

MongoDB的事务机制和数据安全

TJ（唐建法）
MongoDB 大中华区首席技术顾问

关于我

TJ Tang 唐建法

MongoDB 大中华区技术顾问

原联邦快递首席架构师

开源项目anoogose作者

1

MongoDB 事务支持

2

MongoDB 数据安全

MongoDB 事务支持

MongoDB ACID 事务支持

事务类型	MongoDB的支持	MySQL的支持 (InnoDB)
Atomicity	单行/文档级原子性	多行原子性
Consistency	强一致或最终一致	强一致
Isolation	提交读 *	可重复读
Durability	日志及复制	日志

* MongoDB 3.2 起支持

单行/文档级原子性 – 支持

```
db.users.update({ username: "tj.tang" },  
               { $set: {  
                   salary: 500,000,  
                   jobs: [ { }, { }, { } ],  
                   hours: 18  
                 }  
               });
```

All or Nothing

多行、多文档、多语句原子性 – 尚不支持

```
db.users.update({ salary: {$lt: 500,000} },  
               { $set: { salary: 500,000}  
               } });
```

username	salary
James	300,000
Julia	450,000
Melody	400,000
Frank	230,000
Taylor	560,000

开始状态

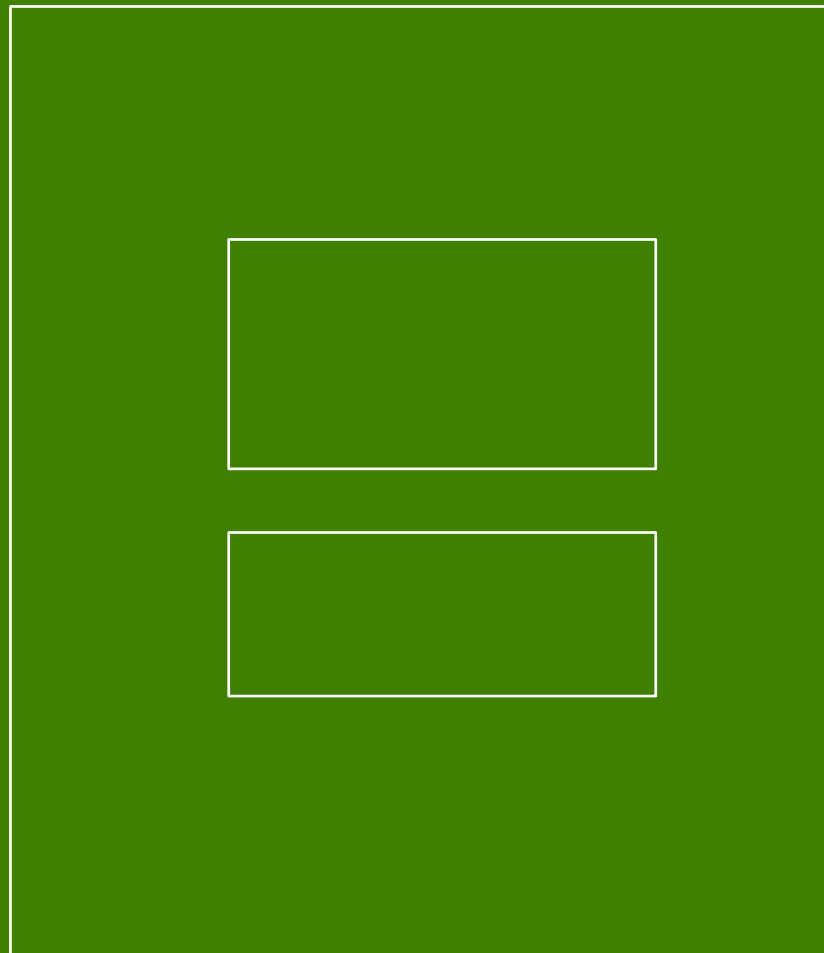
username	salary
James	500,000
Julia	500,000
Melody	500,000
Frank	230,000
Taylor	560,000

