HZ BOOKS
华章IT

中国Solr领域的资深专家和布道师撰写，权威性毋庸置疑。

以实战为导向，全面、系统、细致、深入地讲解Solr的基础知识、核心技术、进阶知识和扩展知识。

實戰

兰小伟 著

The Definitive Guide of Solr

Solr权威指南

下卷

机械工业出版社
China Machine Press

實戰

The Definitive Guide of Solr

Solr权威指南

下卷

兰小伟 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Solr 权威指南　下卷 / 兰小伟著 . —北京：机械工业出版社，2017.10
（实战）

ISBN 978-7-111-58207-6

I. S…　II. 兰…　III. 搜索引擎 – 程序设计 – 指南　IV. TP391.3-62

中国版本图书馆 CIP 数据核字（2017）第 261590 号

Solr 权威指南　下卷

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：何欣阳　　责任校对：李秋荣
印　　刷：北京诚信伟业印刷有限公司　　版　　次：2018 年 1 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：20.5
书　　号：ISBN 978-7-111-58207-6　　定　　价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Preface 序　言

Apache Solr 是使用最广泛的全文检索解决方案，大部分网站都在使用 Solr 来实现搜索功能。然而国内关于 Solr 的资料太少，无奈我只能一点点地啃 Solr 官方提供的 User Guide PDF 文档、Solr Wiki 以及一些纯英文的技术书籍，希望能够借由本书将我学习积累的所有经验倾情传授给那些由于学习 Solr 曲线太陡峭而束手无策的同学们。本书致力于帮助 Java 开发人员更简单、深入地学习 Solr。同时本书还提供了随书源码，其中包含大量可运行的示例代码。本书与随书源码搭配在一起学习会事半功倍！由于目前大数据、云计算的发展如火如荼，各种大数据生态框架如雨后春笋般涌现，给人一种无形的压力。为此，本书也介绍了 Solr 与大数据框架的集成，如果你正好有这方面的需求，希望本书能够给你带来帮助。

为什么写这本书

转眼间，我已经跌跌撞撞走过了 5 个年头，由起初的那个 Java 迷途小书童变身为程序员届的一根老油条，不由感慨万千。由于深谙一个非高校毕业的“正规军”一路走来有多么的艰辛，因此我一直秉持爱开源、爱分享的个性。这么多年来帮助过的程序员太多太多，本着一颗乐于助人的心，我不想大家重走我的弯路。从 2015 年 3 月中旬开始，我在 ITEye 技术社区发布与 Lucene 和 Solr 相关的技术博客，深受大家喜爱。每天联系、咨询我问题的网友越来越多。疲于应付的我，开始意识到仅靠一个人这样一对一地指导是行不通的。而且刚好 Solr 这方面的中文技术书籍在中国还是一片空白，于是萌生了写一本 Solr 中文书籍的想法，希望能够帮助更多的 Solr 技术爱好者。

2015 年 8 月我联系到了华章的杨福川，向他提出了写这本书的想法，得到了他的大力支持。我深知自己过往没有显赫耀眼的工作经历，在一些前辈面前还只是一个晚辈。因此，在创作本书的过程中，查阅了 Solr 官网提供的 Apache-Solr-Ref-Guide、Solr Wiki，并通读了《Solr in action》《Apache Solr 4 Cookbook》《Apache Solr Essentials》《Apache Solr High

Performance》等英文技术书籍。为了能够编写 Solr 与大数据集成相关章节，我又耗费了大量时间通读了《Apache Flume Distributed Log Collection for Hadoop》《Hadoop in Action》《HBase in Action》《Learning Spark》等大数据相关的英文技术书籍。写作本书的过程也成为本人学习提升的过程，为此我花费了整整 1 年的时间。资历尚浅仍可以通过自身努力来弥补，所以我时时刻刻以严谨缜密的态度对待写进书里的每一段文字，除了怀揣着对技术的一种敬畏之情，我知道我还必须为读者负责。

