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

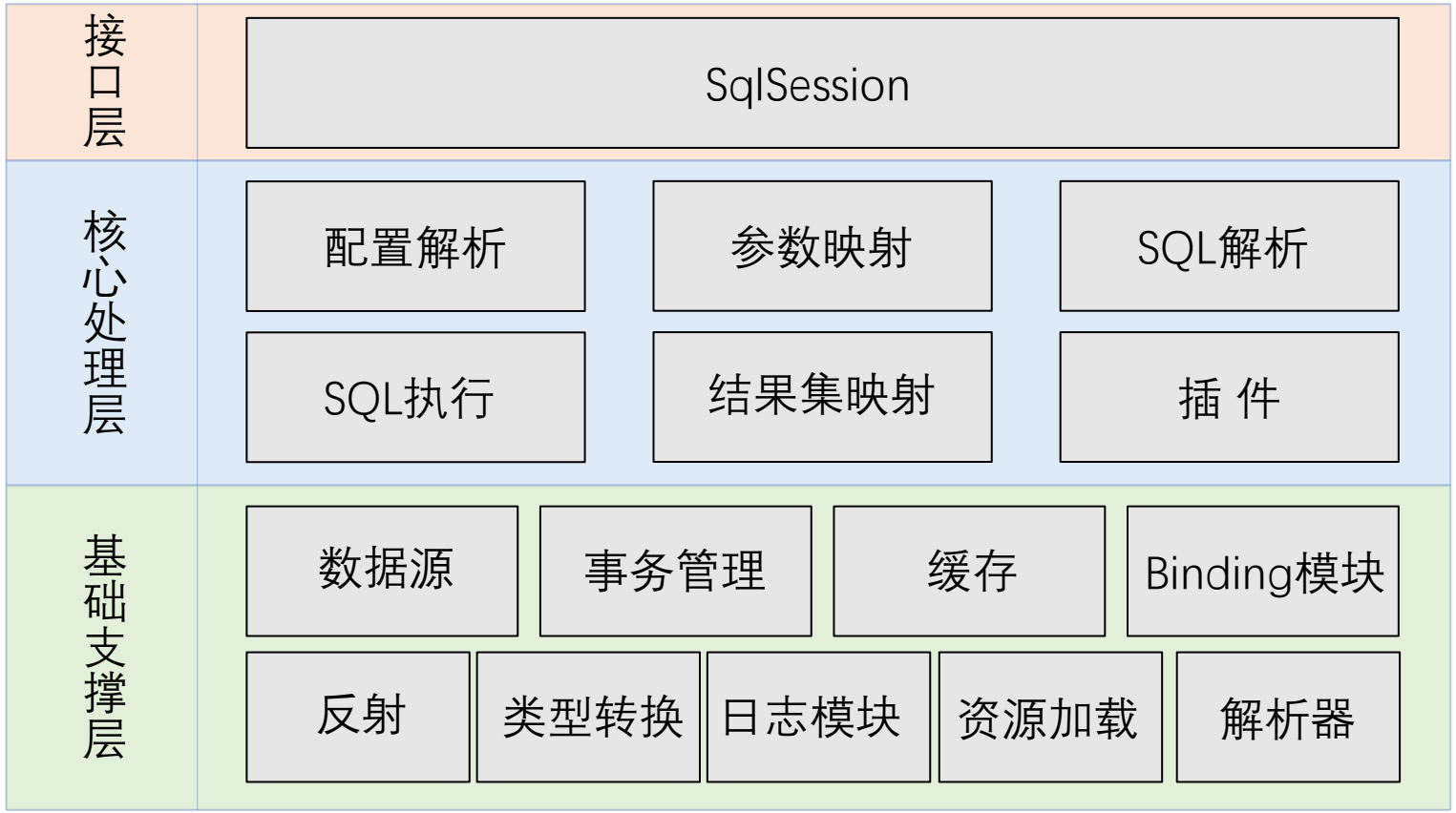
源码包分析

MyBatis 源码下载地址：https://github.com/mybatis/mybatis-3

MyBatis源码结构x

- MyBatis源码导入过程：
  1. 下载MyBatis的源码
  2. 检查maven的版本，必须是3.25以上，建议使用maven的最新版本
  3. mybatis的工程是maven工程，在开发工具中导入，工程必须使用jdk1.8以上版本；
  4. 把mybatis源码的pom文件中<optional>true</optional>，全部改为false；
  5. 在工程目录下执行 mvn clean install -Dmaven.test.skip=true,将当前工程安装到本地仓库（pdf插件报错的话，需要将这个插件屏蔽）；
  6. 其他工程依赖此工程

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；
- 依赖倒转原则：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

