第三周学习总结

首先是课程中的一些笔记

3.1 设计模式：使用设计模式优化排序包工具

1. 设计模式是一种可重复使用的解决方案，是达到框架设计各项原则所需的重要工具

2. 学习设计模式关键是要弄清楚这种设计模式是用来解决什么问题的

3. 以一个排序工具包的设计为例介绍了设计模式的应用。（不过个人感觉这个地方经验成分非常多，非一朝一夕能够掌握啊。。）

3.2 设计模式：Singleton单例模式和adaptor适配器模式

1. 关于单例模式：

    1.1 概念：一个类只产生一个实力

    1.2 性能方面的考虑：减少实例频繁创建和销毁带来的资源消耗

    1.3. 功能上的考虑，用户使用这个实例的时候，可以统一控制，比如打印机这个对象

2. 单例的构造和使用：

    2.1 饿汉模式：私有构造函数+公有静态的创建代码和获取函数

    2.2 懒汉模式：私有构造函数+公有获取函数，创建在第一次获取的时候

    2.3 尽量使用第一种，简单而且有效

    2.4 考虑到多线程环境，单例内尽量不要有状态

3. 适配器模式：用于衔接方法和接口的定义

    3.1 适配器可以分为对象适配器和类适配器，由于继承带来的问题和不便，应该优先使用对象适配器

3.3 junit中的设计模式（上）

1. 父类TestCase依次调用setUp，textXyz，tearDown，然后子类中实现这三个函数即可

2. 模版方法模式：

    2.1 父类中定义好方法的骨架，子类中进行具体的实现。（这个我自己平时在工作中使用的那些protect方法都算）。

    2.2 HTTP servlet拆出来成为模板的继承以后调用链看起来就非常清晰了

3. 策略模式：应用程序针对策略接口进行编程，真正使用的时候进行注入策略实现类

3.4 junit中的设计模式（下）

1. 组合模式：树形结构。以testSuit为例，和TestCase一样实现test函数，内部则是循环调用各个子test。

2. window视窗空间也应用了组合模式，所有的组件都实现底层接口Component，拥有draw方法，其中Container是一个复合Component，内部有很多其他的Component，Container本身也可以draw

3. 装饰器模式：这个有点像aop。关键是要装饰的类和想要实现的类都实现同一个接口，这样可以不断进行装饰。比如RepeatedTest对TestCase进行装饰，实现了循环10000次的逻辑

3.5 spring中的设计模式

1. 依赖注入：A依赖B，但是A不创建B，而是由外部创建B以后注入到A里面来
2. Spring MVC设计模式

3.6 设计模式案例：intel大数据SQL引擎&Panthera设计模式

视频中描述了一个例子，用设计模式把不同的语法点的转换和生成工作做了隔离和抽象，最终变成了几十个transformer和几百个generator，用装饰器模式进行组合，每个transformer和generator相互之间完全不会相互影响，不仅简化了代码，减少了代码量，还促进了团队合作。

个人思考

李老师在课程中说了，学好设计模式，关键还是要理解每一种设计模式都是为什么被设计出来的。其本质其实是为了解决某一类重复的问题。所以，下面用自己的话对于课程中说到的每一种设计模式做一个用法和场景的总结：

1. 单例模式：一个类在全局只创建一个对象。从功能上考虑，这个有利于统筹管理某一类逻辑，比如与db某个表进行交互的某一个业务实体，除了通过这个对象没有其他方式利用与db的这个表交互。从性能上考虑，避免了频繁的类的创建和删除行为，提高了性能。单例常用于多线程中，因此最好不要存储状态。
2. 适配器模式：主要是在实现某个接口功能时，如果发现如果已经有其他的类实现了类似的功能，但是接口却不吻合时，可以用组合模式把其他类的功能组合进来。组合模式分为类适配以及对象适配，由于类适配要承担继承所带来的不方便性，所以优先还是选择对象适配。
3. 模版方法模式：当某些业务或者流程有通用的执行步骤的时候，可以把这些执行步骤提取出来，实现在基类中，然后把不同的部分抽象成一个个的protected方法，由子类去实现或者覆盖，以实现类似功能的扩展
4. 策略模式：和模版方法类似，也是在一个逻辑模块里面先开发完基础的通用流程。只不过业务是通过一个个的接口抽象出来，然后实际使用的时候，这些接口可以注入具体的实现类，并且这些实现类可以灵活替换
5. 组合模式：当有很多类或者对象需要实现同一个接口，并且它们之间通过叠加组合实现一个复杂功能时，就需要使用组合模式，组合模式中往往有一个执行suit功能的类存在，通过suit实现类似于一棵树的复杂组合，最终完成一连串的功能
6. 装饰器模式：当很多功能实现了统一个接口，并且某些功能是在某一个已有的功能上的叠加时，可以用装饰器模式包装已有接口，复用已有逻辑，从而简化代码，并使得逻辑清晰
7. 依赖注入：不同的功能模块可以独立地进行实现和初始化，并且在使用时非常方便地有框架进行组合

8. MVC：Model，View和Controller，通过MVC的方式可以把业务逻辑和视图的实现逻辑分开，也就是，一边只管画图，一边只管计算。Model是业务逻辑部分，View是视图部分，它们通过controller进行衔接分发，独立但是又有机地组合完成任务。