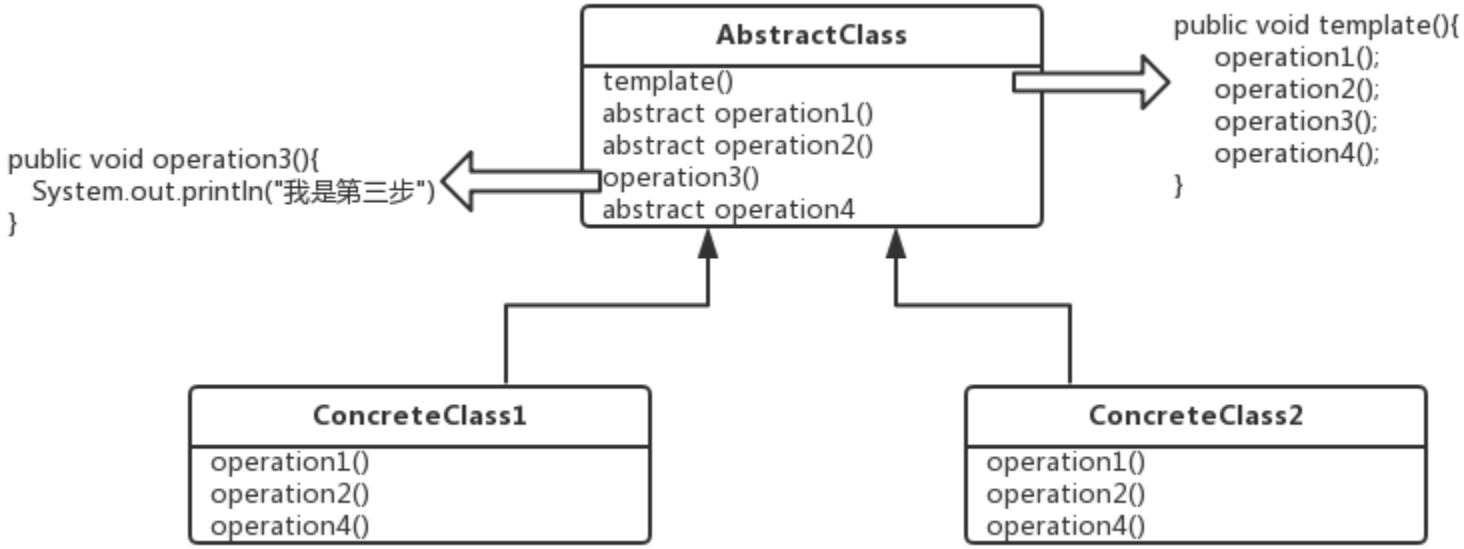
Executor组件分析

- **Executor**是MyBaits核心接口之一，定义了数据库操作最基本的方法，SqlSession的功能都是基于它来实现的；



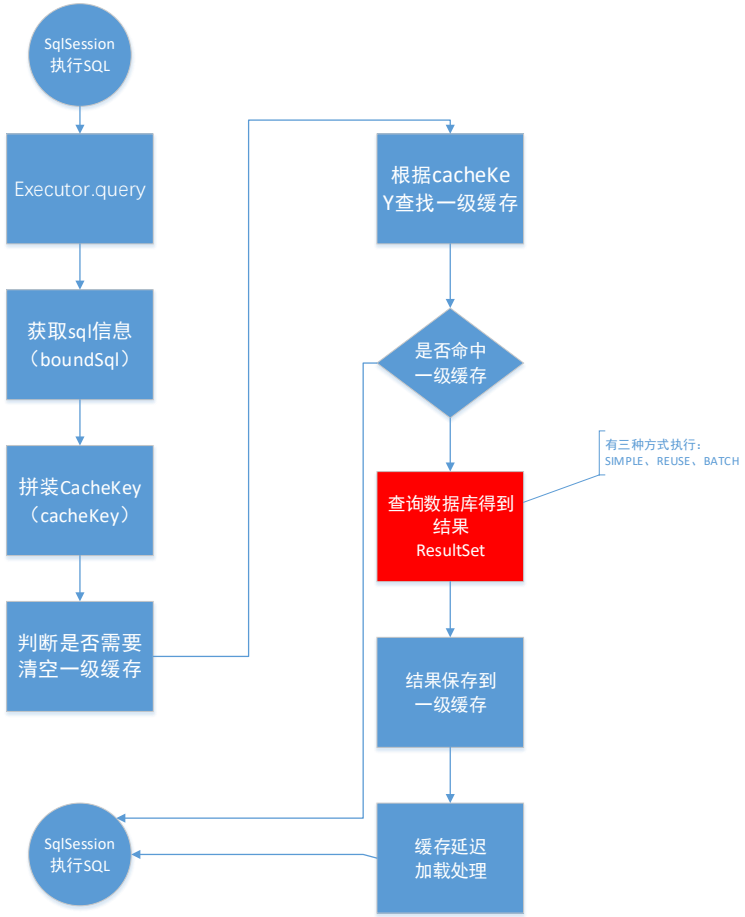
模板模式

- **模板模式（Template Pattern）**：一个抽象类公开定义了执行它的方法的方式/模板。它的子类可以按需要重写方法实现，但调用将以抽象类中定义的方式进行。定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定实现；



