

MongoDB技术分享

@zhangyu



mongoDB

什么是MongoDB?
为什么我会使用它?

MongoDB是什么？

```
{name:'mongo',type:'DB'}
```

MongoDB (from “humongous”)

MongoDB是一个开源、可扩展、高性能、面向文档文的数据库，用C++编写。

浅探原理

内存映射存储引擎

MongoDB采用内存文件映射引擎（MMAP）将文件映射到进程空间，当查询某块数据时操作系统会以Page方式把这块数据加入物理内存。写入数据时，会先把数据保存到内存中，然后Flush一次进行持久化存储。MongoDB中默认每分钟Flush一次。这部分的管理工作由操作系统完成。



浅探原理

数据文件 DBname.0, DBname.1, DBname.2 ...

DBname.0

DBname.1

DBname.2

DBname.3

目录:

1. 面向文档存储 (Document-Oriented Storage)
2. 全索引支持 (Full Index Support)
3. 复制&高可用性 (Replication & High Availability)
4. 自动分片 (Auto-Sharding)
5. 查询 (Querying)
6. Map / Reduce
7. GridFS

面向文档存储 (**Document**-Oriented Storage)

文档是什么?

```
{name:'zhangyu', email:'zhangyuu@gall.me'}
```

看起来是 **JSON** 其实是 **BSON**



二进制编码序列化的JSON扩展

```
{"hello": "world"}
```

```
"\x16\x00\x00\x00\x02hello\x00  
\x06\x00\x00\x00world\x00\x00"
```

优点：轻巧、高效、灵活

缺点：BSON的开销在序列化上

BSON是JSON扩展

JSON表现力有限，因为只有类型：

null、布尔、数字、字符串、数组、对象

BSON扩展了类型：

时间、正则表达式、函数...

BSON是JSON扩展

日期: { "time" : new Date() }

正则表达式: { "regex" : /test/ }

函数: { "function" : function(){/**/} }

.....

<http://docs.mongodb.org/manual/reference/bson-types/>

```
{
  //传统
  "Status":0,
  //数组，包含2个元素
  "Messages":[
    "0",
    "1"
  ],
  //内嵌文档
  "ResponseBody":{
    "name":"zhangyu",
    "email":[
      "zhangyuu@gall.me",
      "uuzhangyu@gmail.com"
    ]
  }
}
//Java newDate()
"Date":ISODate("2013-11-25T17:32:45.427+-800")
}
```

BSON是JSON扩展

NULL	<u>没有值</u>
Boolean	true、false
数字	仅支持64位浮点数,其他类型的数字会自动转化为此格式
字符串	支持UTF-8字符集
日期	存储标准纪元开始的毫秒数,不含时区 new Date();
Regex	符合JavaScript语法的正则表达式
Code	任何JavaScript代码
Binary	可存储任意字节数的字节数组,但在Shell中无法使用
数组	对象集合或列表在MongoDB可以被存储为数组
嵌套文档	一个文档可以包含另外一个文档
其他	<u>详情点击这里</u>

MySQL

数据库

表

记录行

MongoDB

数据库

Collection

Document

面向文档存储 (Document-Oriented Storage)

```
{name:"zhangyu", age: 18, sex:"男"}  
{name:"malong", age: 28}  
.....
```

Collection 用户

```
{name:"zhangyu", like:"读书"}  
{name:"malong", like:"电影"}  
.....
```

Collection 喜好

为什么Document中没有主键ID？

ObjectId

ObjectId是_id(_id是MongoDB在每个文档中的默认唯一标识的名称)的默认类型。

```
{"_id": ObjectId("52610458e4b0975542d34753"), name:"zhangyu"...}
```

0	1	2	3	4	5	6	7	8	9	10	11
Time				Machine			PID		INC		

Time: 时间戳

Machine: 一般是机器主机名的散列值，确保了不同主机生成不同的机器hash值

PID: 进程ID，一台机器不同的mongo进程产生ID不冲突

INC: 自增计数器，确保在同一秒内产生的objectId也不会发现冲突，允许256的3次方等于16777216条记录的唯一性。

初识增删改查

添加文档

```
db.users.insert({  
  "_id": ObjectId("52c3c518498a9646a48133a2"),  
  "name": "zhangyu",  
  "email": "zhangyuu@gall.me"  
});
```

```
db.users.save({  
  "_id": ObjectId("52c3c518498a9646a48133a2"),  
  "name": "zhangyu_2",  
  "email": "zhangyuu@gall.me_2"  
});
```

insert 当_id存在时报错

save 当_id存在时覆盖更新

删除文档

//删除全部

```
db.user.remove();
```

//删除指定记录

```
db.user.remove({"name":"zhangyu"});
```

//删除user集合

```
db.user.drop();
```

更新文档

原文档:

```
{
  "_id": ObjectId("52c3c518498a9646a48133a2"),
  "name": "zhangyu",
  "email": "zhangyuu@gall.me"
}
```

