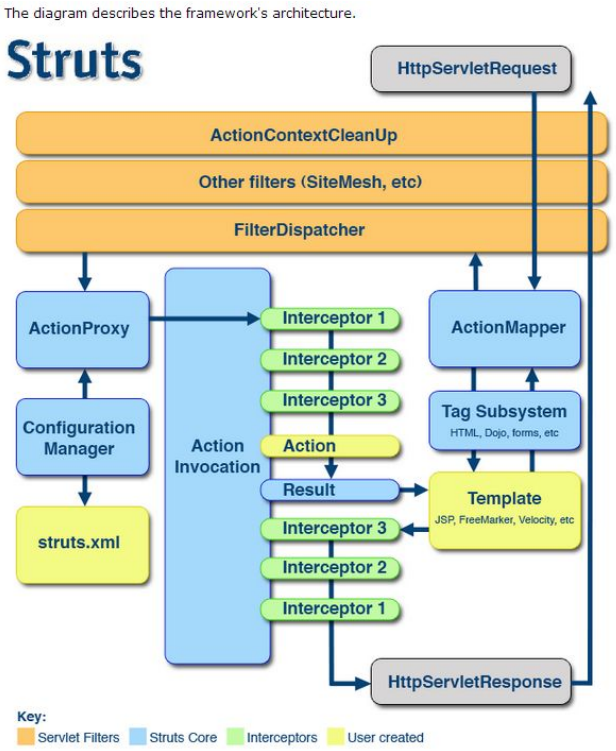
1. 描述一下**Hibernate**的三个状态?

a. transient(瞬时状态)：new出来一个对象，还没被保存到数据库中

b. persistent(持久化状态)：对象已经保存到数据库中并且在hibernate session也存在该对象

c. detached(离线状态)：对象在数据库中存在，hibernate session不存在

2. struts工作流程



1、客户端浏览器发出HTTP请求。

2、根据web.xml配置，该请求被FilterDispatcher接收。

3、根据struts.xml配置，找到需要调用的Action类和方法， 并通过IoC方式，将值注入给Aciton。

4、Action调用业务逻辑组件处理业务逻辑，这一步包含表单验证。

5、Action执行完毕，根据struts.xml中的配置找到对应的返回结果result，并跳转到相应页面。

6、返回HTTP响应到客户端浏览器。

3. Hibernate对一二级缓存的使用， **Lazy-Load**的理解;

a. 一级缓存：hibernate的一级缓存是由session提供的，因此它只存在session的生命周期中。也就是说session关闭的时候该session所管理的一级缓存也随之被清除。hibernate的一级缓存是session所内置的，默认开启，不能被卸载，也不能进行任何配置。在缓存中的对象,具有持久性,session对象负责管理.一级缓存的优点是使用同一个session对象多次查询同一个数据对象,仅对数据库查询一次。一级缓存采用的是Key-Value的MAP方式来实现的。在缓存实体对象时，对象的主关键字ID是MAP的Key，实体对象就是对象的值。所以说一级缓存是以实体对象为单位进行存储的。访问的时候使用的是主键关键字ID。一级缓存使用的是自动维护的功能。但可以通过session提供的手动方法对一级缓存的管理进行手动干预。evict()方法用于将某个对象从session的一级缓存中清除。clear()方法用于将session缓存中的方法全部清除。

b. 二级缓存：二级缓存的实现原理与一级缓存是一样的。也是通过Key-Value的Map来实现对对象的缓存。二级缓存是作用在SessionFactory范围内的。因此它可被所有的Session对象所共享。需要注意的是放入缓存中的数据不能有第三方的应用对数据进行修改。二级缓存默认关闭，需要程序员手动开启，默认为ehcache实现.

c. 懒加载：当用到数据的时候才向数据库查询，这就是hibernate的懒加载特性。延迟加载策略能避免加载应用程序不需要访问的关联对象，以提高应用程序的性能。

4. mybatis如何实现批量提交?

a. 通过标签：

```
insert into sys_user_role
(
```

```
`user_id`,
`role_id`
)
values
<foreach collection="roleIdList" item="item" index="index" separator=",">
    (
    #{userId},
     #{item}
     )
</foreach>
```

b. 通过ExecutorType.BATCH：

```
public void batchInsert(List<T> list, Class<? extends SqlMapper<T>> mapper) {
    sessionTemplate = new SqlSessionTemplate(sqlSessionFactory,ExecutorType.BATCH);

    SqlSession session = sqlSessionFactory.openSession(ExecutorType.BATCH,false);

    try {
      for (T vo : list) {
        sessionTemplate.getMapper(mapper).insert(vo);
      }
       session.commit();
     } catch (Exception e) {
       session.rollback();
     } finally {
       session.close();
     }
}
```

5. session机制?