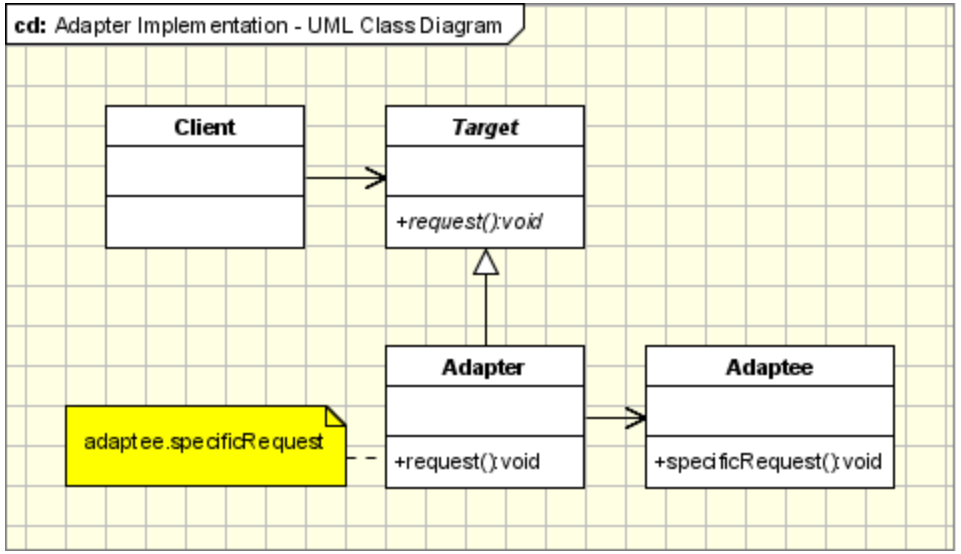
CONTENTS

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

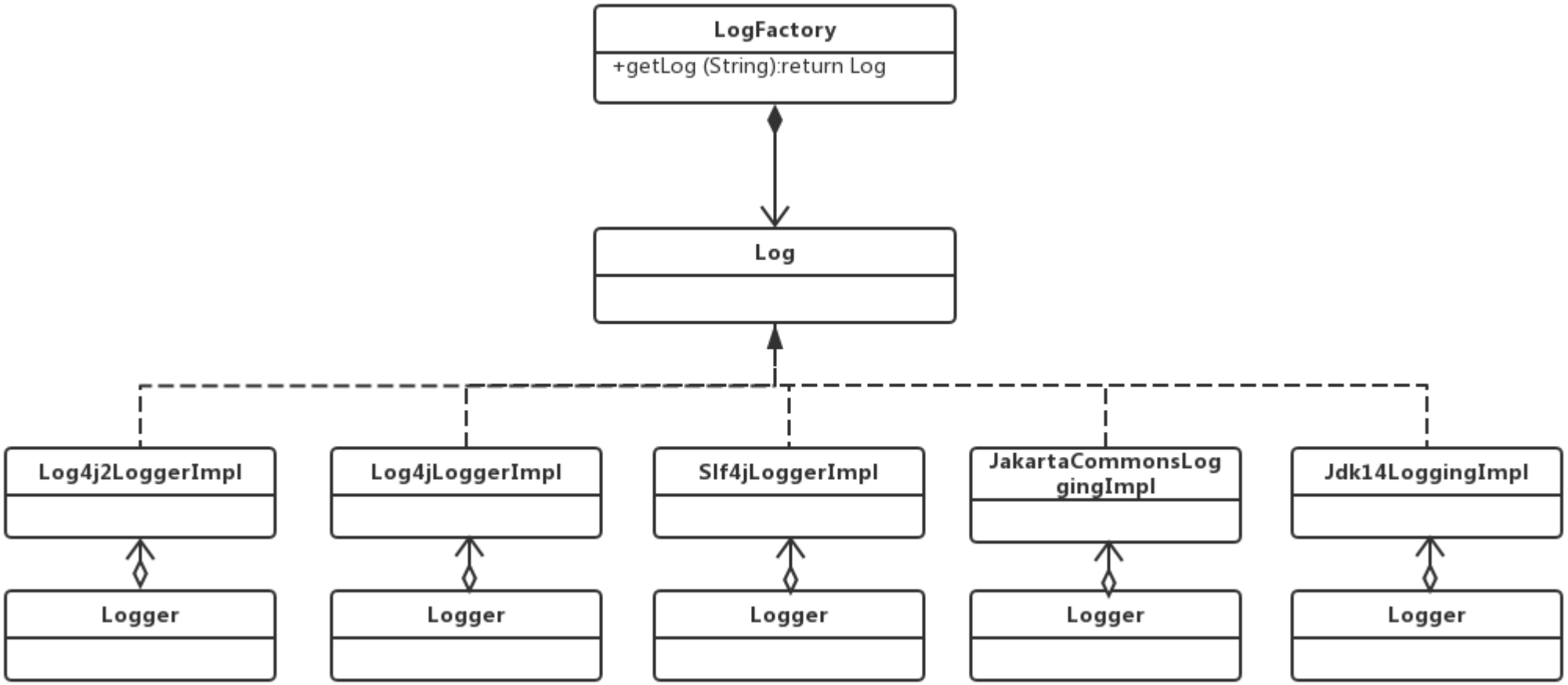
- 适配器模式（Adapter Pattern）是作为两个不兼容的接口之间的桥梁，将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作；



- ✓ 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

why proxy?