结束状态

宕机

如何处理多文档一致性

- 通过建模来避免
- 二阶段提交
- 记录日志，人工干预



一致性

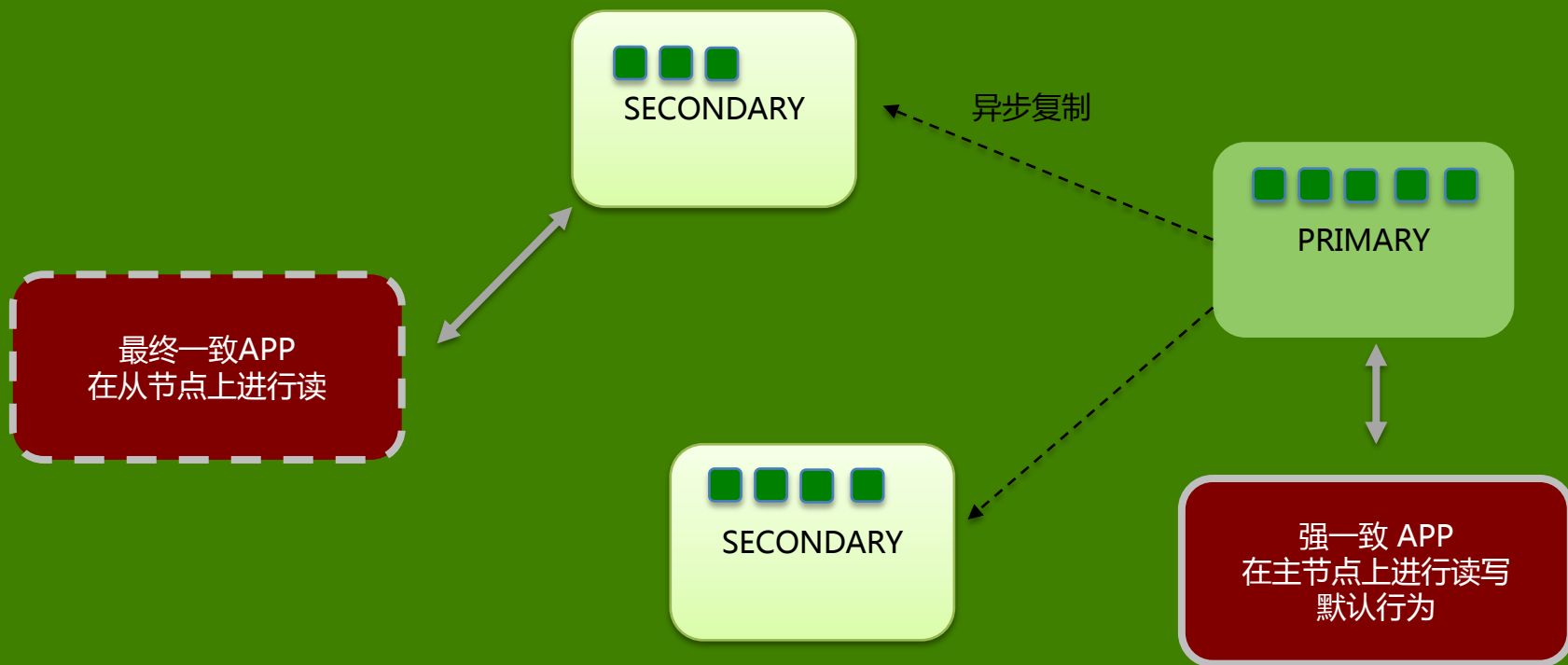
规则校验
主外键

传统数据库


多节点数据一致
Read your writes

分布式数据库

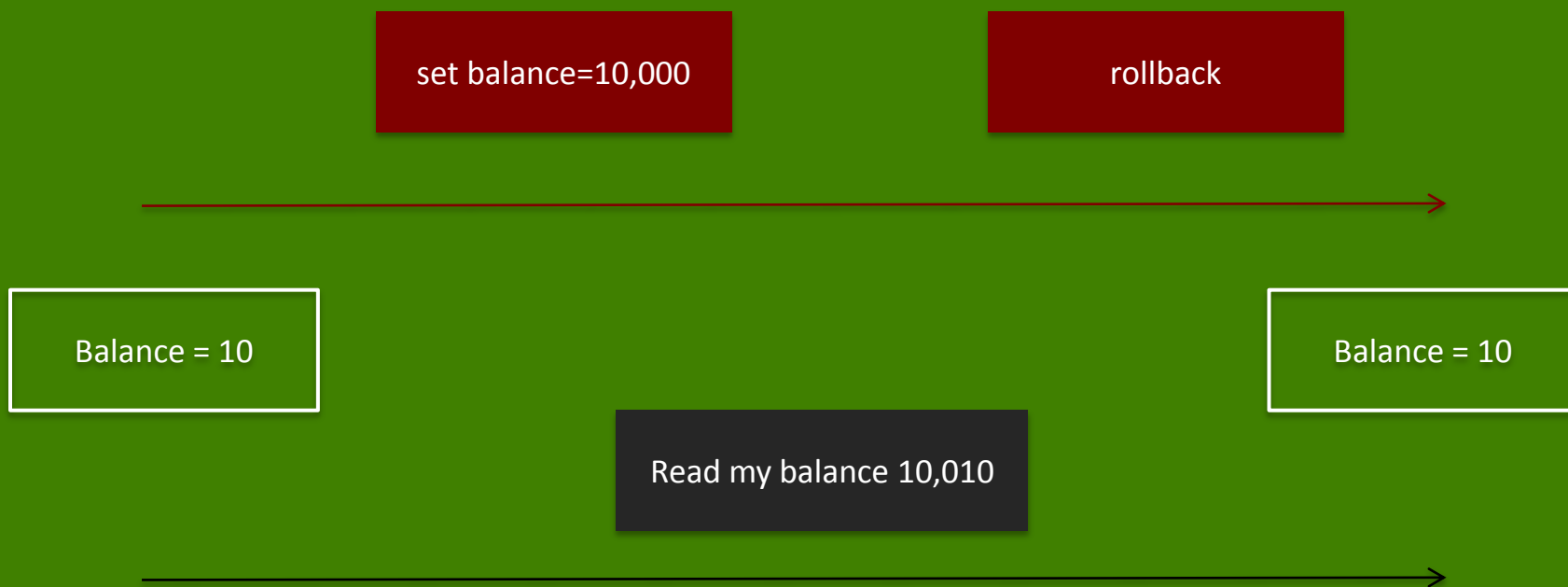
MongoDB : 可调一致性



隔离级别

ISOLATION LEVEL	默认设定
SERIALIZABLE	
REPEATABLE READ	
READ COMMITTED	 
READ UNCOMMITTED	

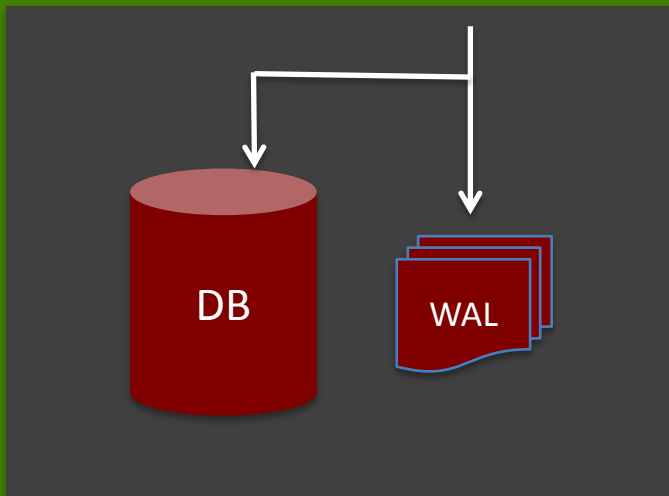
READ UNCOMMITTED 的问题（脏读）



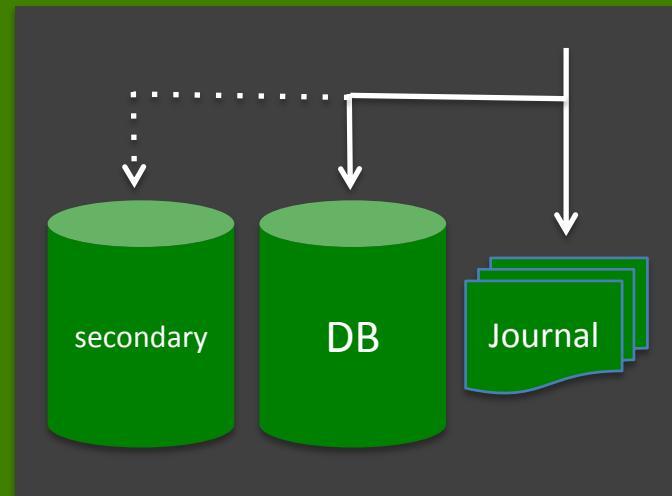
隔离级别

ISOLATION LEVEL	3.2的支持
SERIALIZABLE	
REPEATABLE READ	
READ COMMITTED	 mongoDB
READ UNCOMMITTED	 mongoDB

数据持久性 - Durability



传统数据库



MongoDB

百度为您找到相关结果约82,600个

🔍 搜索工具

[为什么MongoDB会丢数据 - longxibendi的专栏 - 博客频道 - CSDN.NET](#)

2014年10月8日 - MongoDB 丢数据的说法已经出现很久很久了,传言甚多。这里简单总结下场景。1.在MongoDB很早的版本,2.0之前,没有journal,加上默认不是安全写,系统一宕...

[blog.csdn.net/longxibe...](#) - 百度快照 - 88%好评

[别再用MongoDB了!](#)

2015年7月22日 - 近日,他在个人博客上发表了一篇博文《为什么你应该永远、永远、永远不要再使用MongoDB》。在文中,他列举了如下理由:丢失数据(见1、2); 默认忽略...

[www.infoq.com/cn/news/...](#) - 百度快照 - 89%好评

[千万别用MongoDB?真的吗? - 博客 - 伯乐在线](#)

2011年11月10日 - 某人发了一篇《Don't use MongoDB》的血泪控诉,我把原文翻译如下,你可以看看...3.主从结点间的数据复制有缺口,导致从结点丢失主结点有的数据。是的,...