然而造化弄人，在 2016 年的 2 月份，我的颈部莫名其妙长了一个肿瘤，这严重影响了我的身心健康。由于辗转于北京协和医院、解放军总医院等地投医救治，所以这本书的编写工作不得不临时中断。还好我没有放弃，于是在修养了半年之后，又进入了“挑灯夜战”的状态，开始以夜继日地赶稿子。因为已经立下了写书的豪言壮志，所以再苦再累我也是要写完的！由于生病，当初所在的公司要求我立即停薪修养，在看尽了世态炎凉之后，我毅然选择了辞职，打算专职将这本书写好，给读者一个交代。没有了经济来源，只靠自己多年来的积蓄维持生活。我顶着巨大的压力，在大病初愈的情况下，决定倾注全部精力打造这本书。很庆幸我坚持下来了。每天叫醒我的不是闹钟不是鸡汤，也不是其他竞争对手，而是我的决心，因为父母已两鬓白发，快要三十的我还孑然一身。所以我不能虚度光阴，需要为了我爱的人和爱我的人努力奋斗，从而改善他们的生活。这本书也算是给自己 30 岁生日提前备下的一份礼物，并借以纪念不悔的青春岁月。我知道和我有着类似经历的同学太多太多，因此希望这本书能够为学习 Solr 的你们带来帮助和鼓励：定好一个 Target，就永远不要放弃！

准备工作

随书提供了大量的示例代码（本书随书示例源码下载地址：https://github.com/yida-lxw/solr-book），其中涉及 MongoDB、ZooKeeper、Hadoop、HBase、Flume、Kafka、Storm、Spark、Scala 等知识点，不仅限于 Solr，所以对于 Java 初学者而言会有一定压力。尽管书中提供了部分大数据框架的集群搭建步骤，但是由于篇幅的限制不可能面面俱到，你还是需要另外查阅其他相关书籍或资料来补充大数据这方面的知识。由于随书源码是基于 Maven 构建的，因此你还需要掌握 Maven 的基本使用方法。为了尽最大努力满足大部分用户的需求，所以从第 14 章开始我将以 Solr 6.2.1 版本为例进行讲解，而 Solr6.x 是要求 JDK 1.8+ 版本的，那么在学习本书之前，你需要提前安装好 JDK 1.7 和 JDK 1.8。如果你有将 Solr 部署在 Tomcat 下的需求，那么你还应安装 Tomcat 环境。对于企业而言，SolrCloud 集群通常会部署在 Linux 环境下，因此本书 SolrCloud 部分是以 CentOS 6.5 为例进行讲解的，或许你还需要掌握 Linux 操作系统的基础知识以及一些 Linux 的常用命令。另外，由于 Solr 是基于 Lucene 构建的，

因此你最好拥有一定的 Lucene 基础再来学习本书内容会感觉更轻松。因为本书自始至终是以由浅入深的原则进行编写的，尽量细致入微地讲解每一步。当然，Solr 源码是使用 Java 编写的，这也要求你能够熟练掌握 Java 编程语言的知识，并拥有良好的编码基本功以及编程悟性。而 Solr 中的数据往往来自于关系型数据库，因此你最好是对关系型数据库有一定的了解。

如何阅读本书

全书分为上下两卷，总共 16 章，涵盖了 Solr 各个方面的知识点。本书从前到后按内容的难易程度以循序渐进的方式呈现出来。因此你只需要拥有足够的毅力将它阅读完，当然最好是能够边读边上机实践，就可以掌握 Solr。此外每章之间都是相互独立的，如果你对于某章的内容已经非常熟悉，那么可以直接跳过选择感兴趣的章节进行学习。当然还是建议大家能够通读本书，系统学习 Solr，这样才会对 Solr 有一个更完整的理解，为你日后从事 Solr 相关的开发工作打下夯实的基础。本书每章开头部分都列举了该章的主要知识点，可以让你快速了解本章能够学习到的内容。虽然本书中演示的示例代码在随书源码中都可以找到，但是我还是建议大家能够实际动手去敲一遍，毕竟只有亲手实践过，才能将遇到的各种问题真正悟透并彻底解决。这个过程虽是艰辛的，但也是深刻的，因为解决问题对于程序员来说就是积累经验的机会。

面向的读者

- Java 开发工程师；
- 架构师；
- Solr 技术爱好者；
- 各大高校或 IT 培训机构的学弟学妹们。

勘误与反馈

在编写本书的过程中，尽管我倾注了大量时间与精力，但是由于水平有限，书中难免会存在不足与疏漏之处，还请大家多多批评指正。如果你在阅读本书过程中有任何疑问或者建议需要向我反馈，可直接发送 E-mail 至 736031305@qq.com 或者添加个人微信（13476669029）联系我。