修改后的文档:

```
{
  "_id": ObjectId("52c3c518498a9646a48133a2"),
  "name": "zhangyu",
  "email": [
    "zhangyuu@gall.me",
    "uuzhangyu@gmail.com"
  ]
}
```

```
var doc = db.users.findOne({"name" : "zhangyu"});  
doc.email = [  
    "zhangyu@gall.me",  
    "uuzhangyu@gmail.com"  
];  
db.users.update( { "name" : "zhangyu" }, doc);
```

PS: 红色部分为查询条件

// 更新:指定第三个参数为true可以开启upsert模式
//根据条件查找不到数据则创建一条新的

```
db.users.update( { "name" : "zhangyu" }, doc, true);
```

查询文档

作用	SQL	MongoDB
所有记录	SELECT * FROM users;	db.users.find();
age = 18	SELECT * FROM users WHERE age = 18;	db.users.find({"age":18});
筛选字段	SELECT age FROM users WHERE age = 18;	db.users.find({"age":18},{age:1});
排序	SELECT * FROM users ORDER BY name ASC	db.users.find().sort({name:1});
比较	SELECT * FROM users WHERE age > 18;	db.users.find({"age":{"\$gt":18}});
正则	SELECT * FROM users WHERE name LIKE zha;	db.users.find({"name":""});
忽略	SELECT * FROM users LIMIT 10 SKIP 20;	db.users.find().limit(10).skip(20);
or	SELECT * FROM users WHERE a=1 or b=2;	db.users.find({"\$or" : [{ a : 1 } , { b : 2 }]});
distinct	SELECT DISTINCT name FROM users;	db.users.distinct('name');
count	SELECT COUNT(name) FROM users;	db.users.find({name: {'\$exists': true}}).count();
引用		

索引

- MongoDB的索引机制与传统的关系型数据库索引几乎是一样的,绝大多数优化SQL索引的技巧也都适用于MongoDB。
- 索引会增加数据插入、更新、删除的性能开销,应避免为每个键都创建索引。

创建索引

//创建索引

```
db.users.ensureIndex({'name': 1 });
```

// 创建子文档索引

```
db.users.ensureIndex({  
    'user.books': -1  
});
```

// 创建复合索引

```
db.users.ensureIndex({  
    'name': 1, // 升序  
    'age': -1 // 降序  
});
```

/*

如果find操作只用到了一个键,那么
索引方向是无关紧要的;

创建复合索引的时候,一定要谨慎斟酌
每个键的排序方向。

*/

修改索引

// 原来的索引会重建

```
db.users.ensureIndex({  
  'user.books': -1,  
  // 新增一个升序索引  
  'user.likes': 1  
}, {  
  // 打开后台执行  
  'background': true  
});
```

// 重建索引

```
db.users.reIndex();
```


删除索引

//删除索引

```
db.users.dropIndexes();
```

//选择删除索引 根据索引名称

```
db.users.dropIndexes('byname');
```

MongoDB VS MySQL

架构设计

我们看一个简单的例子，如何为MySQL（或任何关系数据库）和MongoDB中创建一个数据结构。

MySQL设计：

People 人物信息表 包含ID 和名字 字段

passports 护照表 包含 对应**people**表外键ID 所属国家和护照有效期

```
mysql> select * from people;
```

id	name
1	Stephane
2	John
3	Michael
4	Cinderella

于是你接下来可以操作如下基本功能：

一共有多少人： **SELECT COUNT(*) FROM people;**

查询出 **stephane** 的护照有效期：

```
mysql> select * from passports;
```

id	people_id	country	valid_until
4	1	FR	2020-01-01
5	2	US	2020-01-01
6	3	RU	2020-01-01

SELECT valid_until from passports ps join people pl ON ps.people_id = pl.id WHERE name = 'Stephane'

有多少人木有护照：

SELECT name FROM people pl LEFT JOIN passports ps ON ps.people_id = pl.id WHERE ps.id IS NULL

MongoDB的设计:

1、“直筒式”的设计，和关系型数据库的理解区别不大

```
{
  "_id" : ObjectId("51f7be4dd6189a56c399d3c1"),
  "name" : "Michael",
  "country" : "RU",
  "valid_until" : ISODate("2019-12-31T23:00:00Z")
}
{ "_id" : ObjectId("51f7be5cd6189a56c399d3c2"), "name" : "Cinderella" }
```