[blog.jobbole.com/5701/](#) - 百度快照 - 评价

[MongoDB数据丢失问题-CSDN论坛-CSDN.NET-中国最大的IT技术社区](#)

4条回复 - 发帖时间: 2014年12月17日

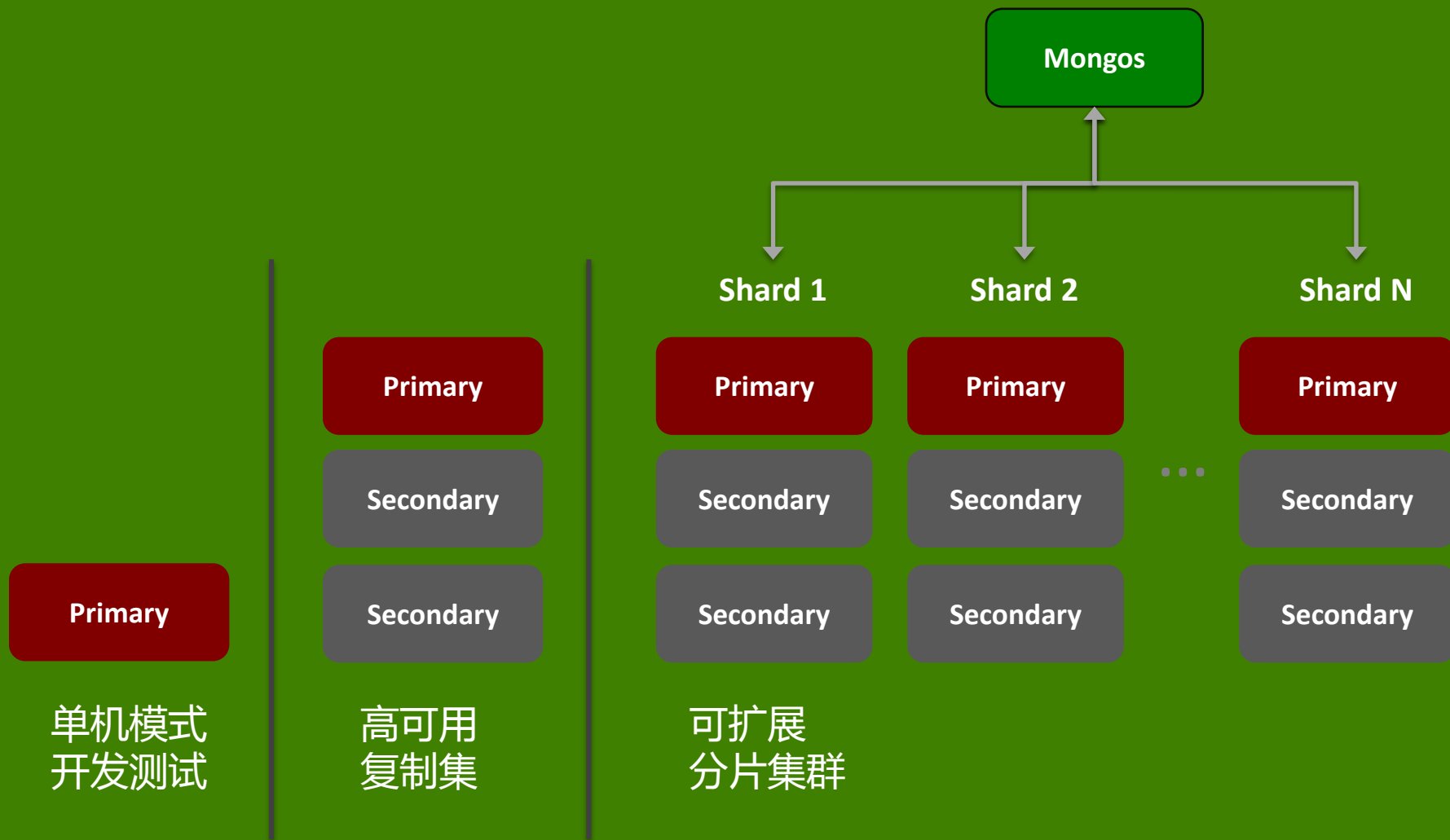
2014年12月17日 - MongoDB数据丢失 我是做前端的,这两天碰到一个问题:我发送数据给后台,后台给我的提示是成功的(查看连接请求的状态是成功的、网页返回也是成功的),因...