1．session是服务器的生成,并传至客户端浏览器,后续请求,都会通过URL重写传至服务器进行session比较.

2．session是基于cookie

3．session可以保存用户信息,但cookie如果浏览器被禁用,则无法保存用户信息

4．如果浏览器禁用会话cookie，则每次请求都无法将第一次请求获得的sessionId传至后台服务器.所以每次请求刷新页面服务器都会生成新的sessionid给到浏览器

5．因HTTP协议为无状态的协议（一旦数据交互完毕，客户端和服务端的连接就会关闭，再次交换数据时需要建立新的连接），需要通过session或者cookie保存和跟踪用户信息

6、sessionID放在浏览器客户端cookie,其它信息放在服务器内存中，也可以做持久化管理memcached、redis中；

6. Struts2表单重复提交问题（token拦截器）

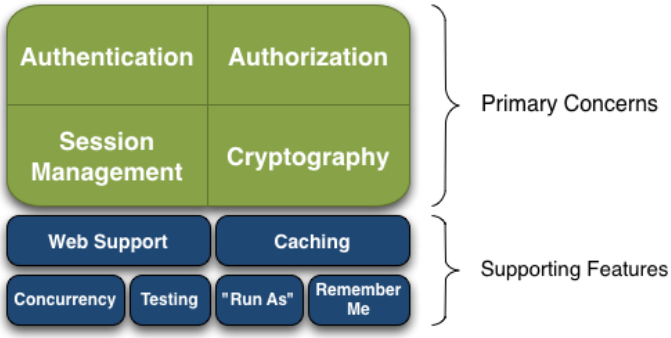
1.访问页面前保存token(服务器后台生成的一串序列时间串放到session中)，并传至前台jsp页面中的隐藏域

2.提交时验证token,将前台的隐藏的token传至后台进行验证是否一致，提交随机生成新的token，可以防止重复提交；

7. shiro：

1、定义：apache shiro是java的一个安全框架，简单易用，基本功能有：认证、授权、加密、会话管理、与Web集成、缓存等。Shiro不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给Shiro即可(五张表)。

2、功能点：



Authentication：身份认证/登录，验证用户是不是拥有相应的身份；

Authorization：授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；

Session Manager：会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通JavaSE环境的，也可以是如Web环境的；

Cryptography：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；

Web Support：Web支持，可以非常容易的集成到Web环境；

Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；

Concurrency：shiro支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；

Testing：提供测试支持；

Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

Remember Me：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

3、工作流程：

1、应用代码通过Subject(主体，代表当前"用户")来进行认证和授权，而Subject又委托给SecurityManager(安全管理器，shiro核心)；

2、我们需要给Shiro的SecurityManager注入Realm(域，Shiro从Realm获取安全数据（如用户、角色、权限）)，从而让SecurityManager能得到合法的用户及其权限进行判断。

8. Tomcat Filter过滤器责任链模式,过滤器拦截器区别?

1、责任链模式：将一个事件处理流程分派到一组执行对象上去，这一组执行对象形成一个链式结构，事件处理请求在这一组执行对象上进行传递。

2、过滤器和拦截器：

1、过滤器filter：是在java web中，你传入的request,response提前过滤掉一些信息，或者提前设置一些参数，然后再传入servlet或者struts的 action进行业务逻辑，比如过滤掉非法url（不是login.do的地址请求，如果用户没有登陆都过滤掉），或者在传入servlet或者 struts的action前统一设置字符集，或者去除掉一些非法字符；

2、拦截器interceptor：是在面向切面编程的就是在你的service或者一个方法，前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。拦截是AOP的一种实现策略；

3、区别：

a. 拦截器是基于java的反射机制的，而过滤器是基于函数回调。

b. 拦截器不依赖于servlet容器，过滤器依赖与servlet容器。

c. 拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。

d. 拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。

e. 在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次

9. Git与Svn的区别：

1、Git是分布式的，而Svn不是；

2、GIT把内容按元数据方式存储，而SVN是按文件

3、分支不同：git分支切换很方便；svn分支就是版本库的另外一个目录；

4、GIT没有一个全局的版本号，而svn有，SVN的版本号实际是任何一个相应时间的源代码快照。

5、GIT的内容完整性要优于SVN(GIT的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。)