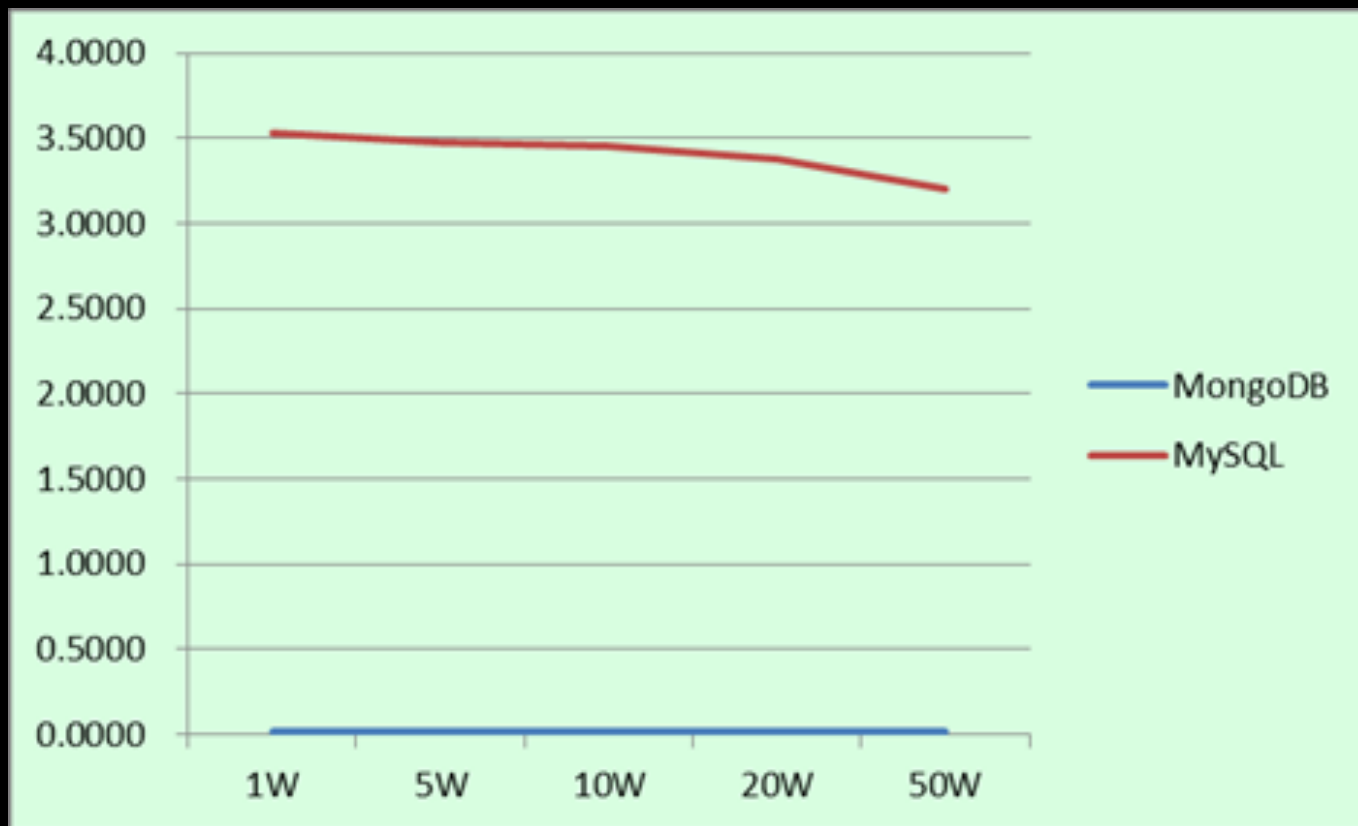
2、MongoDb特征的设计方法，既：把people信息和护照信息柔和在一起

```
{
  "_id" : ObjectId("51f7c71b8ded44d5ebb83776"),
  "name" : "Michael",
  "passport" : {
    "country" : "RU",
    "valid_until" : ISODate("2019-12-31T23:00:00Z")
  }
}
{ "_id" : ObjectId("51f7c7258ded44d5ebb83777"), "name" : "Cinderella" }
```

性能测试

- 1、分别生成1w, 5w, 10w, 20w, 50w个互不相同UUID值, 并保存。
- 2、每查询1000条数据保存一下时间

性能测试



1. 在读取的数据规模不大时，MongoDB的查询速度真是一骑绝尘，甩开MySQL好远好远。
2. 在查询的数据量逐渐增多的时候，MySQL的查询速度是稳步下降的，而MongoDB的查询速度却有些起伏。

