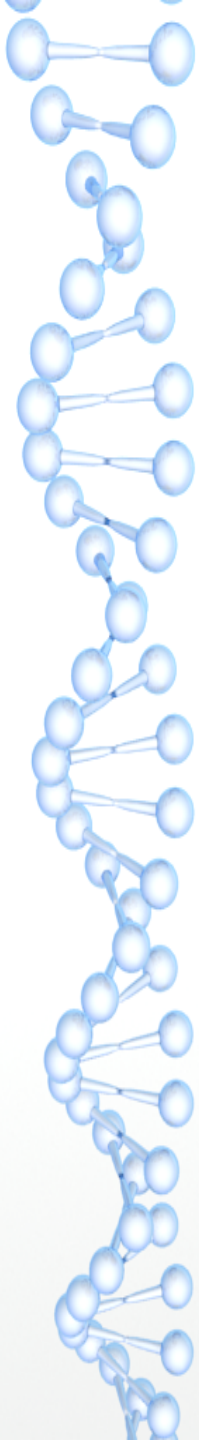


微服务架构下

CDI在领域驱动设计中的精妙应用

张建锋 Jeff Zhang
永源中间件(千岛源)
weibo.com/findapple

日程

- DDD用Java语言的实现设计思路
- 应用CDI规范定义对象
- DDD开发框架的实现介绍

CDI - JavaEE规范之一

- 依赖注入 Dependency Injection
- 松耦合，强类型 Lose couping, strong typing
- 上下文管理 Context management
- 拦截器和装饰器 Interceptors and decorators
- 事件总线 Event bus
- 扩展 Extensions

CDI - 发展过程

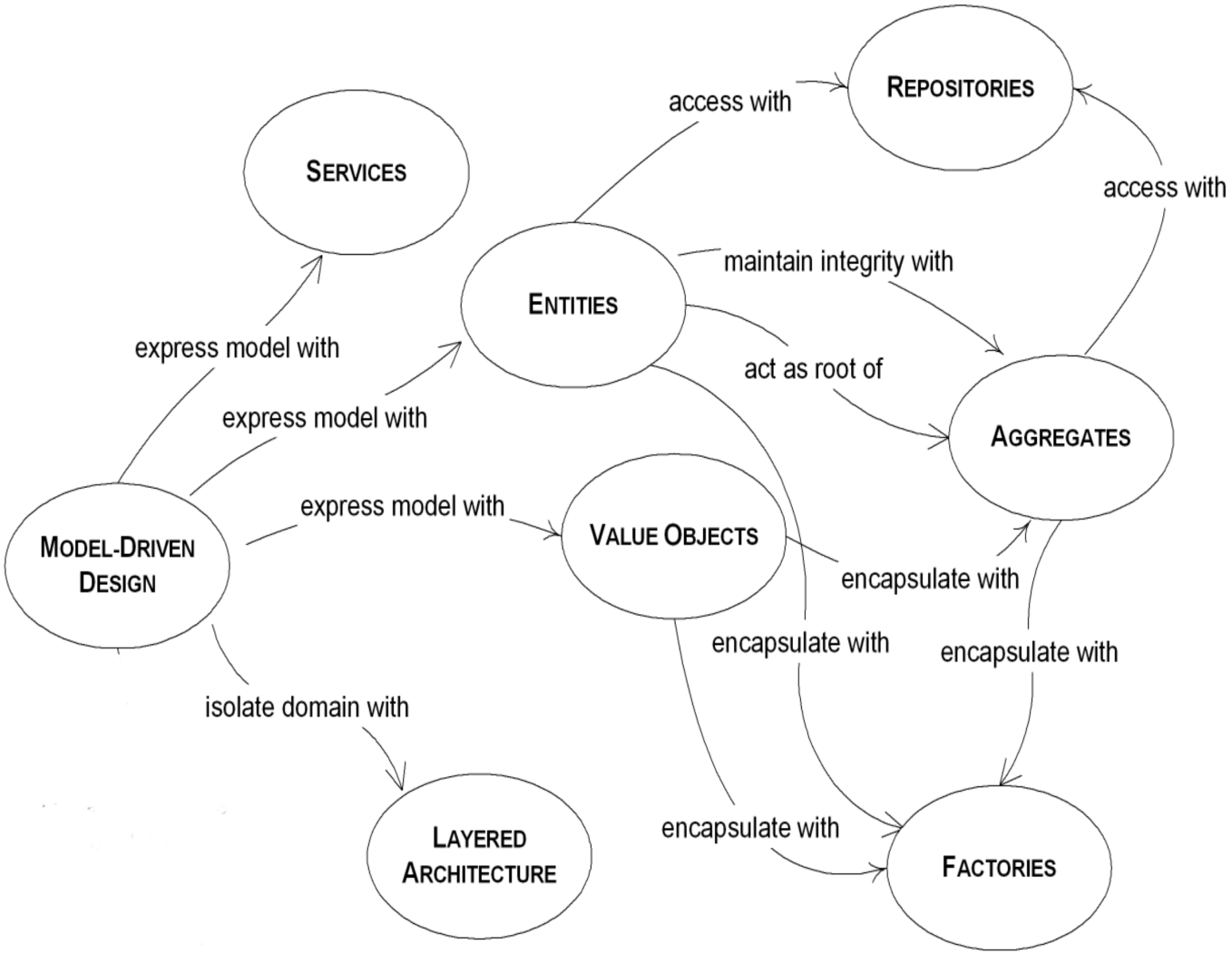
- 源出于 Seam 框架
  - Spring / Guice 灵感
- JSR 299 (Web beans/CDI 1.0, Java EE 6)
- JSR 346 (CDI 1.1, Java EE 7)
  - Weld (参考实现)
  - Apache DeltaSpike
- JSR 365 (CDI 2.0, Java EE 8)

