红领巾项目文档

王杰 141250138

王家玮 141250136

田琦 141250126

王新宇 141250141

王泽霖 141250144

熊凯奇 141250156

杨关 141250166

2016-3-22

目录：

1. 系统主要功能点

1） 上传资源：用户通过系统将自己所拥有的资源上传到互联网上,方便其他对该资源有需求的用户下载。
2） 下载资源：用户通过系统搜索自己想要的资源并从系统上下载资源。
3） 用户管理：系统管理员可以对用户进行权限更改，可以通过系统注册普通用户，普通用户可以修改个人账号信息。
4） 权限管理：用户对访问的资源有权限限制。
5） 资源管理：系统将用户上传的资源进行分类管理。
6） 评论管理：用户可以对其他人上传的资源进行评论和打分，方便其他用户了解该资源。
7） 统计分析：统计每个用户的上传和下载的资源量，资源类型，积分的使用情况，生成统计分析图表。

2. P2P 架构设计与评估

2.1 ADD 过程

2.1.1 第一次迭代

- 步骤一

  获得样本输入，筛选掉不必要的需求，列出 ASR：

性能：场景 1:用户使用客户端进行下载

场景组成部分	可能的值
源	用户
刺激	用户使用客户端进行资源下载
制品	服务器、客户端中涉及到网络访问的模块
环境	系统运行时
响应	良好的下载速度
响应度量	服务器带宽至少为 2Gbps 系统可以保证用户同时下载至少 3 个资源 系统可以保证同时响应 10000 名用户的同时下载 用户下载总速度不能低于 500KB/s 客户端请求下载的缓冲时间应不大于 2s

性能：场景 2:用户使用客户端进行上传

场景组成部分	可能的值
源	用户
刺激	用户使用客户端进行资源上传
制品	服务器、客户端中涉及到网络访问的模块
环境	系统运行时
响应	稳定的上传速度
响应度量	服务器带宽至少为 1Gbps 系统支持批量上传，即至少可以保证用户同时上传 100 个资源 用户上传速度不能低于 1M B/s 客户端请求上传的缓冲时间应不大于 2s

性能：场景 3:用户正常使用客户端进

场景组成部分	可能的值
源	用户
刺激	用户正常使用客户端查看资源
制品	客户端
环境	系统运行时
响应	良好的用户体验
响应度量	客户端中用户操作的响应时间应小于 1s 系统可以保证同时响应 10000 名用户的操作

安全性：场景 4:客户进行上传文件

场景组成部分	可能的值
源	用户
刺激	用户使用客户端进行破坏性病毒或木马资源上传
制品	系统上传服务，系统的数据
环境	系统联网状态下运行时
响应	识别用户上传文件是否含有恶意病毒或信息 判断用户上传文件是否包含违法资源或淫秽资源 文件存储在服务器中需要保持一个不可执行的状态 数据库遭受到侵害时需要保证数据可以安全修复
响应度量	成功识别恶意上传率不得低于 99% 成功判断文件中不包含违法资源和淫秽信息率不低于 95%

	服务器出现问题数据的恢复可以在 3 人日内完成

安全性：场景 5:未经授权的访问文件

场景组成部分	可能的值
源	用户或者黑客等涉嫌攻击系统者
刺激	未经过正规渠道进行文件的上传和下载
制品	服务器、客户端中涉及到网络访问的模块
环境	系统联网状态运行时
响应	对用户的身份进行验证 阻止对服务器和数据的访问
响应度量	当有外界使用其他方式进行文件上传和下载的成功率低于 0.1% ； 任何对服务器和数据的访问需要经过有效的验证 在拒绝提供服务的情况下，依然能进行数据的修改和访问的概率低于 0.01%

安全性：场景 6:经过授权的下载文件

场景组成部分	可能的值
源	用户
刺激	用户经过授权后进行文件的下载
制品	服务器、客户端中的文件访问
环境	系统联网状态运行时
响应	对用户的身份进行验证 保证下载链接的安全性和可靠性，对传输的内容头部进行加密
响应度量	对用户身份验证失败的概率低于 0.05% 在资源有效的情况下下载失败的概率低于 0.5% 下载链接的可靠性和安全给用户主机造成病毒感染的概率低于 0.001%

可伸缩性：场景 7:用户使用上传和下载数量加剧

场景组成部分	可能的值
源	用户
刺激	用户大规模频繁的使用上传和下载功能进行资源的传输
制品	客户端、服务器
环境	系统正常运行时
响应	稳定的下载速度和上传速度

响应度量	可支持的并发上传数不得低于 30000 下载造成影响的用户数量不得高于 5% 服务器崩溃或卡死的概率低于 0.001%

易用性：场景 8:正常操作

场景组成部分	可能的值
源	用户
刺激	用户执行一项操作，如上传文件，下载文件，用户登录等操作
制品	系统
环境	系统运行时
响应	正确引导用户操作步骤 高效且成功地完成该项操作 从错误中及时恢复，防止数据丢失
响应度量	完成单项细节操作的时间不多于 2 分钟 操作成功率高于 99% 错误发生后恢复到重新正常使用所需时间少于 10 分钟 错误发生后数据丢失率低于 1%

可扩展性：场景 9 需要在系统增加一个新的功能

场景组成部分	可能的值
源	开发人员
刺激	需要在系统增加一个新的功能
制品	新的功能模块
环境	系统在构建中或系统已经上线运行
响应	编写新功能代码，修改原有代码 测试代码 发布维护变更
响应度量	代码复用率达到 30% 维护时间不应该超过 3 小时 维护所影响到的源代码量不能超过 2%

可用性：场景 10 下载过程中网络中断

场景组成部分	可能的值

源	网络
刺激	网络状况不稳定或网络中断
制品	下载过程文件
环境	系统在无网络环境下运行
响应	记录下当前下载进度 保存临时文件 暂停下载告知用户下载暂停
响应度量	下载进度记录在 0.5 秒内保存 临时文件在 0.5 秒内保存 1 秒内通知用户暂停下载

可用性：场景 11 系统服务器崩溃

场景组成部分	**可能的值**
源	系统
刺激	系统服务器崩溃
制品	服务器提供的服务
环境	服务器在故障情况下运行
响应	查明系统故障原因 排查系统故障 无法解决故障则重启服务器的服务 记录故障日志 告知访问服务器的用户系统暂时无法使用
响应度量	系统正常运行时间占 99% 系统可以自行排查的故障占 90% 查明故障原因不超过 2 分钟 解决故障时间不超过 10 分钟

可维护性：场景 12 客户端升级

场景组成部分	**可能的值**
源	开发者
刺激	客户端需要更新
制品	更新后的客户端
环境	系统已上线运行
响应	修改或添加客户端代码 测试客户端代码

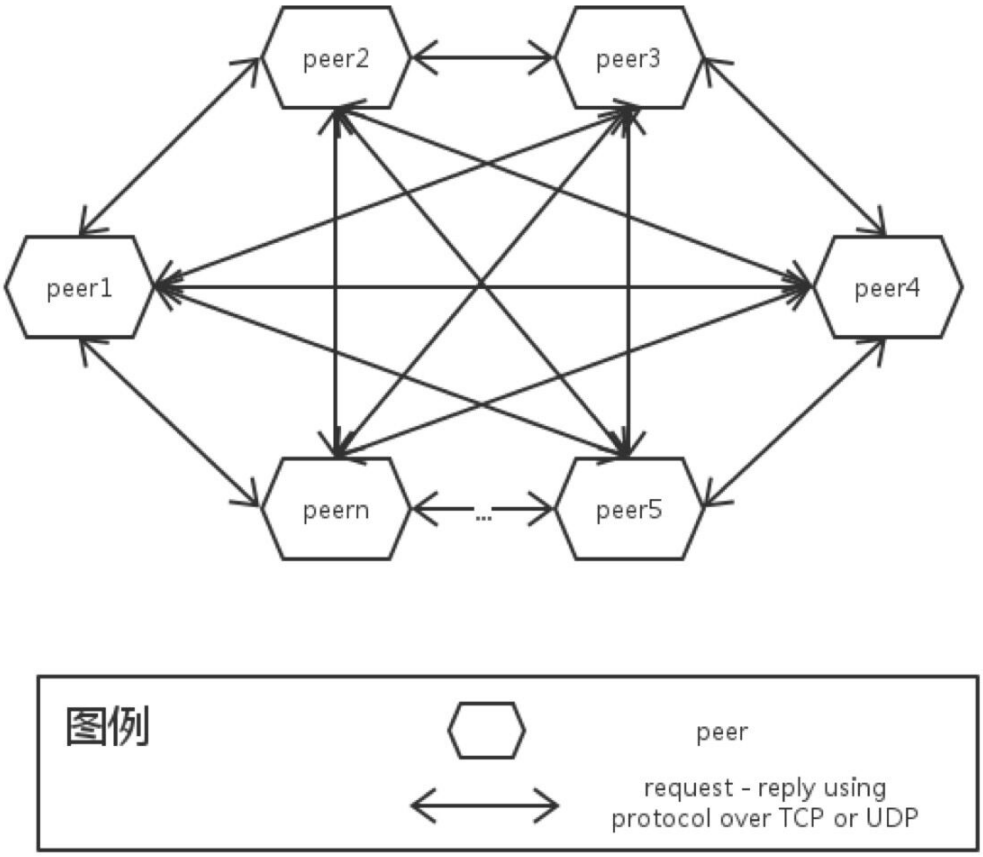
	发布更新内容
响应度量	改动的接口不超过原来的 5% 测试用例通过率达到 99% 维护时间不超过 3 天

可维护性：场景 13 系统出现 bug

场景组成部分	**可能的值**
源	开发者
刺激	开发者发现正在运行的系统存在 bug
制品	修复 bug 后的系统
环境	系统正在运行
响应	修改系统出现的 bug 测试修改后代码 部署新的代码，并重启
响应度量	修正 bug 的时间不超过 3 小时 从发现 bug 到部署成功不超过一天

