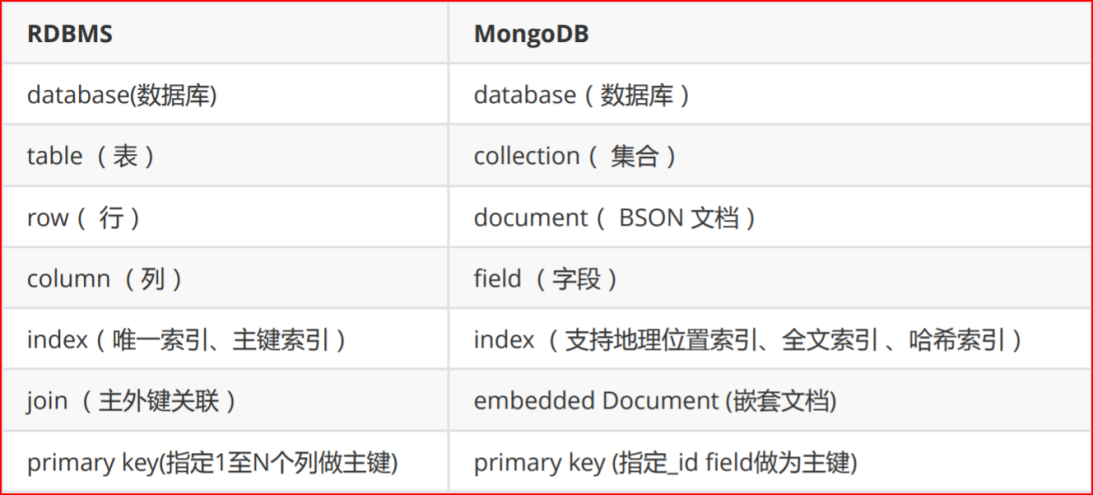
**MongoDB**

▶ **MongoDB 和RDBMS(关系型数据库)对比**



▶**什么是MongoDB**

MongoDB是一个文档数据库，非关系型数据库。采用BSON存储文档数据。 BSON是一种类json的一 种二进制形式的存储格式，简称Binary JSON. 相对于json多了date类型和二进制数组。

▶**BSON**

◇BSON是一种类json的一种二进制形式的存储格式，简称Binary JSON，它和JSON一样，支持内嵌 的文档对象和数组对象，但是BSON有JSON没有的一些数据类型，如Date和Binary Data类型。

◇优点是灵活性高，但它的缺点 是空间利用率不是很理想

◇三个特点：轻量性、可遍历性、高效性

▶**MongoDB的特点**

◇面向文档的存储：以 JSON 格式的文档保存数据。

◇任何属性都可以建立索引。

◇复制以及高可扩展性。

◇自动分片。

◇丰富的查询功能。

◇快速的即时更新。

▶**MongoDB支持的数据类型**

◇String

◇Integer

◇Double

◇Boolean

◇Object

◇Object ID（存储文档id）：一共有四部分组成:时间戳、客户端ID、客户进程ID、三个字节的增 量计数器

◇Array:数组

◇Timestamp

◇Code（存储 JavaScript 代码）

◇File

▶**MongoDB操作**

◇查看数据库 show dbs;

◇切换数据库 如果没有对应的数据库则创建 use 数据库名;

◇创建集合 db.createCollection("集合名")

◇查看集合 show tables; show collections;

◇删除集合 db.集合名.drop();

◇删除当前数据库 db.dropDatabase();

◇如何添加索引

db.collection.createIndex()

◇如何查询集合中的文档

db.collectionName.find({key:value})

◇用什么方法可以格式化输出结果

db.collectionName.find().pretty()

◇如何使用"AND"或"OR"条件循环查询集合中的文档

db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

◇更新数据

db.collectionName.update({key:value},{$set:{newkey:newValue}})

◇如何删除文档

db.collectionName.remove({key:value})

◇在MongoDB中如何排序

并使用 1 和 -1 来指定排序方式，其中 1 表示升序，而 -1 表示降序。

db.connectionName.find({key:value}).sort({columnName:1})

▶**索引**

◇索引类型

•单键索引

•复合索引

•地理空间索引，针对地理空间坐标数据创建索引

•全文索引

•哈希索引 Hashed Index，hash index仅支持等于查询，不支持范围查询

◇慢查询分析

•开启内置的查询分析器,记录读写操作效率

•查询监控结果

•分析慢速查询

•解读explain结果 确定是否缺少索引

◇索引底层实现原理

•MongoDB使用B-树，所有节点都有Data域，只要找到指定索引就可以进行访问， 单次查询从 结构上来看要快于MySql

•B-树

○多路 非二叉树

○每个节点 既保存数据 又保存索引

○搜索时 相当于二分查找

▶**常见问题**

◇为什么MongoDB的数据文件很大

MongoDB采用的预分配空间的方式来防止文件碎片。

◇数据在什么时候才会扩展到多个分片(shard)里

mongodb 分片是基于区域(range)的.所以一个集合(collection)中的所有的对象都被存放到一 个块(chunk)中.只有当存在多余一个块的时候,才会有多个分片获取数据的选项.现在,每个默 认块的大小是 64mb,所以你需要至少 64 mb 空间才可以实施一个迁移.