[bbs.csdn.net/topics/39...](#) - 百度快照 - 88%好评

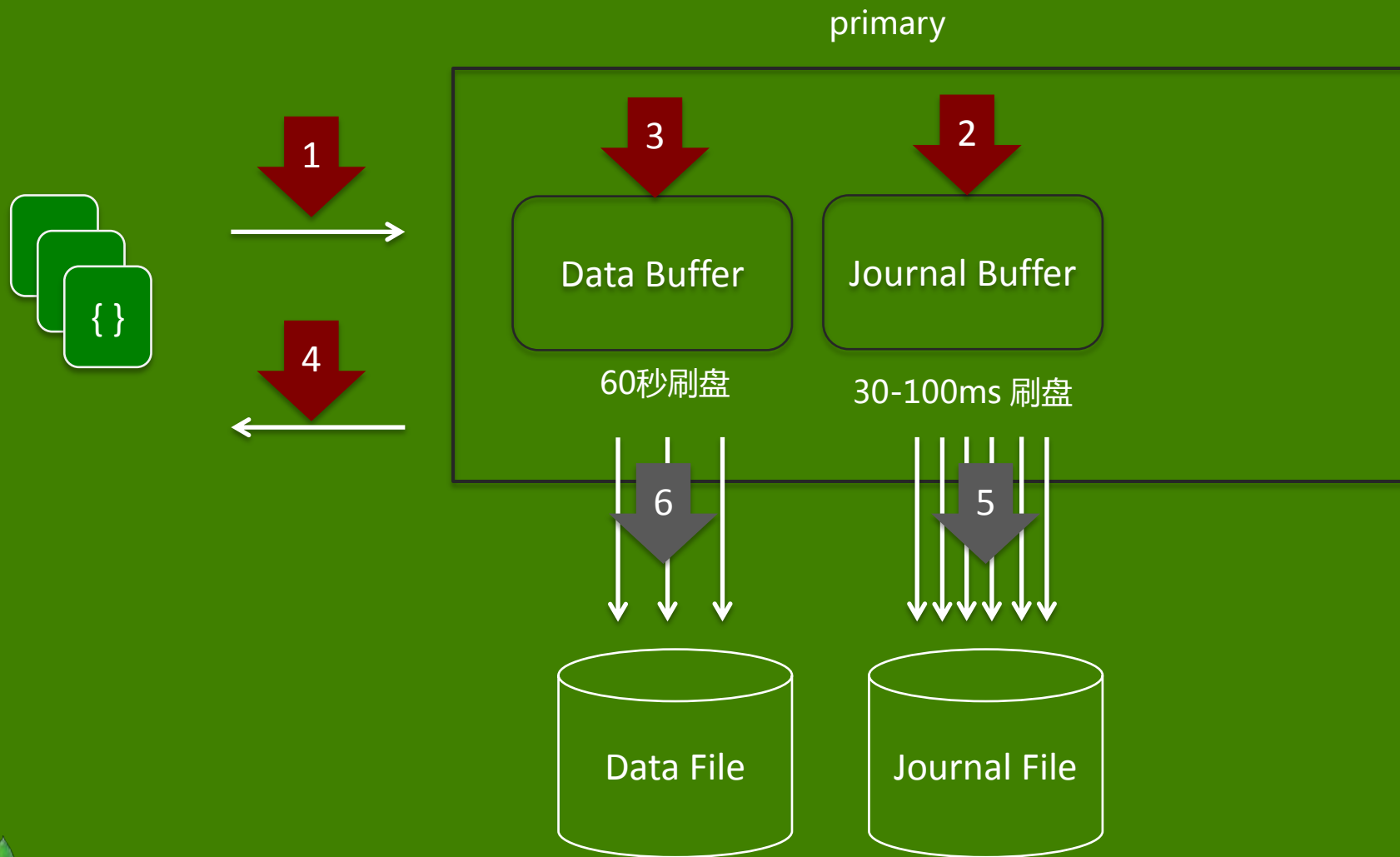
都是 default 惹的祸

MongoDB 数据安全和持久性

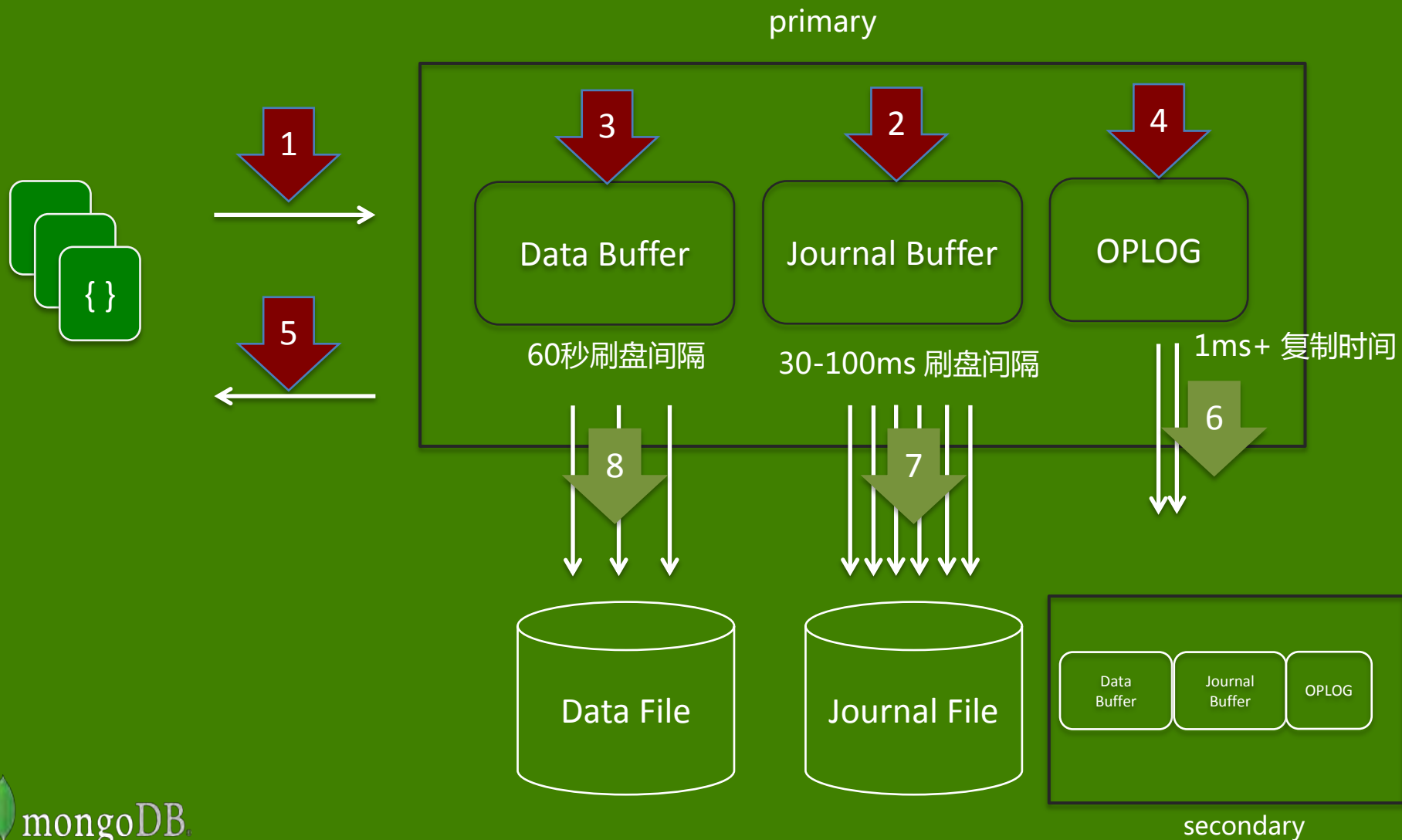
MongoDB 部署模式



MongoDB 的单节点写操作



MongoDB 的复制集写操作



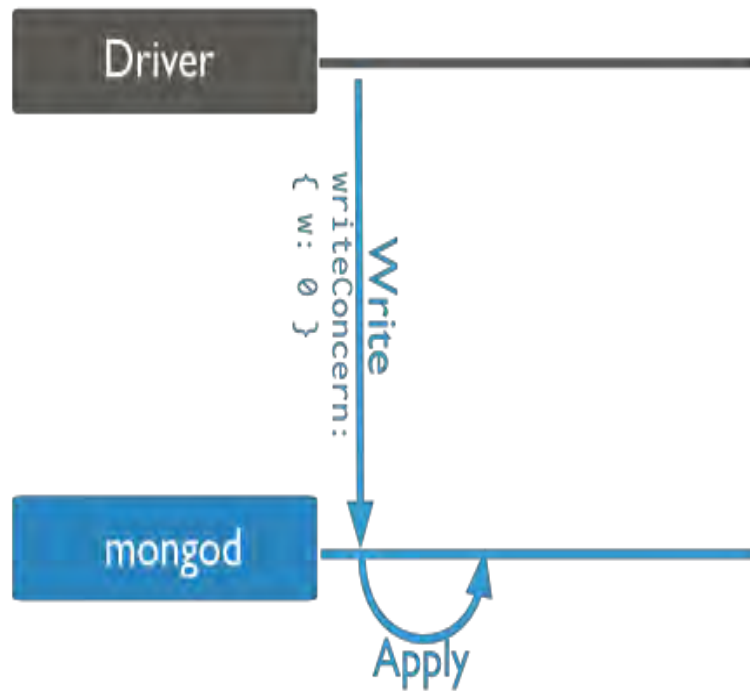
恢复日志 Journal

- 用于系统宕机时恢复内存数据
- 默认为异步刷盘
- 刷盘间隔：
 - MMAP : 30~100ms
 - WiredTiger : 100MB or Checkpoint
- 可使用 j:1 来强制同步刷盘

写关注机制 Write Concern

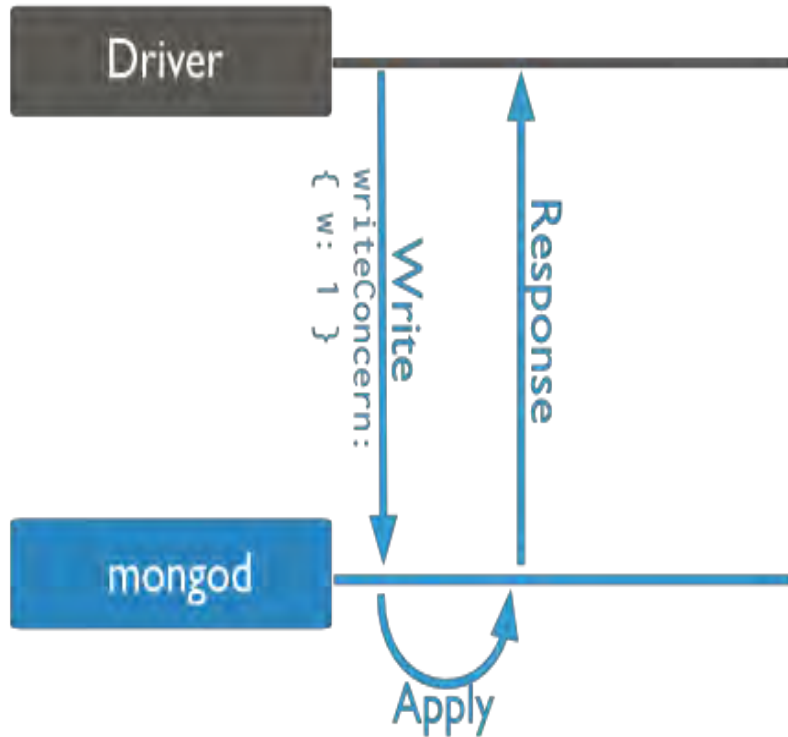
- 用来指定mongod对写操作的回执行为
- 可在connection level或者写操作level指定
- 支持以下值：
 - w: 0 | 1 | n | majority | tag
 - j: 1
 - wtimeout: millis