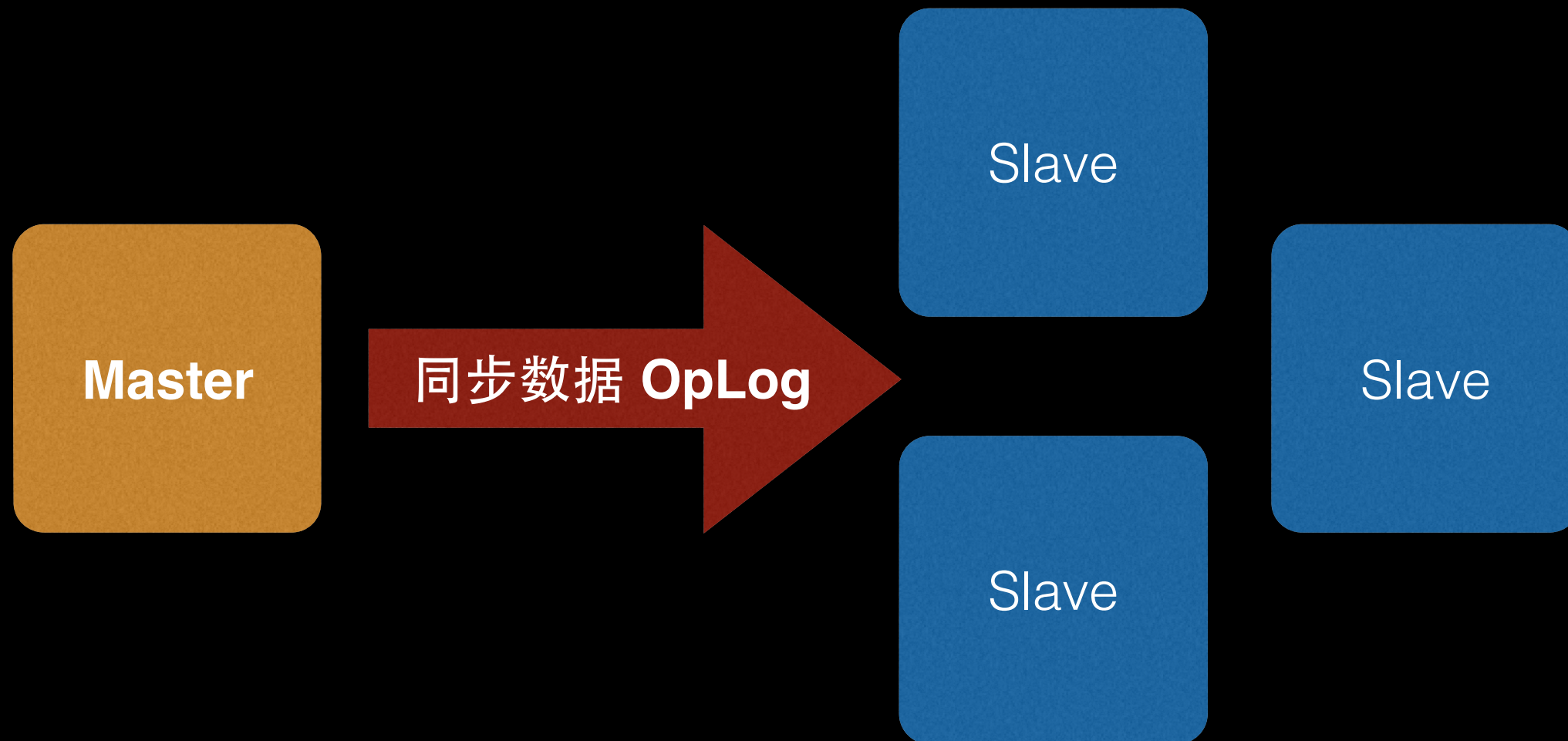
复制&高可用性

1. Master-Slave Replication (主从)

2. Replica Sets (副本集)

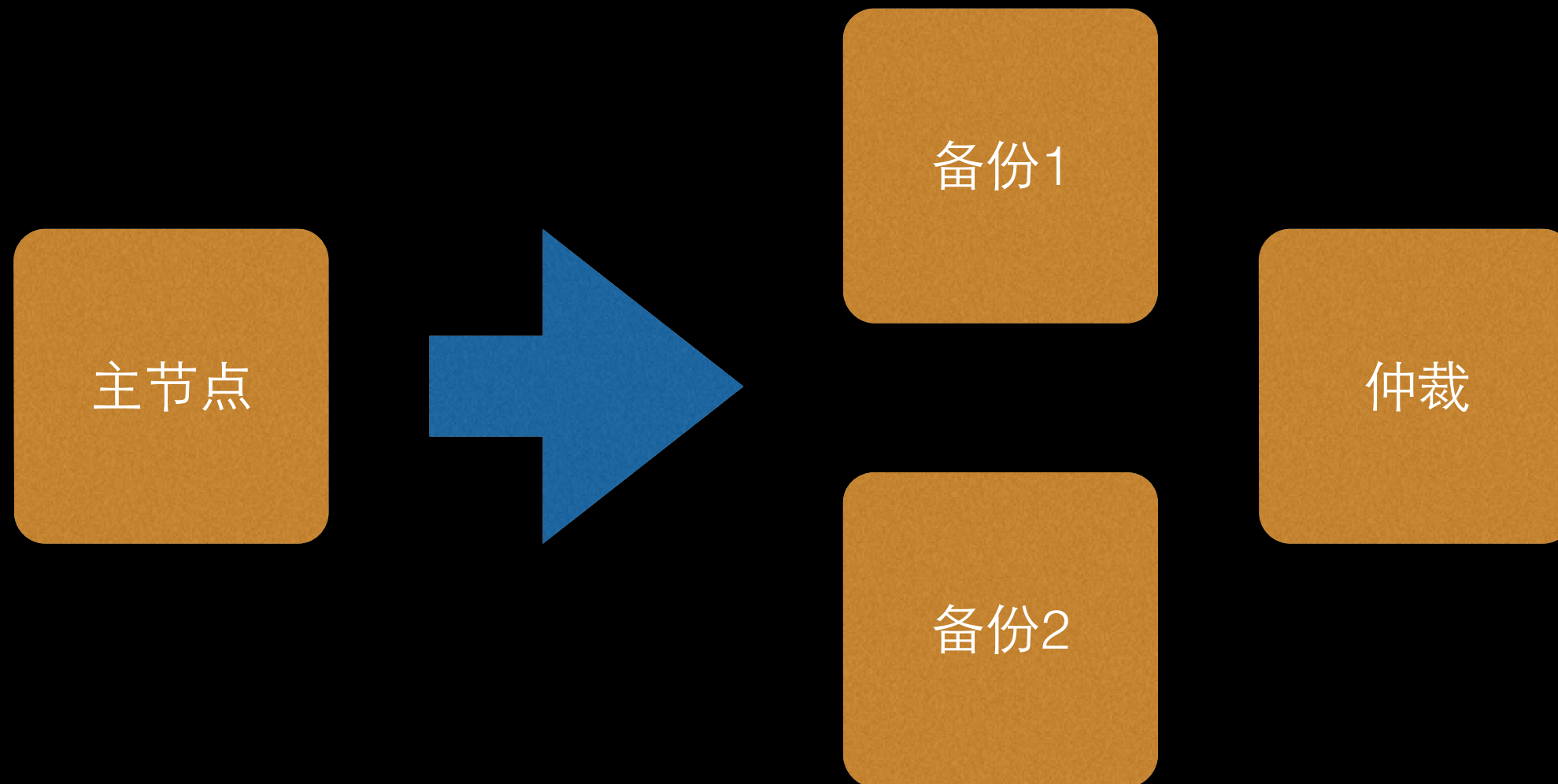