▶**Spring Boot 访问 MongoDB**

◇MongoTemplate 的方式

◇MongoRepository 的方式

**TOMCAT**

▶**Tomcat**

◇**Tomcat初始化流程**

• 启动tomcat ，需要调用 bin/startup.bat (在linux 目录下 , 需要调用 bin/startup.sh)， 在startup.bat 脚本中, 调用了catalina.bat；

• 在catalina.bat 脚本文件中，调用了BootStrap 中的main方法；

• 在BootStrap 的main方法中调用了 init 方法 ，来创建Catalina及初始化类加载器；

• 在BootStrap的main方法中调用了 load 方法 ，在其中又调用了Catalina的load方法；

• 在Catalina 的load方法中 , 需要进行一些初始化的工作, 并需要构造Digester 对象, 用 于解析 XML，最后初始化Server；

• 在Server的initInternal()方法中，初始化了Engine、Executor、Connector;

• 在Connector中的initInternal()方法中，设置了CoyoteAdapter适配器并初始化了 ProtocolHandler；

• ProtocolHandler是个接口，调用AbstractHttp11Protocol的init()方法，在此方法中调用 了AbstractProtocol的init()方法，此方法中初始化了EndPoint;

◇**Tomcat启动流程**

• 在BootStrap 的main方法中调用了start()方法；

• 通过反射又调用了Catalina的start()方法；

• 在Server中的startInternal()方法又调用了Engine、Executor、Connector的start()方 法；

• 在Connector的startInternal()方法中调用了ProtocolHandler的start()方法；

• 在AbstractProtocol的start()方法中调用了EndPoint的start()方法；

◇**Tomcat请求处理流程**

• Connector组件Endpoint中的Acceptor监听客户端套接字连接并接收Socket；

• 将连接交给线程池Executor处理，开始执行请求响应任务；

• Processor组件读取消息报文，解析请求行、请求体、请求头，封装成Request对象；

• Mapper组件根据请求行的URL值和请求头的Host值匹配由哪个Host容器、Context容器、 Wrapper容器处理请求；

• CoyoteAdaptor组件负责将Connector组件和Engine容器关联起来，把生成的Request对象 和响应对象Response传递到Engine容器中，调用 Pipeline；

• Engine容器的管道开始处理，管道中包含若干个Valve、每个Valve负责部分处理逻辑。执 行完Valve后会执行基础的 Valve–StandardEngineValve，负责调用Host容器的Pipeline；

• Host容器的管道开始处理，流程类似，最后执行 Context容器的Pipeline；

• Context容器的管道开始处理，流程类似，最后执行 Wrapper容器的Pipeline；

• Wrapper容器的管道开始处理，流程类似，最后执行 Wrapper容器对应的Servlet对象的处 理方法；

◇ **Tomcat 配置调优**

• 调整tomcat线程池

• 调整tomcat的连接器

• 禁用 AJP 连接器

AJP连接；WEB服务器和Servlet容器通过TCP连接

• 调整 IO 模式

NIO

◇ **谈谈 Tomcat 顶层架构**

• Tomcat中只有一个 Server，一个 Server 可以有多个Service，一个 Service 可以有多个 Connector 和一个 Container；

• Server 掌管着整个 Tomcat 的生死大权；

• Service 是对外提供服务的；

• Connector 用于接受请求并将请求封装成 Request 和 Response 来具体处理；

• Container 用于封装和管理 Servlet，以及具体处理 request 请求；

◇ **Container的组件和职责**

分别是 Engine、 Host、 Context 和 Wrapper。 这 4 种组件（容器）是⽗⼦关系

• Engine：表示整个 Catalina 的 Servlet 引擎， ⽤来管理多个虚拟站点， ⼀个 Service 最 多只能有⼀ 个 Engine，但是⼀个引擎可包含多个 Host

• Host：代表⼀个虚拟主机，或者说⼀个站点，可以给 Tomcat 配置多个虚拟主机地址， ⽽⼀ 个虚拟主机下可包含多个 Context

• Context：表示⼀个 Web 应⽤程序， ⼀个 Web 应⽤可包含多个 Wrapper

• Wrapper ：表示⼀个 Servlet， Wrapper 作为容器中的最底层，不能包含⼦容器

**Nginx**

▶**Nginx如何处理请求**

◇ Nginx 大量使用多路复用和事件通知

◇ Nginx 启动以后，会在系统中以 daemon 的方式在后台运行，其中包括一个 master 进程，n(n>=1) 个 worker 进程。

◇ 所有的进程都是单线程（即只有一个主线程）的，且进程间通信主要使用共享内存的方式。

◇ master 进程用于接收来自外界的信号，并给 worker 进程发送信号，同时监控 worker 进程的工 作状态。

◇ worker 进程则是外部请求真正的处理者，每个 worker 请求相互独立且平等的竞争来自客户端的 请求。

◇ 请求只能在一个 worker 进程中被处理，且一个 worker 进程只有一个主线程，所以同时只能处 理一个请求。

▶**Nginx的轮询策略**

◇轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉，能自动剔除。

◇weight

指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况。

◇ip\_hash

每个请求按访问 ip 的 hash 结果分配，这样每个访客固定访问一个后端服务器，可以解决 session 的问题。

◇fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配

◇url\_hash（第三方）

按访问 url 的 hash 结果来分配请求，使每个 url 定向到同一个后端服务器，后端服务器为 缓存时比较有效。

**Web**

▶**转发和重定向区别**

◇从地址栏显示来说:

forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来, 然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址 栏还是原来的地址.

redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显 示的是新的URL.

◇从数据共享来说:

forward:转发页面和转发到的页面可以共享request里面的数据. redirect:不能共享数据.

◇从运用地方来说:

forward:一般用于用户登陆的时候,根据角色转发到相应的模块.

redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等

◇从效率来说:

forward:高. redirect:低.

▶**HTTP 1.0和HTTP 1.1的主要区别**

◇长连接：HTTP 1.1起，默认使用长连接 ,默认开启Connection： keep-alive。

◇错误状态响应码：HTTP1.1中新增了24个错误状态响应码

◇缓存处理：HTTP1.1则引入了更多的缓存控制策略

◇带宽优化及网络连接的使用 ：HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部 分。

▶**URI和URL的区别**

◇URI： 是统一资源标志符，可以唯一标识一个资源。

◇URL： 是统一资源定位符，可以提供该资源的路径。它是一种具体的 URI

▶**基于Jwt Token进行身份验证**

◇用户向服务器发送用户名和密码用于登陆系统。

◇身份验证服务响应并返回了签名的 JWT，上面包含了用户是谁的内容。

◇用户以后每次向后端发请求都在Header中带上 JWT。

◇服务端检查 JWT 并从中获取用户相关信息。

▶**Http请求的完**全过程

◇浏览器根据域名解析IP地址（DNS）,并查DNS缓存

◇浏览器与WEB服务器建立一个TCP连接

◇浏览器给WEB服务器发送一个HTTP请求（GET/POST）：一个HTTP请求报文由请求行（request line）、 请求头部（headers）、空行（blank line）和请求数据（request body）4个部分组成。

