《Redis 缓存》实训指导书

院系名称：经管信息学院

课程代码：31091330

总学时数：32

适用专业：大数据技术与应用

编 制 人：韦祥

编制日期：2020 年 2 月

审 核 人：

审 定 人：

《Redis 缓存》实训指导书

一、实训目的与要求

《Redis 缓存》主要目的是让学生通过这门实践技能课程的学习了解和掌握 nosql 和 sql 的区别，mongodb，redis 的基本方法，通过不断上机实训练习达到解决实际的问题。因此，在本学期特设置此课程设计，集中一段时间使学生综合运用所学习的 SSM 知识及以前所学习的计算机方面的知识，按照企业流程，完成一个相对具体、综合的网站，全面巩固学生的知识，培养学生解决实际问题的能力，从而达到学以致用的目的。

二、实训内容

（一）实例实训

以电 nosql 设计的实例指导学生如何独立完成设计和制作。让学生在机房实际操作，按照给定的实例完成实例中站点的创建和设计制作。

（二）自建站点实训

让学生自行选择站点的主题，从规划站点到上传文件一步一步完成整个项目的创建、调试和上传工作。

（三）总结

对学生的全部作品进行考核，并选择典型的案例对实训的结果进行考核。

二、参考课时

标题	实训内容	实训课时
预备	实训内容概述	3
实训一	基本知识回顾	3
实训四	mongodb 操作	3
实训五	redis 操作	9
总计		60

三、实训材料准备

（一）软件准备

Eclipse 3.6 或以上版本，jdk1.8 或以上版本。

（二）硬件准备

教师用机：Windows 7 及以上版本。

学生用机：Windows 7 及以上版本。

四、综合实训考核办法：

系统文档	20 分
编写代码	30 分
程序调试	10 分
实训出勤	20 分
技术含量	10 分
美工设计	10 分

目　　录

实训一 基本知识回顾

一、 实训目的和要求

温习java的课程重点难点，使学生对eclipse各方面的操作知识系统的由“片”的认识转向“面”的认识。

二、 实训内容

eclipse的基本操作：创建项目，类，执行等。

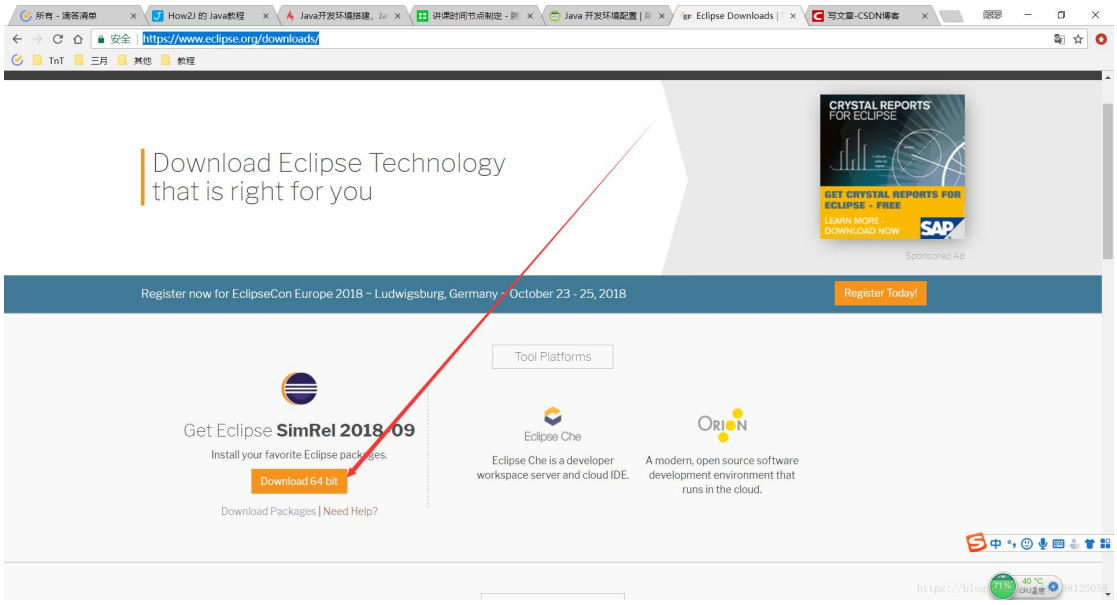
三、 实训准备

eclipse中文版。

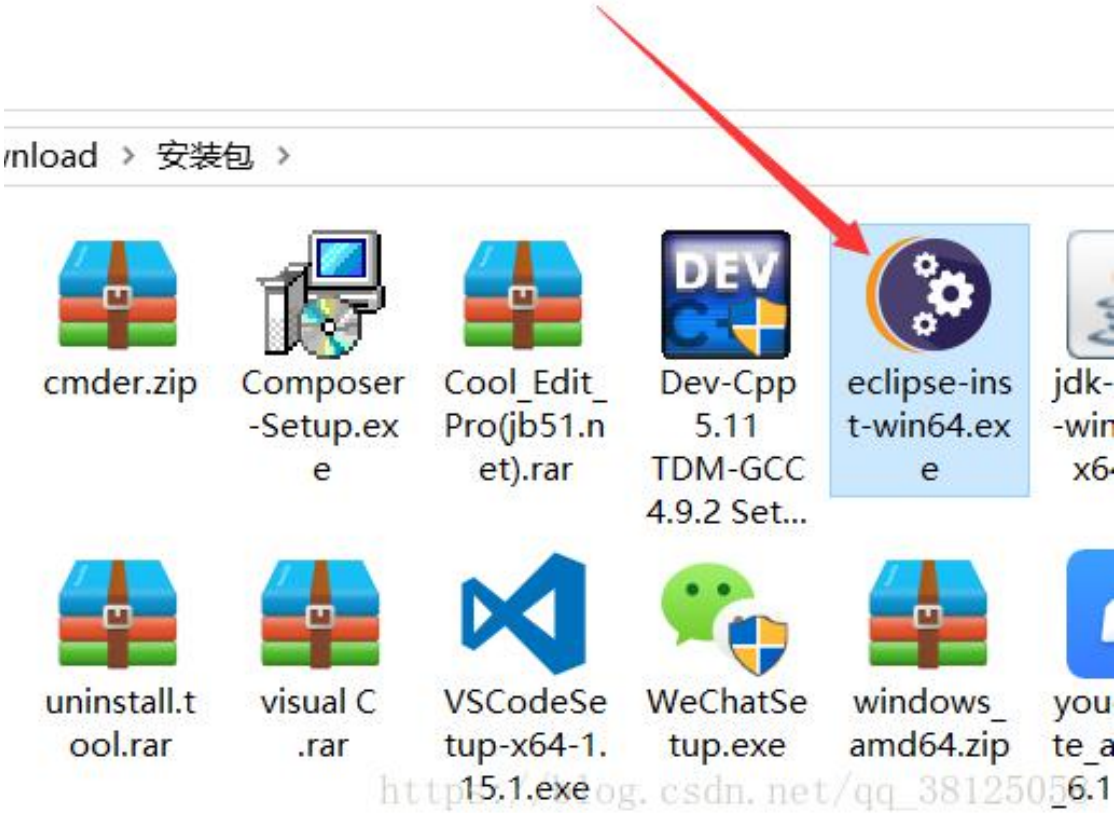
四、 实训步骤

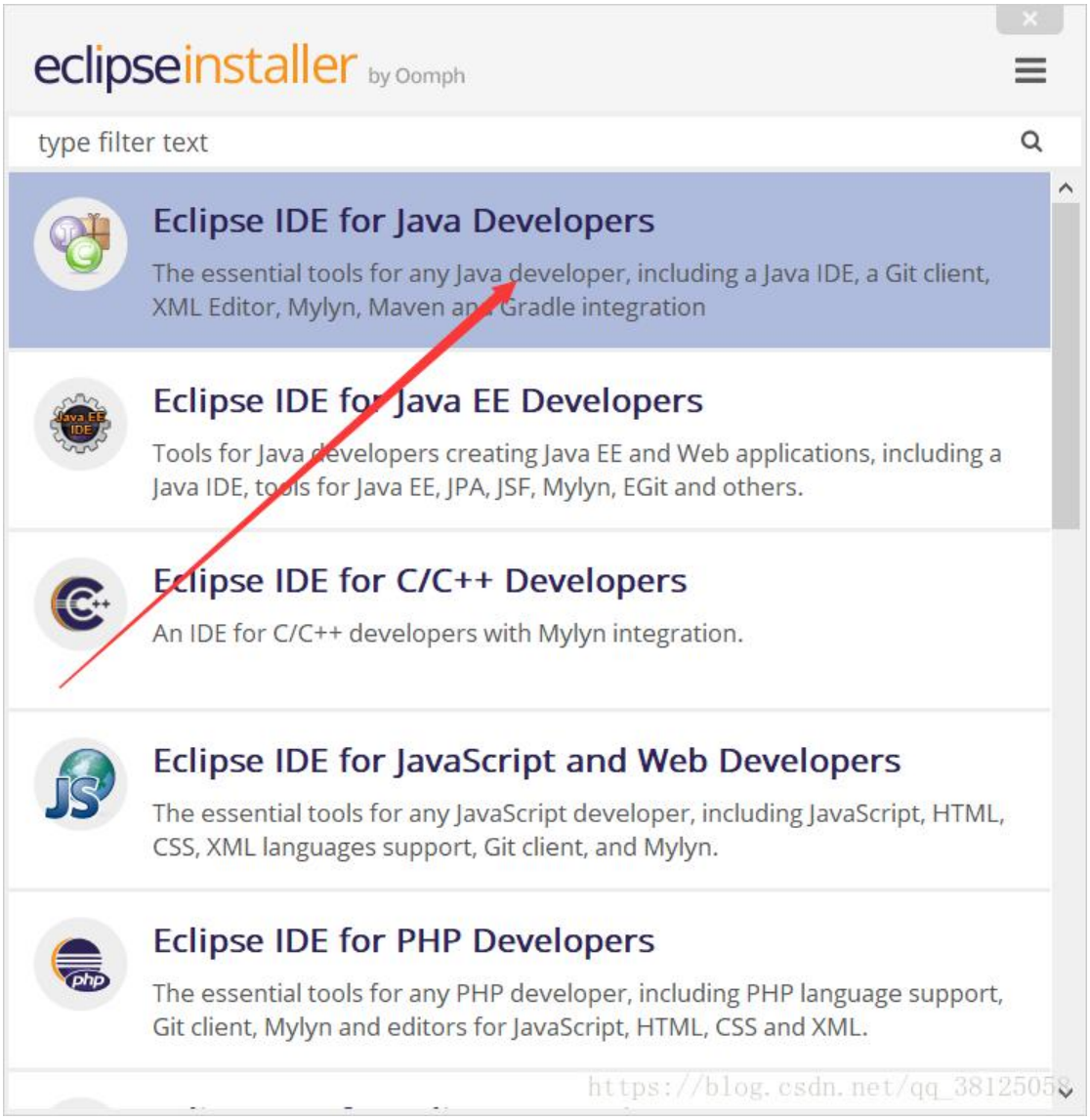
各实训指导教师按照所代学生的情况不同选择性地按下列步骤温习eclipse的重点难点知识。