CDI代码范例

```
public class WeldTeam extends OpenSourceCommunity {

  @Inject
  @AwesomeNews
  Event<String> event;

  public void release() {
    // Fire asynchronously so that we don't need to wait for observer notification b
efore we start celebrating!
    event.fireAsync("CDI 1.2 is dead, long live CDI 2.0!");
    celebrate();
  }

}
```

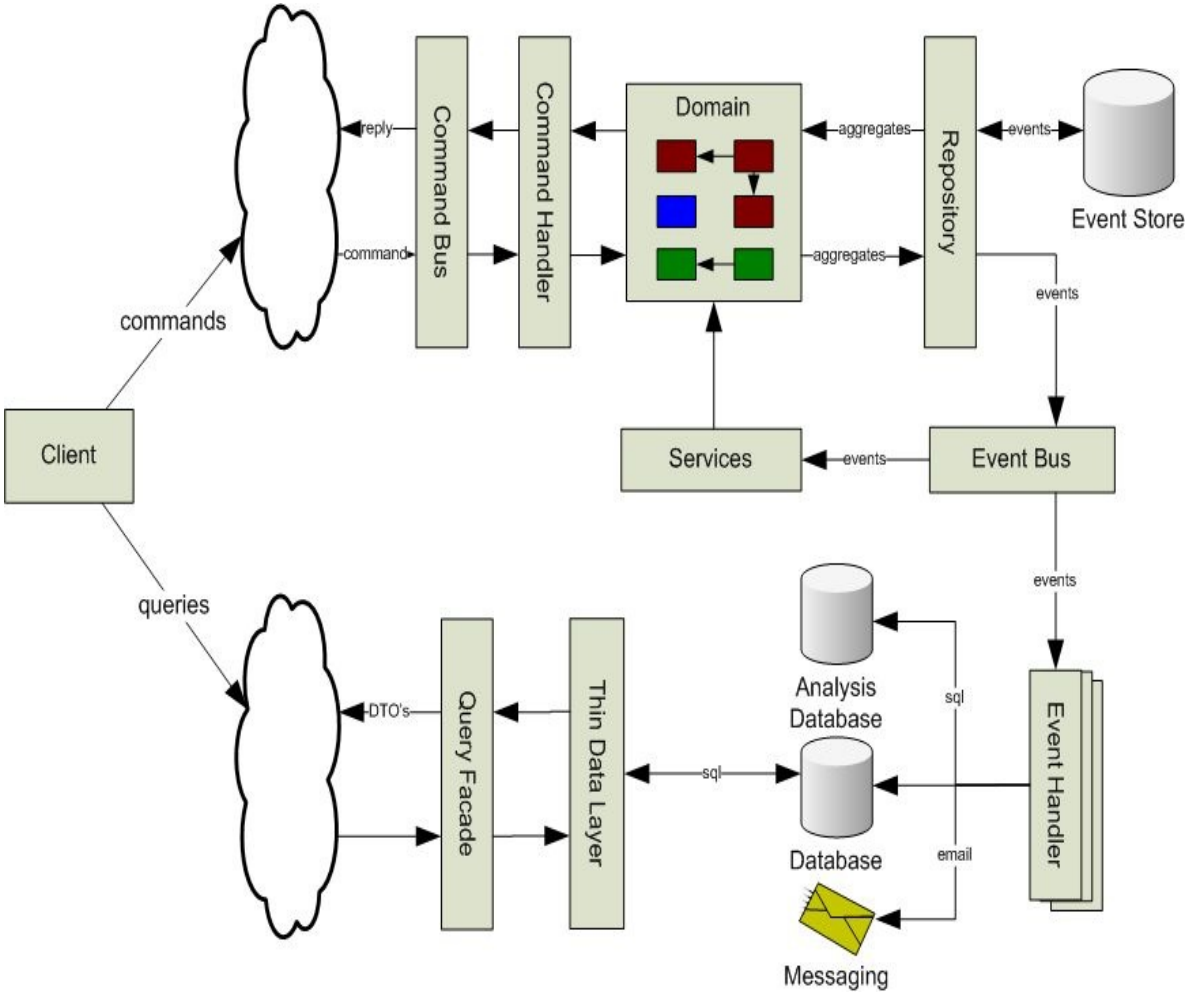
DDD组成

- 领域概念
- 理论性强
- 如何落地?