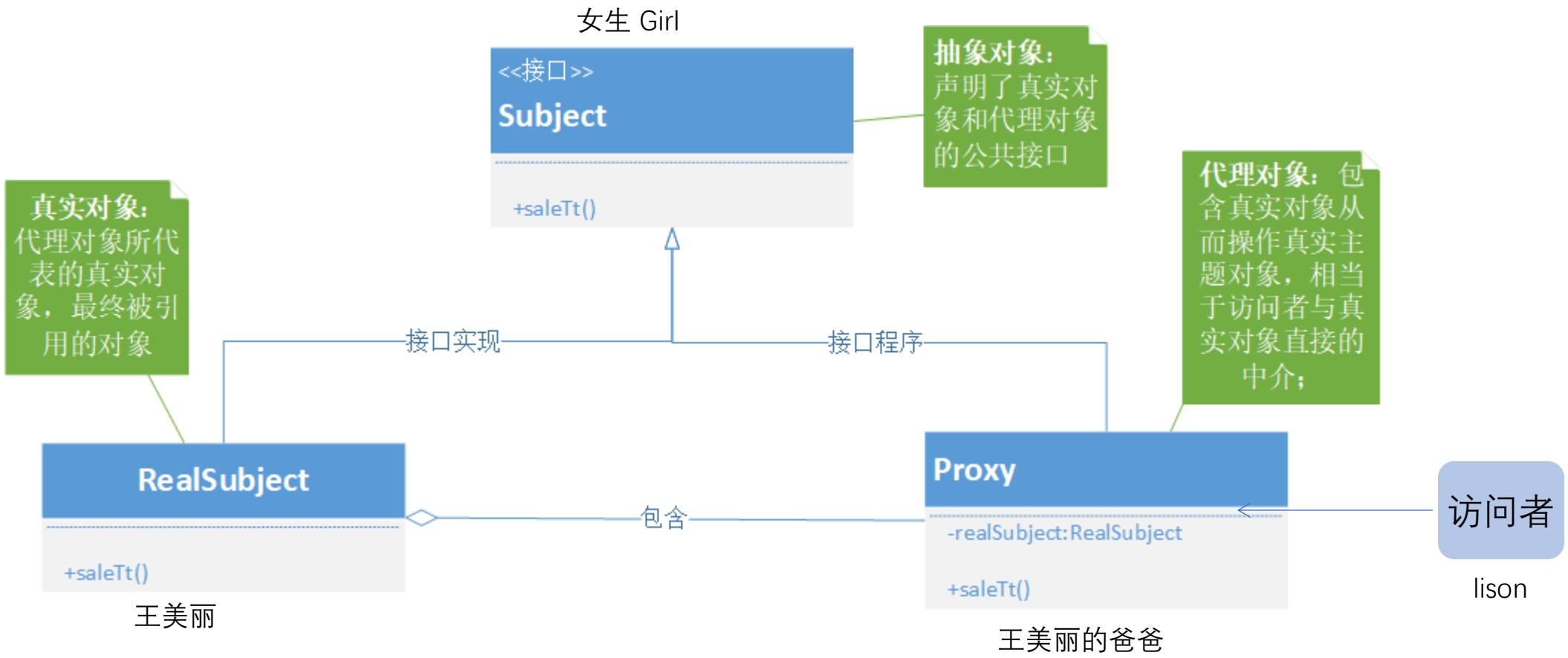
- **代理模式给我们带来的便利：**

✓ 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）

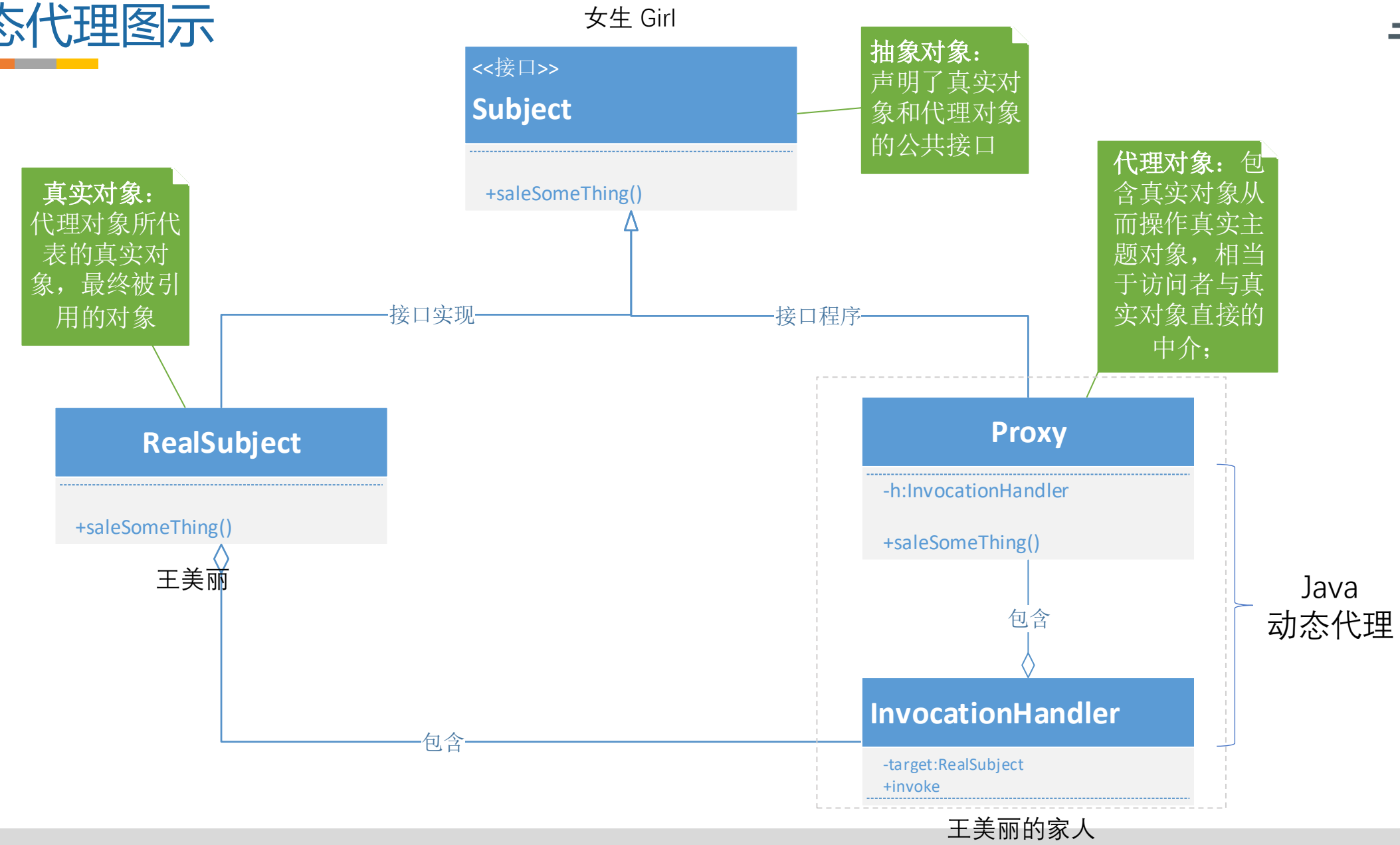
✓ 防止直接访问目标对象给系统带来的不必要复杂性；（海外代购,SSH）

✓ 对业务进行增强，增强点多样化如：前入、后入、异常；（AOP）

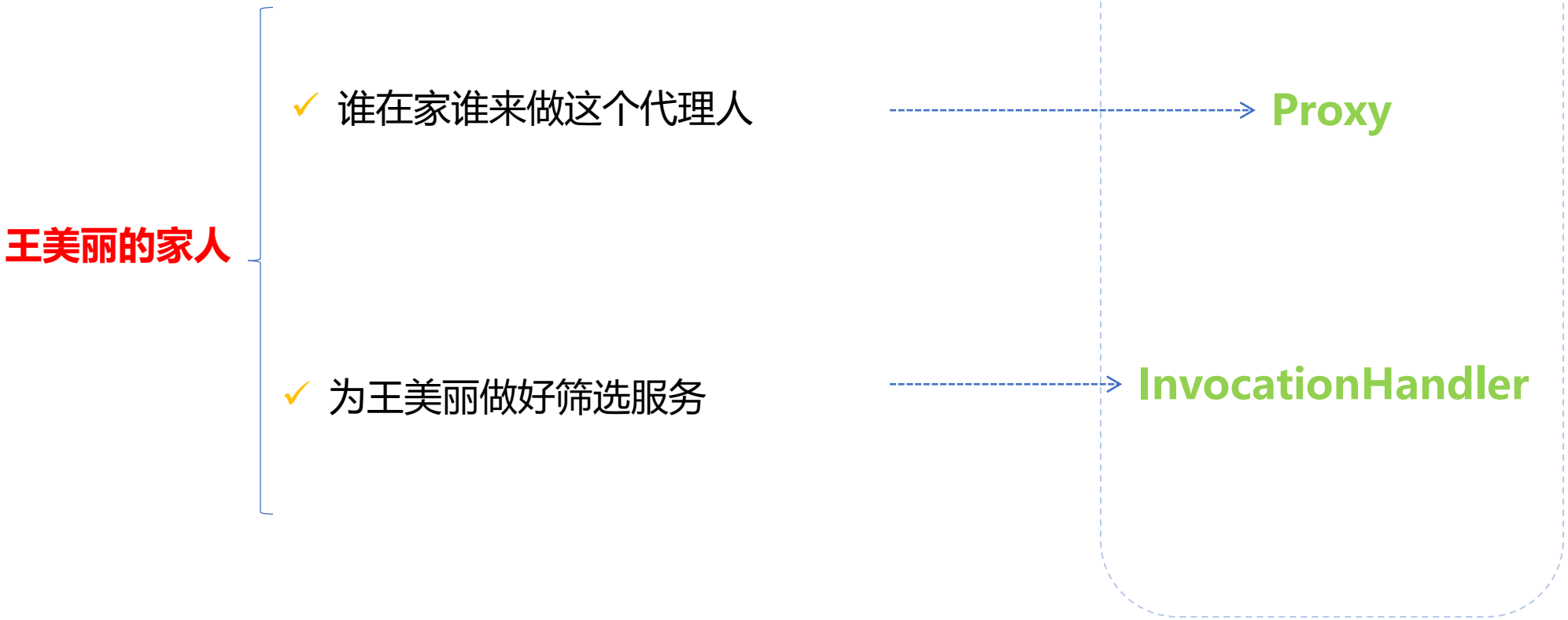
代理模式类图


女生 Girl
<<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()
王美丽
包含
Proxy
-realSubject:RealSubject
+saleTt()
王美丽的爸爸
访问者
lison