w:0 Unacknowledged



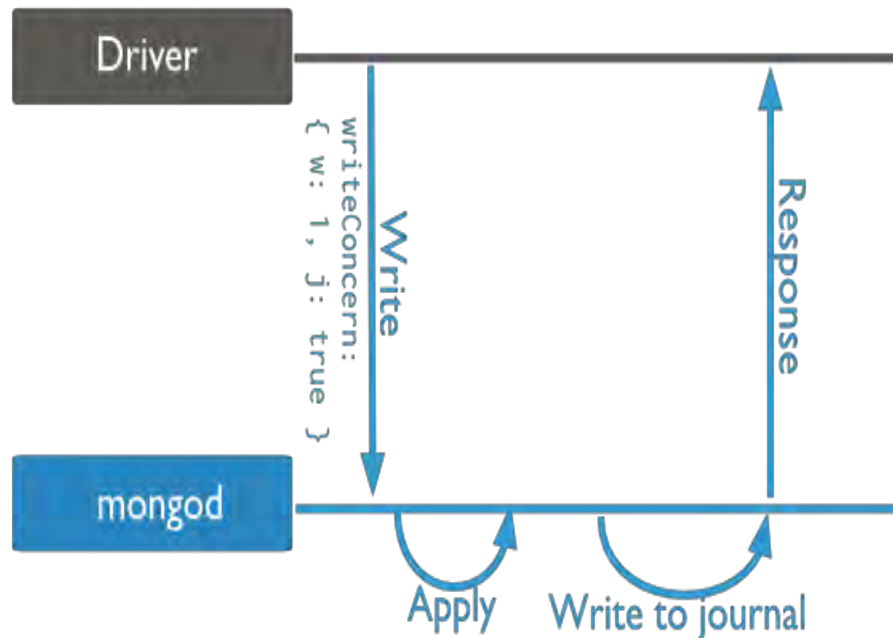
- 无任何回执
- 2.2 及之前版本默认行为
- 网络丢包、系统崩溃、无效数据
- 早期版本丢数据之罪魁祸首

w:1 Acknowledged



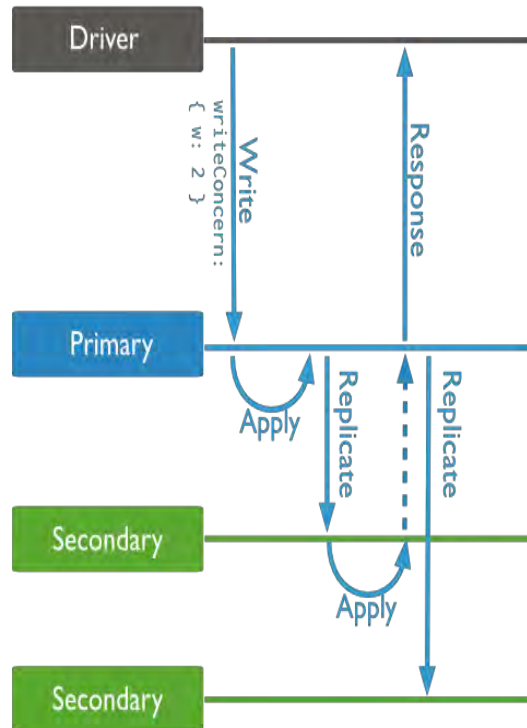
- Mongod在写完内存后发送确认
- 2.4之后默认行为
- 能够处理网络故障、无效数据等错误状态
- 系统崩溃时可能会丢失最多100ms数据

j:1 Journalled



- Journal刷盘之后再发送写回执
- 30ms 间隔Group Commit – 单个请求可能会等最多30ms才返回

w:2/n/majority Replica Acknowledged



- 等待数据复制到n个节点再发送回执

数据“丢失”场景分析

场景A：未使用写关注

重现

使用MongoDB 2.2，或者，
指定 {w:0} 写关注

插入一些无效数据，如10个文档
同一个 _id

检查实际插入数据数目

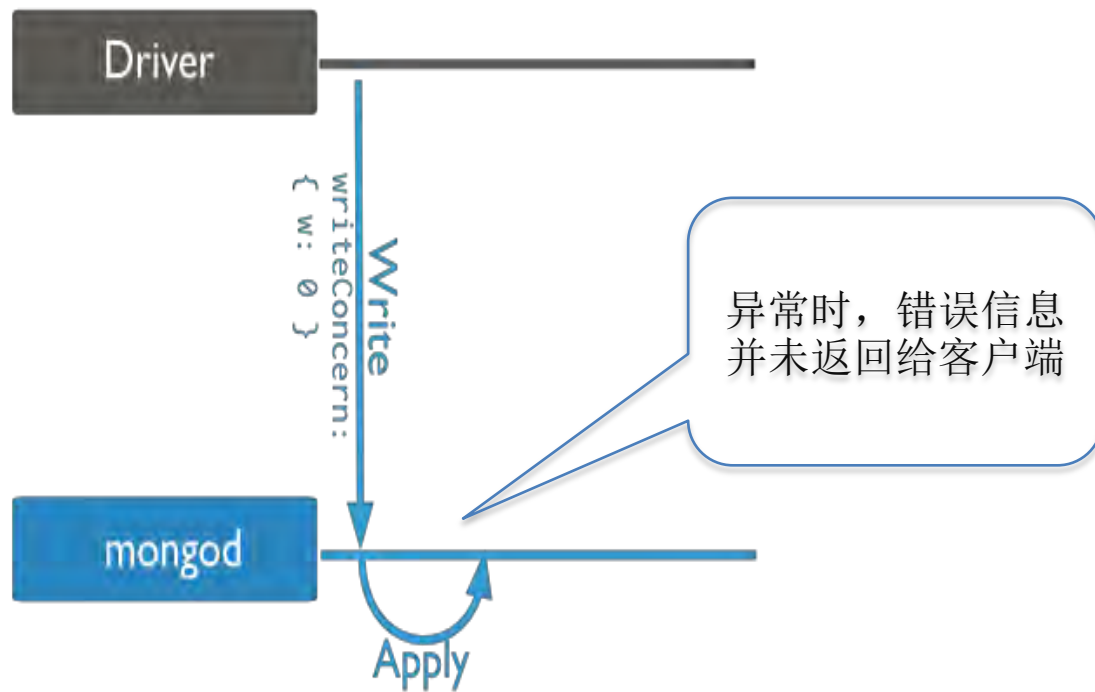
```
>
> db.test.count()
0
> for(var i=0;i<10;i++) {
    var res=db.test.insert({_id:10, a:i}, {writeConcern:{w:0}});
    if(!res.getWriteError())
        print("Inserted doc #"+(i+1));
    else
        print(res.getWriteError().errmsg);
}
```

```
Inserted doc #1
Inserted doc #2
Inserted doc #3
Inserted doc #4
Inserted doc #5
Inserted doc #6
Inserted doc #7
Inserted doc #8
Inserted doc #9
Inserted doc #10
```

```
> db.test.count()
1
>
```

← 期望值：10

原因分析：未 acknowledge 写操作



```
>
> db.test.count()
0
> for(var i=0;i<10;i++) {
    var res=db.test.insert({_id:10, a:i}, {writeConcern:{w:1}});
    if(!res.getWriteError())
        print("Inserted doc #"+(i+1));
    else
        print(res.getWriteError().errmsg);
}
```