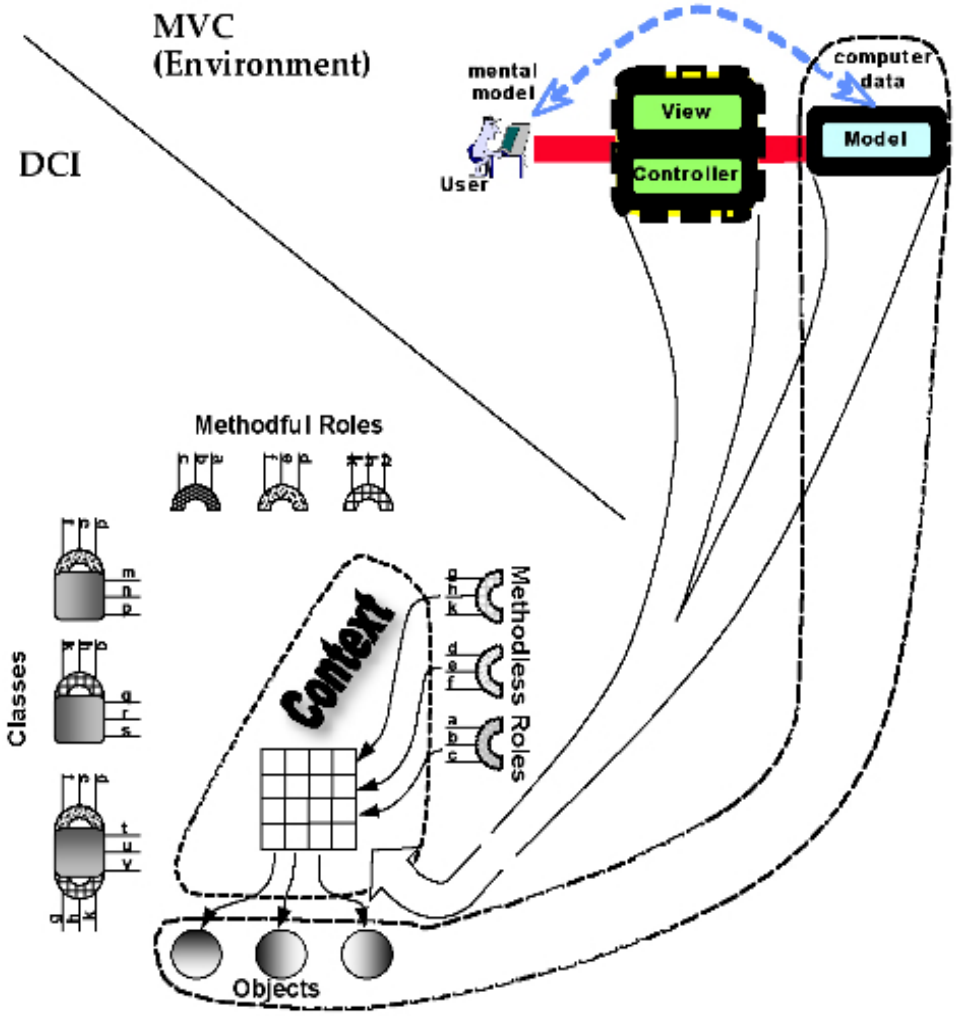
CQRS架构

- Command Query Responsibility Segregation
- 读写分离
- 事件机制解偶
- 适用于复杂应用
- 分布式系统



DCI上下文模型

- Data, Interactions, Context
- 数据在不同的上下文中有不同的行为方式
- 语言和框架的设计 Mixin,Trait



CDI规范技术说明一

- Inject and Qualifiers
- 注入和限定
-
- 组织成一张对象网络

```
public class BookBean implements Serializable {

   @Inject @ThirteenDigits
   private NumberGenerator numberGenerator;

   @Inject
   private ItemService itemService;


   // ...
}
```

CDI规范技术说明二

- Scope
- 范围

- ApplicationScoped, SessionScoped, RequestScoped
- ConversationScoped

```
@Named
@ConversationScoped
@Transactional
public class BookBean implements Serializable {

   @Inject
   private Conversation conversation;

   public void update() {
      conversation.begin();
   }

   public void delete() {
      conversation.end();
   }
}
```

CDI规范技术说明三

- Event
- 事件

- 对象之间解偶
- 统一编程模型

```
public class BookBean implements Serializable {

   @Inject @Paper
   private Event<Book> boughtEvent;

   public void update() {
      boughtEvent.fire(book);
   }
}
```

```
public class InventoryService {

    private void observeBooks (@Observes @Paper Book book) {
        logger.info("Book recevied " + book.getTitle());