下载地址：https://www.eclipse.org/downloads/



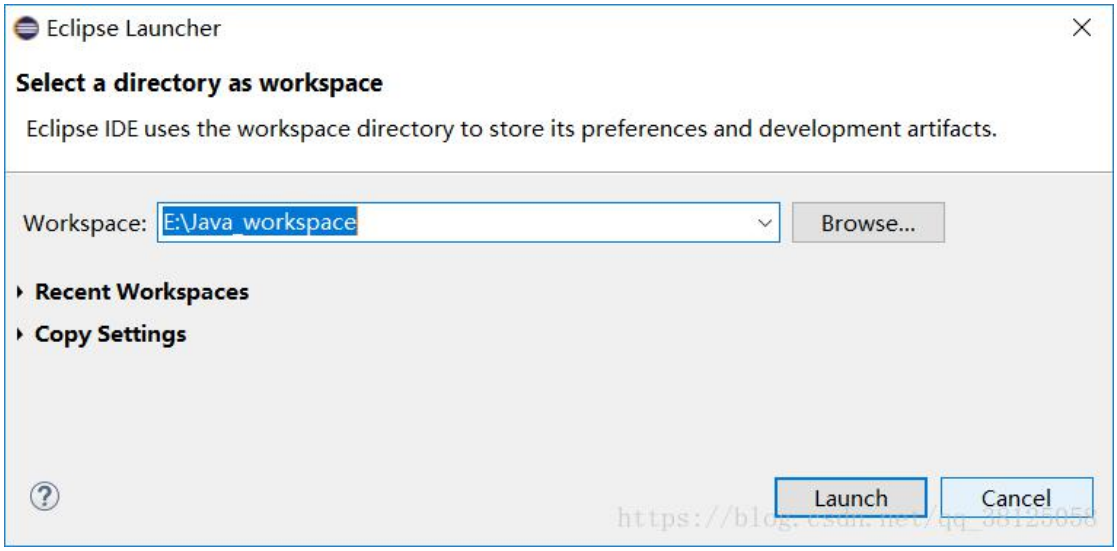
下载完成之后，双击运行，下载安装。

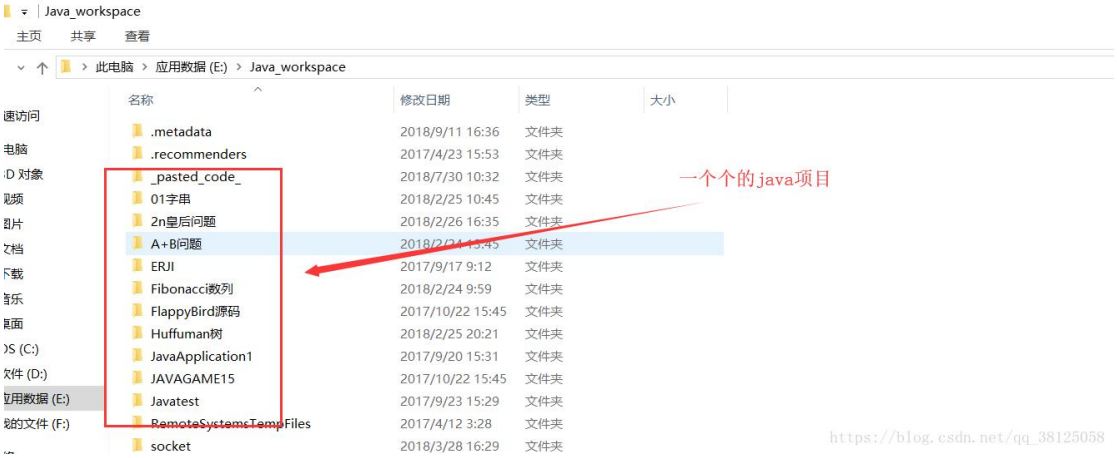




安装完成之后，桌面上就会多了这么一个图标，双击运行

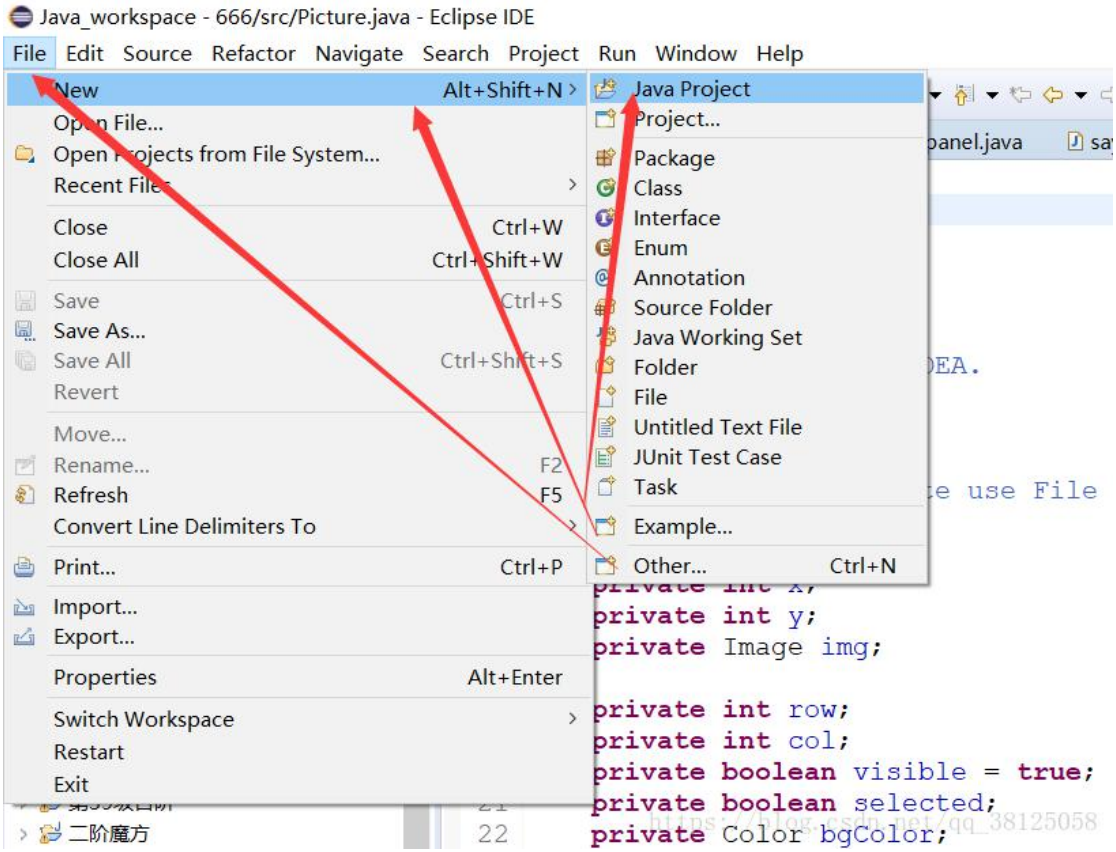
记得设置自己的工作空间，以后自己所有写的代码都会存放到这里。

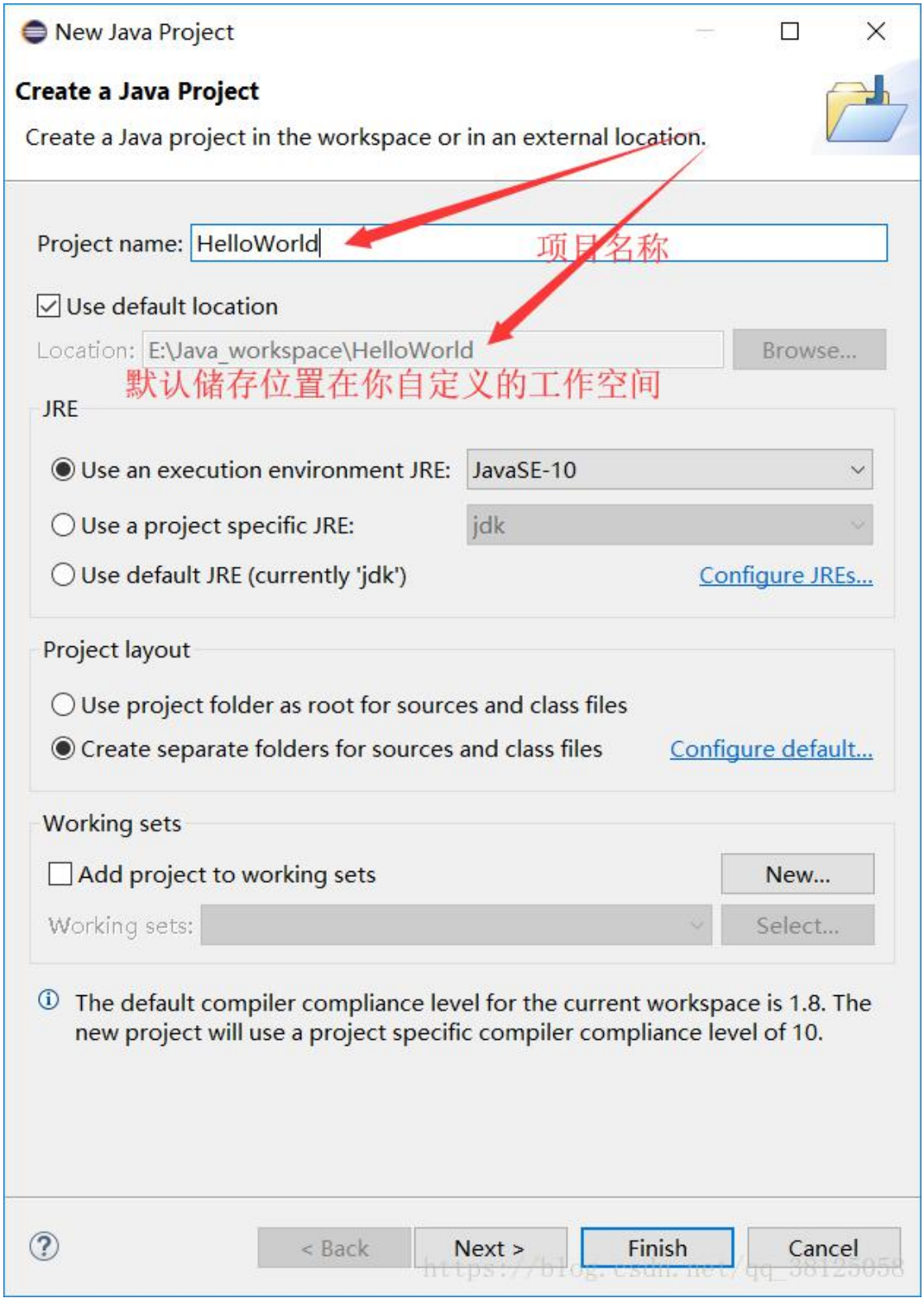


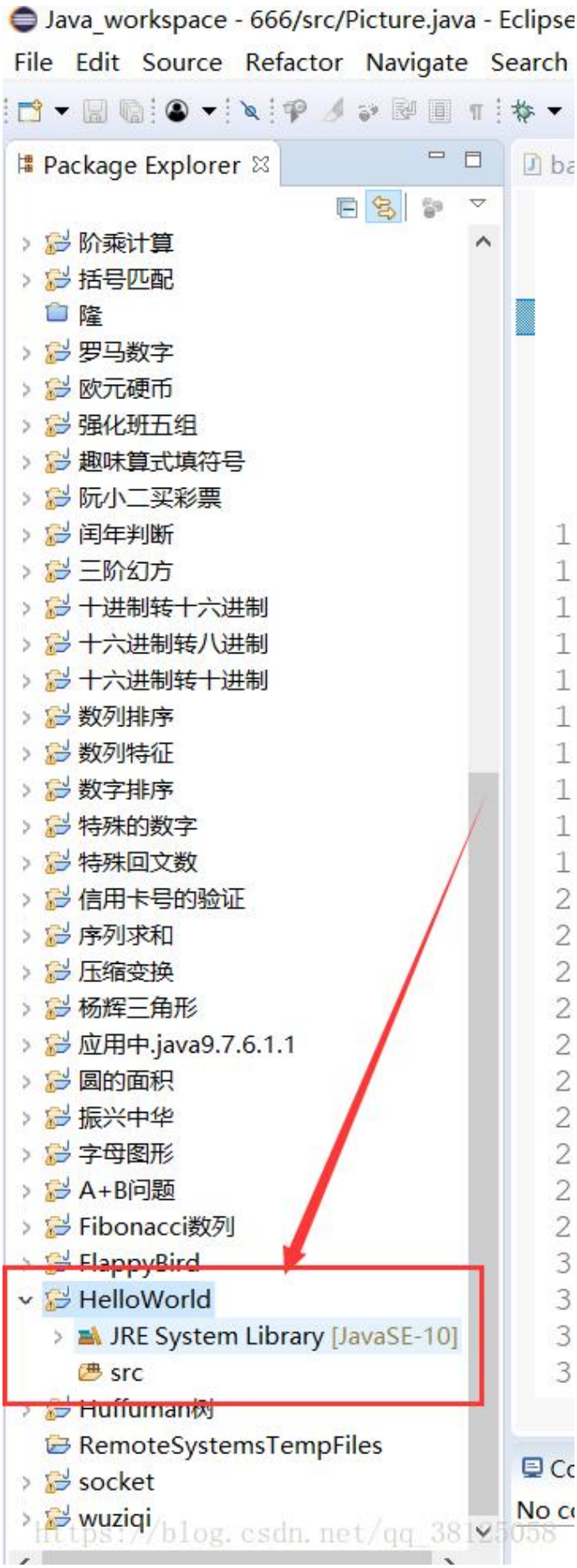


使用：

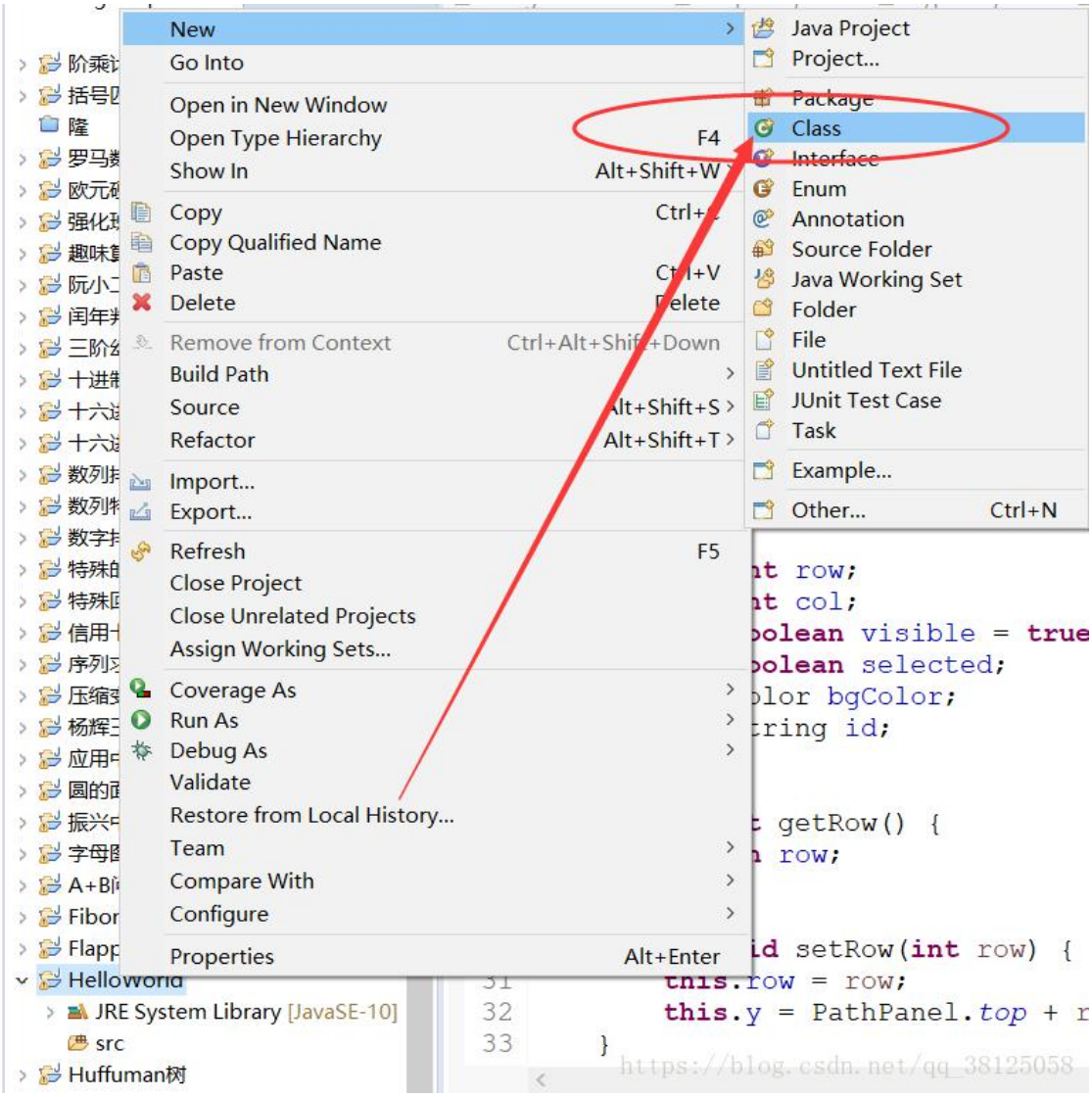
一、新建 java 项目