10. Git命令底层原理：

https://www.jianshu.com/p/2b47a3078a46

1. git init：使用git init初始化一个新的目录时，会生成一个.git的目录，该目录即为本地仓库。一个新初始化的本地仓库

是这样的：

```
├── HEAD
├── branches
├── config
├── description
├── hooks
├── objects
|   ├── info
|   └── pack
└── refs
    ├── heads
    └── tags
```

- description用于GitWeb程序
- config配置特定于该仓库的设置（还记得git config的三个配置级别么）
- hooks放置客户端或服务端的hook脚本
- HEAD传说中的**HEAD指针**，指明当前处于哪个分支
- objectsGit对象存储目录
- refsGit引用存储目录
- branches放置分支引用的目录

其中description、config和hooks这些不在讨论中，后文会直接忽略。

1. git add：Git commit之前先要通过git add添加文件：

```
├── HEAD
├── branches
├── index
├── objects
|   ├── 9f
|   |   └── 4d96d5b00d98959ea9960f069585ce42b1349a
|   ├── info
|   └── pack
└── refs
    ├── heads
    └── tags
```

可以看到，多了一个index文件。并且在objects目录下多了一个9f的目录，其中多了一个4d96d5b00d98959ea9960f069585ce42b1349a文件。

其实9f4d96d5b00d98959ea9960f069585ce42b1349a就是一个Git对象，称为blob对象。

1. git commit：

```
├── HEAD
├── branches
├── index
├── logs
|   ├── HEAD
|   └── refs
|       └── heads
|           └── master
├── objects
|   ├── 88
|   |   └── 23efd7fa394844ef4af3c649823fa4aedefec5
|   ├── 91
|   |   └── 0fc16f5cc5a91e6712c33aed4aad2cfffccb73
|   ├── 9f
|   |   └── 4d96d5b00d98959ea9960f069585ce42b1349a
|   ├── info
|   └── pack
└── refs
    ├── heads
    |   └── master
    └── tags
```

11. JSP的执行过程：

1）客户端发出请求。

2）Web容器将JSP转译成Servlet源代码。

3）Web容器将产生的源代码进行编译。

4）Web容器加载编译后的代码并执行。

5）把执行结果响应至客户端。

12. ZK高可用：

1. ZooKeeper 运行期间，集群中至少有过半的机器保存了最新数据。集群超过半数的机器能够正常工作，集群就能够对外提供服务。

13. zookeeper有什么功能，选举算法如何进行：

1、选举算法：

1、Fast Leader(领导者选举)选举算法：

1.server启动时默认选举自己，并向整个集群广播

2.收到消息时，通过3层判断：选举轮数，zxid，server id大小判断是否同意对方，如果同意，则修改自己的选票，并向集群广播

3.QuorumCnxManager负责IO处理，每2个server建立一个连接，只允许id大的server连id小的server，每个server启动单独的读写线程处理，使用阻塞IO

4.默认超过半数机器同意时，则选举成功，修改自身状态为LEADING或FOLLOWING

5.Obserer机器不参与选举

2、原理：选举结果的影响权重关系是:首先看数据id,数据id大者胜出;其次再判断leader id,leader id大者胜出。

3、举例：

假设有五台服务器组成的zookeeper集群,它们的id从1-5,同时它们都是最新启动的,也就是没有历史数据,在存放数据量这一点上,都是一样的.假设这些服务器依序启动,来看看会发生什么.

1) 服务器1启动,此时只有它一台服务器启动了,它发出去的报没有任何响应,所以它的选举状态一直是LOOKING状态；

2) 服务器2启动,它与最开始启动的服务器1进行通信,互相交换自己的选举结果,由于两者都没有历史数据,所以id值较大的服务器2胜出,但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是3),所以服务器1,2还是继续保持LOOKING状态.

3) 服务器3启动,根据前面的理论分析,服务器3成为服务器1,2,3中的老大,而与上面不同的是,此时有三台服务器选举了它,所以它成为了这次选举的leader.

4) 服务器4启动,根据前面的分析,理论上服务器4应该是服务器1,2,3,4中最大的,但是由于前面已经有半数以上的服务器选举了服务器3,所以它只能接收当小弟的命了.