Inserted doc #1

```
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
E11000 duplicate key error index: test.test.$_id_ dup key: { : 10.0 }
```

```
> db.test.count()
1
>
```

← 期望值：1

场景A：未使用写关注 解决方案

升级到2.4以后版本

指定 {w:1} 写关注

场景B：系统崩溃导致数据丢失

重现

w:1 高速持续写入数据

Kill -9 mongod

检查程序汇报写入的数据和
实际插入的数据

```
function journalDataLoss(){
  var count=0, start = new Date();
  try{
    var docs= [];
    for(var i=0;i<1000;i++) docs.push({a: i});
    while(true){
      var res = db.test.insert(docs);
      count += res.nInserted;
      if(count % 100000 ==0 ) print("inserted "+ count+" time used: " + ((new Date()).
      getTime() - start.getTime())/1000) +" seconds");
    }
  }
  catch(error){ print("Total doc inserted successfully: "+ count); }
}
```

```
> journalDataLoss()
inserted 100000 time used: 1.691 seconds
inserted 200000 time used: 3.391 seconds
inserted 300000 time used: 5.223 seconds
inserted 400000 time used: 7.066 seconds
inserted 500000 time used: 8.696 seconds
inserted 600000 time used: 10.357 seconds
inserted 700000 time used: 12.197 seconds
```

kill -9 monod

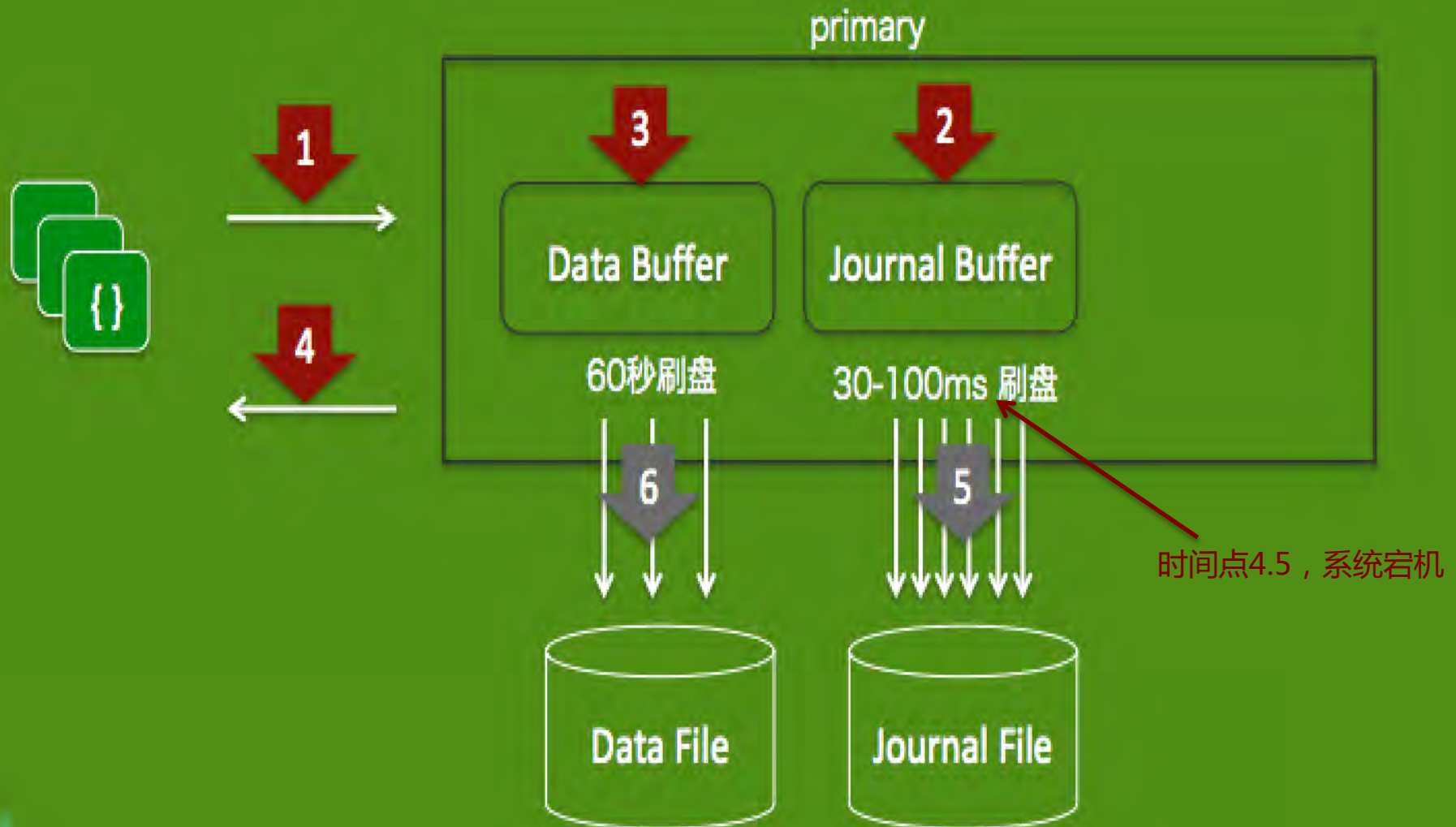
```
2015-10-14T10:27:37.142+0800 I NETWORK DBClientCursor::init call() failed
Total doc inserted successfully: 715000
```

重启
mongod

```
> db.test.count()
2015-10-14T10:27:45.109+0800 I NETWORK trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2015-10-14T10:27:45.114+0800 I NETWORK reconnect 127.0.0.1:27017 (127.0.0.1) ok
713000
>
```

数据丢失！

原因分析：未实时刷日志



```

function journalDataLossWithJ1(){
  var count=0, start = new Date();
  try{
    var docs= [];
    for(var i=0;i<1000;i++) docs.push({a: i});
    while(true){
      var res = db.test.insert(docs,{writeConcern:{j:1}});
      count += res.nInserted;
      if(count % 100000 ==0 ) print("inserted "+ count+" time used: " + ((new Date().
      getTime() - start.getTime())/1000) +" seconds");
    }
  }
  catch(error){ print("Total doc inserted successfully: "+ count); }
}

```

```

> journalDataLossWithJ1()
inserted 100000 time used: 3.579 seconds
inserted 200000 time used: 7.123 seconds
inserted 300000 time used: 10.761 seconds
inserted 400000 time used: 14.31 seconds
inserted 500000 time used: 17.845 seconds
inserted 600000 time used: 21.454 seconds
inserted 700000 time used: 25.004 seconds

```

kill -9 mongod

```

2015-10-14T10:38:02.475+0800 I NETWORK DBClientC::call() failed
Total doc inserted successfully: 726000

```

重启
mongod

```

> db.test.count()
2015-10-14T10:38:12.077+0800 I NETWORK trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2015-10-14T10:38:12.078+0800 I NETWORK reconnect 127.0.0.1:27017 (127.0.0.1) ok
726000
>