致谢

不知道你拿到这本书的时间是哪一年哪一个季节，但是对我来说，这都是我在自己 30

岁之前完成的一个最大的心愿。这是国内真真正正全面介绍 Solr 技术的第一本中文书籍，很开心我做到了。

想感谢的人很多，首先要谢谢爸妈，在我生病期间无微不至地照顾我，并无条件地支持我。

谢谢一路以来理解并鼓励我的朋友和粉丝们，是你们让我不断坚持前行。

谢谢机械工业出版社华章公司的杨福川、高婧雅、李艺在这一年当中对我写作的信任与帮助，没有你们辛勤的付出，就不可能有这本书的面市。非常开心和幸运能够与你们共同完成这样一本书籍。

谢谢我的 Java 启蒙老师习晨龙，是您带我进入了 Java 世界，从此我在汲取知识的路上甘之如饴。

谢谢在这么多年的工作中所有帮助过我的同事，我会一直记得你们。

最后需要感谢的还是我自己，感谢曾经的年少轻狂，感谢一直都存在的梦想，对于梦想我从来没有也永远不会放弃。所以如果你还有梦想，为了你爱的人，为了你自己，请永远不要放弃！

Contents 目　录

第 11 章 Chapter 11

Solr 高级查询

通过第 11 章，你将可以学习到以下内容：

- 掌握如何使用 Function Query 以及如何自定义 Function Query；
- 掌握如何使用 Geospatial Query；
- 掌握如何使用 Pivot Facet 和 Subfacet；
- 掌握如何使用 JSON Facet API 来实现复杂的数据统计查询；
- 掌握如何使用 Solr 中的其他查询组件，比如 Elevation（竞价排名组件）；
- 掌握如何使用 Solr 中的 Result Clustering 组件实现自动结果集聚类分组。

Solr 作为一个强大的文本搜索平台，能够根据输入关键字查询并返回索引文档，你可能也已经了解到了 Solr 的一些核心功能，比如文本分词、关键字高亮、结果集分组等。尽管对于大多数搜索程序来说，将那些与用户查询最佳匹配的索引文档返回给用户是非常重要的，但是 Solr 还有另外一个比较常见的使用场景：聚集结果集用于数据统计分析。Solr 的 Pivot Facet 支持叠加统计多个 Facet（维度），它能够在单个查询中对任意的聚合分类进行计算统计，这使得 Solr 在提供数据分析报告方面变得很有用并且还十分高效。Solr 另一个核心功能就是在查询时能够对数据执行一个 Function（函数）进行动态计算，函数计算后的结果可以被用于 Filter Query、文档的相关性评分、文档的排序、作为文档的“伪域”被返回。Solr 还提供了强大的 Geospatial（地理空间）查询功能，Geospatial 查询允许你根据一个点或者一个区域进行多边形查询，或以经纬度为圆心在指定半径的圆内进行查询，实现附近的位置查询（比如查询当前用户所处位置附近的酒店或饭店）。有时候，你期望在返回的索引文档的域中引用外部数据源，Solr 提供了这个功能。Solr 还支持在同一个 Solr 实例内跨 Core 在一个外键域上执行 Join 操作，这类似于 SQL 里的两个表根据外键进行多表连接查询。上

述每个复杂的功能都会在本章中进行讲解。

11.1 Solr 函数查询

Solr 中的 Function Query（函数查询）允许你为每个索引文档执行一个函数进行动态计算值。Function Query 是一个比较特殊的查询，函数动态计算后得到的值可以作为一个关键字添加到查询中，也可以作为文档的评分，就像是一个普通的关键字查询同时还能生成相关性评分。通过使用 Function Query，函数动态计算值可以被用于修改索引文档的相关性评分，以及查询结果集排序，而且函数动态计算值还可以作为一个“伪域”被动态添加到每个匹配的索引文档中并返回给用户。Function Query 还支持嵌套，意思就是一个 Function 的输出可以作为另一个 Function 的输入，Function 支持任意深度的嵌套。