5) 服务器5启动,同4一样,当小弟.

4、zookeeper管理员指南：

1、集群中过半存活即可用，故集群选择奇数台机器；

14. RPC、RMI、MQ、SOAP：

1、RPC:远程过程调用协议，采用C/S模式，分布式跨语言平台，更多用于同步调用，比如Web Service(SOAP)；

2、RMI:远程方法调用，依赖于java远程消息交换协议，要求服务端与客户端都为java编写；每个方法都具有方法签名，只有签名匹配才可以调用，返回值是基本类型和对象；

3、MQ:队列，更多用于异步传输；

4、SOAP最主要的工作是使用标准的XML描述了RPC的请求信息(URI/类/方法/参数/返回值)。理论上，SOAP就是一段xml，你可以通过http,smtp等发送它(复制到软盘上，叫快递公司送去也行?)。同样SOAP也是跨语言的。

15. Netty高性能：

1、NIO异步非阻塞通信

2、“零拷贝”

3、内存池ByteBuf

4、Netty提供了多种内存管理策略，通过在启动辅助类中配置相关参数，可以实现差异化的定制。

5、高效的Reactor线程模型：Reactor单线程(多线程、主从)模型，指的是所有的IO操作都在同一个NIO线程上面完成

6、为了尽可能提升性能，Netty采用了串行无锁化设计，在IO线程内部进行串行操作，避免多线程竞争导致的性能下降。表面上看，串行化设计似乎CPU利用率不高，并发程度不够。但是，通过调整NIO线程池的线程参数，可以同时启动多个串行化的线程并行运行，这种局部无锁化的串行线程设计相比一个队列-多个工作线程模型性能更优。

7、高效的并发编程：Netty的高效并发编程主要体现在如下几点：

1) volatile的大量、正确使用；

2) CAS和原子类的广泛使用；

3) 线程安全容器的使用；

4） 通过读写锁提升并发性能。

8、高效的序列化框架：

9、灵活的TCP参数配置能力：合理设置TCP参数在某些场景下对于性能的提升可以起到显著的效果，例如SO_RCVBUF和SO_SNDBUF。如果设置不当，对性能的影响是非常大的。

16. 如何保证服务的幂等性?

1、概念：接口的幂等性实际上就是接口可重复调用，在调用方多次调用的情况下，接口最终得到的结果是一致的。有些接口可以天然的实现幂等性，比如查询接口，对于查询来说，你查询一次和两次，对于系统来说，没有任何影响，查出的结果也是一样。

2、GET幂等：值得注意，幂等性指的是作用于结果而非资源本身。怎么理解呢？例如，这个HTTP GET方法可能会每次得到不同的返回内容，但并不影响资源。

3、POST非幂等：因为它会对资源本身产生影响，每次调用都会有新的资源产生，因此不满足幂等性。

4、如何保证幂等性：

1、全局唯一id：如果使用全局唯一ID，就是根据业务的操作和内容生成一个全局ID，在执行操作前先根据这个全局唯一ID是否存在，来判断这个操作是否已经执行。如果不存在则把全局ID，存储到存储系统中，比如数据库、redis等。如果存在则表示该方法已经执行。

从工程的角度来说，使用全局ID做幂等可以作为一个业务的基础的微服务存在，在很多的微服务中都会用到这样的服务，在每个微服务中都完成这样的功能，会存在工作量重复。另外打造一个高可靠的幂等服务还需要考虑很多问题，比如一台机器虽然把全局ID先写入了存储，但是在写入之后挂了，这就需要引入全局ID的超时机制。

使用全局唯一ID是一个通用方案，可以支持插入、更新、删除业务操作。但是这个方案看起来很美但是实现起来比较麻烦，下面的方案适用于特定的场景，但是实现起来比较简单。

2、去重表：这种方法适用于在业务中有唯一标的插入场景中，比如在以上的支付场景中，如果一个订单只会支付一次，所以订单ID可以作为唯一标识。这时，我们就可以建一张去重表，并且把唯一标识作为唯一索引，在我们实现时，把创建支付单据和写入去去重表，放在一个事务中，如果重复创建，数据库会抛出唯一约束异常，操作就会回滚。