- 步骤二

  对整个系统进行分解，分为对等的各个节点。



2.1.2 第二次迭代

- 步骤一

  第二次迭代可以不用此步骤。

- 步骤二

对单个节点进行分解，并分为上传模块、下载模块、权限管理模块、资源管理模块，用户管理模块

2.1.3 第三次迭代

- 步骤一

  第三次迭代可以不用此步骤。

- 步骤二

  在此次迭代中针对上传模块进行分解。

- 步骤三

  确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 2：用户使用客户端进行上传	高	高
2	场景 7：用户使用上传和下载数量加剧	中	高
3	场景 11：系统服务器崩溃	中	中
4	场景 12 客户端升级	中	中

- 步骤四

选择满足架构驱动因素的架构模式。

1） 针对性能相关的架构驱动因素

i. 设计关注点

设计关注点	子关注点
上传速度体验	上传速度快且稳定
	可支持较多用户同时上传

ii. 各子关注点的候选架构模式

a） 上传速度快且稳定

#	模式名称	开发时间开销	成本开销	带宽要求
1	小包传输	中	中	低
2	动态路由优化	中	中	中

✧ 选择模式：小包传输，动态路由优化

✧ 选择理由：小包传输和动态路由优化两者在开发时间与成本开销上基本一样，且没有显著缺点，因此都作为备选方案。

b） 可支持较多用户同时上传

#	模式名称	开发时间开销	成本开销	系统资源开销
1	CDN （内容分发网络）	中	中	中
2	使用第三方软件管理负载均衡	高	中	高
3	使用硬件管理负载均衡	低	高	低

✧ 选择模式：CDN （内容分发网络）

✧ 选择理由：硬件管理负载均衡成本过高，不列入选择。软件管理负载均衡要考虑兼容性等一系列问题，开发时间过高，不列入选择。

iii. 候选架构模式综合评估

	上传**速度快且稳定**		**可支持较多用户同时**上传	
	优点	缺点	优点	缺点
小包传输	带宽要求低	网络利用率低，有额外资源开销	无显著影响	
动态路由优化	效率高	消耗额外计算资源	无显著影响	容易造成拥堵
CDN	减少源站点负载，节省网络分	无显著缺点	通过提高网站响应速度，改善用户体验，增强用户满意度和粘合度；	

	布式框架的支出成本和维护成本		轻松应对突发流量，随时开展网络推广

✧ 评估结果：动态路由优化在性能提升方面比小包传输更有优势；CDN 是支持较多用户同时上传的备选架构模式，且对上传速度快且稳定也有一定优点，因此作为最终决策。

2） 针对可用性相关的架构驱动因素

i. 设计关注点

设计关注点	子关注点
上传体验	上传稳定不中断
	上传中断可以恢复继续上传

ii. 各子关注点的候选架构模式

a） 上传稳定不中断

#	模式名称	稳定性	网络依赖
1	采用 HAPROXY	较低	低
2	采用 LVS 集群	高	高

✧ 选择模式：LVS 集群

✧ 选择理由：HAPROXY 稳定性较低，且不支持虚拟主机，因此不作为选择方案 。

b） 上传中断可以恢复继续上传

#	模式名称	系统资源开销	失效恢复时间
1	系统存储临时文件	中	短
2	云服务存储	低	中

✧ 选择模式：系统存储临时文件

✧ 选择理由：云服务存储系统资源开销小，但恢复时间长而其不能保证云服务存储的稳定性与性能，所以不选取；系统存储临时文件开销略大，但实效恢复时间短且稳定，因此作为备选方案。

iii. 候选架构模式综合评估

	上传稳定不中断		上传中断可以恢复继续上传	
	优点	缺点	优点	缺点
采用 LVS 集群	性能好，接近硬件设备的网络吞吐和连接负载能力；其 DB 模式支持通过广域网进行负载均衡。	配置困难；操作人要求高；	性能好；工作稳定	配置困难；操作人要求高；
系统存储临时文件	无显著影响		能快速恢复下载	需要一定资源开销

✧ 评估结果：系统存储临时文件是上传中断可以恢复继续上传这一子关注点的备选架构模

式，且对上传稳定不中断，因此作为最终决策。LVS 集群稳定，并且对上传中断可以恢复继续上传也无显著缺点，因此作为最终决策。

2.1.4 第四次迭代

- 步骤一

  第四次迭代可以不用此步骤。

- 步骤二

  在此次迭代中针对下载模块进行分解。

- 步骤三

  确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 1：用户使用客户端进行下载	高	中
2	场景 5：未经授权的访问文件	中	中
3	场景 6：用户进过授权的下载文件	高	中
4	场景 7：用户使用上传和下载数量加剧	高	高
5	场景 8：正常操作	中	中
6	场景 10：下载过程中网络中断	高	中

-

  选择满足架构驱动因素的架构模式。

1） 针对可维护性相关的架构驱动因素

3） 针对性能相关的架构驱动因素

iii. 设计关注点

设计关注点	子关注点
资源下载体验	对未经授权的用户拒绝指定资源下载
	可支持多用户同时下载文件资源
	资源下载速度达到 500kb/s.
	支持无网络环境下对资源进行缓存

iv. 各子关注点的候选架构模式

c） 对未经授权的用户拒绝指定资源下载

#	模式名称	开发时间开销	成本开销	系统资源开销
1	数据库表设计	中	低	高
2	拦截器模式	中	中	中

✧ 选择理由：通过数据库中对用户表和文件表的等级进行设置系统资源消耗高

拦截器模式实现简单，筛选能力强，易于管理。

d） 可支持多用户同时下载文件资源

#	模式名称	开发时间开销	成本开销	系统资源开销
1	CDN （内容分发网络）	高	中	中
2	使用 BT 对等协议	中	中	中
3	使用硬件管理负载平衡	低	高	中
4	资源随机存储	中	中	低

选择理由：硬件管理负载平衡，成本也高，资源随机存储管理起来比较复杂，实现困难。使用 BT 对等协议和 cdn 都是成熟的方案，可备用。

e） 资源下载速度达到 500kb/s

#	模式名称	开发时间开销	成本开销	带宽要求
1	提高 CPU 利用率	中	中	不要求
2	动态路由优化	中	高	中
3	多线程分段下载	中	中	不要求

✧ 选择理由：提高 CPU 利用率实现难度较大，不列入选择。动态路由优化和多线程分段下载这三个在开发时间与成本开销上各有优势，且没有显著缺点，因此都作为备选方案。

d）支持无网络环境下对资源进行缓存

#	模式名称	开发时间开销	成本开销	数据丢失风险
1	记录文件下载信息	中	中	很小

e）资源下载速度稳定

#	模式名称	配置难度	成本开销	网络依赖
1	采用 Nginx	配置简单	较低	低
2	采用 LVS 集群	配置困难	高	高

v. 候选架构模式综合评估

	对未经授权的用户拒绝指定资源下载		可支持多用户同时下载文件资源		资源下载速度达到 500kb/s.		支持无网络环境下对资源进行缓存	
	优点	缺点	优点	缺点	优点	缺点	优点	缺点
拦截器模式	实现简单，效率高	消耗多	无影响	无影响	无影响	无影响	无影响	无影响
CDN （内容分发网络）	无影响	无影响	效率高	容量小	解决网络链路问题，有效抗击 DDOS 攻击		无影响	无影响
使用 BT 对等协议	无影响	无影响	效率高，承载能		资源分段下载，服务器多种选择，	资源消	无影响	无影响

			力强，方案成熟		速度很快	耗多		
动态路由优化	无影响	无影响	无影响	无影响	效率高	额外消耗高	无影响	无影响
多线程分段下载	无影响	无影响	无显著影响	无显著影响		效率不明显	无影响	无影响
记录文件下载信息	无影响	无影响	无影响	无影响	无影响	无影响	效率高	

✧ 评估结果：拦截器模式更有利于对资源的过滤以及限制未知权限对文件的访问；通过记录文件下载信息能够有效实现下载中的断点续传功能，使用 BT 对等协议可以很好地支持多用户同时下载同一个资源，而多线程分段容易和 BT 对等协议结合起来使用。

2.1.5 第五次迭代

- 步骤一

  第五次迭代可以不用此步骤。

- 步骤二

  在此次迭代中针对权限（安全）管理模块进行分解。

- 步骤三

  确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 4：客户进行上传文件	高	高
2	场景 5：未经授权的访问文件	高	高
3	场景 6：经过授权的下载文件	高	中

- 步骤四

  选择满足架构驱动因素的架构模式。

1） 针对安全性相关的架构驱动因素

i. 设计关注点

设计关注点	子关注点

可靠性	阻止未经授权的资源访问
机密性	传输过程中资源内容不泄露
完整性	防止对非法操作对数据的篡改

ii. 各子关注点的候选架构模式

f) 阻止未经授权的资源访问

#	模式名称	开发时间开销	成本开销	局限性	系统资源开销
1	基于密钥认证访问者	高	中	密钥生成过程会损耗性能	高
2	限制访问次数	低	低	无法从本质上阻止未授权访问	低
3	防火墙	低	低	防护可控性低	中

✧ 选择理由： 由于 P2P 架构和系统的功能要求安全性比较高，可以接受一定的开发和成本的开销，因此采用基于密钥的认证作为系统的备选方案。

g） 传输过程中资源内容不泄露

#	模式名称	稳定性	实现难度	可靠性	灵活性
1	传输层安全协议	中	低	中	高
2	更改默认设置	高	中	高	低
3	防火墙	高	低	中	低

✧ 选择理由：由于网盘系统涉及大量用户隐私，传输过程需要尽可能的保证隐私安全，因此以上三种模式均作为备选方案。

h） 防止对非法操作对数据的篡改

#	模式名称	稳定性	实现难度	可靠性	资源占用
1	系统冗余备份	高	高	高	高
2	输入验证，输出编码	高	高	高	低

✧ 选择理由：进行冗余备份能有效通过备份数据复原被篡改的数据，而 P2P 架构是分布式的，数据分存在各个节点中，没有冗余的必要，而对输入验证和输出编码能有效的防止跨站脚本攻击，阻止篡改行为，因此选择输入验证，输出编码作为备选方案。

vi. 候选架构模式综合评估

	阻止未经授权的资源访问		传输过程中资源内容不泄露		防止对非法操作对数据的篡改	
	优点	缺点	优点	缺点	优点	缺点
基于密钥认证访问者	安全性强，难以攻破	服务器端和客户端生成密钥代价较大	信息被加密，监听者无法解析	传输的信息需要加密解密，消耗资源	无显著影响	
限制访问次数	对资源和成本消耗小	难以真正阻止未授权访问	访问次数受限，受到窃听可	无法真正的防止网站受到监	无明显优点	非法操作在受限次数内仍可