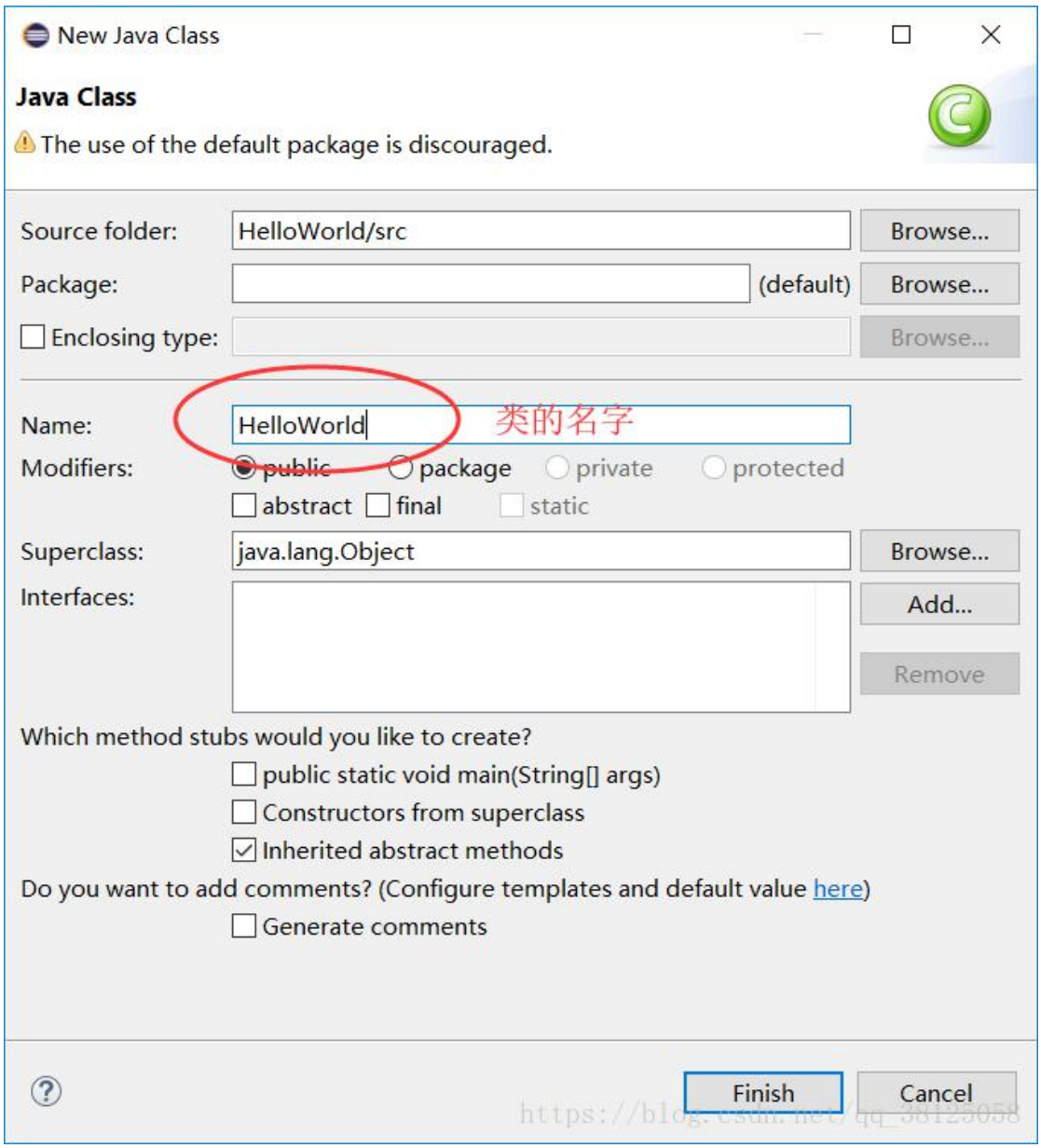

Java_workspace - 666/src/Picture.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N
panel.java
private int y;
private Image img;
private int row;
private int col;
private boolean visible = true;
private boolean selected;
private Color bgColor;
二阶魔方
22


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: HelloWorld
项目名称
Use default location
Location: E:\Java_workspace\HelloWorld
Browse...
默认储存位置在你自定义的工作空间
JRE
Use an execution environment JRE: JavaSE-10
Use a project specific JRE: jdk
Use default JRE (currently 'jdk')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
New...
Working sets:
Select...
The default compiler compliance level for the current workspace is 1.8. The new project will use a project specific compiler compliance level of 10.
< Back
Next >
Finish
Cancel