3、插入或更新：这种方法插入并且有唯一索引的情况，比如我们要关联商品品类，其中商品的ID和品类的ID可以构成唯一索引，并且在数据表中也增加了唯一索引。这时就可以使用InsertOrUpdate操作。在mysql数据库中如下：

```
insert into goods_category (goods_id,category_id,create_time,update_time)
       values(#{goodsId},#{categoryId},now(),now())
       on DUPLICATE KEY UPDATE
       update_time=now()
```

4、多版本控制：这种方法适合在更新的场景中，比如我们要更新商品的名字，这时我们就可以在更新的接口中增加一个版本号，来做幂等

```
boolean updateGoodsName(int id,String newName,int version);
```

在实现时可以如下

```
update goods set name=#{newName},version=#{version} where id=#{id} and version<${version}
```

5、状态机控制：这种方法适合在有状态机流转的情况下，比如就会订单的创建和付款，订单的付款肯定是在之前，这时我们可以通过在设计状态字段时，使用int类型，并且通过值类型的大小来做幂等，比如订单的创建为0，付款成功为100。付款失败为99

在做状态机更新时，我们就这可以这样控制

```
update `order` set status=#{status} where id=#{id} and status<#{status}
```

17. zookeeper工作原理?

1、定义：zookeeper是一种为分布式应用所设计的高可用、高性能且一致的开源协调服务，它提供了一项基本服务：分布式锁服务。后来摸索出了其他使用方法：配置维护、组服务、分布式消息队列、分布式通知/协调等。

2、特点：

1、能够用在大型分布式系统中；

2、具有一致性、可用性、容错性，不会因为一个节点的错误而崩溃；

3、用途：用户大型分布式系统，作协调服务角色；

1、分布式锁应用：通过对集群进行master选举，来解决分布式系统中的单点故障（一主n从，主挂全挂）。

2、协调服务；

3、注册中心；

4、原理：

术语：

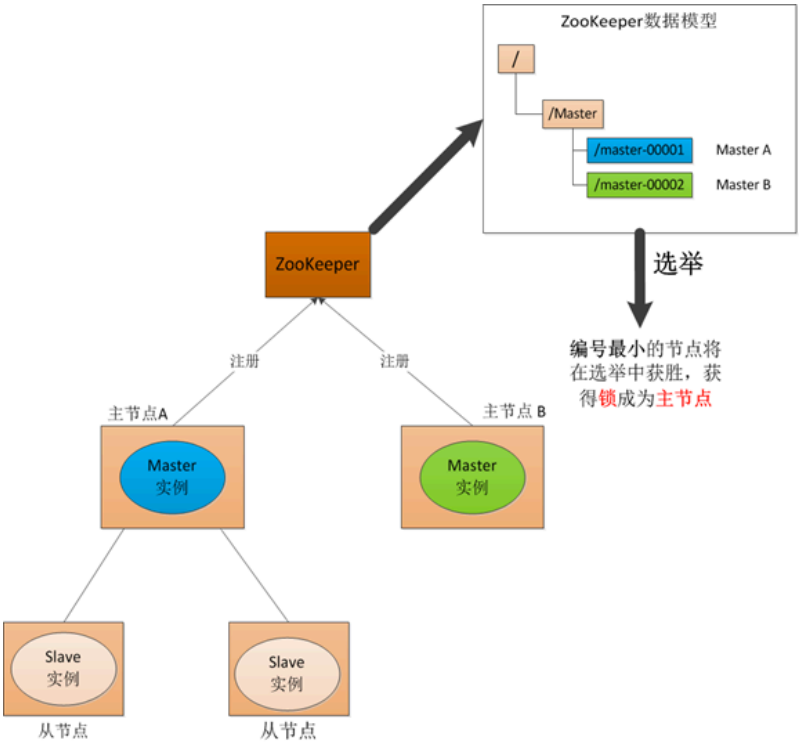
数据结构Znode：zookeeper数据采用树形层次结构，和标准文件系统非常相似，树中每个节点被称为Znode；