    }
}
```

CDI规范技术说明四

- BeanManager
- Bean管理器
- Extension
- 扩展

```
public class SecurityManagerExtension implements Extension {

    void afterBeanDiscovery(@Observes AfterBeanDiscovery abd, BeanManager bm) {

        //use this to read annotations of the class
        AnnotatedType<SecurityManager> at = bm.createAnnotatedType(SecurityManager.class);

        //use this to instantiate the class and inject dependencies
        final InjectionTarget<SecurityManager> it = bm.createInjectionTarget(at);

        abd.addBean( new Bean<SecurityManager>() {

            @Override
            public Class<?> getBeanClass() {
                return SecurityManager.class;
            }

            @Override
            public Set<InjectionPoint> getInjectionPoints() {
                return it.getInjectionPoints();
```

实体/值对象 Entity / Value Object

- Entity
- 有id定义的实体类
- 可以用JPA中 @Entity 来定义

- Value Object
- 和实体对比：没有id，不可变的类，存在多个值对象
- 使用Annotation进行描述

领域服务 Service

- 接口提供服务
- 不同的实现提供不同类别的服务
  - Alternative
- Stereotypes 多个Annotation叠加

```
@RequestScoped
@Transactional(requiresNew=true)
@Secure
@Named
@Stereotype
@Retention(RUNTIME)
@Target(TYPE)
public @interface Action {}
```