复制&高可用性

1. Master-Slave Replication (主从)



复制&高可用性

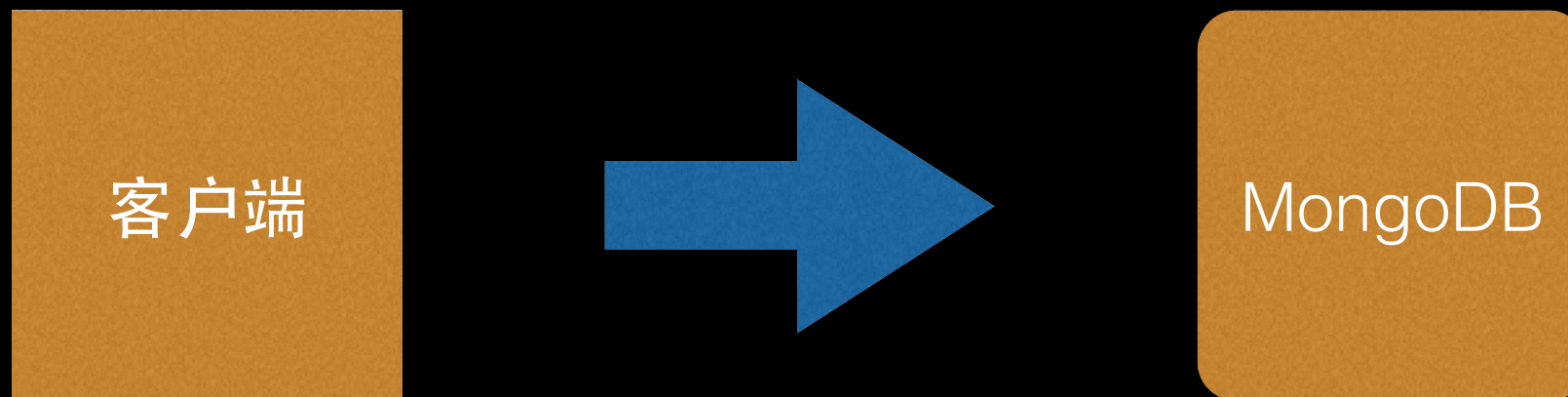
1. Replica Sets (副本集)