◇服务端响应HTTP响应报文，报文由状态行（status line）、相应头部（headers）、空行（blank line）和响应数据（response body）4个部分组成。

◇浏览器解析渲染

▶**4XX 客户端错误**

400 Bad Request ：请求语法错误

401 Unauthorized ：需要认证信息，或者认证失败

403 Forbidden ：请求被拒绝

404 Not Found

▶**5XX 服务器错误**

500 Internal Server Error ：服务器正在执行请求时发生错误。

503 Service Unavilable ：服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

504 Gateway time out : 网关超时。

▶**HTTPS工作原理**

◇浏览器：发起一个https的请求，把自身支持的密钥算法套件发送给服务端

◇服务器：接收到客户端密钥算法套件后与自身支持的对比，如果不支持则连接断开，反之则会从中 选出一种HASH加密算法，以证书的形式返回给客户端。证书中还包含了 公钥 颁证机构 网址 失效 日期等等。

◇浏览器：1、验证证书的合法性 2、生成随机密码，用证书中的公钥加密成随机数秘钥。3、随机密 码对称加密之前通信信息HASH值 和 随机数秘钥 发送给服务端

◇服务器：1、私钥来解密取出随机数密码，再用随机数密码 解密 之前通信信息HASH值，并获取之 前通信信息HASH值，对比确认是否一致。确保HASH算法没有被串改。 2、随机密码对称加密之前 通信信息的HASH值给客户端

◇浏览器：1、客户端用随机数解密之前通信信息HASH值，并获取之前通信信息HASH值，对比确认 是否一致，此时握手过程结束 2、之后通信数据将由之前浏览器生成的随机密码并利用对称加密算 法进行加密

▶**RPC和HTTP的区别**

◇传输协议

RPC，可以基于TCP协议，也可以基于HTTP协议

HTTP，基于HTTP协议

◇传输效率

RPC，使⽤用⾃自定义的TCP协议，可以让请求报⽂文体积更更⼩小，或者使⽤用HTTP2协议， 也可以很好的减少报⽂文的体积，提⾼高传输效率

HTTP，如果是基于HTTP1.1的协议，请求中会包含很多⽆无⽤用的内容，如果是基于HTTP2.0， 那么简单的封装以下是可以作为⼀一个RPC来使⽤用的，这时标准RPC框架更多的是服务治理

◇性能消耗

RPC，可以基于thrift实现⾼高效的二进制传输

HTTP，⼤大部分是通过json来实现的，字节⼤大⼩小和序列列化耗时都⽐比thrift要更更消 耗性能

主要在于序列列化和反序列列化的耗时

◇负载均衡

RPC，基本都⾃自带了了负载均衡策略略

HTTP，需要配置Nginx， HAProxy来实现

◇服务治理

（下游服务新增，重启，下线时如何不不影响上游调⽤用者）

RPC，能做到自动通知，不影响上游

HTTP，需要事先通知，修改Nginx/HAProxy配置

◇使用场景

RPC主要⽤用于公司内部的服务调用，性能消耗低，传输效率⾼高，服务治理理⽅方便便。

HTTP主要⽤用于对外的异构环境，浏览器器接口调⽤用， APP接⼝调⽤用，第三⽅方接⼝调⽤

▶**HTTP长连接**

◇长连接的几种实现方式

•ajax 轮询

ajax 轮询指客户端每间隔一段时间向服务端发起请求，保持数据的同步。

•WebSocket

Websocket 实现了客户端与服务端的双向通信，只需要连接一次，就可以相互传输数据， 很适合实时通讯、数据实时更新等场景。

•iframe 长连接

在网页上嵌入一个 iframe 标签，该标签的 src 属性指向一个长连接请求。这样服务端就 可以源源不断地给客户端传输信息。保障信息实时更新

◇为什么要有Connection: keep-alive

重用连接的机制，就是在http请求头中加入Connection: keep-alive来告诉对方这个请求响 应完成后不要关闭，下一次咱们还用这个请求继续交流。

◇http长连接会一直保持吗

不会的。服务端都会设置keep-alive超时时间。超过指定的时间间隔，服务端就会主动关闭接。

同时服务端还会设置一个参数叫最大请求数，比如当最大请求数是300时，只要请求次数超过 300次，即使还没到超时时间，服务端也会主动关闭连接。

◇Transfer-Encoding

在响应头中使用，因为客户端浏览器都能解析这个响应头

输出的内容长度不能确定，普通的静态页面、图片之类的基本上都用不到这个。

如果结合：Content-Encoding: gzip 使用的时候，Transfer-Encoding: chunked还是比较有用 的。

◇Content-Length

请求方法为POST的时候使用，常常出现在响应头

◇保持长连接后，浏览器如何知道服务器已经响应完成？

一般情况下这个特殊标志就是Content-Length，来指明响应体的数据大小，比如Content-Length: 120表示响应体内容有120个字节，这样浏览器接收到120个字节的响应体后就知道了已经响应 完成。

Transfer-Encoding: chunked，该响应头表示响应体内容用的是分块传输，此时服务器可以将 数据一块一块地分块响应给浏览器而不必一次性全部响应，待浏览器接收到全部分块后就表示 响应结束。

◇TCP keepalive

TCP keepalive指的是TCP保活计时器（keepalive timer）。设想有这样的情况：客户已主动 与服务器建立了TCP连接。但后来客户端的主机突然出故障。显然，服务器以后就不能再收到 客户发来的数据。因此，应当有措施使服务器不要再白白等待下去。这就是使用保活计时器。 服务器每收到一次客户的数据，就重新设置保活计时器，时间的设置通常是两小时。若两小时 没有收到客户的数据，服务器就发送一个探测报文段，以后则每隔75秒发送一次。若一连发送 10个探测报文段后仍无客户的响应，服务器就认为客户端出了故障，接着就关闭这个连接。