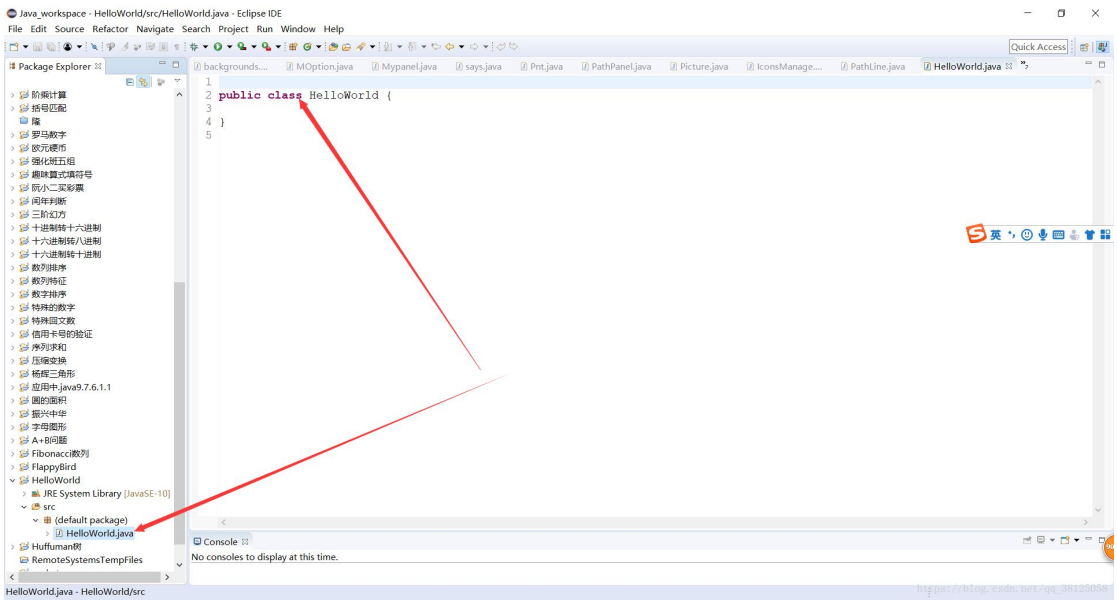

Java_workspace - 666/src/Picture.java - Eclipse
File Edit Source Refactor Navigate Search
Package Explorer
阶乘计算
括号匹配
隆
罗马数字
欧元硬币
强化班五组
趣味算式填符号
阮小二买彩票
闰年判断
三阶幻方
十进制转十六进制
十六进制转八进制
十六进制转十进制
数列排序
数列特征
数字排序
特殊的数字
特殊回文数
信用卡号的验证
序列求和
压缩变换
杨辉三角形
应用中.java9.7.6.1.1
圆的面积
振兴中华
字母图形
A+B问题
Fibonacci数列
FlappyBird
HelloWorld
JRE System Library [JavaSE-10]
src
Huffuman树
RemoteSystemsTempFiles
socket
wuziqi
https://blog.csdn.net/qq_38125058

二、选中刚刚建立的 java 项目，右键点击，新建类



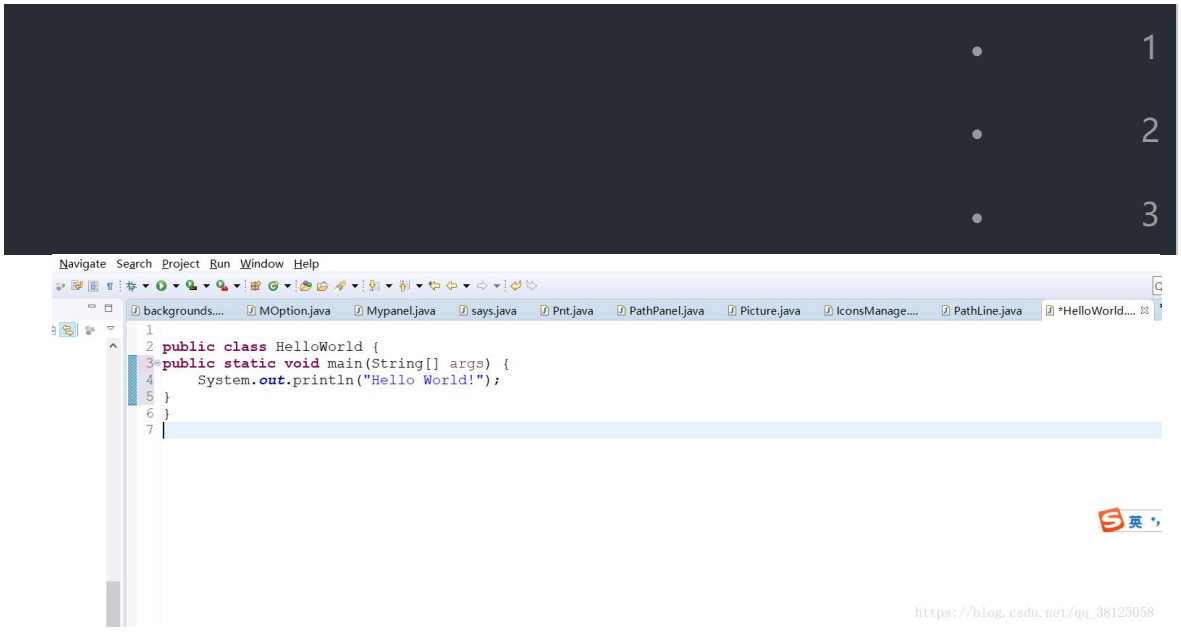


点击 finish，会发现 src 目录下多了一个.java 文件，即我们刚刚建立的 HelloWord 类

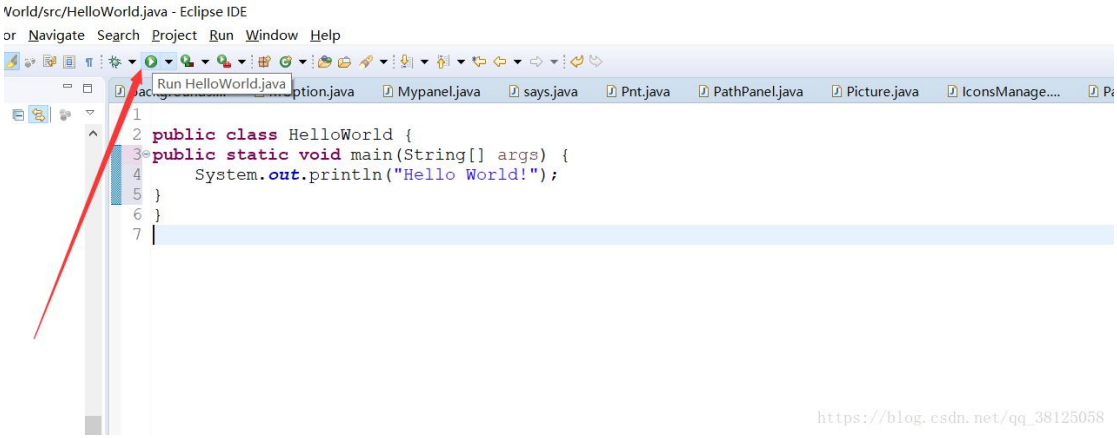


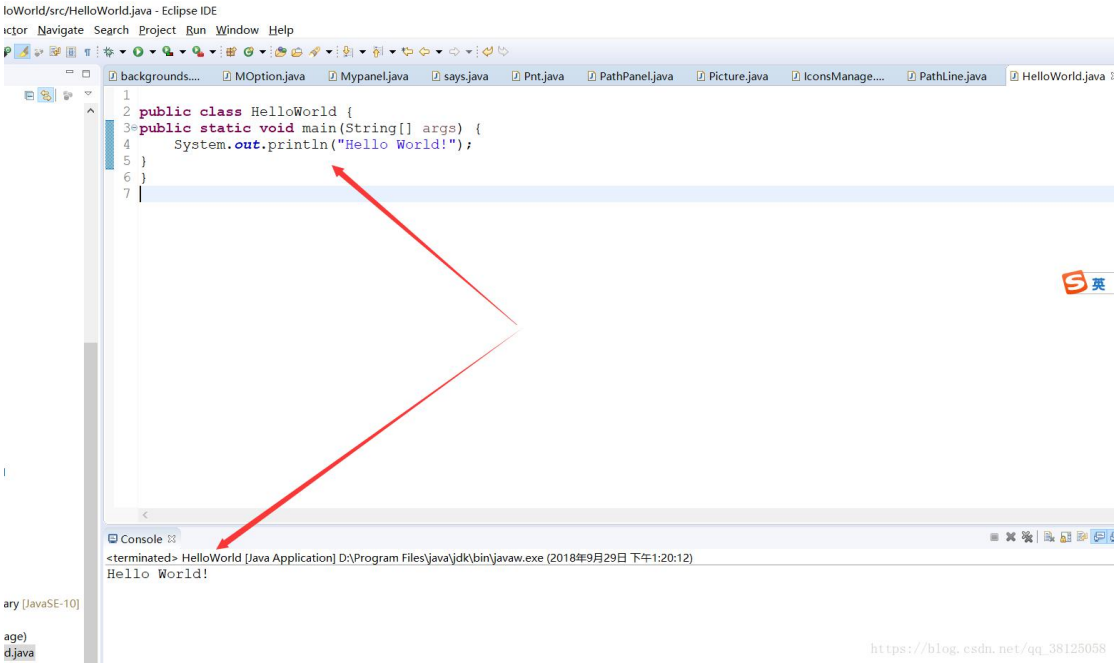
输入如下代码：

```
public static void main(String[] args) {

        System.out.println("Hello World!");

}
```