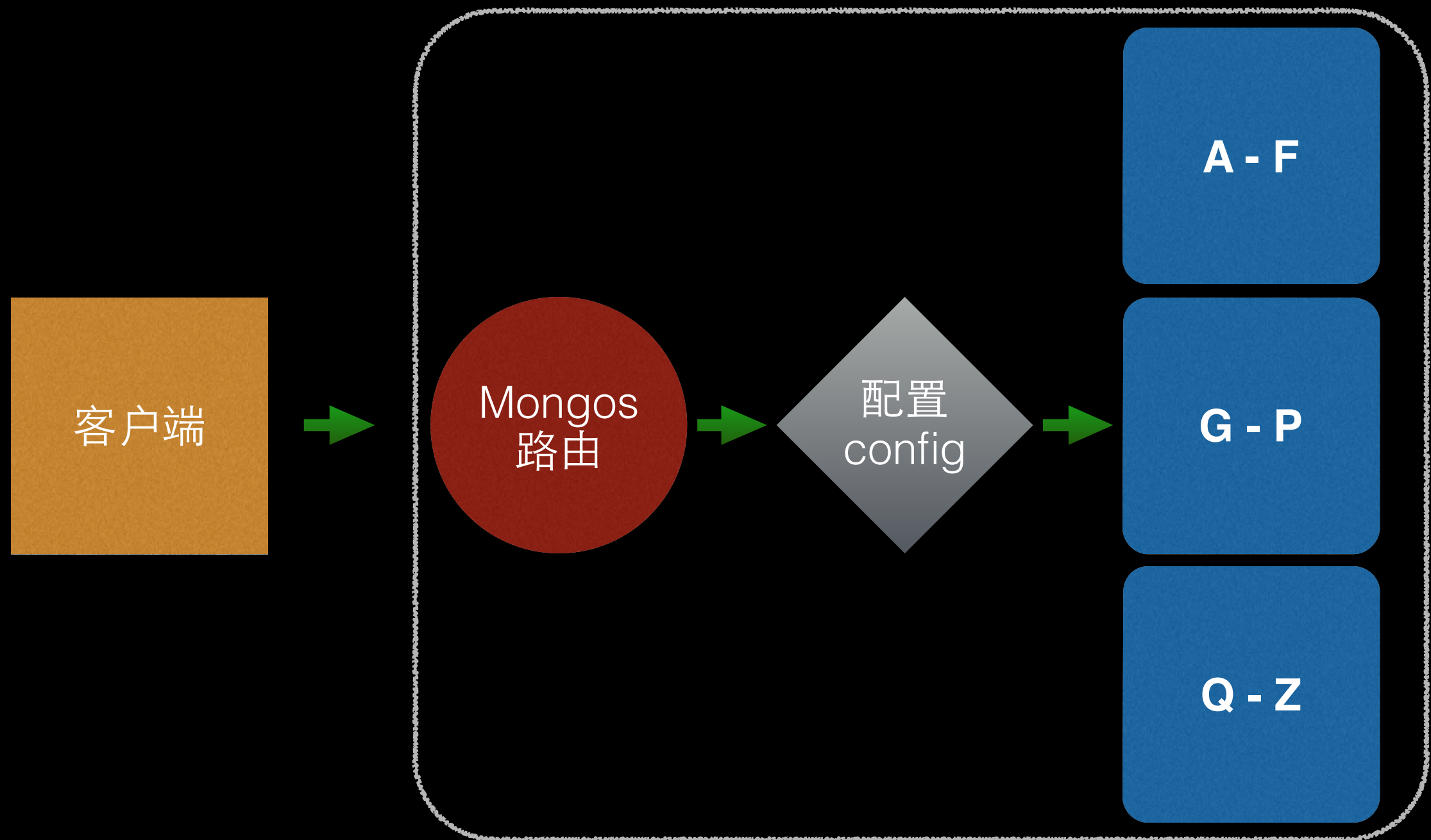
自动分片：

场景： 用户表访问（用户量少）

问题， 当用户量越来越大， 导致一个DB已经无法承担当前数据量....



自动分片:



适用场合

联机分析处理（On-Line Analytical Processing,简称OLAP）

OLAP的概念，泛指一切不对数据进行输入等事务性处理，而基于已有数据进行分析的方法。

缓存：由于性能很高，Mongo也适合作为信息基础设施的缓存层。在系统重启之后，由Mongo搭建的持久化缓存可以避免下层的数据源过载。

存储大尺寸、低价值的数据

Mongo非常适合由数十或者数百台服务器组成的数据库。

不适用场合

联机交易处理（OLTP, Online transaction processing）

OLTP通常被运用于自动化的数据处理工作，如订单输入、金融业务...等反复性的日常交易活动。

高度事物性的系统

需要SQL才能解决的问题

GridFS:

一种存储文件的机制

场景：用户上传图片

量大、文件小、大小不一、碎片多、IO负载重

GridFS:

文件可以分成多个Document存储

可以很好的利用Mongo复制、分片等特性

大文件存储 (2G)

碎片自动处理

可以利用MongoDB丰富的查询来管理文件

Thanks!