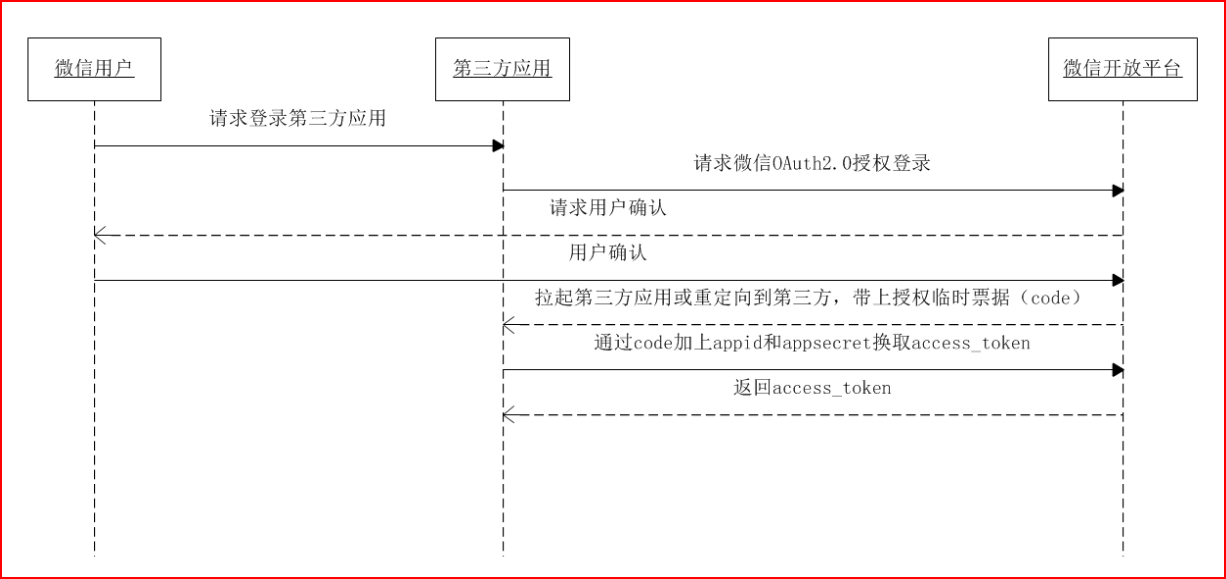
**微信方面**

▶**微信认证（Oauth2授权码模式）**

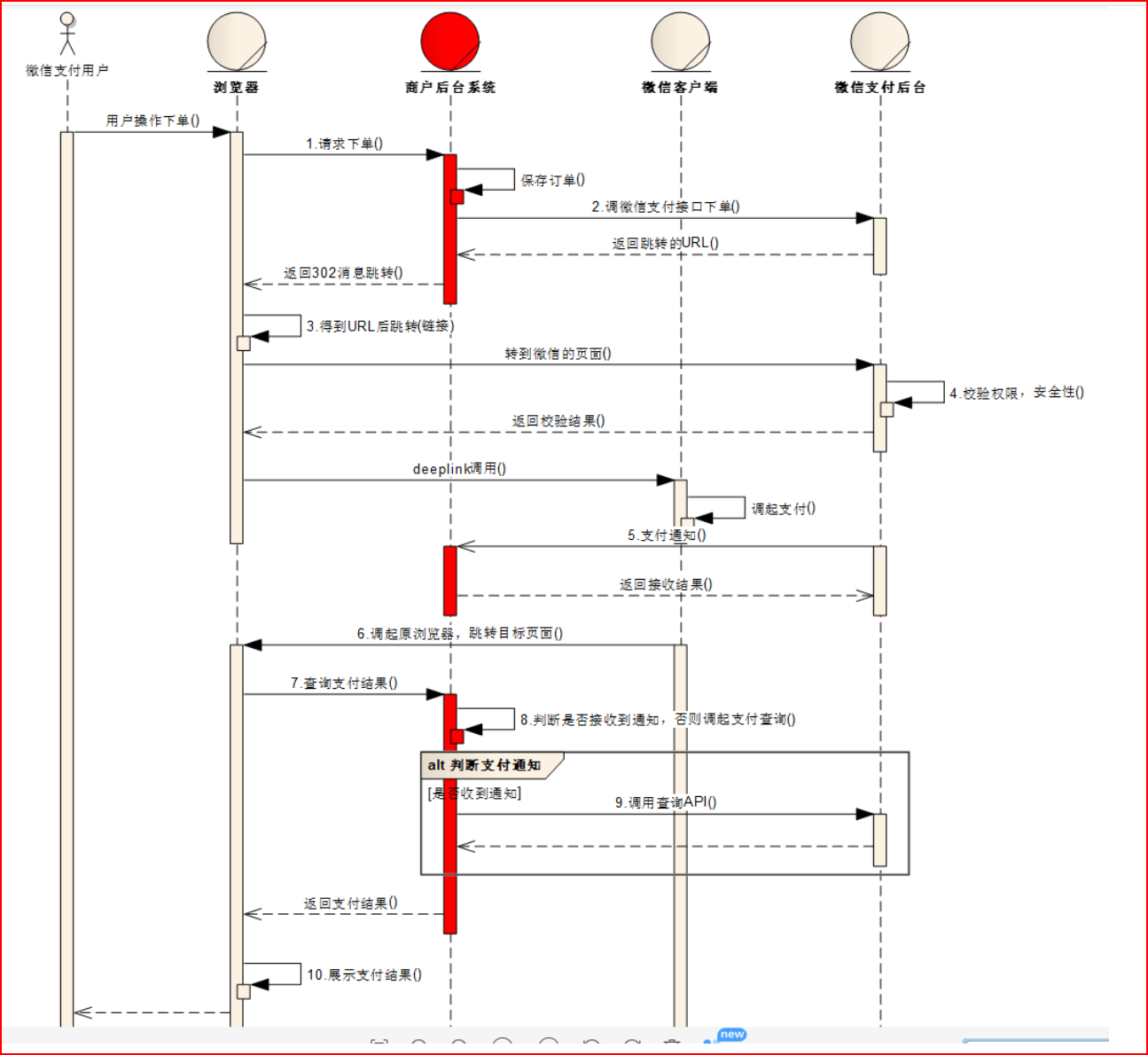
1. 第三方发起微信授权登录请求，微信用户允许授权第三方应用后，微信会拉起应用或重定向到第 三方网站，并且带上授权临时票据code参数；