点击**执行**按钮，如下，就可以看到我们的执行结果了

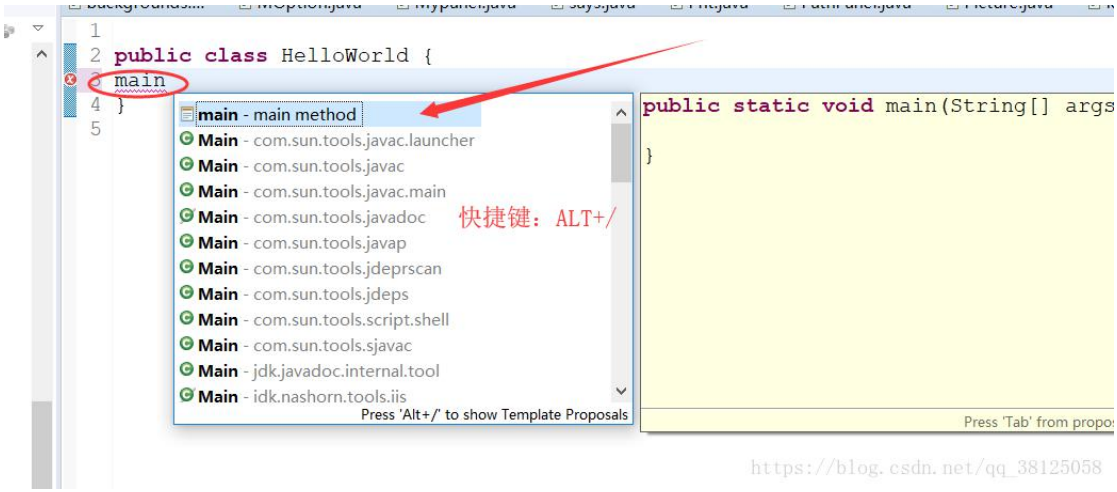


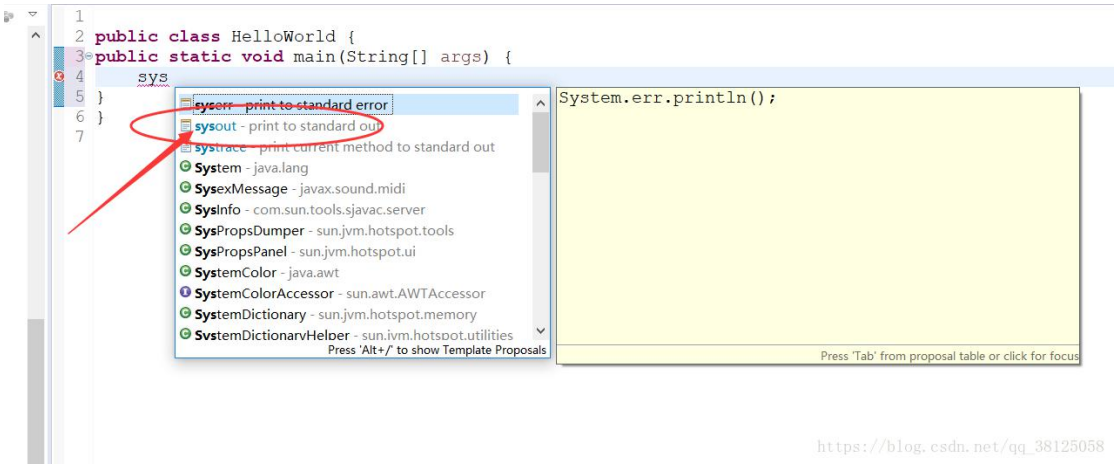


介绍几个快捷键：

1、代码提示：alt+/

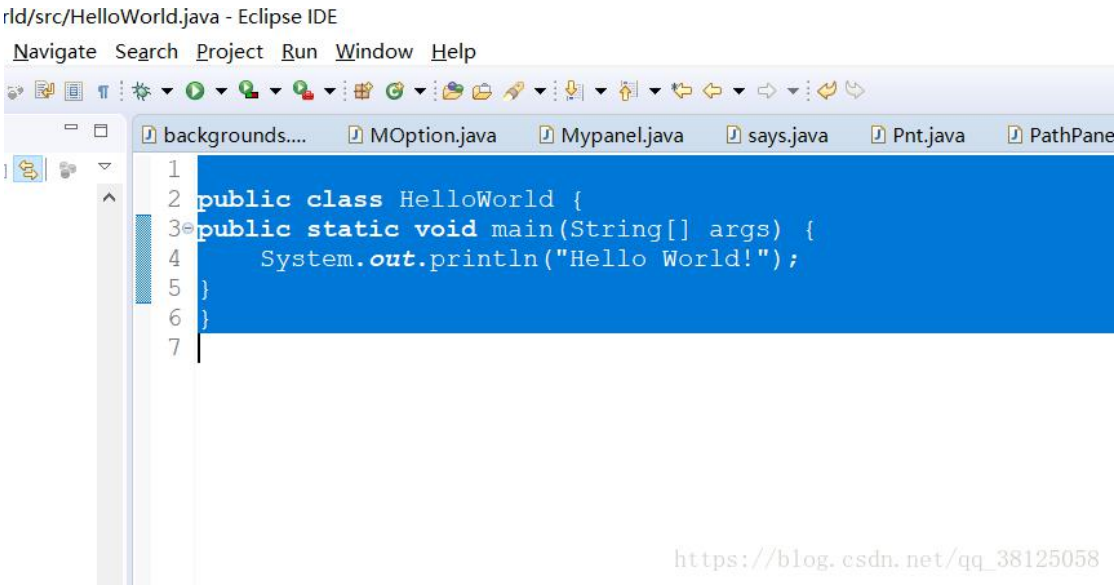
例：





2、全部选中：Ctrl+A

代码对齐：Ctrl+I



3、Ctrl+Z：撤回

Ctrl+Y：反撤回

4、Ctrl+S：保存

5、Ctrl+/：注释

五、 实训方法

使用投影进行讲解演示，并抽样进行检查。

六、 考核办法

此部分实训内容采用抽样考察的方法，考核以操作的熟练程度和正确性为评分标准，以 A（优秀）、B（良好）、C（及格）、D（不及格）为成绩标准。

实训二　mongodb 操作

一、　实训目的和要求

通过对 mogodb 的学习，使学生了解 mongodb 的步骤，让同学们 mogodb 的粗略和宏观认识得到细化，使其懂得如何独立面对 mongodb 的设计。