领域事件 Domain Event

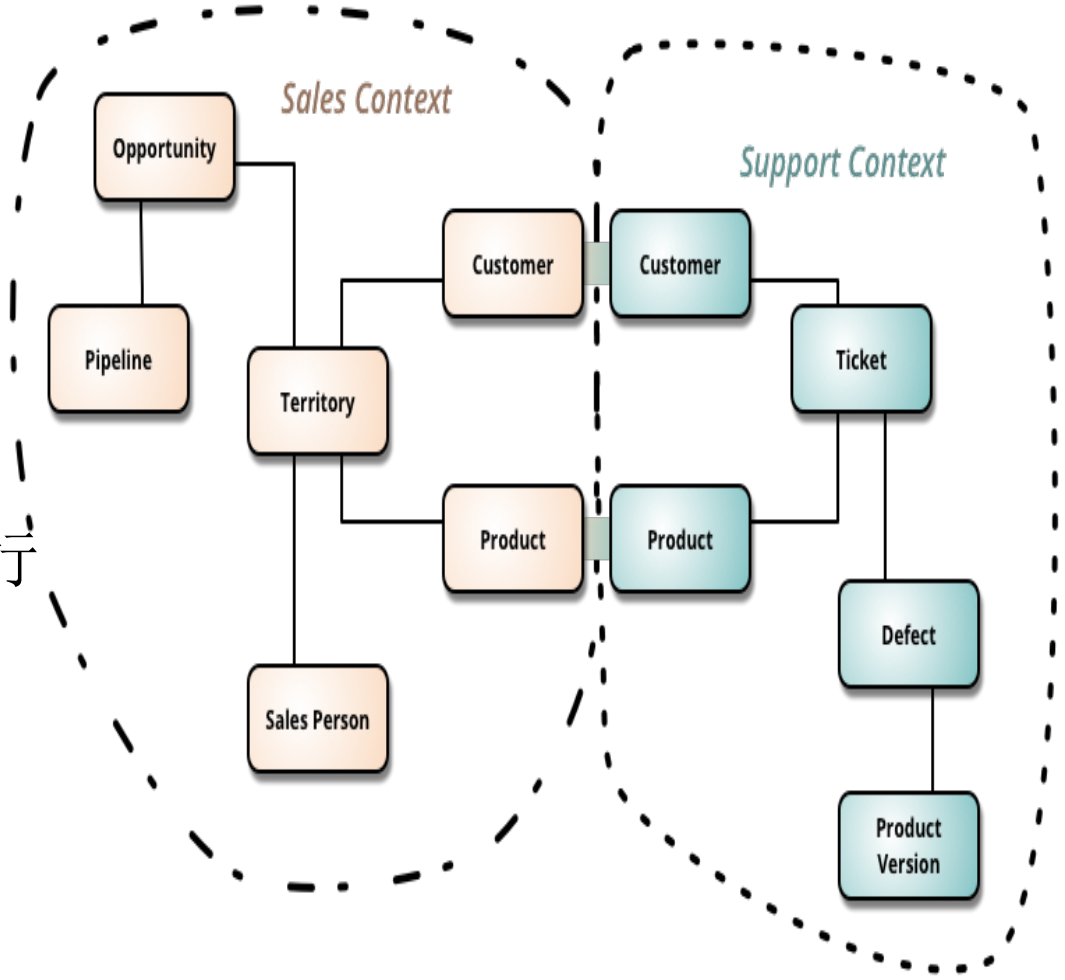
- Event事件
- 解偶，线程和对象生命期由容器控制
- 可以进行事件的筛选

```
import javax.enterprise.event.Event;

@Stateless
public class ProductManager {
   @PersistenceContext EntityManager em;
   @Inject @Any Event<Product> productEvent;

   public void delete(Product product) {
      em.delete(product);
      productEvent.select(new AnnotationLiteral<Deleted>(){}).fire(product);
   }

   public void persist(Product product) {
      em.persist(product);
      productEvent.select(new AnnotationLiteral<Created>(){}).fire(product);
   }
   ...
}
```

```
import javax.ejb.Singleton;

@ApplicationScoped @Singleton
public class Catalog {
   ...
   void addProduct(@Observes(during = AFTER_SUCCESS) @Created Product product) {
      products.add(product);
   }

   void removeProduct(@Observes(during = AFTER_SUCCESS) @Deleted Product product) {
      products.remove(product);
   }
}
```

聚合 Aggregate / 边界上下文 Bounded Context

- Aggregate 聚合根
- 微服务的边界
- 粒度再小，则没有必要，性能也会有较大损失
- 跨越上下文边界的再进行数据复制



工厂 Factory

- Factory 工厂类（设计模式）
- Producer
- 由容器负责对象的创建和生命期管理

```
import javax.enterprise.inject.Produces;

@SessionScoped
public class Preferences implements Serializable {
   private PaymentStrategyType paymentStrategy;
   ...
   @Produces @Preferred
   public PaymentStrategy getPaymentStrategy() {
       switch (paymentStrategy) {
           case CREDIT_CARD: return new CreditCardPaymentStrategy();
           case CHECK: return new CheckPaymentStrategy();
           case PAYPAL: return new PayPalPaymentStrategy();
           default: return null;
       }
   }
}
```