```

无数据丢失

场景B：未使用 j:1 系统崩溃 解决方案

使用 j:1

场景C：主备置换导致数据丢失

重现

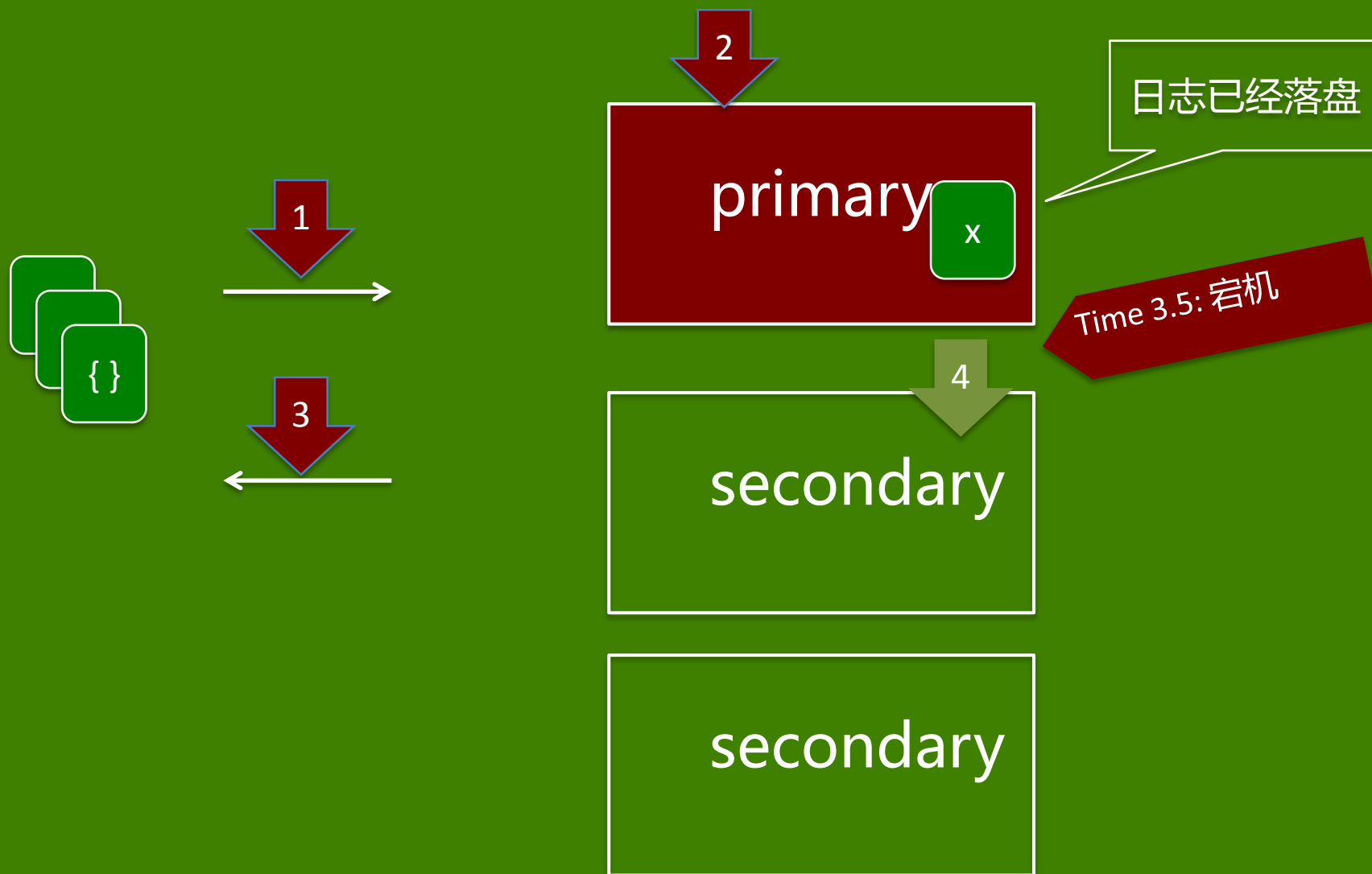
w:1, j:1 高速持续写入数据

kill -9 mongod 主节点

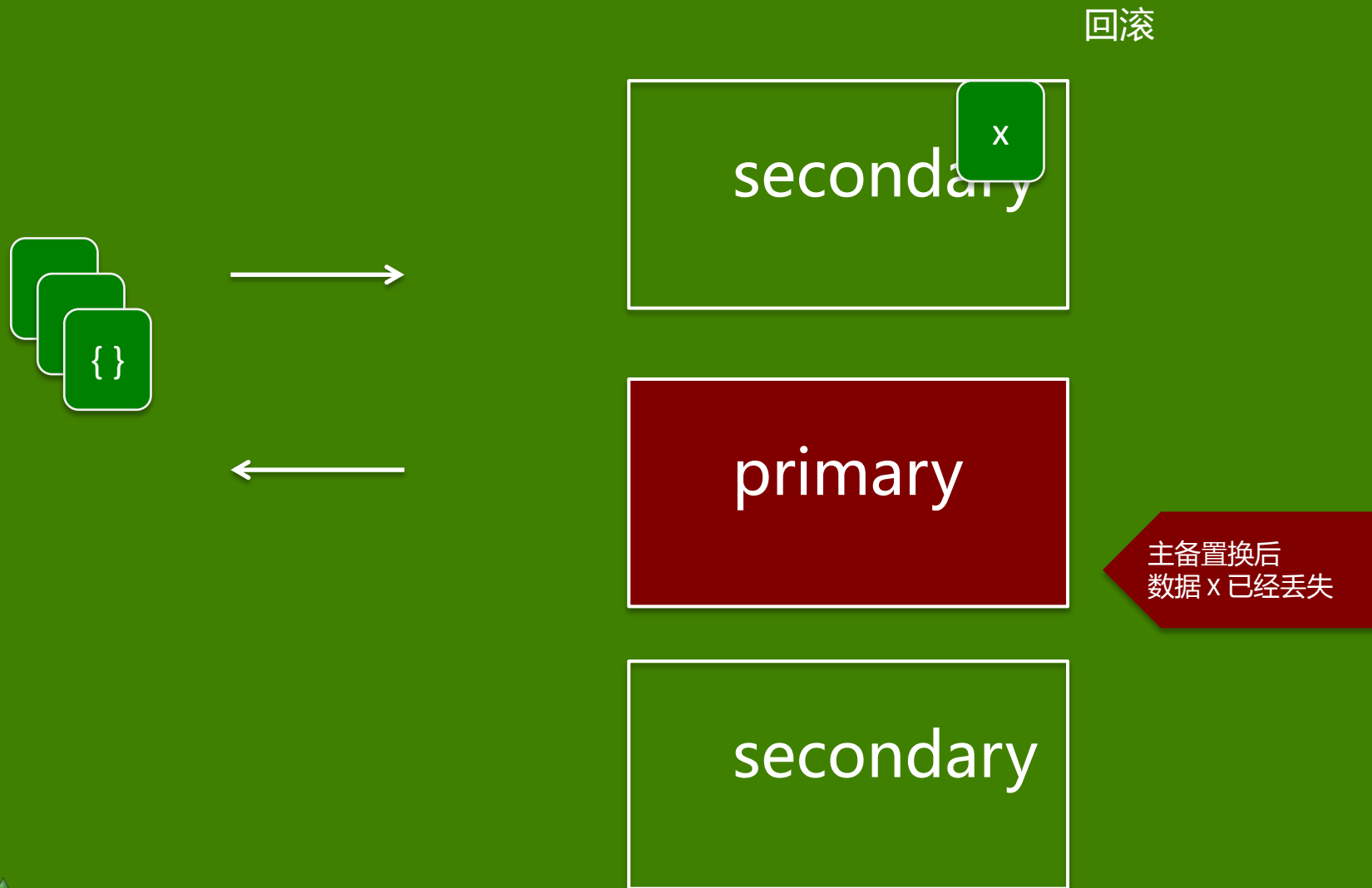
连接到新的主节点

实际插入的数据少于程序汇报的
插入数据

主备置换时的写操作



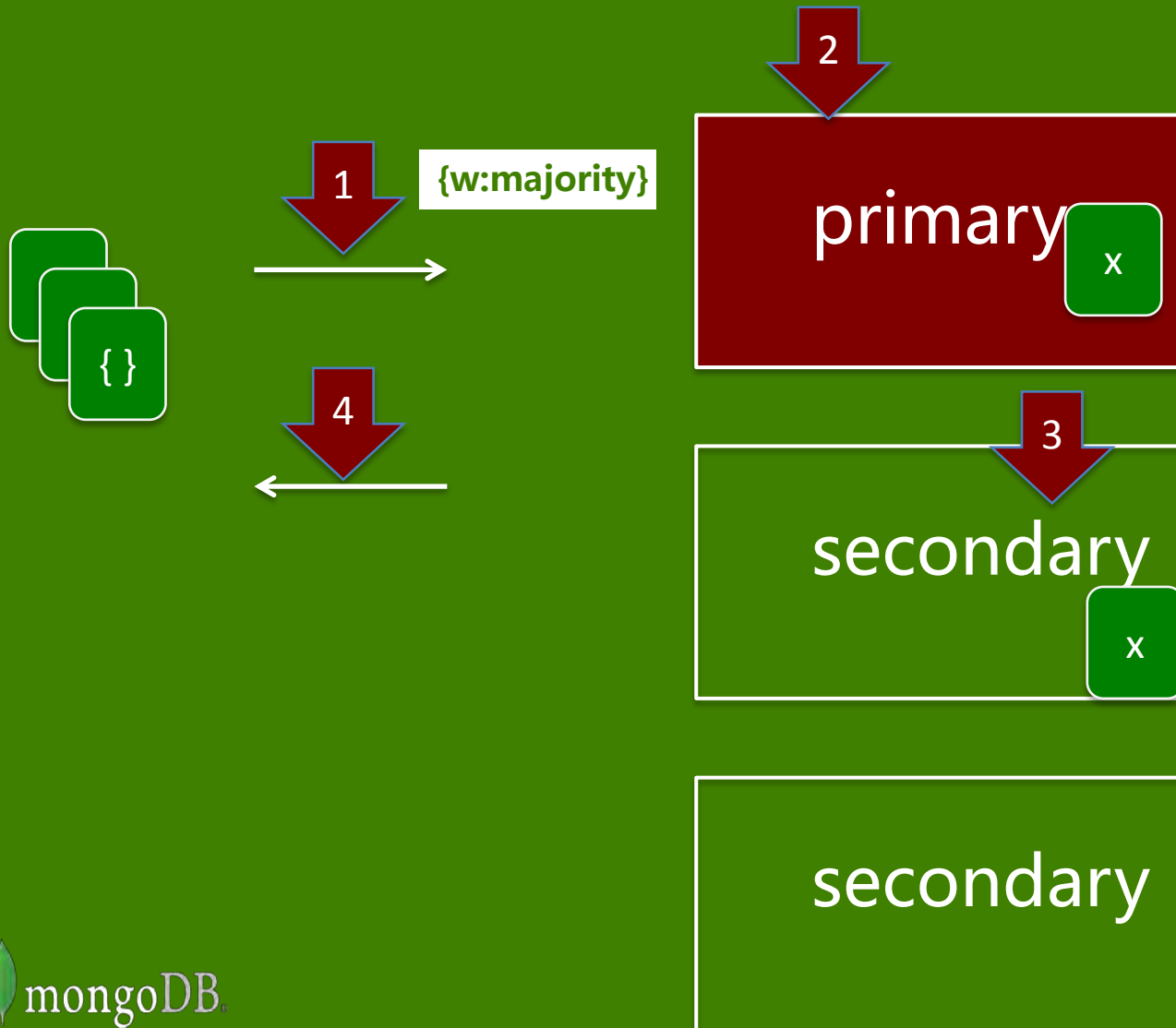