二、 实训内容

讲解和演示 spring 设计方法。

三、 实训准备

eclipse。

四、 实训步骤

五、 实训方法

使用投影讲解演示，并抽样进行检查。

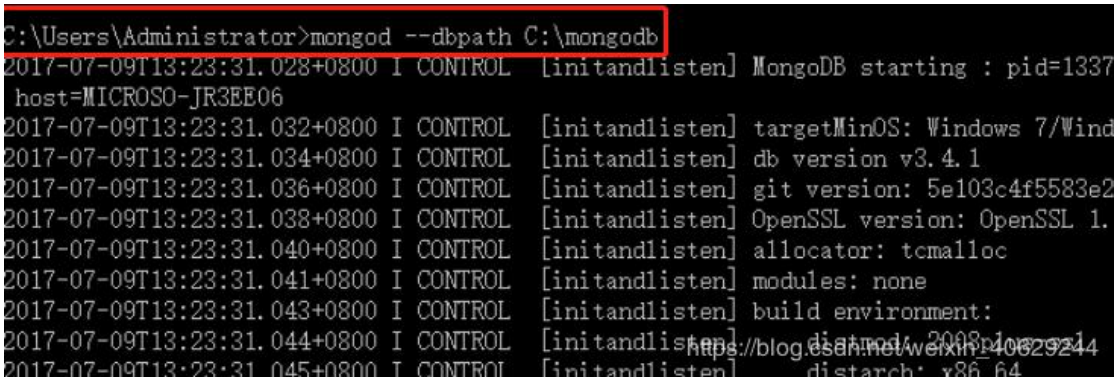
六、 考核办法

此部分实训内容采用抽样考察的方法，考核以操作的熟练程度和正确性为评分标准，以 A（优秀）、B（良好）、C（及格）、D（不及格）为成绩标准。

1. MongoDB 数据库使用

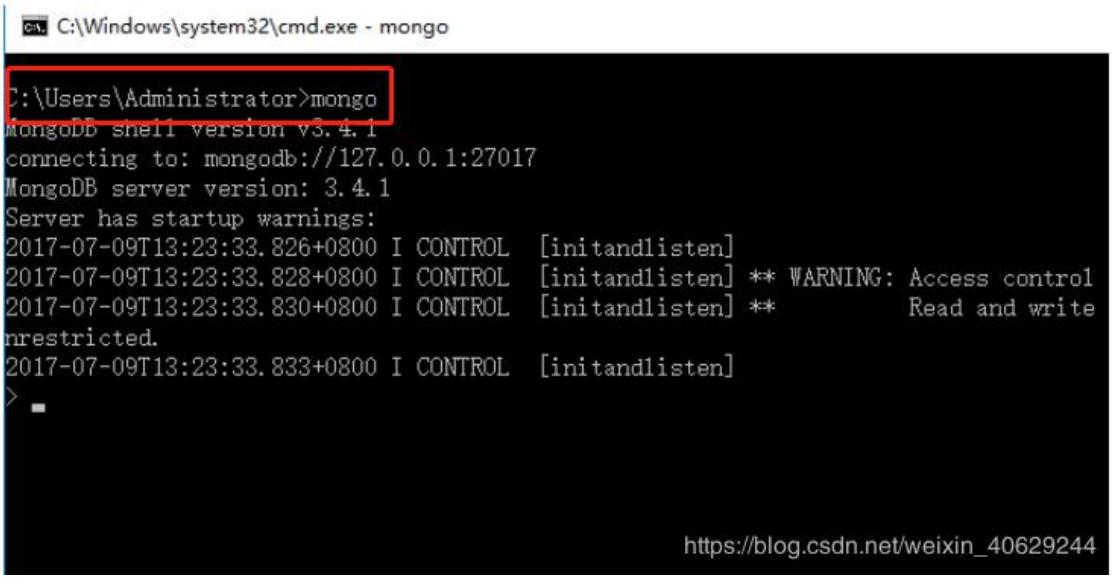
(1). 开启 mongodb 服务。

要管理数据库，必须先开启服务，开启服务使用 mongod --dbpath c:\mongodb，"c:\mongodb"为当前自己的数据库所在路径。



(2). 管理 mongodb 数据库。

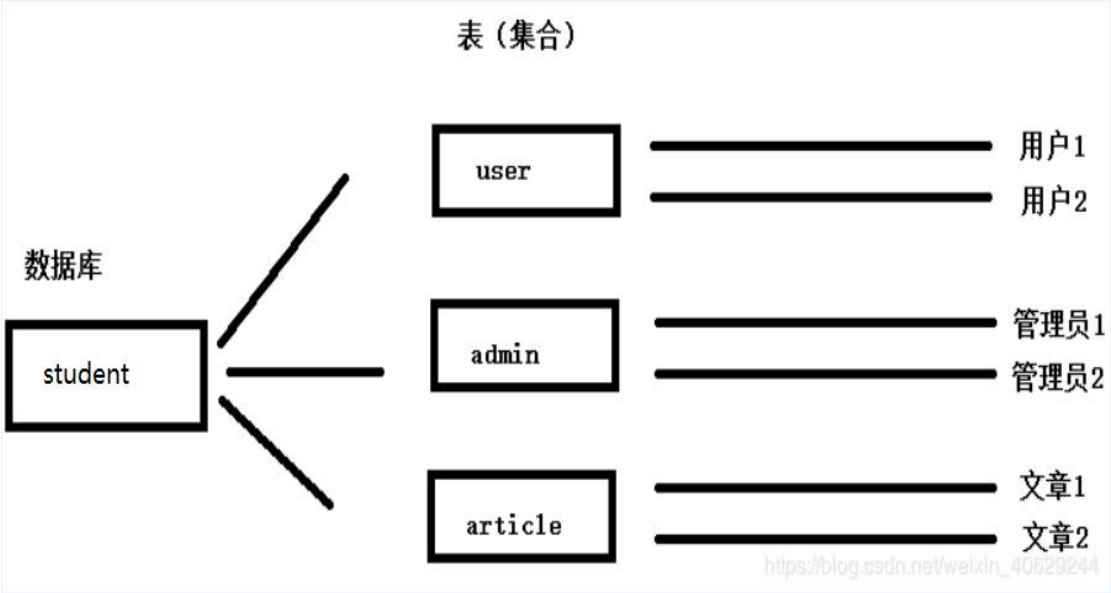
在新的 cmd 窗口中输入 mongo



(3). 查看所有数据库列表 show dbs

(4). 清屏　cls

2 . MongoDB 数据库的创建与删除



(1). 创建、使用数据库

use student

如果之前没有这个数据库，当前为创建一个新的数据库，那么必须插入一条数据，才能保证数据库创建成功。

数据库中不能直接插入数据，只能往集合(collections)中插入数据。

不需要专门创建集合，只需要声明集合并插入数据就会创建成功(以下示例均已 user 集合为例)。

db.user.insert({"name":"augus"});

系统发现 user 是一个陌生的集合名字，所以就自动创建了。

(2). 显示当前数据库中的数据集合（mysql 中叫表）

show collections

(3). 删除当前所在的数据库

db.dropDatabase()

(4). 删除指定的集合(当前为 user 集合)

db.user.drop()

3. MongoDB 数据库插入数据

db.user.insert({"name":"yuki"});

4. MongoDB 数据库查找数据(重点)

(1). 查询所有记录

db.user.find();

类似 SELECT* FROM user;

(2). 查询当前集合中的某列去重后的数据

db.user.distinct("name");

以上会过滤掉 name 中的相同数据

类似 SELECT DISTINCT name FROM user;

(3). 查询 age = 22 的记录

db.user.find({"age": 22});

类似　SELECT* FROM user;

(4). 查询 age > 22 的记录

db.user.find({age: {$gt: 22}});

类似 SELECT* FROM user WHERE age >22;

(5). 查询 age < 22 的记录

db.user.find({age: {$lt: 22}});

类似 SELECT* FROM user WHERE age <22;

(6). 查询 age >= 25 的记录

db.user.find({age: {$gte: 25}});

类似 SELECT* FROM user WHERE age >= 25;

(7). 查询 age <= 25 的记录

db.user.find({age: {$lte: 25}});

类似 SELECT* FROM user WHERE age <= 25;

(8). 查询 age >= 23 并且 age <= 26

db.user.find({age: {$gte: 23, $lte: 26}});

类似 SELECT* FROM user WHERE age >= 25 AND age<=26;

(9). 查询 name 中包含 mongo 的数据，多用于模糊查询搜索

db.user.find({name: /mongo/});

类似 SELECT* FROM user WHERE name LIKE '%mongo%';

(10). 查询 name 中以 mongo 开头的数据

db.user.find({name: /^mongo/});

类似 SELECT* FROM user WHERE name LIKE 'mongo%';

(11). 查询指定列 name、age 数据

db.user.find({}, {name: 1, age: 1});

类似 SELECT name, age FROM user;

name 也可以用 true 或 false，ture 的情况下与 name:1 效果一样，如果用 false 就 是排除 name，显示其它列的信息。

(12). 查询指定列 name、age 数据， 并且 age > 25

db.user.find({age: {$gt: 25}}, {name: 1, age: 1});

类似 SELECT name, age FROM user WHERE age >25;

(13). 按照列排序 1 升序 -1 降序

升序：db.user.find().sort({age: 1});

降序：db.user.find().sort({age: -1});

(14). 查询 name = zhangsan, age = 22 的数据

db.user.find({name: 'zhangsan', age: 22});

类似 SELECT * FROM user WHERE name = 'zhangsan' AND age = '22';

(15). 查询前 5 条数据

db.user.find().limit(5);

类似 SELECT TOP 5 * FROM user;

(16). 查询 10 条以后的数据

db.user.find().skip(10);

类似 SELECT * FROM user WHERE id NOT IN (SELECT TOP 10 *

FROM user);

(17). 查询在 5-10 之间的数据

db.user.find().limit(10).skip(5);

可用于分页，limit 是页大小，skip 是第几页乘以页大小的值

类似 SELECT * FROM user LIMIT 5,5

(18). OR 查询

db.user.find({$or: [{age: 22}, {age: 25}]});

