后台技术选型

- 后台采取***单机分层模型***，通过Restful风格的RPC接口实现与前端的解耦。与前端交互采用JSON作为序列化的标准，利用开源的JSON序列化和反序列化工具实现
- 后台采用了Spring MVC + Spring + MyBatis的展示层、业务逻辑层、持久层分层的架构。
- 数据库选择了开源的Mysql，支持主流的SQL标准，索引、内外连接等基本需求都能满足，支持单机事务。
- 数据库、后台系统均采取UTF-8编码，避免因为编码问题造成中文不兼容的问题
- 日志系统采用log4j（应该采用slf4j抽象接口层+log4j实现）

Spring MVC

优势

- 开源框架，有开源社区提供的丰富的文档，方便问题的排查和解决。
- 支持Resultful的RPC接口实现，与Spring的兼容性好，提供了简化配置的注解，开发效率高，对于开源工具的集成简单
- filter+controller的处理机制，能通过责任链的机制，利用filter对请求统一进行过滤，解决跨域问题。
- 采用经典的MVC设计模式
- 兼容Java的官方注解标准

缺点

- 没有集成Velocity等模板引擎，不方便模板与数据的解耦。
- View层的架构只是简单的界面，而没有区分control(组件),layout(容器),screen(主体)，造成代码冗余为维护带来困难

选型原因

相对于WebX学习成本较低，而且文档较丰富，性价比较高;相对于Structs2，配置简洁，与Spring配合使用粘合成本更低

Spring

优势

- 依赖注入，不用再很麻烦的写一堆工厂模式的代码
- 进行类间解耦，方便使用Mockito+Junit进行单元测试，因为只需要Mock注入的对象即可，不需要修改代码
- 能够利用注解简化配置代码，对业务逻辑没有侵入性
- 能够提供AOP的支持，方便书写切面代码。
- 对ibatis等持久层框架有较好的封装
- 开源框架，文档丰富，方便解决问题

缺陷

- 采用Java的反射机制实现，是运行时的动态代理，相对于编译器的注入技术，牺牲了效率

Mybatis

优势

- 轻量持久层框架
- 相对于Hibernate学习成本更低，而且更轻量，可以自己写SQL，便于优化
- 对数据源进行了封装

缺陷

没有提供分库分表等大数据场景下的解决方案

Log4j

优势

- 轻量
- 配置简单
- 开源解决方案，文档丰富

缺点

不如LogBack效率高，接口层面的缺陷可以通过slf4j改善

Mockito+Junit

- 方便单元测试和数据库测试
- Mockito支持Mock、Stub、Spy，提供了一站式解决方案，反模式的架构设计很精巧
- 与Spring配合起来，写单测很顺滑