主备置换时的回滚行为



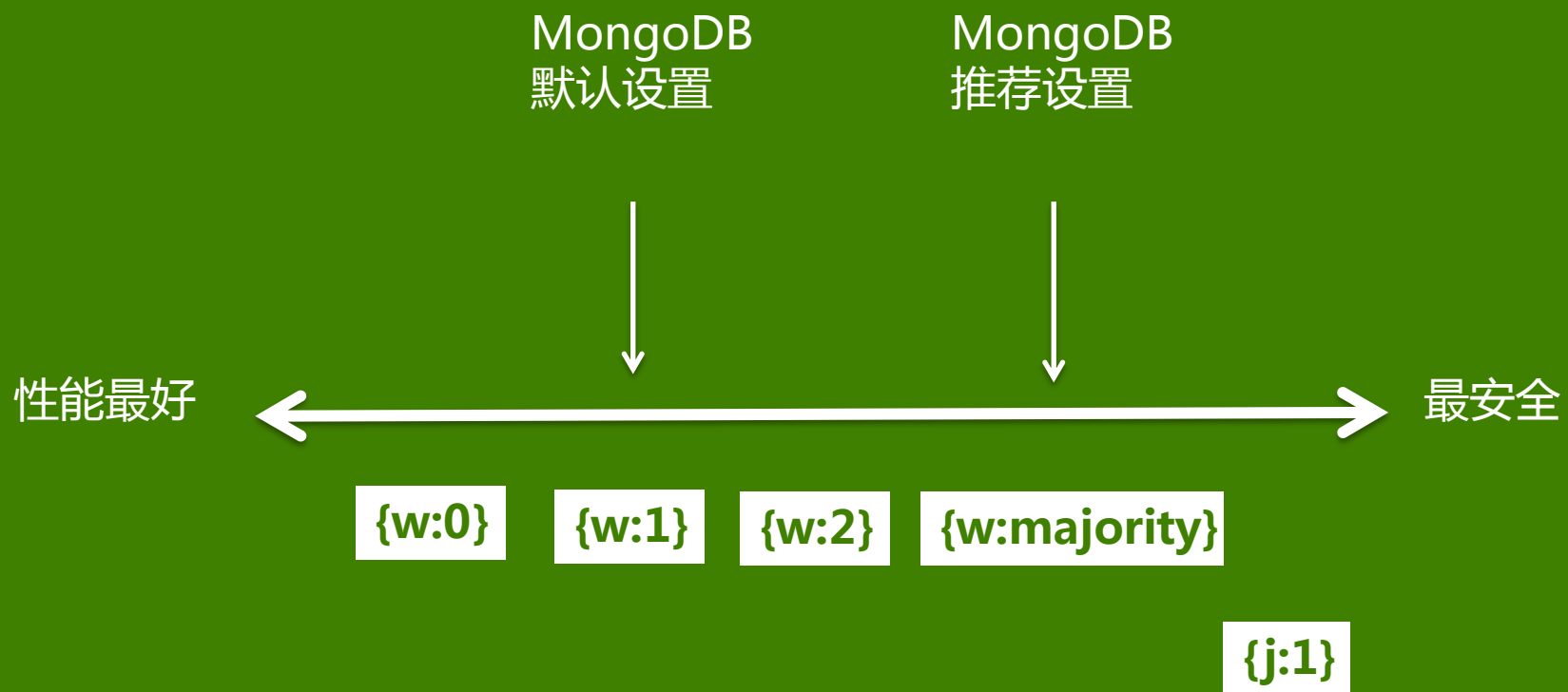
场景C：主备倒换导致数据丢失 解决方案

使用 w: majority

确认数据写到大部分节点再返回



MongoDB 数据安全总结





MongoDB中文社区 <http://www.mongoing.com>

MongoDB在线课 <http://university.mongodb.com>

jianfa.tang@mongodb.com