			能性降低	听		篡改数据
防火墙	拦截成功率高	防火墙不能防范来自内部网络的攻击	屏蔽被保护的信息，有效阻止监听	不能阻止内部泄密行为	拦截成功率高	非法操作可能绕过防火墙
传输层安全协议	客户机和服务器需要认证，安全性强	需要服务器端加密算法和浏览器支持	传输过程加密，难以监听	需要复杂的认证过程	安全性强	加密解密消耗资源
更改默认设置	无显著影响		使监听者难以获取到数据，有可靠性	成本开销较大	减少网站自身防御负担	成本开销较大
系统冗余备份	无显著影响		无显著影响		备份可以增加系统数据的可靠性	服务器更新数据负担加重
输入验证，输出编码	输入验证可以阻止未授权的访问	编码解码过程消耗系统资源	无显著影响		高效防止跨站脚本攻击，保护隐私	增加验证和解码过程，加重负担

✧ 评估结果：基于密钥的认证安全性强，虽然资源消耗较大但是对用户隐私有保障，添加防火墙对认证和传输过程的安全都有一定安全保护作用，传输层安全协议则是比较成熟的传输加密协议，保证了机密性，可靠性和完整性，输入验证，输出编码则可以保证网站免受 XSS 跨站脚本的攻击，因此选择以上四种模式。

2.1.6 第六次迭代

- 步骤一

  第六次迭代可以不用此步骤。

- 步骤二

  在此次迭代中针对资源管理模块进行分解。

- 步骤三

  确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 3：用户正常使用客户端	中	中
2	场景 4：客户进行上传文件	高	高
3	场景 7：用户使用上传和下载数量加剧	高	高
4	场景 8：正常操作	高	中

- 步骤四

选择满足架构驱动因素的架构模式。

4） 针对可用性相关的架构驱动因素

vii. 设计关注点

设计关注点	子关注点
资源分类体验	服务器对文件格式与权限限制进行分类管理
资源评论体验	评论方式简洁且便于显示
资源排名体验	资源按照热度、评论数等进行排名无误

viii. 各子关注点的候选架构模式

i） 服务器对文件格式与权限限制进行分类管理

#	模式名称	开发时间开销	成本开销	系统资源开销
1	拦截器模式	中	低	低
2	文件处理	中	中	低

✧ 选择理由：对文件格式的分类选择以及资源的访问权限，在前端使用拦截器模式进行拦截，没有硬件成本，软件实现相对较为简单。

j） 评论方式简洁且便于显示

#	模式名称	开发时间开销	成本开销	系统资源开销
1	数据库分表	中	中	中
2	服务端 MVC	中	中	低
3	前端双向绑定	低	高	中

✧ 选择理由：资源内容评论选择存储在数据库中，对数据库进行分表处理。服务端采用 MVC 便于评论的 ajax 方法，前端双向绑定有利于页面的局部刷新。

k） 资源按照热度、评论数等进行排名

#	模式名称	开发时间开销	成本开销	技术难度
1	数据库集成框架	中	中	较小
2	数据库读写分离+分区	高	中	有

✧ 选择理由：采用数据库集成框架方便于对热度与评论进行快速排名，数据库读写分离加快检索的速度，提高读的效率

ix. 候选架构模式综合评估

	服务器对文件格式与权限限制进行分类管理		评论方式简洁且便于显示		资源按照热度、评论数等进行排名	
	优点	缺点	优点	缺点	优点	缺点
拦截器模式	对系统资源开销较	网络利用率低，有	无显著影响		无显著影响	网络利用率低，有

	低	额外资源开销				额外资源开销
文件处理	效率高	消耗额外计算资源	无显著影响	容易造成拥堵	效率高	消耗额外计算资源
数据库集成框架	效率高	无显著缺点	无显著缺点		效率高	容量小
服务端MVC	使用灵活且可以满足普通需求	运行消耗资源，架构较为复杂	无显著影响	无显著影响	使用灵活且可以满足普通需求	软件运行消耗资源
前端双向绑定	无显著影响		数据快速更新，界面显示较为合理	效率较低，实现复杂度升高	带宽要求低	无显著缺点
数据库读写分离+分区	提高读的效率	无显著影响	读资源丰富	修复部分故障时间长，影响大	读数据效率高	有数据丢失风险

✧ 评估结果:拦截器模式更有利于对资源的过滤以及对未知权限对文件的访问;前端的数据双向绑定以及对数据库的读写分离+分区是实现评论快速更新以及文件资源排序的不二选择。数据库集成框架有利于数据的检索与排序更加便捷。

5） 针对可拓展性相关的架构驱动因素

iii. 设计关注点

设计关注点	子关注点
增加功能	资源管理的分类进行增加功能
增加接口	资源管理需要对外提供其他的接口api

iv. 各子关注点的候选架构模式

c） 资源管理的分类进行增加功能

#	模式名称	实现难度	稳定性	依赖
1	面向接口编程	实现简单	高	低
2	面向切面编程	实现较难	高	低

✧ 选择理由:面向接口编程与面向切面编程在实际业务逻辑之中都多有应用，我们根据项目需求在这里选择面向接口编程为主，面向切面编程为辅

d） 资源管理需要对外提供其他的接口api

#	模式名称	系统资源开销	重要性
1	面向接口编程	低	高
2	异常输入处理	高	高

✧ 选择理由:面向接口编程在提供api中占据了主要，同时我们也不能忽视异常输入与异常错误的处理，当调用者传入非法参数时需要有应对手段

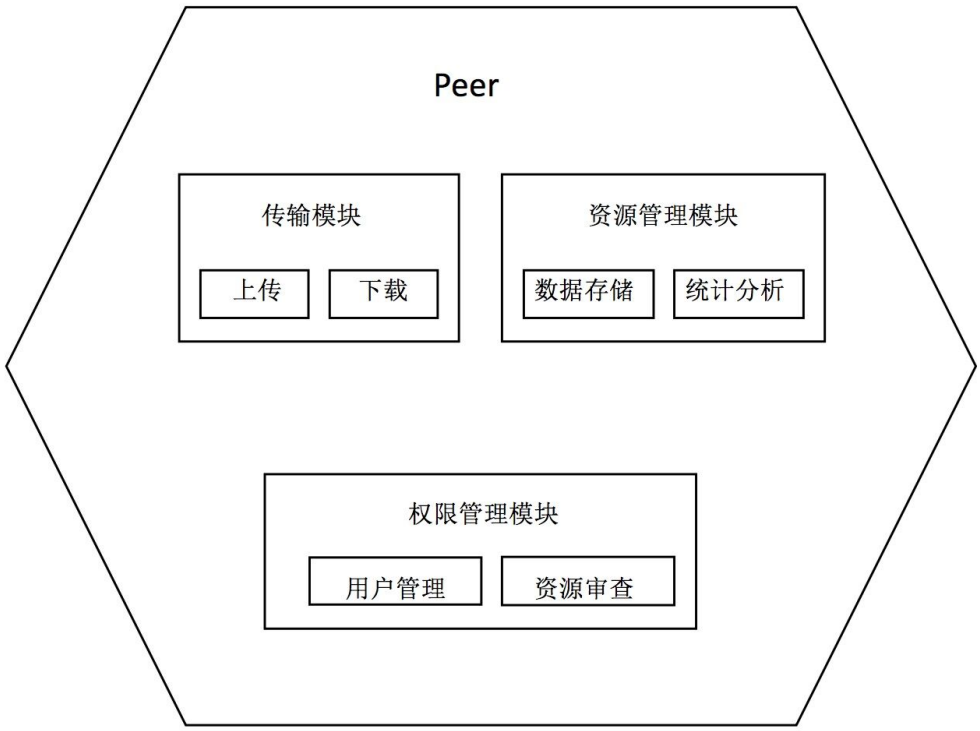
v. 候选架构模式综合评估

	资源管理的分类进行增加功能		资源管理需要对外提供其他的接口 api	
	优点	缺点	优点	缺点
面向接口编程	面向接口编程，耦合度低，聚合度高，代码易懂，可拓展性强。封装性强	增加了系统的复杂度，降低了代码的可复用性	接口经过合理设计后，有利于程序设计的规范化，可以并性开发，提高了工作效率	暂无
面向切面编程	减少代码的重复读，统一管理，降低耦合	配置相对较为复杂	代码较为清晰，减少了重复代码	增加了代码复杂度
异常输入处理	有利于系统的可靠性能力		提升系统的稳定性	增加了代码的复杂度