动态代理图示

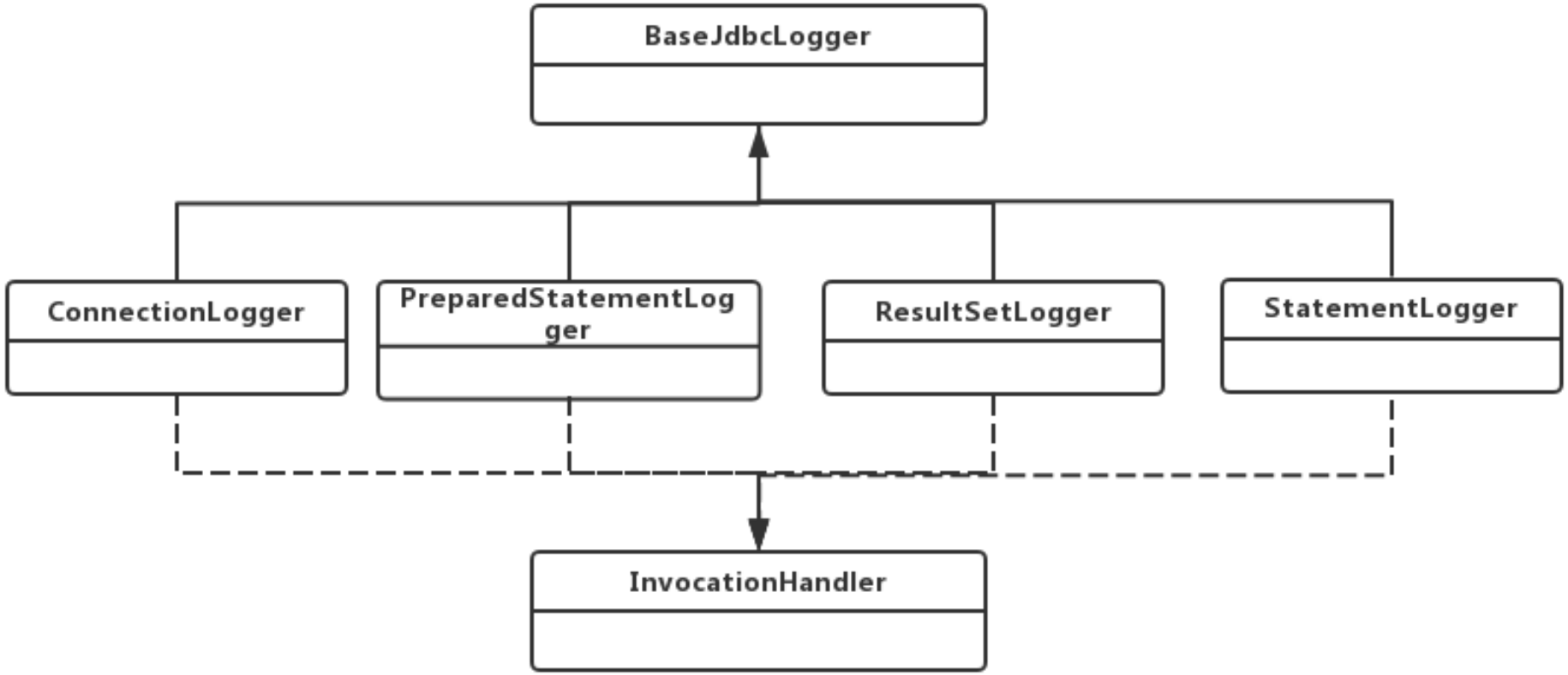

女生 Girl
<<接口>>
Subject
+saleSomeThing()
抽象对象：声明了真实对象和代理对象的公共接口
真实对象：代理对象所代表的真实对象，最终被引用的对象
代理对象：包含真实对象从而操作真实主题对象，相当于访问者与真实对象直接的中介；
接口实现
接口程序
RealSubject
+saleSomeThing()
王美丽
Proxy
-h:InvocationHandler
+saleSomeThing()
包含
InvocationHandler
-target:RealSubject
+invoke
包含
Java
动态代理
王美丽的家人

解密动态代理


JDK动态代理
王美丽的家人
✓ 谁在家谁来做这个代理人
Proxy
✓ 为王美丽做好筛选服务
InvocationHandler

日志模块JDBC包类图

- ✓ **ConnectionLogger**：负责打印连接信息和SQL语句，并创建PreparedStatementLogger；
- ✓ **PreparedStatementLogger**：负责打印参数信息，并创建ResultSetLogger
- ✓ **ResultSetLogge**：r负责打印数据结果信息；