仓库Repository

- 数据存储
- 多种实现方式，比如
  - JPA Persistent
  - Spring Data
  - Apache DeltaSpike Data

```
@Repository
public interface PersonRepository extends EntityRepository<Person, Long>
{

    List<Person> findByAgeBetweenAndGender(int minAge, int maxAge, Gender gender);

    @Query("select p from Person p where p.ssn = ?1")
    Person findBySSN(String ssn);

    @Query(named=Person.BY_FULL_NAME)
    Person findByFullName(String firstName, String lastName);

}
```

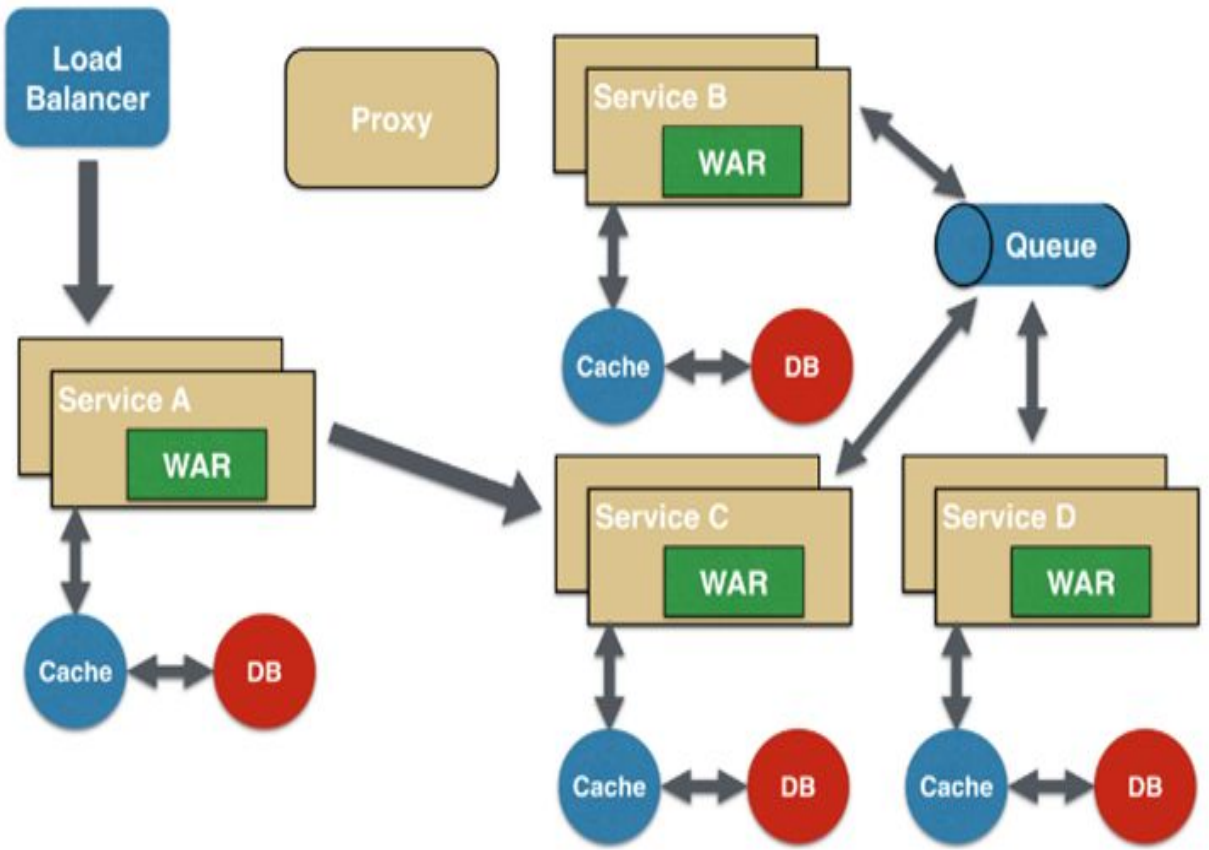
Side effect 副作用 / Composite 组合

- Intercept 拦截器
-
- Decorator装饰器
  - 对象包装，加强功能