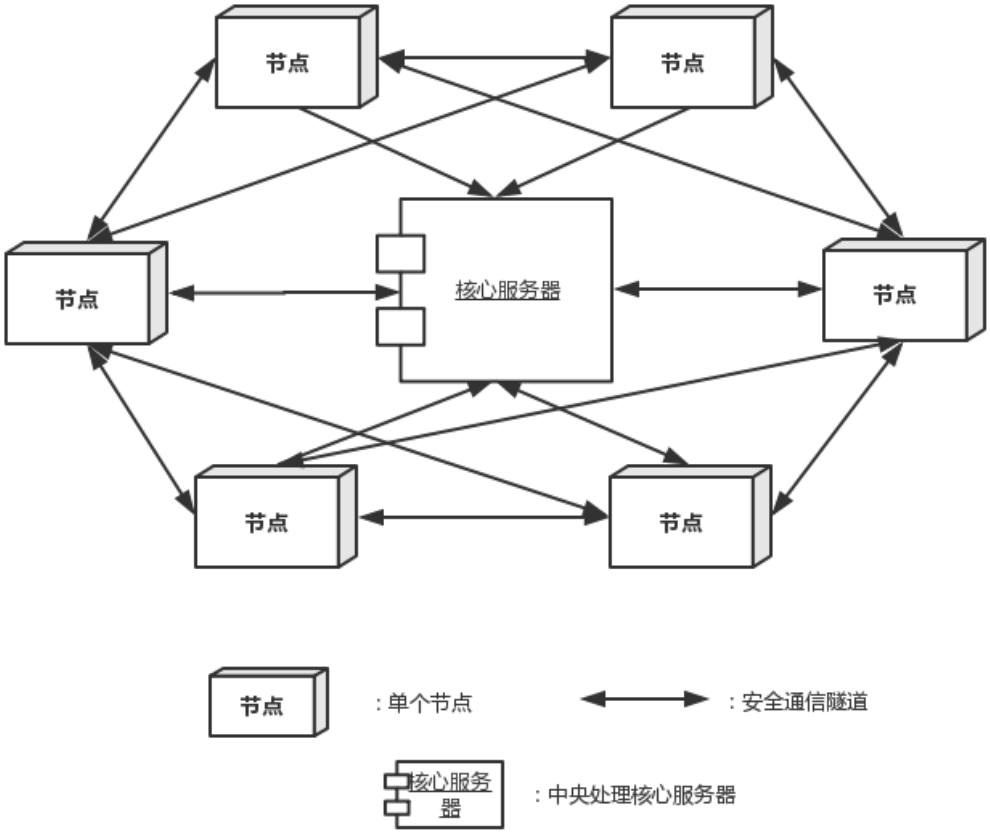
评估结果：相对来说面向接口编程更有利于可拓展性的修改，降低了耦合度增加了内聚性，使代码通俗易懂。同时在编程过程中如果需要的话可以辅助一些面向切面的编程。异常处理增加了代码的稳定性与健壮性，提高了系统的可靠性，是一个必选的选项。

2.2 架构图

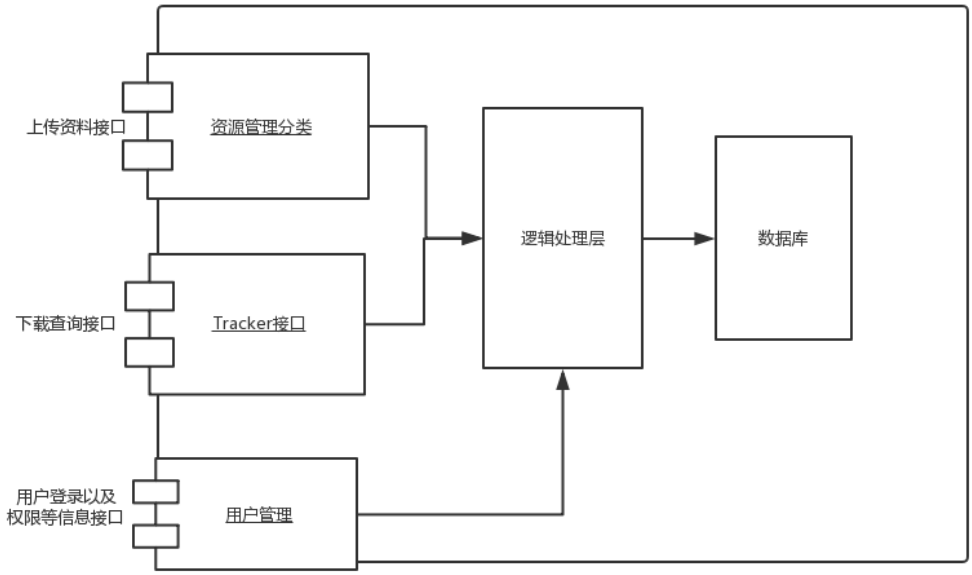
1） 模块视图



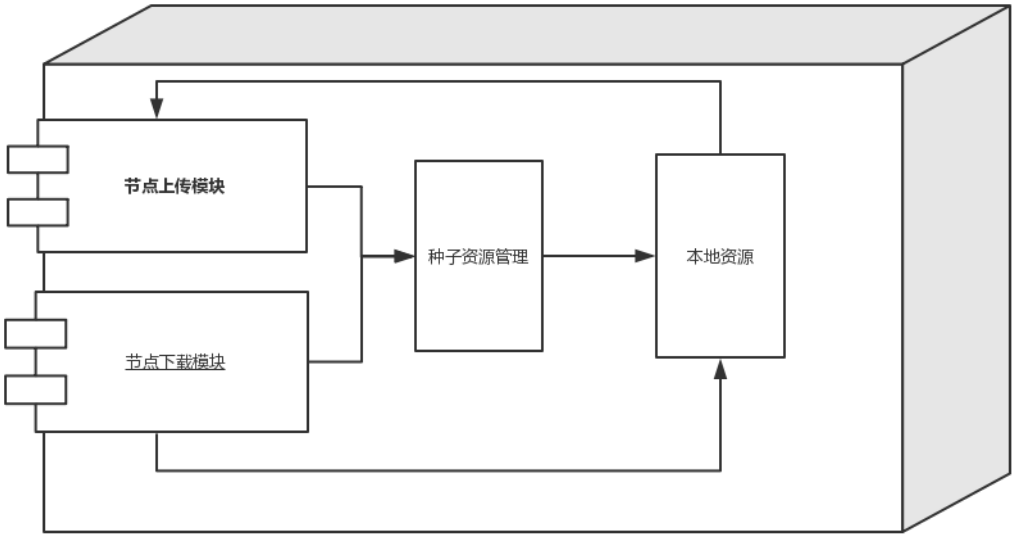
2） C& C 视图



P2P 模式框架节点与服务器视图



中央核心服务器视图



节点模块视图

3.Client/Server架构设计

3.1 ADD 过程

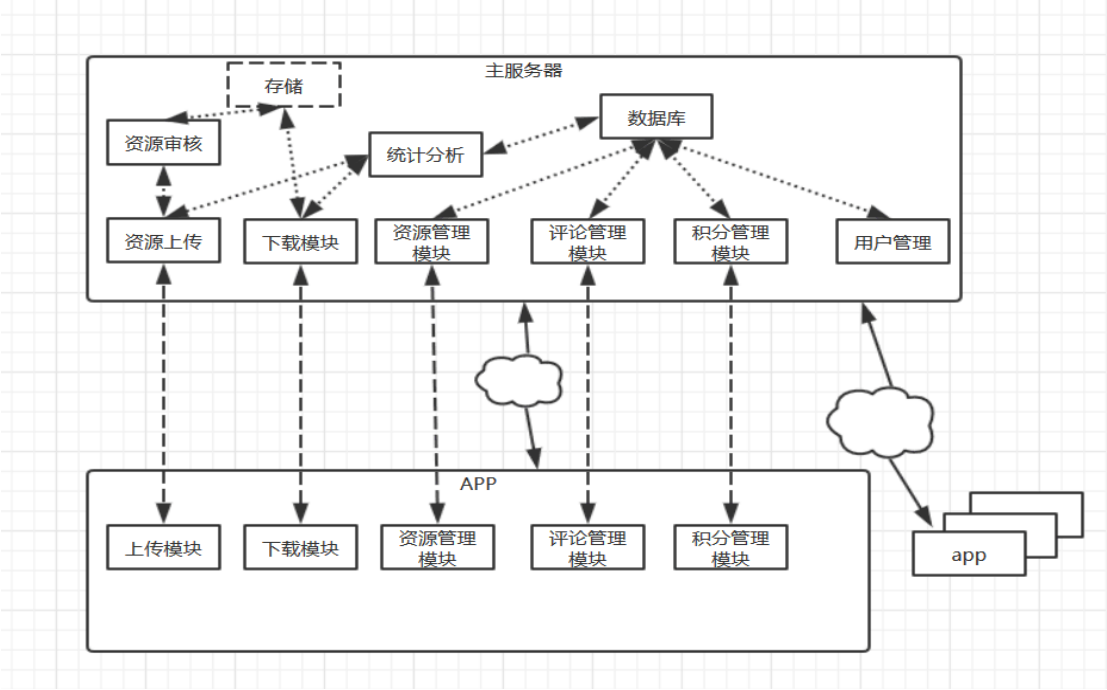
3.1.1 第一次迭代

步骤一

获得样本输入，筛选掉不必要的需求，列出AS。此处的ASR 与P2P 架构设计ADD 过程中所列相同。

步骤二

对整个系统进行分解，得到系统组成元素视图如下：



3.1.2 第二次迭代

步骤一：第二次迭代可以不用此步骤

步骤二：我们选择系统（服务器和客户端）的上传模块进行分解。上传模块是系统的核心功能模块之一，它可以分为数据源采集（客户端）、上传进度（客户端和服务器）、上传数据传输（客户端和服务器之间）、上传数据处理（服务器）四个分解功能点。对上传模块进行分解，对于进一步设计该系统的体系结构有很大的帮助。