目 录

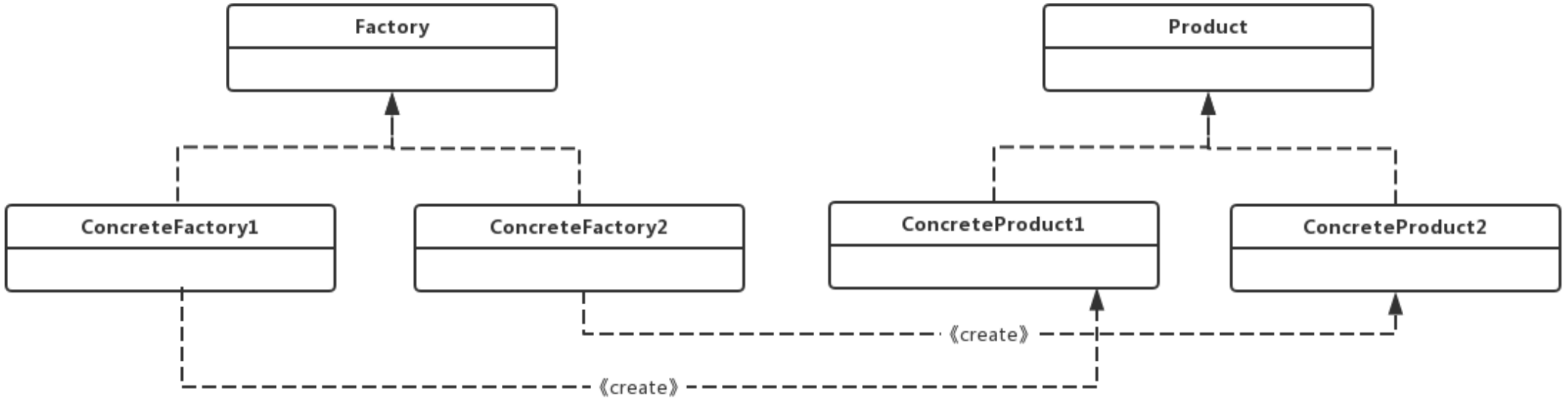
CONTENTS

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；

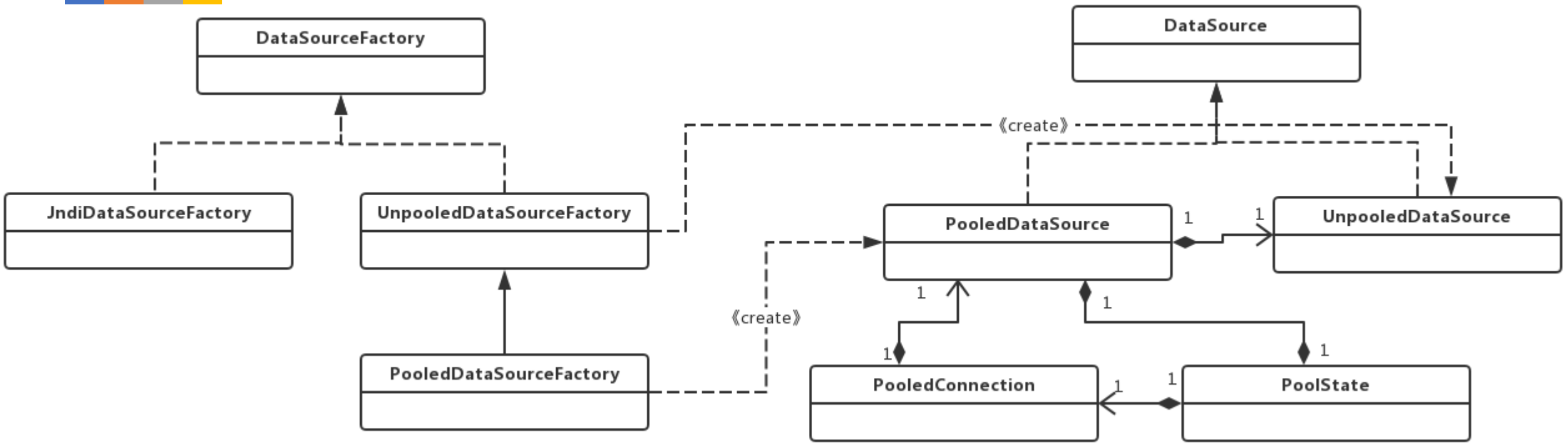
为什么要使用工厂模式？

- 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；

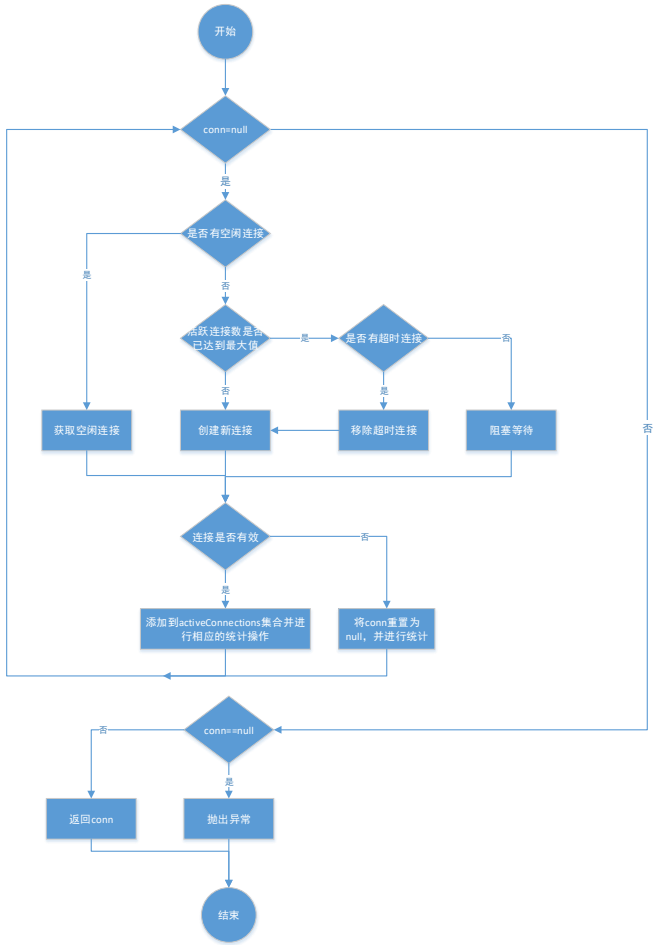
优点：
- ✓ 把对象的创建和使用的过程分开，对象创建和对象使用使用的职责解耦；
- ✓ 如果创建对象的过程很复杂，创建过程统一到工厂里管理，既减少了重复代码，也方便以后对创建过程的修改维护；
- ✓ 当业务扩展时，只需要增加工厂子类，符合开闭原则；

数据源模块类图

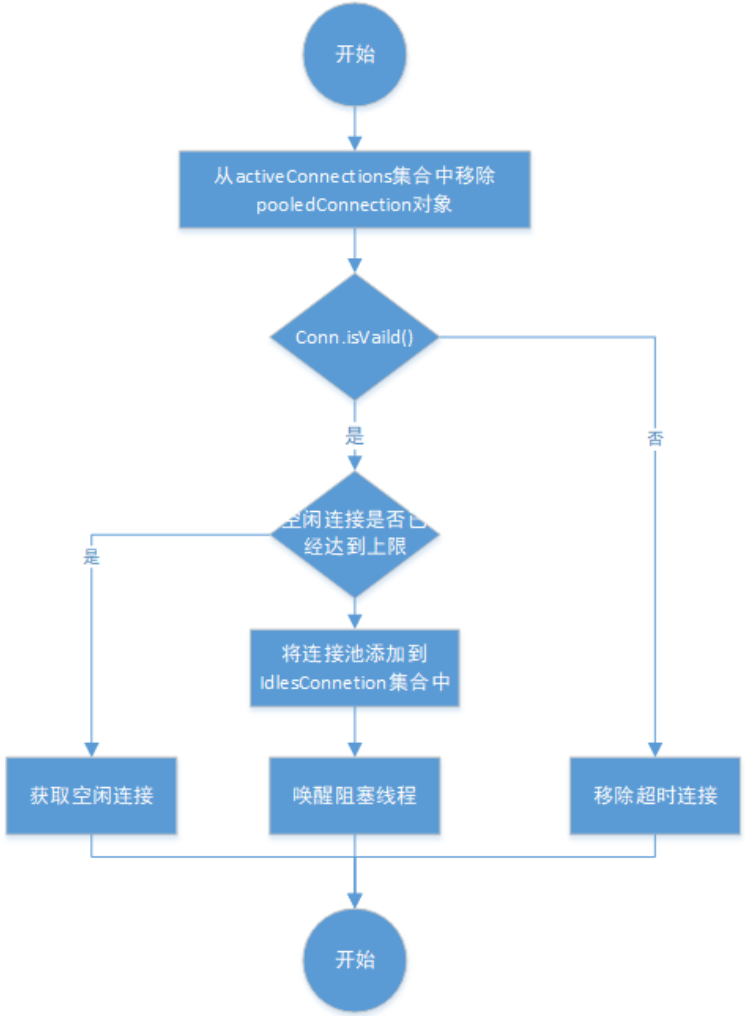


- ✓ **PooledConnection**：使用动态代理封装了真正的数据库连接对象；
- ✓ **PoolState**：用于管理PooledConnection对象状态的组件，通过两个list分别 管理空闲状态的连接资源和活跃状态的连接资源
- ✓ **PooledDataSource**：一个简单，同步的、线程安全的数据库连接池

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

反射模块分析

反射过程分析

反射核心类

基础支撑层源码分析 缓存模块需求

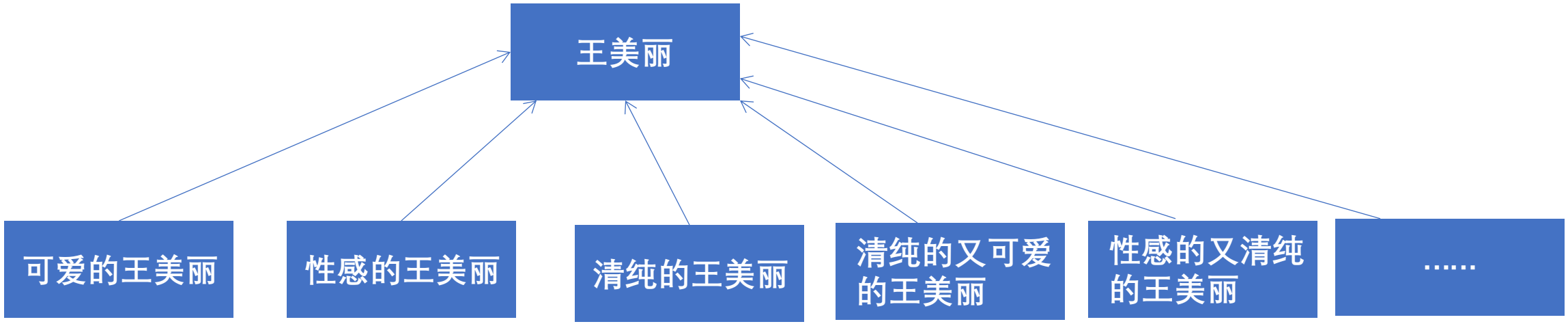
- Mybatis缓存的实现是基于Map的，从缓存里面读写数据是缓存模块的核心基础功能；
- 除核心功能之外，有很多额外的附加功能，如：防止缓存击穿，添加缓存清空策略（fifo、lru）、序列化功能、日志能力、定时清空能力等；
- 附加功能可以以任意的组合附加到核心基础功能之上；