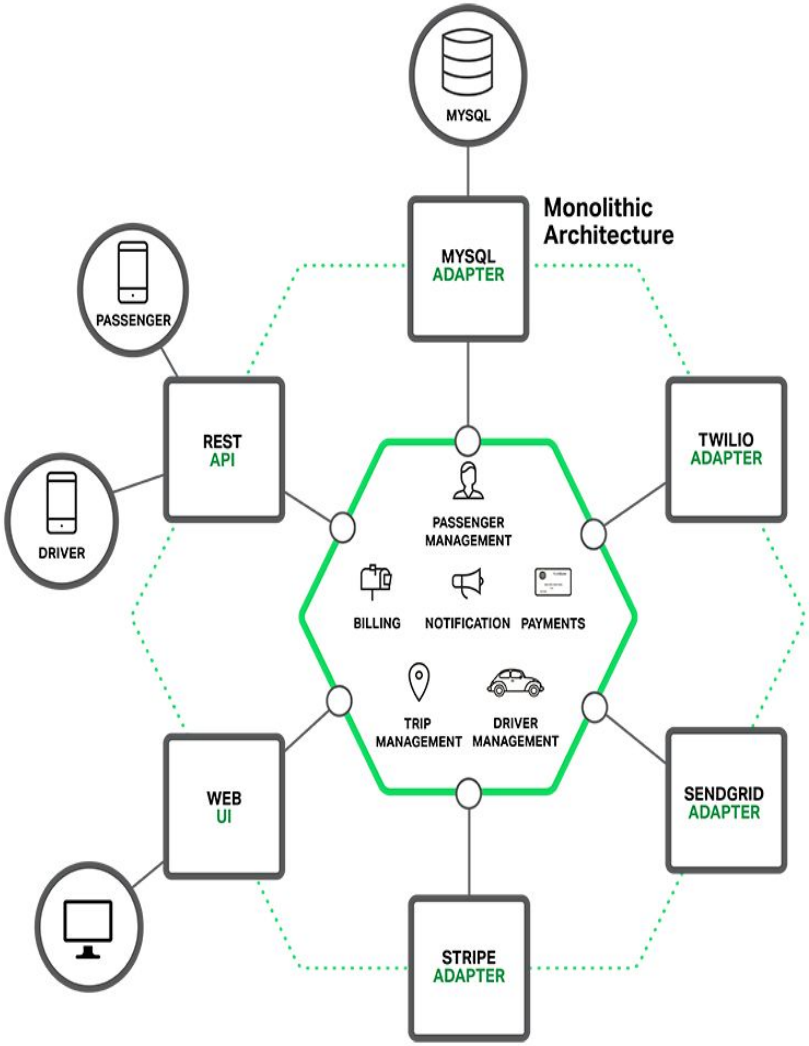
```
@Decorator
public abstract class LargeTransactionDecorator
      implements Account {
   @Inject @Delegate @Any Account account;

   @PersistenceContext EntityManager em;

   public void withdraw(BigDecimal amount) {
      account.withdraw(amount);
      if ( amount.compareTo(LARGE_AMOUNT)>0 ) {
         em.persist( new LoggedWithdrawl(amount) );
      }
   }

   public void deposit(BigDecimal amount);
      account.deposit(amount);
      if ( amount.compareTo(LARGE_AMOUNT)>0 ) {
         em.persist( new LoggedDeposit(amount) );
      }
   }
}
```