步骤三：

确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 2：用户使用客户端进行上传	高	中
2	场景 4：客户进行上传文件	高	高
3	场景 5：未经授权的访问文件	高	中
4	场景 6：用户使用上传数量加剧	中	高
5	场景 8：正常上传操作	高	低
6	场景 11：系统服务器崩溃	高	高

步骤四：

选择满足架构驱动因素的架构模式。

i. 设计关注点

设计关注点	子关注点
数据源采集	数据源压缩编码
上传数据传输	上传数据传输协议
	上传数据效率
上传数据处理	上传数据处理方式
	上传数据处理效率
上传进度	上传断点记录
	上传数据拆分方式

ii. 各子关注点的候选架构模式

a) 数据源压缩编码

☆用于区分的参数：

- 安全性，对用户上传的文件有识别功能，确保服务器不被上传的文件破坏（场景 4）

- 性能，对客户端压缩的性能要求以及网络传输的容错性
- 易用性，保证在采集过程中用户的操作是正常不卡顿的
- 可伸缩性，用户上传数量加剧（场景 6）

编号	方案名称	压缩率	容错能力	CPU 能力要求	压缩时间
1	RAR 压缩	高	高	高	中
2	ZIP 压缩	中	低	高	高

☆ 选择方案：RAR 压缩

☆ 选择原因：ZIP 的安装包比较大，并且仅仅只有英文版加汉化包，而 RAR 有官方的简体中文版，并且安装包很小，不足 1M 。且 RAR 在最高压缩率压缩的情况下，RAR 的压缩率要更高一点。除此之外，ZIP 支持的格式很多，但已经比较老并且不大流行，而 RAR 的支持格式要比 ZIP 多并且也支持很多流行的压缩格式。ZIP 仅仅只能压缩成 zip 格式，并不能鸡鸭 RAR 格式，而 RAR 不仅有自己的格式，还可以压缩成 zip 格式并解压 zip 格式。最后也是最重要的一个原因，在传输超大型文件时，例如几十个 G 的情况下，RAR 支持分卷压缩，这很好的降低了上传文件是客户端和服务器的压力，而 zip 不支持分卷压缩，所以选择 RAR 压缩方式。

b）上传数据传输协议

☆用于区分的参数：

- 性能，用户使用客户端进行上传（场景 2）
- 可伸缩性，用户上传数量加剧（场景 6）
- 可用性，上传过程中，网络中断（场景 10）

编号	方案名称	可靠性	实时性	连续性	完整性
1	TCP 协议	强	弱	一般	强
2	UDP 协议	弱	强	一般	弱

☆ 选择方案：TCP 协议

☆ 选择原因：TCP 协议是面向连接的可靠协议，如果发现分组有丢失现象，会请求进行重传，保证了数据的准确性，保证了数据有序传输。而 UDP 协议是无连接的，它本身没有数据校验机制,不会进行数据的校准和分组确认,且传输数据时是无序的,所以在传输数据时，如果少了一些数据是不会重传的。而我们的系统需要传输的可靠性和完整性，否则可能有些严重的缺失就会导致文件运行失败。综上我们选择 TCP 协议进行文件数据的上传。

c）上传数据效率

☆用于区分的参数：

- 性能，用户使用客户端进行上传（场景 2）
- 可用性，上传数据流中断（场景 10）
- 可伸缩性，用户上传数量加剧（场景 6）

编号	方案名称	传输效率	可用性	可同时上传数量

1	提升服务器带宽	高	中	中
2	使用分布式服务器	高	高	多
3	提高客户端的上行带宽	中	中	中

☆ 选择方案：使用分布式服务器，并根据需要提升各服务器带宽

☆ 选择原因：上传数据效率主要考虑传输速率和传输可用性。传输效率主要由服务器的总带宽和客户端的上行带宽决定,单一提升某一服务器或者客户端的带宽的确可以提高传输效率，但使用分布式服务器不仅可以提升总带宽，而且有更高的可用性和容错性。不仅如此，在整个分布式系统中，同一个资源，只保存一个在系统中，只要用户在上传的时候，系统发现有相同资源，则可以立即结束上传，故可以提高上传的并发量，使用户可以在同一时间上传更多的数据。如果只是单一的提高客户端的上行带宽,则会增加用户的开销且没有普遍性,只有提高了上行带宽的用户上传速率才会增加,所以我们使用分布式服务器加提升服务器带宽的方案。

d）上传数据处理方式

☆用于区分的参数：

- 性能，上传数据处理的性能要求
- 成本，存储上传的数据需要较大的成本
- 安全性，识别客户上传文件（场景 4）

编号	方案名称	性能要求	成本	安全性	处理速率
1	直接存储	低	低	低	高
2	把文件解压缩并扫描，检查是否有违禁文件或病毒，若没有文件则存储	高	高	高	低
3	只解压一小部分文件，检查是否有违禁文件或者病毒，若没有则存储	中	中	中	中

☆ 选择方案：只解压一小部分，检查是否有违禁文件或者病毒，若没有则直接存储

☆ 选择原因：如果选择直接存储，虽然成本降低了，处理速率也提高了，但是安全性大大降低，因为用户有可能上传了违禁的文件或者一个病毒文件，直接导致系统崩溃，造成了更大损失，而如果把文件全部解压缩去扫描，又太过于消耗系统资源，降低了系统的性能，使

得系统的处理速率大大降低。而如果只解压一部分的话，虽然会导致安全性没有第二种方案那么高，但是系统性能得到打打改善，处理速率也将大大提高，并且成本也控制在了一个可接受的范围里面。

e）上传数据处理效率

☆ 用于区分的参数

- 性能，服务器对于上传数据处理的性能要求
- 可伸缩性，服务器对于大量上传的数据的处理（场景 7）
- 可用性，系统服务器的崩溃处理（场景 11）

编号	解决方案	处理速率	可伸缩性	可用性
1	使用 RAID5 存储上传的数据	中	中	高
2	边解压边分析数据	高	高	低
3	解压完成后再分析数据	中	中	低
4	直接存储	高	高	低

☆ 选择方案：使用 RAID5 存储上传的数据，边解压边分析数据

☆ 选择原因：使用 RAID5 存储数据，在系统突然崩溃，上传的数据丢失的时候，RAID5 可以使用另外一个校验信息,恢复之前上传的数据,提高了系统的可用性,而直接存储的话，在系统崩溃之后，上传的数据将损毁也无法修复，可能会导致一些无法挽回的损失。与此同时，使用边解压边分析数据方法有效提高了数据的处理效率，一旦分析到了违禁或者病毒，就可以立即停止解压并且丢弃该文件，具有高处理速率和高的可伸缩性。

f）上传断点记录

☆ 用于区分的参数：

- 性能，用户使用客户端进行上传（场景 2）
- 可用性，下载过程中网络中断（场景 10）
- 可伸缩性，用户使用上传数量加剧（场景 7）

编号	解决方案	上传速率	可用性	可同时上传数量
1	自定义断点交互协议	中	高	高
2	使用 http 断点下载协议	高	高	中

☆ 选择方案：自定义断点交互协议

☆ 选择原因：自定义断点续传协议，可以让浏览器记住最近一次成功传输的位置，也可以把上传的断点放在服务器端,而 http 协议的断点只能记录在浏览器中,一旦关闭了浏览器，将不可能在实现断点续传，而且在自定义断点协议中，可以配合前后端配套开发，使用缓存

技术，能得到更大的同时上传数量，所以我们选择自定义断点交互协议。

g）上传数据拆分方式

☆ 用于区分的参数：

- 性能，用户使用客户端进行上传（场景 2）
- 可用性，下载过程中网络中断（场景 10）
- 可伸缩性，用户使用上传数据加剧（场景 7）

编号	解决方案	上传速率	可用性	可同时上传数量
1	定义拆分单元为 1M	中	高	高
2	定义拆分单元为 10M	高	中	中

☆ 选择方案：定义拆分单元为 1M

☆ 选择原因：将拆分单元定义为 1M ，将有助于在断点续传的时候，少传一些文件，并且因为传输单元为 1M ，没有占用太多的传输通道，可以增加更多的上传并发，使得同一用户能在同一时间，上传更多的文件。虽然拆分为 10M 的少了很多次交互次数，上传速率会比拆分为 1M 的快，但是我们更加看重上传的并发量和断点的传输量，所以我们选择了拆分单元为 1M 大小的方案为解决方案。

3.1.3 第三次迭代

步骤一：

第三次迭代可以不用此步骤

步骤二：

我们选择系统（服务器和客户端）的资源管理模块进行分解。资源管理模块是系统的核心功能模块之一，它可以分为统计模块（客户端）、存储模块（服务器）和推荐模块（服务器）三个分解功能点。对资源管理模块进行分解，对于进一步设计该系统的体系结构有很大的帮助。

步骤三：

确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 3：用户正常使用客户端进行资源查看	高	中
2	场景 8：正常操作	高	低

