**第一周学习笔记**：

2.10-2.14

本周主要学习方面是spring进阶：

结合前段时间在信贷系统的工作任务，进一步理解了各种配置，以及架构的设计思想。

Aop的理解：

AOP 即 Aspect Oriented Programming 面向切面编程在面向切面编程的思想里面，把功能分为核心业务功能，和周边功能。

所谓的核心业务，比如登陆，增加数据，删除数据都叫核心业务   
所谓的周边功能，比如性能统计，日志，事务管理等等

周边功能在Spring的面向切面编程AOP思想里，即被定义为切面   
在面向切面编程AOP的思想里面，核心业务功能和切面功能分别独立进行开发   
然后把切面功能和核心业务功能 "编织" 在一起，这就叫AOP

在统一开发平台的设计中，日志以及事务管理体现的十分明显。

对此作了很多相关的代码练习。

**为什么在做微服务设计的时候需要 DDD**

在DDD的学习中，感觉主要体现在讲各种功能以及相关实体设计为相关的领域以及各种相关的模块。整体来说就是分而治之；

分“主要考虑如何划分，治意味着分出来的每个部分要能够独立运行，且能够相互协作来完成整体目标。

在划分微服务时，需要考虑功能维度，质量维度和工程维度三个方面。而面对复杂的功能，需要采用 DDD 的方式来进行功能维度的划分。

**如何快速且深入的学习一门新技术**

先快速了解整体内容，宏观了解技术全貌：快速、跳跃性查资料，重点关注反复 而且结合前段时间的敏捷培训，感觉一定要进行刻意练习。

实战：先以实现 helloworld 为开端，接着不断丰富内容，每次实现一个小功能，

逐步积累案例库

考试：检验知识的掌握程度，通过做微服务案例来检验学习成果，进行技术整合，

防止遗漏，技术分析

**JVM 问题定位典型案例分析**

1）类加载死锁：线程Dump没有检测到死锁，有不少的线程Block在Class.forname()

执行 jstack -m <pid>

2)finalReference 堆积：finalize 被调用

3）堆外内存泄露：xmx 设置过大，在统一开发平台的设计中，也需要考虑相关内存的问题。

4）YGC 拉长：发现 ParNew GC 的时间不断拉长，StringTable主要记录了经过了String.interm后的String对象指针，SystemDictionary主要是记录了加载的类，两者都是hashTable结构，因此存在hash碰撞的风险。

**数据库事务的相关学习**

**由于本人在数据库方面感觉有些薄弱，对这方面需要着重学习**

虽然事务通过确保对数据的更改仅在事务被成功提交之后才成为永久性的，从而提供了一般的数据库一致性，但还是须要用户或应用程序来确保每个事务中执行的SQL操作序列始终会导致一致的数据库。

Serializable(串行化)：一个事务在执行过程中完全看不到其他事务对数据库所做的更新。

Repeatable Read(可重复读)：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，但是不能看到其他事务对已有记录的更新。

Read Commited（读已提交数据）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，而且能看到其他事务已经提交的对已有记录的更新。

Read Uncomitted（读未提交数据）：一个事务在执行过程中可以拷打其他事务没有提交的新插入的记录，而且能看到其他事务没有提交的对已有记录的更新。

隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也越大。对于多数应用程序，可以有优先考虑把数据库系统的隔离级别设为Read Commited，它能够避免脏读，而且具有较好的并发性能。尽管它会导致不可重复读、虚读和第二类丢失更新这些并发问题，在可能出现这类问题的个别场合，可以由应用程序采用悲观锁或乐观锁来控制。

当数据库系统采用read Commited隔离级别时，会导致不可重复读和第二类丢失更新的并发问题，可以在应用程序中采用悲观锁或乐观锁来避免这类问题。从应用程序的角度，锁可以分为以下几类：

A.悲观锁：指在应用程序中显示的为数据资源加锁。尽管能防止丢失更新和不可重复读这类并发问题，但是它会影响并发性能，因此应该谨慎地使用。

B.乐观锁：乐观锁假定当前事务操作数据资源时，不会与其他事务同时访问该数据资源，因此完全依靠数据库的隔离级别来自动管理锁的工作。应用程序采用版本控制手段来避免可能出现的并发问题。