11.1.1 Function 语法

Solr 中标准的 Function 语法是先指定一个 Function 名称，后面紧跟着一对小括号，小括号内可以传入零个或多个输入参数，语法使用示例如下：

```
functionName()
functionName(input1)
functionName(input1, input2)
functionName(input1, input2, ..., inputN)
```

Function 的输入参数可以是以下任意一种形式：

- 一个常量值（数字或者字符串），语法：

```
100, 1.45, "hello world"
```

- 一个域名称，语法：

```
fieldName, field(fieldName)
```

- 另外一个 Function，语法：

```
functionName(...)
```

- 一个变量，语法：

```
q={!func}min($f1,$f2)&f1=sqrt(popularity)&f2=1
```

尽管 Solr Function 乍一看让人有点不知所措，其实 Solr 文档中定义了每个 Function 的输入参数的类型，大部分的 Function 都遵循 Function 的标准语法，但是 Constant Function（常量函数）、Field Function（域函数）、Parameter Substitution（替换变量）这些属于特例，它们支持另一种简单语法。Constant Function（常量函数）的语法就是值本身。Field Function（域函数）的语法就是域的名称被一个名称为“field”的函数包裹。Parameter Substitution

（替换变量）的语法就是函数的输入变量使用的是一个 $ 开头的变量，该变量引用自请求 URL 的查询文本中定义的变量。除了这 3 个特例，其他函数都使用标准的 Function 语法。

因为 Function 的所有输入参数可以被看作是一个 Function（函数）（常量值可以被看作常量函数），所以 Function 的标准语法在概念上来讲，就可以理解为 functionName(function1, ..., functionN)。假设索引文档中有个 fieldContainingNumber 域，它其中有个值为 - 99，那么请思考下面几个 Function 的使用示例：

```
max(2, fieldContainingNumber)                    //输出结果：2
max(fieldContainingNumber, 2)                    //输出结果：2
max(2, -99)                                      //输出结果：2
max(-99, 2)                                      //输出结果：2
max(2, field(fieldContainingNumber))             //输出结果：2
max(field(fieldContainingNumber), add(1,1))      //输出结果：2
```

从上面示例你会注意到，你可以为 Constant Function 常量函数（甚至你可以为其他任意标准函数）使用 Field Function 进行包装，尽管输入的参数顺序以及每个输入参数的含义会有所不同，但是它们最终都是用于计算 - 99 和 2 之间的最大值。将一个函数的输入参数看作另外一个函数的好处就是它允许你用任意的嵌套函数来实现复杂计算。并不是所有的 Function（函数）都支持同样类型的输入参数，有些 Function（函数）期望接收字符串类型常量参数，而另外一些 Function（函数）可能期望接收 Integer 或者 Float 类型的数字。假设 fieldContainingString 域的域值为 "hallo"，请思考下面的函数调用示例：

```
strdist("hello", fieldContainingString, edit)  //输出结果：0.8
strdist("hello", "hallo", "edit")              //输出结果：0.8
```

strdist 函数用于计算两个字符串之间的相似度，相似度计算是基于一个指定的算法进行计算，使用哪种算法是通过函数的第 3 个参数进行指定，我们示例中的 "edit" 表示采用编辑距算法。假如我们将参数的数据类型指定为错误的，函数将会返回什么呢？

```
strdist("hello", 1000, edit)                   //输出结果：0
strdist(1000, "1000", edit)                    //输出结果：1
strdist("1001", 1000, edit)                    //输出结果：0.75
```

你可能会觉得函数会抛出异常，然而实际上函数内部会适当地自动进行数据类型转换，比如在示例中，将数字常量 1000 转换成字符串 "1000"。在大多数情况下，你并不能安全地将一个字符串转换成一个数字，此时 Solr 可能会抛出一个异常。因此，需要谨记：函数嵌套确实很好用，但是并不是所有的函数都可以随意嵌套，你需要考虑每个函数的输入参数类型是否正确。

Solr 的 Function 可以影响相关性评分，可以被用于 Filter Query 过滤结果集，可以基于函数计算值进行排序，可以将函数计算值作为索引文档的“伪域”并返回，甚至可以基于函数计算值进行 Facet 查询统计。下一节我们将深入学习这些用法。