4	场景 5：未经授权的访问文件	高	中
5	场景 9：系统增加一个新功能	高	高
6	场景 12：客户端升级	中	高

步骤四：

选择满足架构驱动因素的架构模式。

i. 设计关注点

设计关注点	子关注点
统计	统计方式
存储	存储方式
推荐	推荐策略

ii. 各子关注点的候选架构模式

a）统计方式

☆ 用于区分的参数：

- 性能，用户正常使用客户端
- 易用性，正常操作
- 可扩展性，系统增加一个新的功能

编号	方案名称	性能	易用性	可扩展性
1	列表统计	高	低	高
2	图表统计	中	高	中

☆ 选择方案：图表统计

☆ 选择原因：资源管理主要是方便用户尽快尽早的了解系统的资源情况，所以在性能、易用性和可扩展性中，最重要的是易用性，虽然列表统计具有较高的性能和可扩展性，在新增一个统计的时候，可以更快地复用代码，而图标统计只能重新设计图表，但是用户的感官体验是第一位的，所以我们选择图表统计作为解决方案。

b）存储方式

☆ 用于区分的参数：

- 性能，用户正常使用客户端
- 可维护性，系统升级
- 可用性，系统服务器崩溃
- 成本，存储开销

编号	方案名称	性能	可维护性	可用性	成本
1	RAID0 存取	高	高	低	低

2	RAID1 存取	中	中	高	高
3	RAID5 存取	高	高	高	中

☆ 选择方案：RAID5 存取

☆ 选择原因 :RAID5 试讲校验码平均分到各个磁盘中，这样就将校验盘的访问压力平均分配到每一个块磁盘中降低了校验盘的访问压力。这种磁盘结构既提升了 IO 能力，又有冗余能力，还能从损坏的数据中恢复。对比 RAID0，RAID5 具有恢复能力，如果服务器崩溃，倒数数据丢失，则 RAID0 不能将损失恢复，所以不选择 RAID0 存储方式。对比 RAID1，因为 RAID1 只将同一份数据存储在两个盘中，导致其成本提高，写速度变为原来的两倍，可导致上传速度降低，所以不选 RAID1 的存储方式。所以，最后我们选择 RAID5 存取的方案。

c）推荐策略

☆ 用于区分的参数：

- 性能，用户正常使用客户端（场景 3）
- 易用性，正常操作（场景 8）

编号	方案名称	性能	易用性
1	最热推荐	高	中
2	喜好推荐	中	高
3	随机推荐	中	中

☆ 选择方案：一般情况下最热推荐，用户需要时，给予喜好推荐

☆ 选择原因：最热推荐只需要知道哪些资源被下载的次数最多，就能给出一张推荐列表，所以这种推荐方式并不消耗多少资源，而且性能较高，下载次数最多的不一定是用户最喜欢的，所以我们选择在用户需要的时候，用户能得到喜好推荐，而喜欢推荐是根据用户之前的下载记录，判断用户喜好类型，然后再根据喜好类型，给予用户一个推荐列表，这样的推荐，就需要后台服务器的算法支撑，不单单只是读写操作，所以性能会比最热推荐低，但是这符合用户的需求，所以我们选择了一个折中的方式，就是在用户需要的时候，才会进行喜好推荐，默认情况下是最热推荐。

3.1.4 第四次迭代

步骤一：

第三次迭代可以不用此步骤

步骤二：

此次迭代选择系统（服务器和客户端）中的资源下载模块进行分解。资源下载模块是用户资源分享这一核心功能点的重要子模块，它可以分为资源搜索(客户端与服务器间)，资源数据传输（客户端与服务器之间），资源下载中断恢复（客户端）三个分解功能点。对下载模块

进行分解，对于进一步设计该系统的体系结构有很大的帮助。

步骤三：

确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景一：用户使用客户端下载	中	中
2	场景三：用户使用客户端查看	中	中
3	场景五：未经授权的文件下载	中	中
4	场景六：经过授权的文件下载	高	低
5	场景七：下载数量加剧	高	高
6	场景八：正常操作	高	低
7	场景十：下载过程中网络中断	中	中

步骤四

选择满足架构驱动方式的架构模式

i. 设计关注点

设计关注点	子关注点
资源采集	资源搜索方式
	资源压缩编码
资源数据传输	资源数据传输协议
	资源传输效率
	资源加密
资源下载中断恢复	中断处理和恢复

ii. 各子关注点的候选架构模式

a)资源搜索方式

用于区分的参数：

- 性能，用户使用客户端搜索（场景三）
- 易用性，正常操作（场景八）

编号	方案名称	性能	易用性
1	数据库检索关键字（可使用缓存）获得相应资源信息	中	高
2	使用种子方式直接检索数据库	高	低

☆ 选择方案：数据库检索关键字

☆ 选择原因：数据库检索关键字或许比直接搜索较慢，但是却能让用户直接搜索需要的相关资源，不需要花时间寻找种子，提高了易用性，而且在使用缓存的情况下性能也不会差上太多。

b.资源压缩编码

用于区分的参数：

- 性能，用户使用客户端下载（场景一）
- 易用性，正常操作（场景八）
- 可伸缩性，下载数量加剧（场景七）

编号	方案名称	压缩率	容错能力	CPU 性能要求	压缩时间
1	MPEG2 压缩标准	低	中	中	短
2	H.264 压缩标准	高	高	高	长

☆ 选择方案：H.264 压缩标准

☆ 选择原因：MPEG 压缩标准属于比较古老的技术，而 H.264 压缩标准是比较新的压缩技术，所以在压缩率和网络传输的容错能力上讲，H.264 压缩率高，文件体积小，优于 MPEG2。但 MPEG2 虽然压缩率低，但对 CPU 的运算能力要求比较低，压缩时间也比较短。但同时考虑到需要进行网络传输，网络传输效率的优先级高于客户端运算的优先级，而通过 H.264 压缩的文件体积小，利于网络传输，同时又有很好的网络容错能力。所以选择 H.264 压缩标准。

c.数据传输协议

用于区分的参数：

- 性能，用户使用客户端下载（场景一）
- 安全性，经过授权的文件下载（场景六）
- 可伸缩性，下载数量加剧（场景七）
- 可用性，下载过程中网络中断（场景十）

编号	方案	性能	安全性	可伸缩性	可用性
1	TCP 协议	弱	强	弱	强
2	UDP 协议	强	弱	强	弱

☆ 选择方案：UDP 协议

☆ 选择原因：UDP 协议是无连接的，它本身没有数据校验机制，不会进行数据的校准和分组确认。而 TCP 是面向连接的可靠协议，如果发现分组有丢失现象，会请求进行重传。对于资源分享，数据的安全性和可用性相对更加重要，传送的文件出现问题需要重新下载，反而增加了下载的时间。所以选择 TCP 协议进行下载。

d.数据传输效率

用于区分的参数：

- 性能，用户使用客户端下载（场景一）
- 可伸缩性，下载数量增加（场景六）

- 可用性，下载过程中网络中断（场景十）

编号	方案	性能	可伸缩性	可用性
1	使用镜像服务器	高	高	高
2	离线下载	中	高	高
3	提升服务器带宽	高	中	中
4	限制客户端码率	中	中	中

☆ 选择方案：镜像服务器加速和离线下载

☆ 选择原因：数据传输效率主要考虑传输速率和传输可用性。传输效率主要由服务器总带宽决定，单一提升某一服务器的带宽的确可以提高传输效率，但使用镜像服务器不仅可以提升总带宽，而且有更高的可用性和容错性。单一服务器的连接并发数是固定的，可容纳用户也有数量上的限制，而使用镜像服务器可以容纳更多的用户，并且可以通过镜像服务器的缓存提高性能。如果某台服务器发生故障，单一服务器就再也不能提供服务，而分布式服务器仍可对用户提供资源。而限制客户端的码率并不是长久的方案，首先不满足易用性。所以首选方案是使用镜像服务器，当然也可以根据需求提升镜像服务器的带宽。而离线下载则可以帮助冷门资源的下载，提高性能和可用性，且提高安全性，又不与镜像服务器冲突，所以也会同时使用

e.数据加密方式

用于区分的参数：

- 安全性：未经授权的文件下载（场景五）
- 性能：用户使用客户端进行下载（场景一）

编号	方案	安全性	性能
1	对称加密	中	高
2	非对称加密	高	中

☆ 选择方案：对称加密

☆ 选择原因：场景七要求系统保证数据的安全性，兼顾系统性能，能及时响应客户需求。非对称加密与对称加密都能保证系统数据很好的安全性，但是非对称加密对于客户端和服务端的负担更大，为了保证系统的性能，故选择对称加密方案。

f.中断处理与恢复

用于区分的参数：

- 性能，用户使用客户端下载（场景一）
- 易用性，正常下载（场景八）
- 安全性，网络中断（场景十）

编号	方案	性能	易用性	安全性
1	保存本地备份，可在重新下载时导入恢复	高	中	中
2	删除本地备份，完全重	低	中	高

	新下载			

☆ 选择方案：保存本地备份，重新下载时导入恢复

☆ 选择原因：保存备份虽然会降低安全性但是有前面的加密保护，而重新导入并不困难但对用户下载大文件中断有着极大帮助，重新导入解析或许会消耗时间，但是实际上用户可以通过选择是否继续之前的下载来决定是否导入解析。导入解析本身也可以通过进度的配置文件解决。

3.1.5 第五次迭代

步骤一

第五次迭代可以不用此步骤

步骤二

我们选择系统服务器的数据库接口模块进行分解

步骤三

确定架构驱动因素，如下表所示：

#	架构驱动因素	重要性	难易度
1	场景 1:用户使用客户端进行下载	高	中
2	场景 2:用户使用客户端进行上传	高	中
3	场景 3:用户正常使用客户端	高	中
4	场景 5:未经授权的访问文件	高	高
5	场景 12 客户端升级	高	中