模板模式应用场景

遇到由一系列步骤构成的过程需要执行，这个过程从高层次上看是相同的，但是有些步骤的实现可能不同，这个时候就需要考虑用模板模式了。比如：Executor查询操作流程:



Executor的三个实现类解读

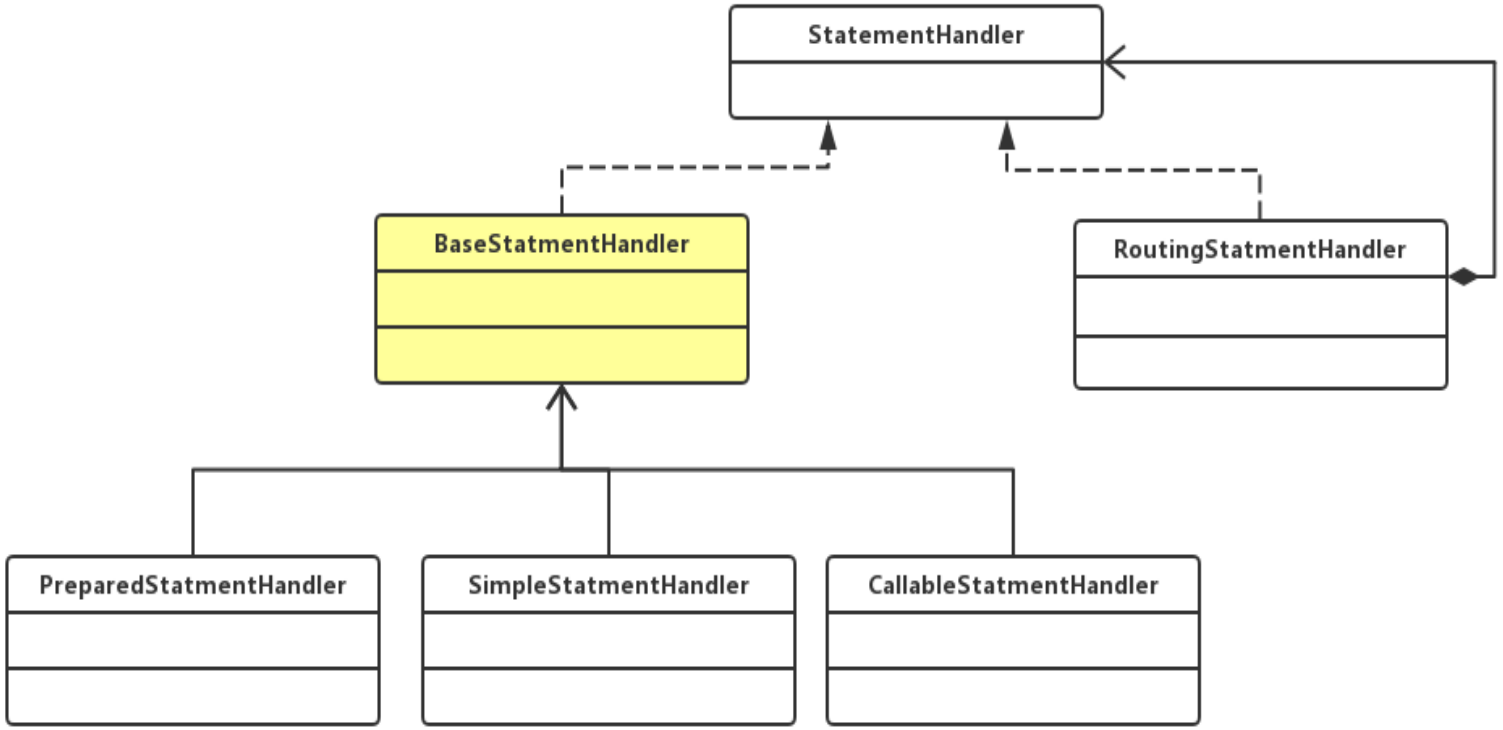
- SimpleExecutor：默认配置，使用statement对象访问数据库，每次访问都要创建新的statement对象；
- ReuseExecutor：使用预编译PrepareStatement对象访问数据库，访问时，会重用缓存中的statement对象；
- BatchExecutor：实现批量执行多条SQL语句的能力；