2. 通过code参数加上AppID和AppSecret等，通过API换取access\_token；

3. 通过access\_token进行接口调用，获取用户基本数据资源或帮助用户实现基本操作。



▶**微信h5支付**



1、用户在商户侧完成下单，使用微信支付进行支付

2、由商户后台向微信支付发起下单请求（调用统一下单接口）注：交易类型trade\_type=MWEB

3、统一下单接口返回支付相关参数给商户后台，如支付跳转url（参数名“mweb\_url”），商户通过 mweb\_url调起微信支付中间页

4、中间页进行H5权限的校验，安全性检查（此处常见错误请见下文）

5、如支付成功，商户后台会接收到微信侧的异步通知

6、用户在微信支付收银台完成支付或取消支付,返回商户页面（默认为返回支付发起页面）

7、商户在展示页面，引导用户主动发起支付结果的查询

8、商户后台判断是否接到收微信侧的支付结果通知，如没有，后台调用我们的订单查询接口确认订 单状态

9、展示最终的订单支付结果给用户

▶**微信支付订单状态**

1、待买家支付——买家还未完成支付

2、订单已关闭——订单已作废

3、买家已支付——买家已完成支付

4、交易结束——交易已完成

5、全额退款——已将订单的全额退还给用户

**网络和协议**

▶**CSRF**

CSRF（Cross-site request forgery）跨站请求伪造，误点攻击链接，获取cookie伪装成受信任用 户的请求来利用受信任的网站

解决 ：瞬时的授权方法。即使点击非法链接发送请求到服务端，非法请求不会携带 token 的

▶**XSS**

跨站脚本攻击(Cross Site Scripting)。恶意攻击者往Web页面里插入恶意的Script代码，当用户 浏览该页之时，嵌入其中Web里面的Script代码会被执行，从而达到恶意攻击用户的目的。

▶**TCP**

◇三次握手

• 客户端–发送带有 SYN 标志的数据包–一次握手–服务端

• 服务端–发送带有 SYN/ACK 标志的数据包–二次握手–客户端

• 客户端–发送带有带有 ACK 标志的数据包–三次握手–服务端

◇四次挥手

• 客户端-发送一个 FIN，用来关闭客户端到服务器的数据传送

• 服务器-收到这个 FIN，它发回一 个 ACK，确认序号为收到的序号加1 。和 SYN 一样，一个 FIN将占用一个序号

• 服务器-关闭与客户端的连接，发送一个FIN给客户端

• 客户端-发回 ACK 报文确认，并将确认序号设置为收到序号加1

▶**TCP,UDP 协议的区别**

UDP 在传送数据之前不需要先建立连接，远地主机在收到 UDP 报文后，不需要给出任何确认。一般 用于即时通信，比如： QQ 语音、 QQ 视频 、直播等等

TCP 提供面向连接的服务。在传送数据之前必须先建立连接，数据传送结束后要释放连接。 TCP 不 提供广播或多播服务。一般用于文件传输、发送和接收邮件、远程登录等场景。

▶**单点登录 SSO**

1、用户访问系统1的受保护资源，系统1发现用户未登录，跳转至sso认证中心，并将自己的地址1 作为参数；

2、sso认证中心发现用户未登录，将用户引导至登录页面；

3、用户输入用户名密码提交登录申请；

4、sso认证中心校验用户信息，创建用户与sso认证中心之间的会话，称为全局会话，同时创建授权 令牌；

5、sso认证中心带着令牌跳转会最初的请求地址（系统1）；

6、系统1拿到令牌，去sso认证中心校验令牌是否有效；

7、sso认证中心校验令牌，返回有效，注册系统1；

8、系统1使用该令牌创建与用户的会话，称为局部会话，返回受保护资源；

9、用户访问系统2的受保护资源；

10、系统2发现用户未登录，跳转至sso认证中心，并将自己的地址作为参数；

11、sso认证中心发现用户已登录，跳转回系统2的地址，并附上令牌；

12、系统2拿到令牌，去sso认证中心校验令牌是否有效；

13、sso认证中心校验令牌，返回有效，注册系统2；

14、系统2使用该令牌创建与用户的局部会话，返回受保护资源

▶**CAS（Central Authentication Service） ：单点登录的解决方案**

◇概念

• CAS Client：需要集成单点登录的应用，称为单点登录客户端

• CAS Server：单点登录服务器，用户登录鉴权、凭证下发及校验等操作

• TGT：ticket granting ticket，用户凭证票据，用以标记用户凭证，用户在单点登录系统中 登录一次后，再其有效期内，TGT即代表用户凭证，用户在其它client中无需再进行二 次登录操作，即可共享单点登录系统中的已登录用户信息

• ST：service ticket 服务票据，服务可以理解为客户端应用的一个业务模块，体现为客户端 回调url，CAS用以进行服务权限校验，即CAS可以对接入的客户端进行管控