步骤四

选择满足架构驱动因素的架构模式

i. 设计关注点

设计关注点	子关注点
数据存储	数据存储方式
	数据架构
	数据库结构
数据传输	数据传输方式
	数据传输过程检测
数据加密	数据加密方式

ii. 各子关注点的候选架构模式

a)数据存储方式

✧ 用于区分的参数

- 安全性：保证数据存储不被篡改泄露（场景 5）
- 可维护性：保证数据存储兼容（场景 12）

编号	方案名	读写速度	操作容易度	安全性
1	在线存储	快	容易	中
2	近线存储	慢	中等	中
3	脱机存储	极快	困难	低
4	异站保护	块	中等	高

✧ 选择方案：在线存储

✧ 选择原因：要求保证数据的可移植性和数据的访问速度，近线存储的数据访问速度较慢，脱机存储需要人为存储，操作过于繁琐，异站存储降低了数据的可移植性，故选择对访问速度和安全性及可移植性都有保障的在线存储方式。

b)数据存储架构

✧ 用于区分的参数

- 安全性：保证数据存储不被篡改泄露（场景 5）
- 可维护性：保证数据存储兼容（场景 12）

编号	方案名	数据量	访问速度
1	嵌入式架构	小	中
2	X86 架构	中	慢
3	云技术	高	快

✧ 选择方案：云存储

✧ 选择原因：系统要求足够的数据存储量以及数据下载上传的速度，嵌入式架构和基于 X86 的架构在数据存储量上无法满足项目要求，故选择云存储架构。

c)数据库结构

✧ 用于区分的参数

- 安全性：保证数据存储不被篡改泄露（场景 5）
- 性能：保证数据传输的迅速（场景 1,2,3）
- 可维护性：保证数据存储兼容（场景 12）

编号	方案名	访问速度	语句复杂度
1	网状数据库	快	高
2	关系数据库	中	低
3	树状数据库	中	高
4	面向对象数据库	快	高

✧ 选择方案：关系数据库

✧ 选择原因：场景 1,2,3 要求系统保证数据上传下载的速度及简明的数据结构。树形数据库和网状数据库结构较复杂，面向对象数据库访问速度较慢，故选择二者能兼顾的关系数据库。

d)数据传输方式

✧ 用于区分的参数

- 安全性：保证数据传输中的安全（场景 4，5，6）
- 性能：保证数据传输的迅速（场景 1,2,3）

编号	方案名	节点数目	传输速度
1	单节点传输	一个	慢
2	多节点传输	多个	快

✧ 选择方案：多节点传输

✧ 选择原因：场景 1,2,3 要求系统能保证上传与下载速度，故选择能多节点传输多个节点同时传输。

e)数据传输过程监控

✧ 用于区分的参数

- 安全性：对不正常的数据请求进行警报（场景 5）
- 性能：保证数据传输分流进行提升传输速度（场景 1,2,3）

编号	方案名	占用带宽	检测覆盖率
1	传输全程监控	高	>90%
2	传输抽样监控	低	>50%

✧ 选择方案：数据传输全程监控

✧ 选择原因：场景 5 要求系统有极高的安全性，需要识别恶意传输的数据流，同时兼顾系统性能。虽然全程监控开销比抽样大，但是能够保证用户数据在传输过程中的安全性，安全性更加重要。故选择全程监控的方案。

f)数据加密方式

✧ 用于区分的参数

- 安全性：保证加密的复杂度，最大限度的防止破解（场景 5）
- 性能：加密和解密不能影响系统的正常响应（场景 1,2,3）

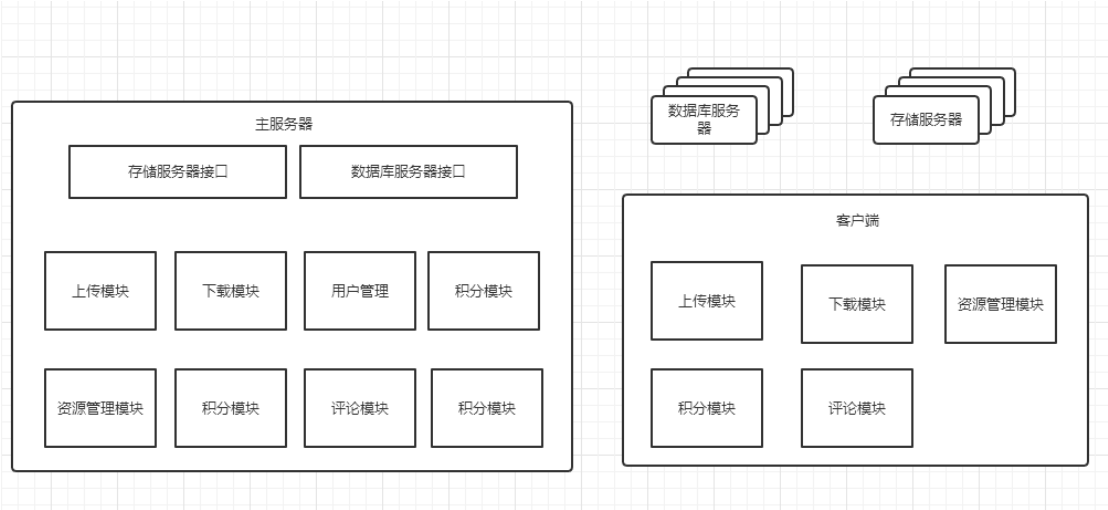
编号	方案名	复杂度	对系统响应的影响
1	对称加密	高	低
2	非对称加密	很高	低

✧ 选择方案：对称加密

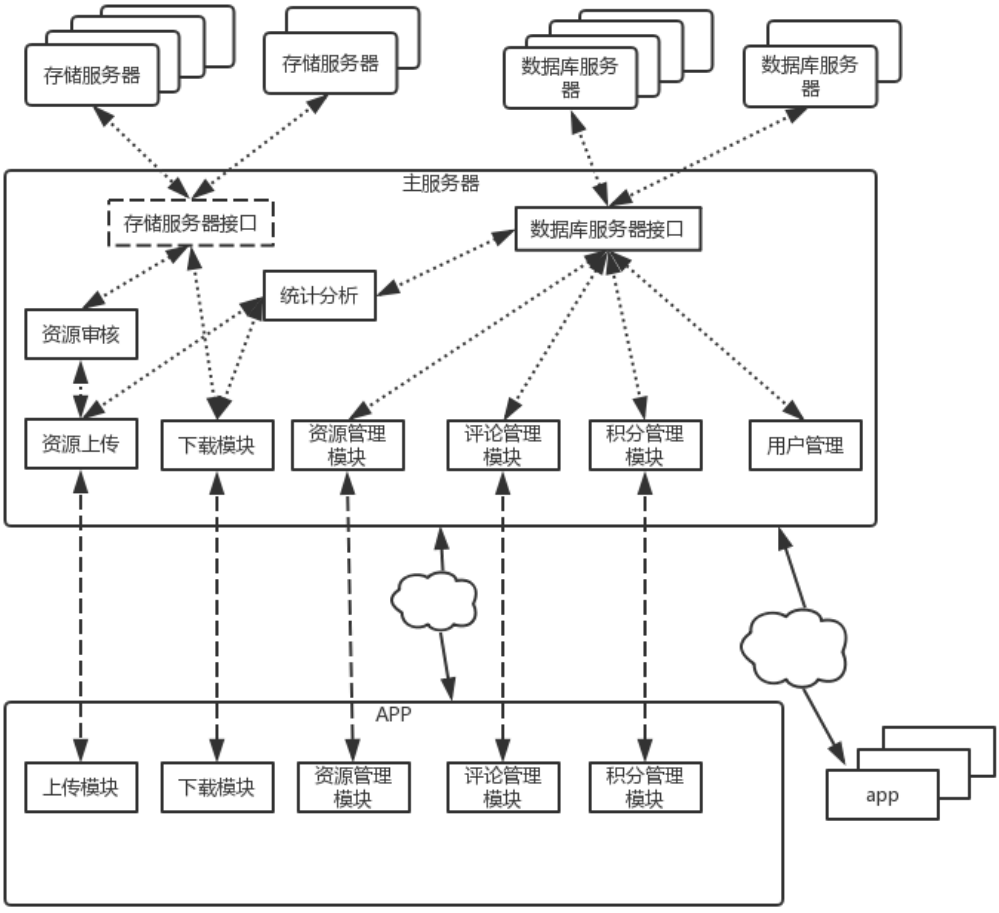
✧ 选择原因：场景 5 要求系统保证数据的安全性，兼顾系统性能，能及时响应客户需求。非对称加密与对称加密都能保证系统数据很好的安全性，但是非对称加密于客户端和服务端的负担更大，为了保证系统的性能，故选择对称加密方案。