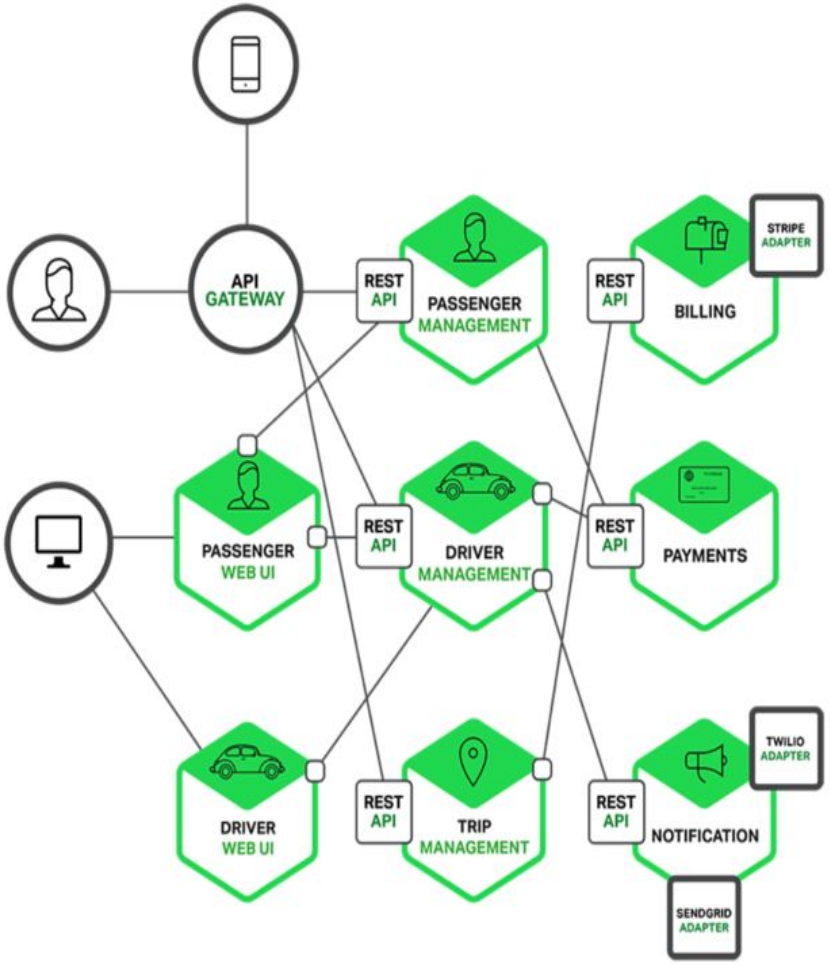
微服务架构

- 采用微服务架构后的变化
- 利用消息队列，缓存等
- 单个CDI应用程序进程



架构六边形





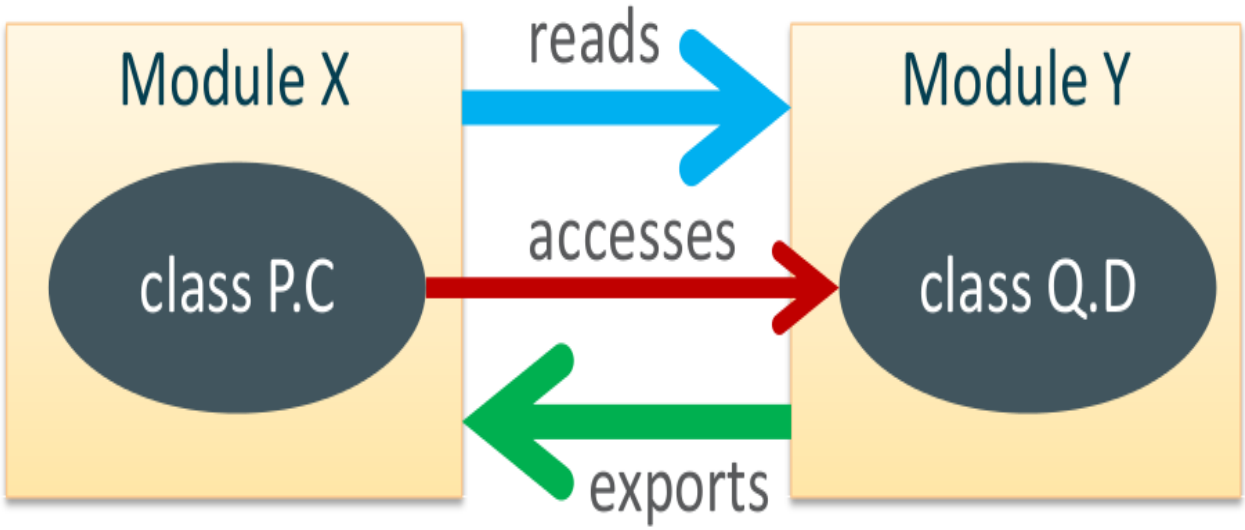
对比Spring框架

- Spring Framework (Data, Security) <=> CDI (Weld, Apache DeltaSpike)
- Spring Boot <=> Wildfly Swarm (…)

- 授之于鱼，还是授之于渔
- 对于复杂的应用，以及大中型独立软件公司，维护自己的领域模型很有必要

模块

- Java 9 Module
- 应用于微服务中，更好的隔离上下文边界




```
module X {
   requires Y;
}
```

```
module Y {
   exports Q;
}
```

落地项目

- Wildfly Swarm
- Payara Micro
- KumuluzEE
- Hammock

Thanks & QA