Executor的三个重要小弟

- 通过对SimpleExecutor doQuery()方法的解读发现，Executor是个指挥官，它在调度三个小弟工作：
  - StatementHandler：它的作用是使用数据库的Statement或PrepareStatement执行操作，启承上启下作用；
  - ParameterHandler：对预编译的SQL语句进行参数设置，SQL语句中的的占位符 “？” 都对应BoundSql.parameterMappings集合中的一个元素，在该对象中记录了对应的参数名称以及该参数的相关属性
  - ResultHandler：对数据库返回的结果集（ResultSet）进行封装，返回用户指定的实体类型；

StatementHandler分析

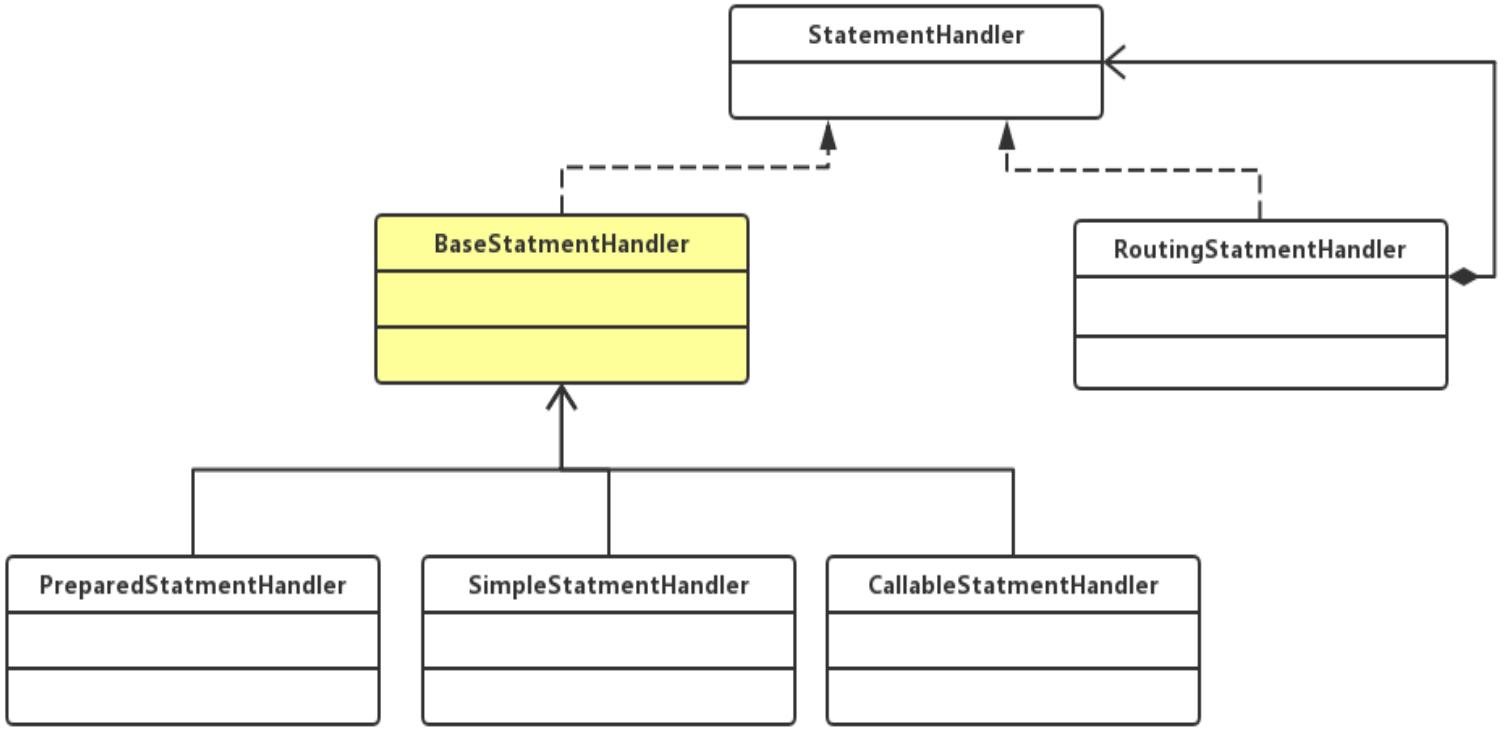
- StatementHandler完成Mybatis最核心的工作，也是Executor实现的基础；功能包括：创建statement对象，为sql语句绑定参数，执行增删改查等SQL语句、将结果映射集进行转化；



- ✓ **BaseStatementHandler**：所有子类的抽象父类，定义了初始化statement的操作顺序，由子类实现具体的实例化不同的statement（模板模式）；
- ✓ **RoutingStatementHandler**：Excutor组件真正实例化的子类，使用静态代理模式，根据上下文决定创建哪个具体实体类；
- ✓ **SimpleStatmentHandler**：使用statement对象访问数据库，无须参数化；
- ✓ **PreparedStatmentHandler**：使用预编译PrepareStatement对象访问数据库；
- ✓ **CallableStatmentHandler**：调用存储过程；

ResultHandler分析

- StatementHandler完成Mybatis最核心的工作，也是Executor实现的基础；功能包括：创建statement对象，为sql语句绑定参数，执行增删改查等SQL语句、将结果映射集进行转化；



- ✓ **BaseStatementHandler**：所有子类的抽象父类，定义了初始化statement的操作顺序，由子类实现具体的实例化不同的statement（模板模式）；
- ✓ **RoutingStatementHandler**：Excutor组件真正实例化的子类，使用静态代理模式，根据上下文决定创建哪个具体实体类；
- ✓ **SimpleStatmentHandler**：使用statement对象访问数据库，无须参数化；
- ✓ **PreparedStatmentHandler**：使用预编译PrepareStatement对象访问数据库；
- ✓ **CallableStatmentHandler**：调用存储过程；

为什么使用mapper接口就能操作数据库？

sqlSession对数据库执行一次查询操作的时序图？描述主要流程