11.1.2 使用函数查询

为了便于后续的示例讲解，请大家从随书源码中获取 Core 的相关配置文件、测试数据及导入测试数据的测试类，根据我们前面章节所学的知识将本节测试环境需要的 "news" Core 搭建好。

在 Solr 中执行一个典型的关键字查询，需要在倒排索引中查找关键字，同时计算每个匹配索引文档的相关性评分，从而决定哪些索引文档与查询关键字比较相关，最后作为结果集返回。然而查询不仅能基于搜索关键字，你可以在查询中插入一个 Function 并将其看作另外一个搜索关键字。为了演示 Function Query，请建立 "news" Core 并运行随书源码中的 IndexNews 类导入测试数据。假如已经成功导入了测试数据，可以执行下面的查询示例：

```
http://localhost:8080/solr/news/select?
q="United States" AND France AND President AND _val_:"recip(ms(date),1,100,100
)"&indent=true
```

上面的查询表示查询包含 "United States" 短语且包含 France 和 President 关键字，并且函数计算值在 [1, 100] 区间范围内的索引文档。这里有 3 个关键点需要引起你的注意：

- _val_ 语法：用于注入一个 Function Query，这里的 _val_ 可以看作主查询中的一个查询 Term。
- Function Query 并不会改变最终返回结果集中索引文档的总数。
- 查询的最后相关性评分一般是查询中每个 Term 的相关性评分的总和，"United States"、France 和 President 这些 Term 的相关性评分是基于 tf-idf 相似度算法进行计算的，但是 Function Query 的评分计算是函数自身的计算值。

基于上述 3 点，你可以了解到示例中的 Function Query 是为了给新添加的索引文档进行加权。最新的索引文档的相关性评分可能是 100，而最旧的索引文档的评分可能是 1，剩下的索引文档的评分会落在 [1, 100] 之间。注意，每个索引文档的最后评分是标准化的，这意味着每个索引文档的最后评分不会都到达 100 分，最近添加的索引文档相比之前显示会更靠前。

Function 在 Solr 中无处不在，它可以对用户的 q 参数进行加权，它还可以在不同的 Query Parser 中使用，比如在 eDisMax Query Parser 中通过 bf 参数指定 Function；它还可以作为 Filter Query 的一部分，用于索引文档排序等。但是最重要的是你需要了解 Function Query 是如何被执行的。前面的示例中你已经见过了 "_val_" 这样的语法，你可能还记得我们之前介绍过的 Function Query Parser，可以通过一个本地参数 !func 来构造一个 Function Query，比如：{!func}functionName(…)。Function Query 本质就是将函数计算值作为构造的函数查询的评分，因此，以下几种查询语法是等价的：

```
q=solr AND _val_:"add(1, boostField)"
q=solr AND _query_:"{!func}add(1, boostField)"
q=solr AND {!func v="add(1, boostField)"}
```

为一个查询，添加一个 Function 看起来非常有用，它能修改查询匹配的索引文档的评分。如果你期望过滤掉函数计算值不在指定范围内的索引文档，可以使用 Function Range Query Parser 来解决函数计算值范围过滤。

如果你需要根据函数计算值的范围来过滤索引文档，那么 Function Range Query Parser（简称 frange）会比较适用你的使用场景，Frange 过滤器通过执行一个指定的 Function Query，过滤掉函数计算值不在指定范围内的索引文档。为了演示这种功能，我们搭建测试环境。这里会用到随书源码中的 "salestax" Core，Core 相关配置文件和测试数据以及导入数据测试类请从相应章节中查找获取。导入完成之后，请看下面这个查询示例：

```
http://localhost:8080/solr/salestax/select?q=*:*&
fq={!frange l=10 u=15}product(basePrice, sum(1, $userSalesTax))&
userSalesTax=0.07
```

以上查询先通过 sum 函数计算 $userSalesTax 和 1 的价格之和，然后将 basePrice 域的域值与 sum 函数计算返回值通过 product 函数求乘积，最后通过 frange 过滤器的 l（即 lower 表示最小值）和 u（即 upper 表示最大值）参数定义了 product 函数计算返回值的取值范围，符合这个区间范围限制的索引文档将会被返回。你还可以设置 incll（即 include lower）和 inclu（include upper）参数来指定是否包含两个边界值。