• TGC：ticket granting cookie，存储用户票据的cookie，即用户登录凭证最终映射的cookies

◇原理

• 用户在浏览器中访问应用

• 应用发现需要索要用户信息，跳转至SSO服务器

• SSO服务器向用户展示登录界面，用户进行登录操作，SSO服务器进行用户校验后，映射出TGC

• SSO服务器向回调应用服务url，返回ST

• 应用去SSO服务器校验ST权限及合法性

• SSO服务器校验成功后，获取用户信息

▶**OAuth2**

◇原理

• 用户打开客户端以后，客户端要求用户给予授权。

• 用户同意给予客户端授权。

• 客户端使用上一步获得的授权，向认证服务器申请令牌。

• 认证服务器对客户端进行认证以后，确认无误，同意发放令牌。

• 客户端使用令牌，向资源服务器申请获取资源。

• 资源服务器确认令牌无误，同意向客户端开放资源。

◇四种授权方式

• 授权码模式（authorization code）

1、登陆

2、/oauth/authorize后重定向code结果

3、/oauth/token

• 简化模式（implicit）

1、登陆

2、/oauth/authorize后重定向token结果

没有refresh\_token

• 密码模式（resource owner password credentials）

1、/oauth/token

• 客户端模式（client credentials）

1、/oauth/token

没有refresh\_token

▶**OSI网络分层协议**



▶**设计模式**

◇23种设计模式

• 工厂模式

应用实例：您需要一辆汽车搬东西，而不用去管这辆汽车是怎么做出来的，以及这个汽车 里面的具体实现。

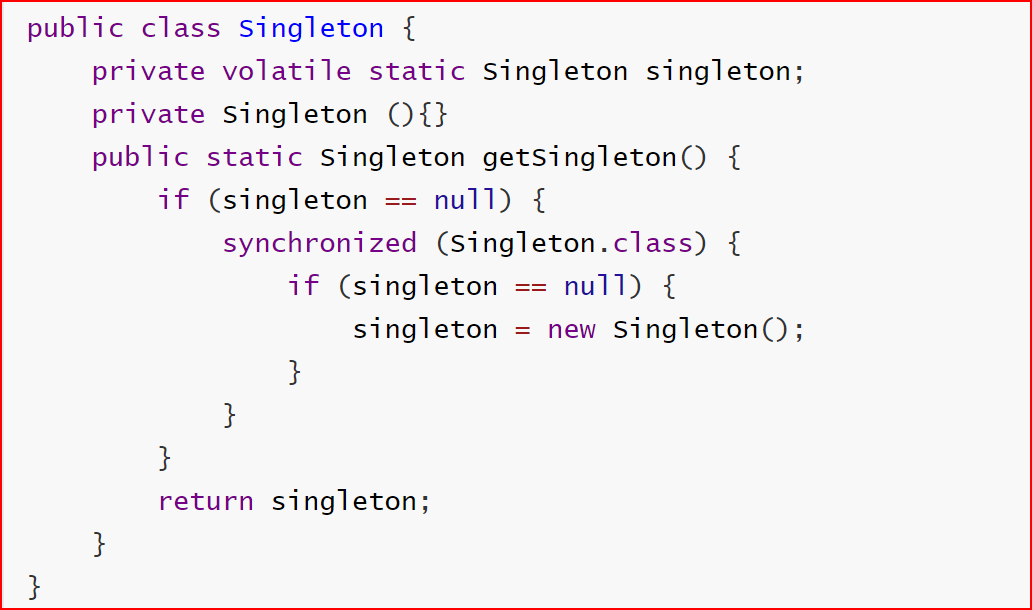
• 抽象工厂模式

应用实例：衣柜里面上衣，裤子，上衣有衬衫、外套；裤子有裤衩、牛仔裤

• 单例模式

应用实例：一个班只有一个班主任

代码



• 建造者模式

应用实例：肯德基里面的薯条、汉堡、可乐、鸡翅是不变的，它们的组合经常变化

• 原型模式

应用实例：细胞分裂

• 适配器模式

应用实例：在linux上运行windows程序

• 桥接模式

使用场景：两个独立变化的维度

• 过滤器模式

实现使用：使用不同的标准来过滤一组对象，通过逻辑运算以解耦的方式把它们连接起来

• 组合模式

应用实例：表单中由输入框、单选、多选、按钮组成

• 装饰器模式

应用实例：画需要有画框装饰才能挂在墙上

• 外观模式

应用实例：医院看病，可能要去挂号、门诊、取药，如果有提供接待人员，只让接待人员 来处理，就很方便。

• 享元模式

应用实例：数据库连接池

• 代理模式

应用实例：远程连接-翻墙、Nginx、Vpn

• 责任链模式

应用实例：JS事件冒泡

• 命令模式

应用实例：模式cmd指令、请求或者指令封装成对象

• 解释器模式

应用实例：编译器、运算表达式计算。

• 迭代器模式

应用实例：JAVA 中的 iterator

• 中介者模式

应用实例：交易要在市场中进行，市场作为中介

代码

• 备忘录模式

应用实例：打游戏时的存档，（tcc中try阶段也很像）

• 观察者模式

使用：一个对象的状态发生改变，所有的依赖对象都将得到通知（spring listener）

• 状态模式

使用场景：行为随状态改变而改变的场景

关键：状态对象化

• 策略模式

使用场景：旅行的出游方式，选择骑自行车、坐汽车，每一种旅行方式都是一个策略

关键：策略对象化

• 模板模式

使用：多个子类共有方法

• 访问者模式

使用：对象结构中对象对应的类很少改变，但经常需要在此对象结构上定义新的操作

◇设计模式的六大原则

• 开闭原则

对扩展开放，对修改关闭

• 单一职责原则

每个类应该实现单一的职责

• 里氏替换原则

任何基类可以出现的地方，子类一定可以出现。

子类对父类的方法尽量不要重写和重载。因为父类代表了定义好的结构，通过这个规范的 接口与外界交互，子类不应该随便破坏它。

• 依赖倒转原则

面向接口编程，依赖于抽象而不依赖于具体。

写代码时用到具体类时，不与具体类交互，而与具体类的上层接口交互。

• 接口隔离原则

使用多个隔离的接口，比使用单个接口要好

• 迪米特法则（最少知道原则）

无论被依赖的类多么复杂，都应该将逻辑封装在方法的内部，降低类的耦合度

• 合成复用原则

尽量首先使用合成/聚合的方式，而不是使用继承。

▶**继承、实现、依赖、关联、聚合、组合的联系与区别**

继承（实线三角）：继承父类

实现（虚线三角）：实现接口

关联（实现箭头）：属性

  聚合（虚四角-实线箭头） 集体和个体关系

  组合（实四角-实线箭头） 整体和部分关系

依赖（虚线箭头）：方法参数

**场景问题**

▶**十万个数据,找出重复次数最多的十个数据并打印**

先通过Hashmap存储, key为数据,value为它出现的次数.然后用优先级队列,存储类型为Map.Entry, 重写比较器类,利用value进行比较.