3.2 架构图

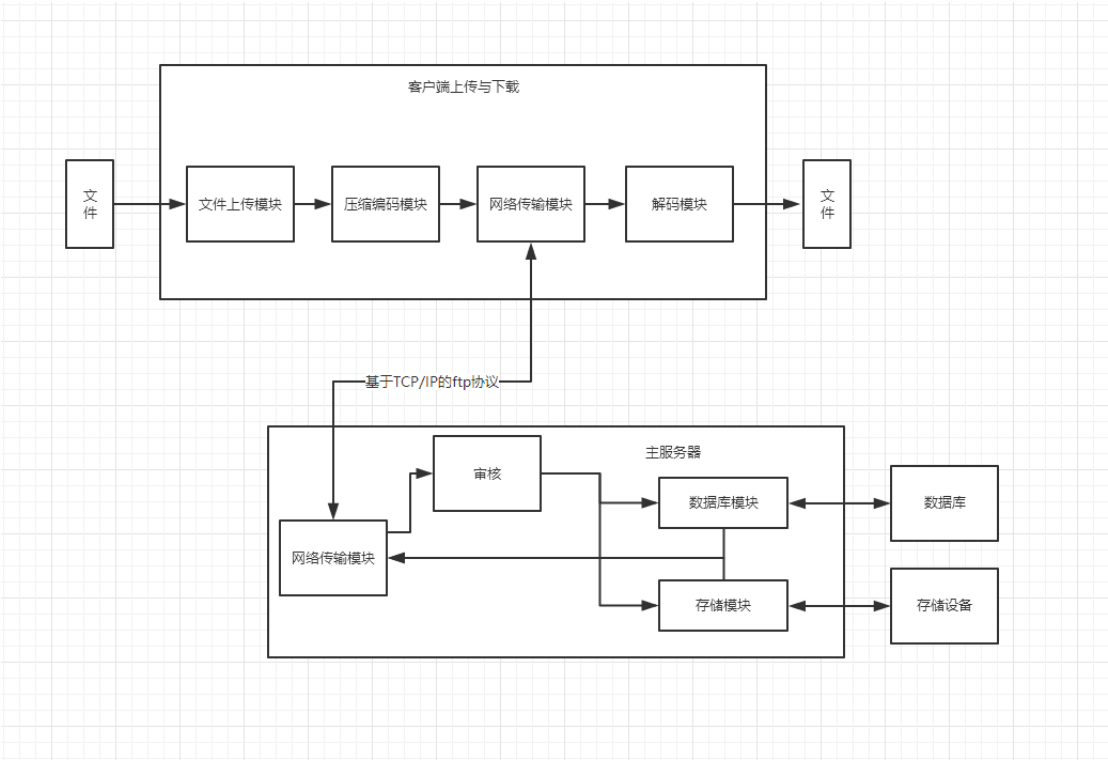
（1）模块视图



（2）C&C 视图



…….. 服务器交互 ------ 客户端与服务器交互



C/S 模式框架客户端与服务器视图

4.P2P 与 C/S 架构对比

4.1 性能对比

P2P	C/S
● 节点越多，网络性能越好，网络随着规模增大而越发稳固 ● P2P 能充分利用 Peer 的资源，提高传输效率和通信质量	● 服务器的存储能力和处理能力以及所在网络的吞吐量是该模式性能的瓶颈，随着节点增加，服务器负载越来越重，一旦服务器崩溃，整个网络也随之瘫痪 ● 需要通过增加服务器数量来提升性能，成本高

4.2 安全性对比

P2P	C/S
● 信息共享，用户自由度高，带来随机性和不确定性，数据安全性难以保证 ● 安全策略、备份策略复杂，成本高	● 数据与控制流方向单一，高度集中，安全措施实现相对简单 ● 一旦服务器被攻击，损失很大

4.3 资源利用对比

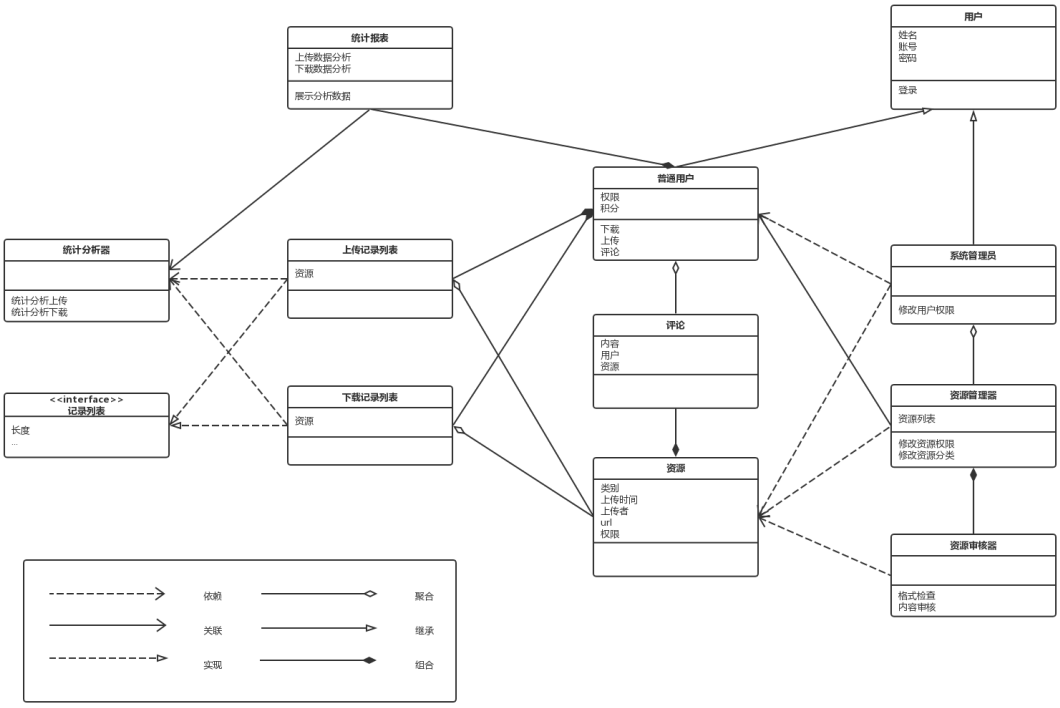
P2P	C/S
● 能够有效的利用所有节点上的各种资源 ● 占用客户端存储、处理资源	● 大量有一定计算与存储能力的客户端的资源被闲置 ● 相同数据的分发造成服务器和网络带宽的浪费 ● 资源更新成本高

4.4 维护成本对比

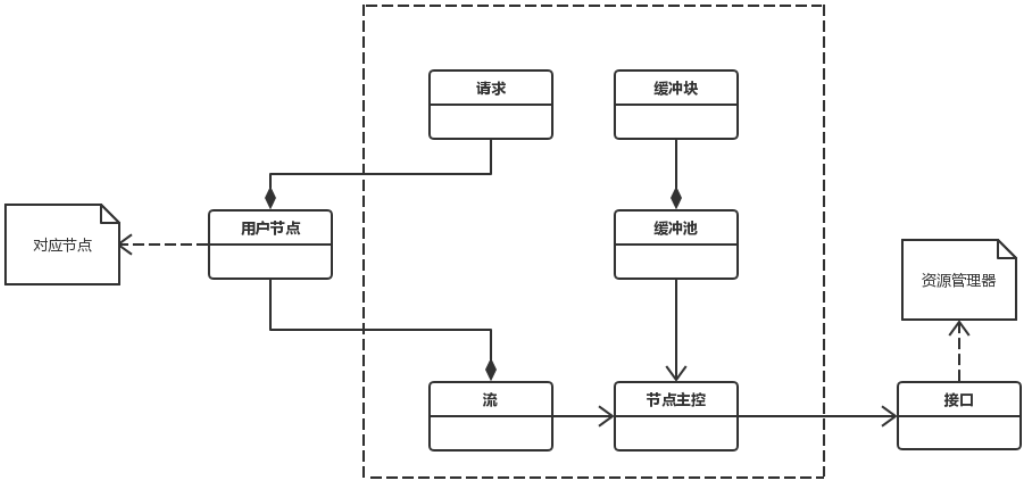
P2P	C/S
● 核心服务器不需要存储大量的资源，不必耗费大量资金，而且每个队等点都可以在网络上发布和分享信息，这使得闲散资源得以充分的利用	● 软件更新快，需要花费大量 ing 里和金钱在软件的更新上，而且工作人员要维护服务器和数据库，也要耗费大量资金

5.系统类图设计

经过 P2P 架构和 C/S 架构的设计、评估、对比过程，最终决定系统采用 P2P 架构，并设计类图如下所示：



类与组件、连接件的映射关系如下图所示：



6.挑战和经验

此次项目最大的挑战来自于对于 P2P 架构以及资源上传下载过程的不熟悉，为了顺利完成架构设计，小组成员查阅了大量资料，在资源传输部分进行了深入的了解，对资源传输系统有了更深的了解。

通过此次项目实践，小组成员对 ADD 方法有了更加全面的认识。ADD 是以质量为主要矛盾，功能为次要矛盾来进行体系结构设计，重点在于质量属性上。

同时我们对架构设计的流程也更加熟悉，但也了解到一个好的架构的设计的过程是困难

和复杂的，既要进行架构设计又要进行评估，才能确保架构设计的质量。在进行架构设计，还要注意多考虑本系统的特性和侧重点，对重要的部分要多花些功夫。作为体系架构师，需要对软件开发的各方面都要十分清楚了解，包括各种决策，解决方案可能的影响等等，需要不断地汲取知识和积累很多的经验才能得到更好的成果。

此外，这次团队的人数比较多，因此如何高效地利用人力资源进行合作成为了另一个侧重点。在每次团队会议中，每位成员都会参与讨论，涉及的话题包括：系统特色功能、两种 c& c 架构的选择、ADD 方法等。我们会提前准备议题，尽可能地保证每次团队会议都围绕着解决问题的思路开展。另外在分工上，我们把两个基本 c& c 架构划分给两个小组（4 人）分别使用 ADD 进行体系结构设计，为的是保证大家尽可能多的考虑到不同架构的特点，便于之后对两个架构详尽比较，再进行选择。由此，我们整个团队合作的过程都相对比较顺利。

7.组员和分工

组员	学号	分工
王杰	141250138	参与质量属性划分，负责易用性场景的描述。基于 ADD 方法进行对 P2P 架构的体系结构设计与评估：参与下载模块性能这一质量属性的 ADD 过程讨论；负责用户管理的安全性这一部分的架构设计。
王新宇	141250141	负责安全性相关场景的编写；参与 P2P 架构设计的 ADD 过程，并设计出系统的类图。
王家玮	141250136	参与模块划分，参与架构设计的讨论，负责可伸缩性相关场景的编写，参与 P2P 架构设计 ADD 过程对资源管理模块进行分解。
田琦	141250126	参与讨论了小组产品- - 红领巾的功能，选取 C/S 和 P2P 两种 C& C 风格的架构来实现。参与讨论了 P2P 架构的 ADD，完成了上传功能模块的 ADD 过程。
熊凯奇	141250156	与小组成员讨论 C/S 架构的总体结构和模块划分，负责架构相关设计图的绘制和上传模块的 ADD 分析。
王泽霖	141250144	负责性能相关场景的编写；参与 C/S 架构设计的 ADD，负责可用性部分；参与下载模块的架构决策讨论；负责最终文档汇总与排版。
杨关	141250166	参与小组需求讨论，参与了 C/S 架构设计的 ADD；完成 ADD 中可维护性的分析；参与 ADD 中资源管理模块的分析；完成可维护性相关 PPT 部分。