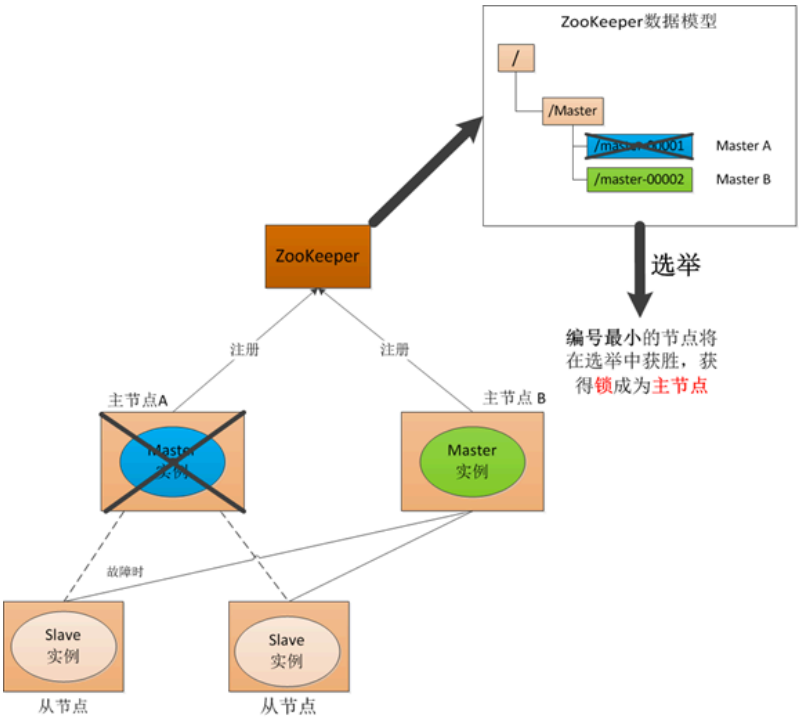
通知机制Watcher：zookeeper可以为所有的读操作（exists()、getChilden()及getData()）设置watch，watch事件是一次性出发器，当watch的对象状态发生改变时，将会触发次对象上watch所对应的事件。watch事件将被异步的发送给客户端，并且zookeeper为watch机制提供了有序的一致性保证。

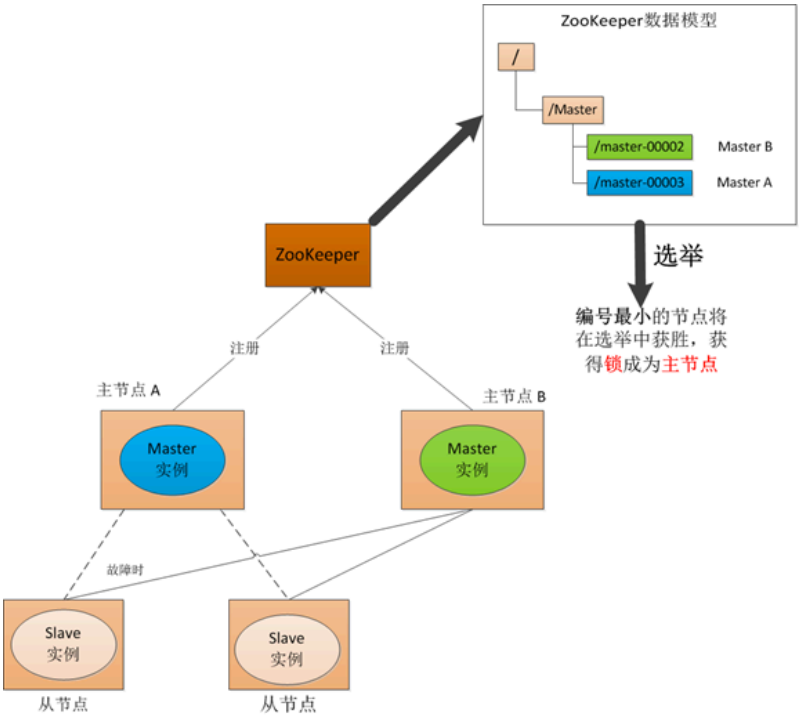
基本流程：分布式锁应用场景

1、传统的一主n从分布式系统，容易发生单点故障，传统解决方式是增加一个备用节点，定期给主节点发送Ping包，主节点回复ack，但是如果网络原因ack丢失，那么会出现两个主节点，造成数据混乱。

2、zookeeper的引入可以管理两个主节点，其中挂了一个，会将另外一个作为新的主节点，挂的节点回来时担任备用节点；







18. cap理论：

1、概念：一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项。

2、一致性：更新操作成功并返回客户端完成后，所有节点在同一时间的数据完全一致，所以，一致性，说的就是数据一致性。

3、可用性：服务一直可用，而且是正常响应时间。

4、分区容错性：分布式系统在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性和可用性的服务。

19.