优先级队列中利用小根堆形式,只存储十个.

接下来遍历其他的,如果比小根堆堆顶的要大,对优先级队列进行remove和add操作.

最终,优先级队列中只会存储下来十个value最大的Entry

代码：

略

▶**4亿个int数，如何找出重复的数**

用hash方法，建一个2的32次方个bit的hash数组，每取一个int数，可hash下2的32次方找 到它在hash数组中的位置，然后将bit置1表示已存在

▶**4亿个url，找出其中重复的**

考虑内存不够，通过hash算法，将url分配到1000个文件中，不同的文件间肯定就不会重复了，再 分别找出重复的

**项目问题**

▶**缓存击穿问题**

查询一个数据库中不存在的数据

**解决：**

**访问频次不大**

当通过某一个key去查询数据的时候，如果对应在数据库中的数据都不存在，我们将此key对 应的value设置为一个默认的值，比如“NULL”，并设置一个缓存的失效时间，这时在缓存失 效之前，所有通过此key的访问都被缓存挡住了。后面如果此key对应的数据在DB中存在时， 缓存失效之后，通过此key再去访问数据，就能拿到新的value了

**互斥锁**

**代码：**



▶**Java poi海量数据导出内存溢出问题**

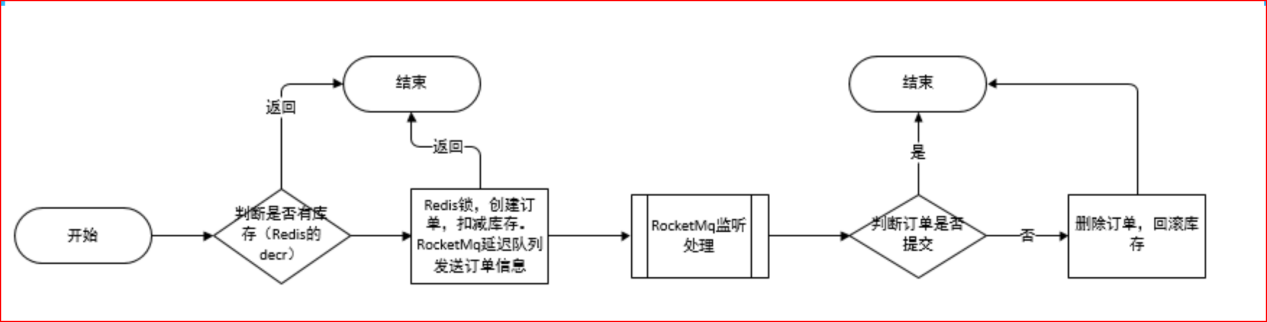
当数据量超出65536条后，在使用HSSFWorkbook或XSSFWorkbook，程序会报OutOfMemoryError： Javaheap space;内存溢出错误。

解决： new SXSSFWorkbook(1000); 内存中保留 1000 条数据，以免内存溢出，其余写入 硬盘

▶**SnowFlake的算法生成的id由于过长导致JS拿到的时候精度丢失（转成字符类型）**

▶**分布式系统中Mybatis需要关闭一级缓存，否则会造成脏读**

▶**秒杀**



▶**如果让你写一个消息队列，该如何进行架构设计？**

◇可伸缩性

消息队列支持可伸缩性，可以快速扩容，增加吞吐量和容量。参照一下 kafka 的设计理念，消 息通过topic发送给broker，每个topic对应多个partition，给topic增加 partition，做 数据迁移，增加机器。

◇持久化

消息队列的的数据要持久化，要利用磁盘顺序读写的性能、多路io复用，零拷贝等特性

◇可用性

消息队列的可用性， kafka的高可用保障机制。partition有多副本，topic维护副本集合，副 本集合中leader、follower。leader对外服务，挂了从副本集合中选举新的leader。

◇可靠性

消息队列的可靠性，确保生产者将消息发送给broker（给生产者的确认反馈），确保消费者将 消息从broker消费掉（手动确认）

▶**RPC流程**

① Client以本地调用的方式调用服务

② Client Stub接收到调用后，把服务调用相关信息组装成需要网络传输的消息体，并找到服务地址 （host:port），对消息进行编码后交给Connector进行发送

③ Connector通过网络通道发送消息给Acceptor

④ Acceptor接收到消息后交给Server Stub

⑤ Server Stub对消息进行解码，并根据解码的结果通过反射调用本地服务

⑥ Server执行本地服务并返回结果给Server Stub

⑦ Server Stub对返回结果组装打包并编码后交给Acceptor进行发送

⑧ Acceptor通过网络通道发送消息给Connector

⑨ Connector接收到消息后交给Client Stub，Client Stub接收到消息并进行解码后转交给Client

⑩ Client获取到服务调用的最终结果

▶**如何设计RPC框架**

◇序列化/反序列化

首先排除Java的ObjectInputStream和ObjectOutputStream，因为不仅需要保证需要序列化或 反序列化的类实现Serializable接口，还要保证JDK版本一致，公司应用So Many，使用的语 言也众多，这显然是不可行的，考虑再三，决定采用Objesess。

◇通信技术

同样我们首先排除Java的原生IO，因为进行消息读取的时候需要进行大量控制，如此晦涩难用， 正好近段时间也一直在接触Netty相关技术，就不再纠结，直接命中Netty。