Google GUAVA

@zhangyu

GUAVA 简介

GUAVA是基于Java的项目，来自Google。

核心库：集合、缓存、基本类型支持、并发、常见注解、字符串处理、I/O 等。

用Guava是因为我喜欢简洁的API、可以少写几行代码。

Collection

JAVA写法:

```
Map<String, Map<Long, List<String>>> map =  
    new HashMap<String, Map<Long, List<String>>>();
```

这些静态工具方法，**Lists**和**Sets**也有:

```
List<String> stringList = Lists.newArrayList();  
Set<String> stringSet = Sets.newHashSet();
```

GUAVA写法:

```
Map<String, Map<Long, List<String>>> map = Maps.newHashMap();
```

操作lists和maps

写单元测试时，会构造一些不可变的测试数据，可能是**list**、**map**、**set**等

JAVA写法：

```
List<String> list = new ArrayList<String>();
```

```
list.add("a");
```

```
list.add("b");
```

Guava写法：

```
ImmutableList<String> list = ImmutableList.of("a", "b", "c");
```

```
ImmutableMap<String,String> map =
```

```
    ImmutableMap.of("key1", "value1", "key2", "value2");
```

CharMatcher 可以非常方便地添加到你的java工具箱中。有些人形容它：“像打了兴奋剂的StringUtils”

从字符串中得到所有的数字，你可以这样：

```
String string =
```

```
    CharMatcher.DIGIT.retainFrom("some text 12345 and more");
```

结果：some text 12345 and more --> 12345

如果你想把字符串中的数据都去掉，可以如下：

```
String string =
```

```
    CharMatcher.DIGIT.removeFrom("some text 89983 and more");
```

结果：some text 89983 and more --> some text and more

更多信息查看[Java doc](#)

Joiner and Splitter

可以这么使用Joiner:

```
String[] sub = { "usr", "local", "lib" };  
String directory = Joiner.on("/").join(sub);  
结果: usr/local/lib
```

或者这样:

```
int[] numbers = { 1, 2, 3, 4, 5 };  
String numAsStr = Joiner.on(";").join(Ints.asList(numbers));  
结果: 1;2;3;4;5
```

得益于Guava对基本型的支持, 可以很方便这么处理:

```
String numbersAsStringDirectly = Ints.join(";", numbers);  
结果: 1;2;3;4;5
```

Splitter 提供了更多的操作, 而且更加健壮, 切割字符串

```
Iterable<String> split = Splitter.on(",").split(numbersAsString);
```

这种字符串的切割很让人头疼

```
String str = "foo , what,,,more,";
```

Splitter允许我们对分割结果做更多的控制:

```
Iterable<String> sp =  
    Splitter.on(",").omitEmptyStrings().trimResults().split(str);
```

结果: [foo, what, more]

直接转为List操作

```
Lists.newArrayList(split);
```

Strings

判断 String不为null,且不为空

Java写法 :

```
if(null != string && !string.isEmpty()) {  
  
}
```

Guava写法:

```
if(!Strings.isNullOrEmpty(str)) {  
  
}
```


Predicate

一个简单的计算。

```
List<Integer> numbers =  
Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8,  
9, 10);
```

很多人会这样写

```
static boolean isEven(int num) {  
    return (num % 2) == 0; // simple  
}  
  
for (int number : numbers) {  
    if (isEven(number)) {  
        // do something  
    }  
}
```

现在可以这样写

```
Predicate<Integer> isEven = new  
Predicate<Integer>() {  
    public boolean apply(Integer number) {  
        return (number % 2) == 0;  
    }  
};  
  
Iterable<Integer> evenNumbers =  
Iterables.filter(numbers, isEven);  
  
for (Integer number : evenNumbers) {  
    System.out.println("Yep!" + number);  
}
```

Function

一个简单的字符类型转换。

JAVA写法

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
  
List<String> toStrings = Lists.newArrayList();  
  
for(Integer i : numbers) {  
    toStrings.add(String.valueOf(i));  
}
```

Guava写法

```
List<String> toString = Lists.transform(numbers, new Function<Integer, String>() {  
    public String apply(Integer number) {  
        return String.valueOf(number);  
    }  
});
```

I/O包

- Files提供了针对文件的工具方法
- ByteStreams提供了针对字节流的工具方法
- CharStreams提供了针对字符流的工具方法
- Resources提供了针对classpath下资源操作的工具方法

```
File file = new File("/Users/zhangyu/a.log");
```

一行代码即可

```
List<String> lines = Files.readLines(file, Charsets.UTF_8);
```

取空格操作

```
List<String> trimmedLines = Files.readLines(file, Charsets.UTF_8,  
    new LineProcessor<List<String>>() {  
        List<String> result = Lists.newArrayList();  
        public boolean processLine(String line) { //处理器  
            return result.add(line.trim());  
        }  
        public List<String> getResult() { //结果  
            return result;  
        }  
    }  
);
```

复制、移动文件操作

```
File from = new File("/U/z/a.log");    File to = new File("/U/z/b.log");  
Files.copy(from, to); //复制  
Files.move(from, to); //移动文件
```

操作资源文件