你可能会说，能不能自定义一个 Function 来灵活地过滤任意查询匹配的结果集？关于自定义 Function 相关内容会在本章的后续章节讲解。现在你已经知道如何为查询添加 Function，并且理解了函数评分是如何计算的，接下来让我们继续学习使用函数动态计算值来代替静态的域值。

11.1.3　将函数计算值作为“伪域”返回

在上一节，你已经了解到函数的输入参数可以被看作是一个函数，既然如此，那么似乎我们可以使用 Function 来替换 Field，因为 Field 和 Function 最终都是返回一个值。事实也是如此，你不仅可以为每个索引文档动态计算得到一个数值，还可以将这个数值作为一个“伪域”随索引文档一起返回。重新回到我们在上一节中的 "salestax" 示例，执行下面这个查询：

```
http://localhost:8080/solr/salestax/select?q=*:*&
userSalesTax=0.07&
fl=id,basePrice,product(basePrice, sum(1, $userSalesTax))
```

上面这个查询返回的结果会是怎样的呢？正如你看到的那样，你会发现返回的索引文档中多了一个“伪域”，“伪域”的域名称就是我们定义的函数表达式，“伪域”的域值就是函数表达式最终的计算值。之所以称为“伪域”，是因为它并不真正存在于我们的索引数据中，但是它仍然会像其他存储域一样被一起返回。“伪域”的名称使用函数表达式可能会显得冗长难看，不过值得庆幸的是，Solr 提供了为“伪域”定义任意你想要的别名的功能，具

体如何为“伪域”定义别名，请看下面这个示例：

```
http://localhost:8080/solr/salestax/select?q=*:*&
userSalesTax=0.07&
fl=id,basePrice,totalPrice:product(basePrice, sum(1, $userSalesTax))
```

这里我们为"product(basePrice, sum(1, $userSalesTax))"这个伪域定义了一个别名totalPrice，最终返回结果里伪域名称就是我们这里定义的别名了。正因为你可以为“伪域”定义任意的别名，因此也就意味着可以将“伪域”的别名定义为索引文档中真实存在的域的域名称，这样就可以直接使用“伪域”的值来冒充该域的真实域值。当你期望根据用户权限来控制某些用户没有权限访问某个域的真实域值的时候，通过“伪域”别名来冒充真实域会对你很有用。

通过使用 Function，你可以在域的域值返回之前对其进行任意操纵，比如经过函数计算变换它的值。你不仅可以通过函数修改文档中任何域的域值，还可以通过函数修改文档的相关性评分，从而影响文档是否应该被返回，或者文档在返回的结果集中的排序。

11.1.4 根据函数进行排序

在上一节，你了解了如何将一个函数的动态计算值添加到索引文档中作为一个“伪域”在查询结果集中返回；你也知道了如何根据函数计算值来对查询结果集进行过滤，以及如何使用函数来修改匹配文档的相关性评分。接下来，让我们继续学习如何基于函数动态计算值来对查询结果集进行排序。根据函数动态计算值来对查询结果集进行排序的语法与普通查询中根据某个域排序的语法没什么太大的不同，具体请看下面这个查询示例：

```
http://localhost:8080/solr/salestax/select?q=*:*&
userSalesTax=0.07&
sort=product(basePrice, sum(1, $userSalesTax)) asc, score desc
```

上面这个查询，根据 product 函数计算值升序进行排序，然后再按文档的评分降序排序，你可以结合其他函数构造更为复杂的 Function Query，比如：

```
http://localhost:8080/solr/salestax/select?
q=_query_:"{!func}recip(ms(date),1,100,100)"&
userSalesTax=0.07&
totalPriceFunc=product(basePrice, sum(1, $userSalesTax))&
fq={!frange l=10 u=15 v=$totalPriceFunc}&
fl=*,totalPrice:$totalPriceFunc&
sort=$totalPriceFunc asc, score desc
```

上面这个查询首先根据 Function Query Parser 对"{!func}recip(ms(date),1,100,100)"查询表达式进行解析，构造成 Function Query；然后通过 _query_ 语法将 Function Query 转换成普通的 Query，转换后的查询并没有过滤任何索引文档，它只是用来根据文档的 date 域的时间远近对索引文档进行加权；然后通过 fq 对 $totalPriceFunc 变量表示的函数最终计算