◇高并发技术

远程调用技术一定会是多线程的，只有这样才能满足多个并发的处理请求。这个可以采用JDK 提供的Executor。

◇服务注册与发现

Zookeeper。当Server启动后，自动注册服务信息（包括host,port,还有nettyPort）到ZK中； 当Client启动后，自动订阅获取需要远程调用的服务信息列表到本地缓存中。

◇负载均衡

分布式系统都离不开负载均衡算法，好的负载均衡算法可以充分利用好不同服务器的计算资源， 提高系统的并发量和运算能力。

◇非侵入式

借助于Spring框架

▶**微服务**

◇微服务的利弊

利

模块化，边界清楚

独立部署

弊

分布式复杂

运维复杂性

◇微服务设计

微服务逻辑分层

微服务基础服务层

微服务聚合服务层

网关

Zuul

Gateway

Nginx

App/H5

◇分布式事务

两阶段

三阶段

TCC

消息队列

◇熔断限流隔离降级

Hystrix

Sentinel

◇集中式配置中心

Spring Cloud Config

百度的 Disconf

阿里的 Diamond

携程的 Apollo

◇调用链监控&日志

Zipkin

Cat

Skywalking

◇容器化部署

Docker

Kubernetes

◇部署到生产，预估容量

评估访问量

评估平均访问量 QPS

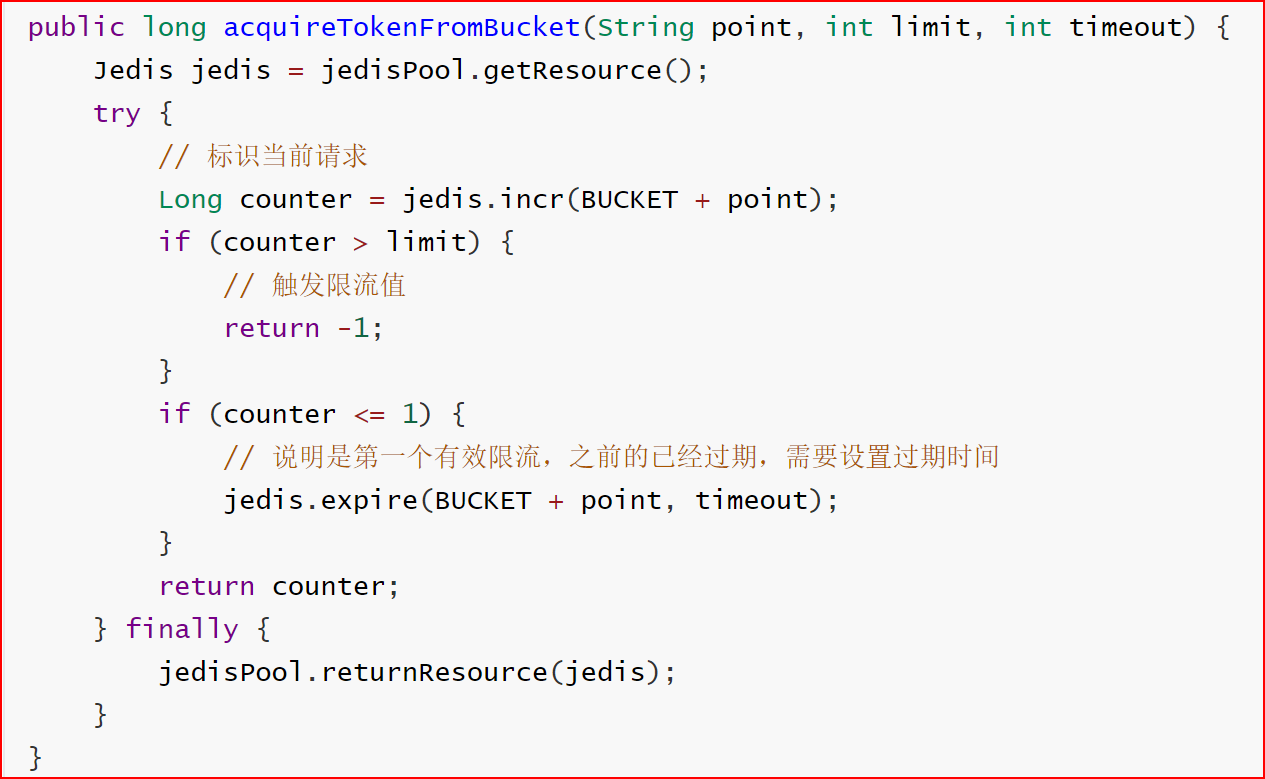
评估高峰 QPS

评估系统，单机极限 QPS

▶**限流策略**

◇固定窗口算法

限制固定时间段内，特定api的最大请求量，如限制每秒最多请求100次，超出100次触发限 流策略。



◇滑动窗口算法

将一个时间区间划分为N个长度固定的小时间窗口，每一次请求考察当前时间往前N个连续时 间窗口的请求量总和，如果超过阈值则触发限流策略。如配置请求阈值100，将1s时间划分为 5个连续的时间窗口，每个窗口长度250ms，则意味当前250ms 加 上一秒的最后750ms请求量 不会100个请求。

◇令牌桶算法

存在一个令牌桶计数器，每隔一定时间放入一块令牌，每次请求取出一块令牌。，令牌桶有最 大容量限制，当计数器令牌数达到最大值，则不再放入，如果计数器令牌数为0，请求则触发限 流策略。如每隔10ms放入一块令牌，最大令牌数为100，假设某个时刻令牌桶放满令牌，则这 个时刻的前10ms能请求100次，清20ms能请求101次。

◇漏桶算法

相对于令牌桶是固定速率放入令牌，在没有令牌的时候拒绝请求，漏桶则是固定数据漏出请求， 当请求量过大，流入请求超过桶容量，则触发限流策略。具体实现时，可以将漏桶想象成一个 有流量限制的先进先出的队列，在队列不为空情况下，每隔一定时间取出一个队列请求进行处 理相应，如果队列已经满了，再来请求则触发限流相关策略。如限制每10ms漏出一个请求，桶 容量为100个请求，则每10ms最多能处理一个请求。