```
URL url = Resources.getResource("foo.txt");  
List<String> list = Resources.readLines(url, Charsets.UTF_8);
```

Thanks!

Web 实战部分

@zhangyu

WebJars简介

- WebJars 是将Javascript、CSS 等类库打包成 jar文件的一套实现
- 明确并轻松管理客户端的依赖于基于JVM的Web应用程序
- 最主要的一点，他可以通过Maven, Gradle, & SBT来管理Web应用中的依赖
- 项目成立于2012年4月28日

首先添加一个WebJar在pom.xml中，建立应用程序的依赖关系。如：

```
<!-- 必须要引入 webjars 定位器 否则找不到引入的web类库-->
```

```
<dependency>
```

```
    <groupId>org.webjars</groupId>
```

```
    <artifactId>webjars-locator</artifactId>
```

```
    <version>0.1</version>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.webjars</groupId>
```

```
    <artifactId>bootstrap</artifactId>
```

```
    <version>2.1.1</version>
```

```
</dependency>
```

然后简单地引用资源，如：

```
<link rel='stylesheet'
```

```
    href='webjars/bootstrap/2.1.1/css/bootstrap.min.css'>
```

PS:已经打包好的类库 <https://github.com/webjars/>

使用文档 <http://www.webjars.org/documentation>

SpringMvc项目需要在配置pom之后 配置Spring配置文件映射静态资源。如：

```
<mvc:resources mapping="/webjars/**"  
    location="classpath:/META-INF/resources/webjars/" />
```

然后简单地引用资源，如：

```
<link rel='stylesheet'  
    href='/webjars/bootstrap/2.1.1/css/bootstrap.min.css'>
```

JQuery 实用插件介绍

基于Bootstrap时间选择插件

1、bootstrap-daterangepicker

<https://github.com/smalot/bootstrap-datetimepicker>

2、bootstrap-datetimepicker

<https://github.com/dangrossman/bootstrap-daterangepicker>

Fuel UX

Fuel UX 是扩展 Twitter Bootstrap 轻量级 JavaScript 控件。其优点包括易于集成到 Web 项目，对 Bootstrap 的支持，项目比较活跃更新快。所有功能都有详细的文档以及单元测试。

<http://exacttarget.github.io/fuelux/>

jQWidgets

<http://www.jqwidgets.com/jquery-widgets-demo/>

重量级创新的 UI 部件库，基于 jQuery 开发，可让用户开发非常专业的、跨浏览器支持的 Web 应用。

NOD

在我看来编写表单验证是痛苦无趣的一件事儿，所以我推荐使用NOD来减少表单验证带来的痛苦。

```
var metrics = [  
  [ '#foo', 'presence', '不能为空' ],  
  [ '#bar', 'min-length:4', '最少四个字符' ]  
];
```

```
$("#form").nod(metrics);
```

PS:这个也是非常强大的 jQuery Validation Plugin

添加应用



名称

描述

✓ 添加

添加应用



名称

内容长度最小为2

描述

内容长度最小为2





















































✓ 添加

The Icons

在项目中我们会遇到各式各样的图标，用在菜单、按钮上，更甚至是项目的Logo。为了减少美工的工作量以及减少项目中图片的使用，推荐下面两个Icon库。

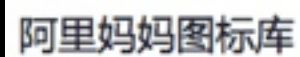
Font Awesome 提供 369个Icons图标(不可订制)

<http://fontawesome.io/icons/>

 fa-adjust	 fa-anchor	 fa-archive	 fa-arrows
 fa-arrows-h	 fa-arrows-v	 fa-asterisk	 fa-ban
 fa-bar-chart-o	 fa-barcode	 fa-bars	 fa-beer
 fa-bell	 fa-bell-o	 fa-bolt	 fa-book
 fa-bookmark	 fa-bookmark-o	 fa-briefcase	 fa-bug
 fa-building-o	 fa-bullhorn	 fa-bullseye	 fa-calendar
 fa-calendar-o	 fa-camera	 fa-camera-retro	 fa-caret-square-o-down
 fa-caret-square-o-left	 fa-caret-square-o-right	 fa-caret-square-o-up	 fa-certificate
 fa-check	 fa-check-circle	 fa-check-circle-o	 fa-check-square
 fa-check-square-o	 fa-circle	 fa-circle-o	 fa-clock-o
 fa-cloud	 fa-cloud-download	 fa-cloud-upload	 fa-code
 fa-code-fork	 fa-coffee	 fa-cog	 fa-cogs
 fa-comment	 fa-comment-o	 fa-comments	 fa-comments-o

2、阿里巴巴 提供 N个 矢量图图标库(可订制)

<http://www.iconfont.cn/>



在线JavaScript/CSS压缩工具

YUI Compressor，用来压缩JavaScript/CSS文件工具，该工具由Java语言开发。

在线版本：<http://refresh-sf.com/yui/> 可以方便的将JavaScript/CSS进行压缩，

Thanks!

E-mail: zhangyuu@gall.me

Google+: uuzhangyu@gmail.com