怎么样优雅的为核心功能添加附加能力？

使用继承的办法扩展附加功能？

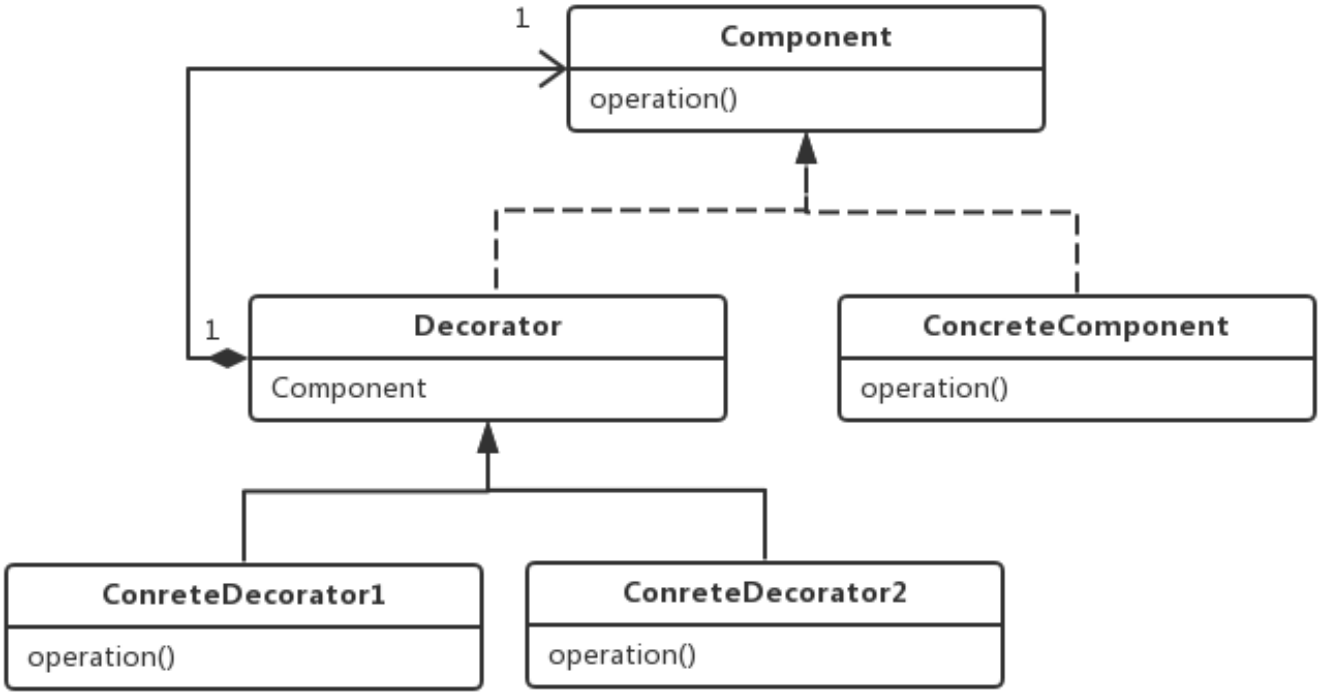
A：继承的方式是静态的，用户不能控制增加行为的方式和时机。另外，新功能的存在多种组合，使用继承可能导致大量子类存在；

使用继承扩展王美丽？



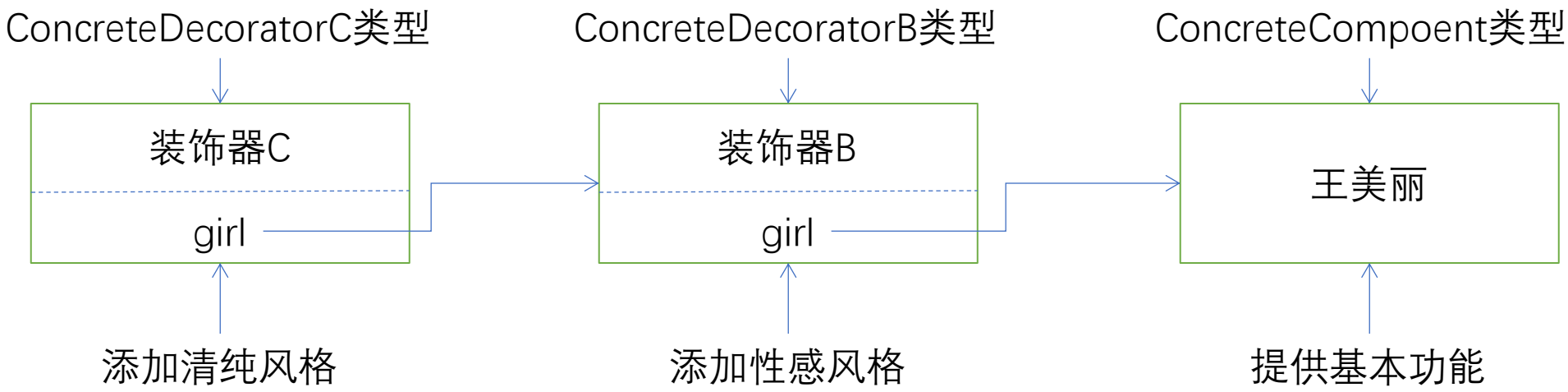
装饰器模式uml类图

- 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；

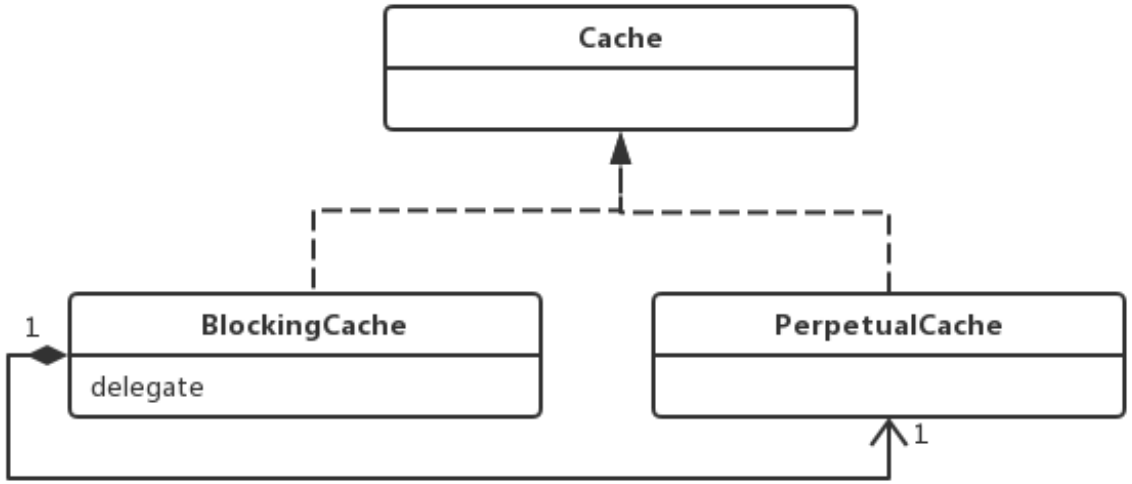
- ✓ 灵活性：装饰器模式将功能切分成一个个独立的装饰器，在运行期可以根据需要动态的添加功能，甚至对添加的新功能进行自由的组合；
- ✓ 扩展性：当有新功能要添加的时候，只需要添加新的装饰器实现类，然后通过组合方式添加这个新装饰器，无需修改已有代码，符合开闭原则；

装饰器模式使用举例

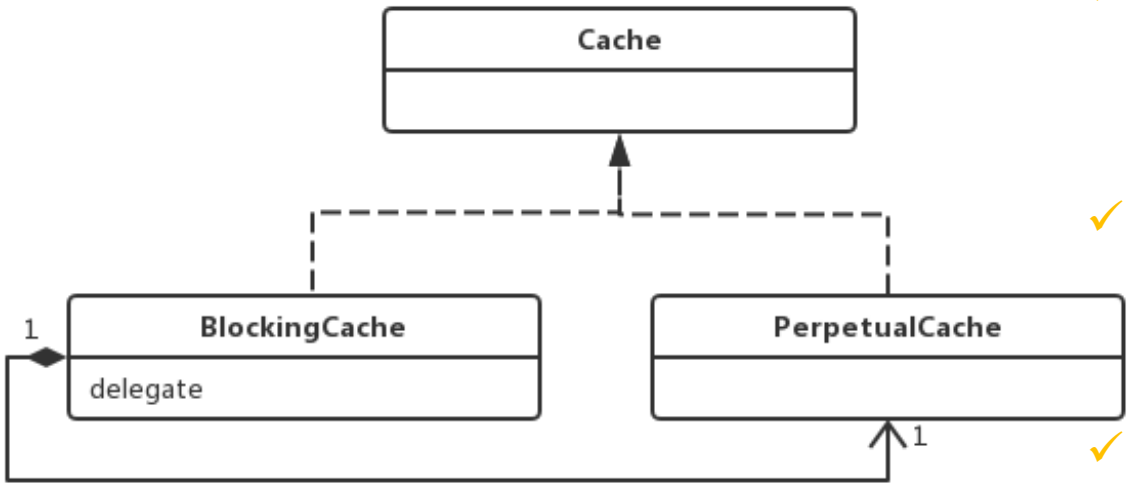
- ✓ IO中输入流和输出流的设计

```
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

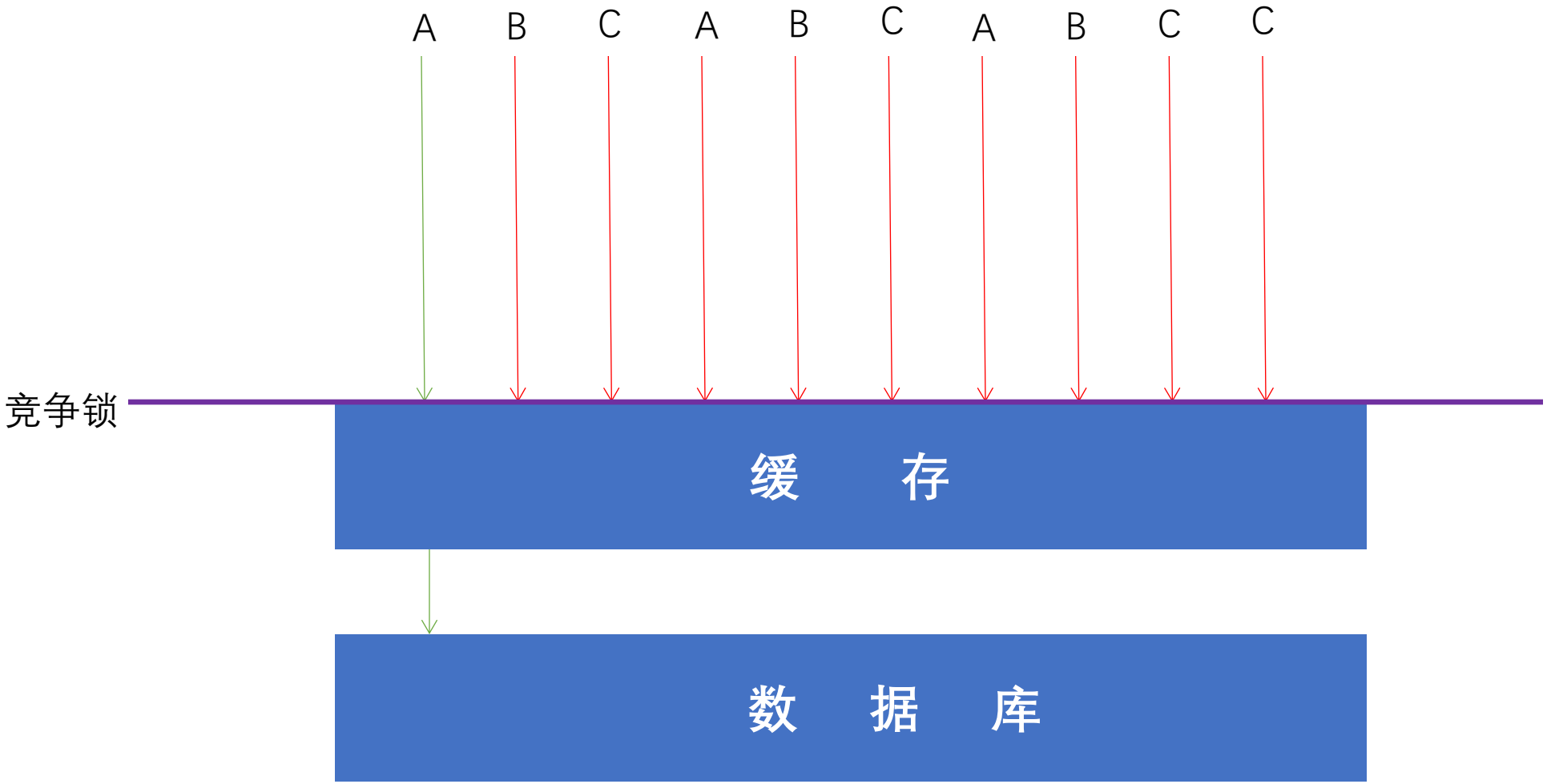


装饰器模式使用举例

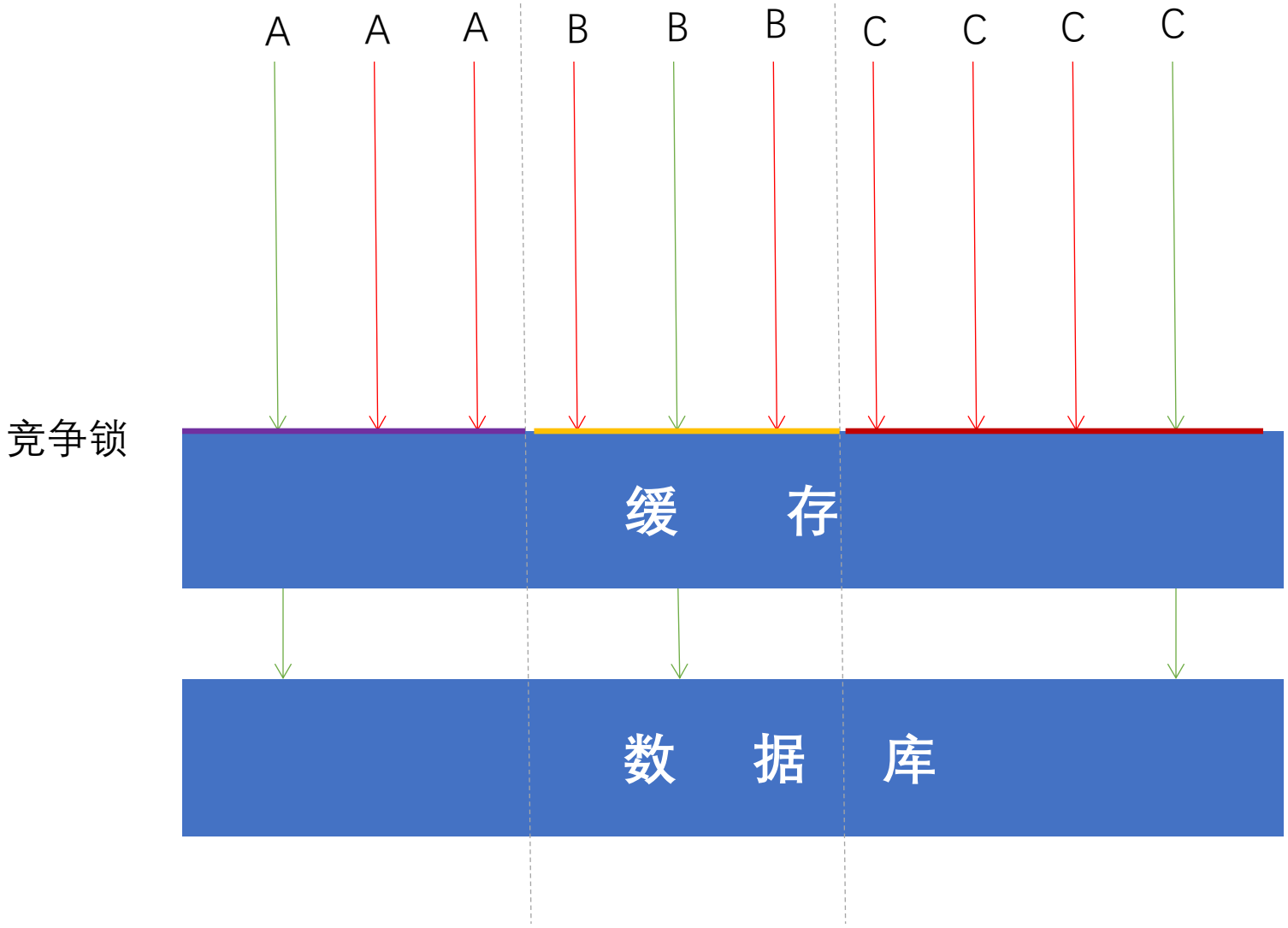


- ✓ **Cache**：Cache接口是缓存模块的核心接口，定义了缓存的基本操作；
- ✓ **PerpetualCache**：在缓存模块中扮演ConcreteComponent角色，使用HashMap来实现cache的相关操作；
- ✓ **BlockingCache**：阻塞版本的缓存装饰器，保证只有一个线程到数据库去查找指定的key对应的数据；

锁粒度的问题 粗粒度锁

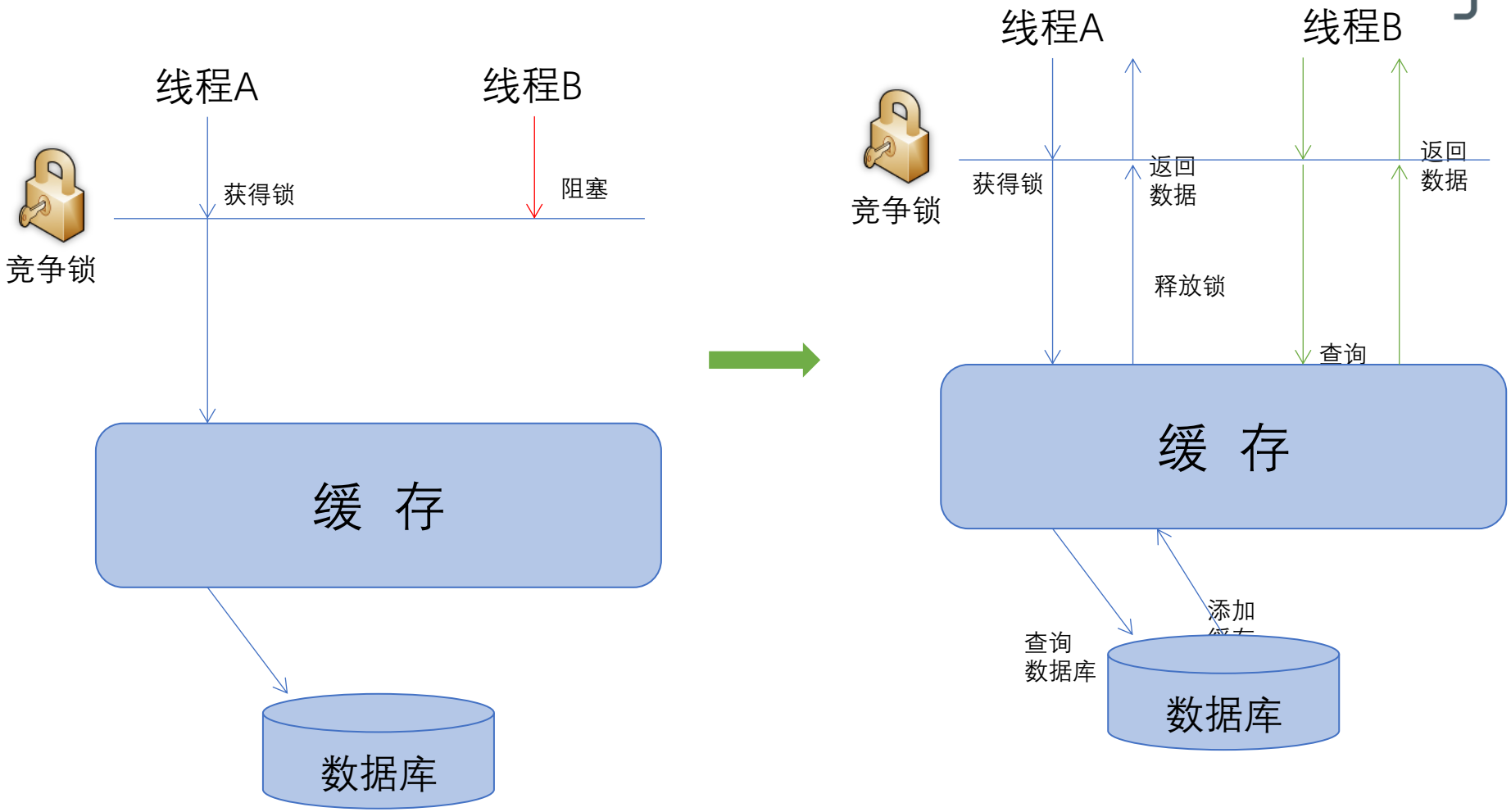


锁粒度的问题 细粒度锁(按key)



缓存装饰器解读

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



CacheKey解读

- Mybatis中涉及到动态SQL的原因，缓存项的key不能仅仅通过一个String来表示，所以通过CacheKey来封装缓存的Key值，CacheKey可以封装多个影响缓存项的因素；判断两个CacheKey是否相同关键是比较两个对象的hash值是否一致；

- 构成CacheKey的对象
  - ✓ mappedStatment的id
  - ✓ 指定查询结果集的范围（分页信息）
  - ✓ 查询所使用的SQL语句
  - ✓ 用户传递给SQL语句的实际参数值

- 重点解读方法
  - ✓ update(Object obj)
  - ✓ equals(Object obj)

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

反射模块分析

反射过程分析
反射核心类

orm框架查询数据过程

反射的核心类

- **ObjectFactory：**MyBatis每次创建结果对象的新实例时，它都会使用对象工厂（ObjectFactory）去构建POJO；
- **ReflectorFactory：**创建Reflector的工厂类，Reflector是mybatis反射模块的基础，每个Reflector对象都对应一个类，在其中缓存了反射操作所需要的类元信息；
- **ObjectWrapper：**对对象的包装，抽象了对象的属性信息，他定义了一系列查询对象属性信息的方法，以及更新属性的方法；
- **ObjectWrapperFactory：** ObjectWrapper 的工厂类，用于创建ObjectWrapper ；

反射的核心类

- **MetaObject**：封装了对象元信息，包装了mybatis中五个核心的反射类。也是提供给外部使用的反射工具类，可以利用它可以读取或者修改对象的属性信息；