类似 SELECT * FROM user WHERE age = 22 OR age = 25;

(19). findOne 查询第一条数据

db.user.findOne();

类似 SELECT TOP 1 * FROM user;

类似 db.user.find().limit(1);

(20). 查询某个结果集的记录条数，用于统计数量

db.user.find({age: {$gte: 25}}).count();

类似 SELECT COUNT(*) FROM user WHERE age >= 25;

如果要返回限制之后的记录数量，要使用 count(true)或者 count(非 0)

```
db.user.find().skip(10).limit(5).count(true);
```

5. MongoDB 数据库更新数据

(1). 查找名字为小明的，把年龄更改为 16 岁

```
db.user.update({"name":"小明"},{$set:{"age":16}});
```

(2). 查找英语成绩为 70 的人，把年龄更改为 33 岁

```
db.user.update({"score.english":70},{$set:{"age":33}});
```

(3). 查找所有性别为男的用户把年龄都改为 33 岁

```
db.user.update({"sex":"男"},{$set:{"age":33}},{multi: true});
```

(4). 查找 name 为小明的用户的信息进行完整替换,注意没有$set 了

```
db.user.update({"name":"小明"},{"name":"大明","age":16});
```

(5). 查找 name 为 Lisi 的用户,将其年龄增加 50 岁

```
db.user.update({name: 'Lisi'}, {$inc: {age: 50}}, false, true);
```

类似 UPDATE user SET age = age + 50 WHERE name = 'Lisi';

(6). 查找 name 为 Lisi 的用户,将其年龄增加 50 岁,姓名改为 Wangwu

```
db.user.update({name: 'Lisi'}, {$inc: {age: 50}, $set: {name: 'Wangwu'}},
false, true);
```

类似 update users set age = age + 50, name = 'Wangwu' where name =
'Lisi';

6. MongoDB 数据库删除数据

(1). 删除年龄为 32 的所有用户

```
db.user.remove({age: 32});
```

(2). 删除年龄为 32 的一个用户

```
db.user.remove( { "age": "32" }, { justOne: true } )
```

实训四 redis 操作

一、 实训目的和要求

使学生在 redis 的规划，让学生对自己即将 redis 进行规划并完成站点规划书的撰写。

二、 实训内容

让学生在因特网上申请免费的个人主页空间，向学生讲解站点规划的重要性以及站点规划书的撰写方法，并由学生自行思考并完成实训指导书的撰写。

三、 实训准备

eclipse。

四、 实训步骤

redis 的介绍：

redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash（哈希类型）。

这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子

性的。在此基础上，redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。
区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 master-slave(主从)同步

redis 的特点：

1. 使用 Redis 有哪些好处？

(1) 速度快，因为数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O(1)

(2) 支持丰富数据类型，支持 string，list，set，sorted set，hash

(3) 支持事务,操作都是原子性,所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

(4) 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

2. redis 相比 memcached 有哪些优势？

(1) memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类型

(2) redis 的速度比 memcached 快很多

(3) redis 可以持久化其数据

3. redis 常见性能问题和解决方案：

(1) Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件

(2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

4. MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据

相关知识：redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。redis 提供 6 种数据淘汰策略：

voltile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

5. Memcache 与 Redis 的区别都有哪些？

1)、存储方式

Memecache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。

Redis 有部份存在硬盘上，这样能保证数据的持久性。

2)、数据支持类型

Memcache 对数据类型支持相对简单。

Redis 有复杂的数据类型。

3），value 大小

redis 最大可以达到 1GB，而 memcache 只有 1MB

6. Redis 常见的性能问题都有哪些？如何解决？

1).Master 写内存快照，save 命令调度 rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以 Master 最好不要写内存快照。

2).Master AOF 持久化，如果不重写 AOF 文件，这个持久化方式对性能的影响是最小的，但是 AOF 文件会不断增大，AOF 文件过大会影响 Master 重启的恢复速度。Master 最好不要做任何持久化工作，包括内存快照和 AOF 日志文件，特别是不要启用内存快照做持久化,如果数据比较关键，某个 Slave 开启 AOF 备份数据，策略为每秒同步一次。

3).Master 调用 BGREWRITEAOF 重写 AOF 文件，AOF 在重写的时候会占大量的 CPU 和内存资源，导致服务 load 过高，出现短暂服务暂停现象。

4). Redis 主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave 和 Master 最好在同一个局域网内

7, redis 最适合的场景

Redis 最适合所有数据 in-momory 的场景，虽然 Redis 也提供持久化功能，但实际更多的是一个 disk-backed 的功能，跟传统意义上的持久化有比较大的差别，那么可能大家就会有疑问，似乎 Redis 更像一个加强版的 Memcached，那么何时使用 Memcached,何时使用 Redis 呢?

如果简单地比较 Redis 与 Memcached 的区别，大多数都会得到以下观点：、Redis 不仅仅支持简单的 k/v 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。

、Redis 支持数据的备份，即 master-slave 模式的数据备份。
、Redis 支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

（1）、会话缓存（Session Cache）

最常用的一种使用 Redis 的情景是会话缓存（session cache）。用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？

幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。

（2）、全页缓存（FPC）

除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。

再次以 Magento 为例，Magento 提供一个插件来使用 Redis 作为全页缓存后端。

此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

（3）、队列

Reids 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。

如果你快速的在 Google 中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

（4），排行榜/计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的 10 个用户－我们称之为“user_scores”，我们只需要像下面一样执行即可：

当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行：

```
ZRANGE user_scores 0 10 WITHSCORES
```

Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

（5）、发布/订阅

最后（但肯定不是最不重要的）是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统！（不，这是真的，你可以去核实）。

Redis 提供的所有特性中，我感觉这个是喜欢的人最少的一个，虽然它为用户提供如果此多功能。

支持的数据类型： 5 种数据类型

```
string  list   set  zset   hash

redis={
        k1:'123',       字符串
        k2:[1,2,3,4],    列表/数组
        k3:{1,2,3,4}      集合
        k4:{name:'michael',age:23}  字典/哈希表
        k5:{('tank',18),('xixi',33)}  有序集合
}
```

特点：
可以持久化
单线程，单进程

redis 的安装使用：

linux 下安装：

```
wget http://download.redis.io/releases/redis-3.0.6.tar.gz
tar xzf redis-3.0.6.tar.gz
cd redis-3.0.6
make

启动服务端：
src/redis-server


启动客户端：
src/redis-cli
redis> set foo bar
OK
redis> get foo
"bar"
```

window 下的安装：参考以下链接

'https://www.cnblogs.com/liuqingzheng/p/9831331.html
Python 操作 Redis 之安装和支持存储类型

安装 redis 模块

```
pip3 install redis
```

Python 操作 Redis 之普通连接

redis-py 提供两个类 Redis 和 StrictRedis 用于实现 Redis 的命令,StrictRedis 用于实现大部分官方的命令，并使用官方的语法和命令，Redis 是 StrictRedis 的子类，用于向后兼容旧版本的 redis-py.

```
import redis
```

```
r = redis.Redis(host='127.0.0.1', port=6379, password='12345')  # 只要设置了 redis 的密码，在连接时需要输入密码
r.set('foo', 'Bar')
print(r.get('foo'))
```

Python 操作 Redis 之连接池

redis-py 使用 connection pool 来管理对一个 redis server 的所有连接，避免每次建立、释放连接的开销。默认，每个 Redis 实例都会维护一个自己的连接池。可以直接建立一个连接池，然后作为参数 Redis，这样就可以实现多个 Redis 实例共享一个连接池。一般情况下可以考虑创建一个单例。

```
import redis

pool = redis.ConnectionPool(host='127.0.0.1', port=6379)
r = redis.Redis(connection_pool=pool)
r.set('foo', 'Bar')
print(r.get('foo'))
```

redis 的 string 操作：

String 操作，redis 中的 String 在在内存中按照一个 name 对应一个 value 来存储。如图：

set(name, value, ex=None, px=None, nx=False, xx=False)

在 Redis 中设置值，默认，不存在则创建，存在则修改
参数：
ex，过期时间（秒）
px，过期时间（毫秒）
nx，如果设置为 True，则只有 name 不存在时，当前 set 操作才执行,值存在，就修改不了，执行没效果
xx，如果设置为 True，则只有 name 存在时，当前 set 操作才执行，值存在才能修改，值不存在，不会设置新值

setnx(name, value)

设置值，只有 name 不存在时，执行设置操作（添加）,如果存在，不会修改

setex(name, time, value)

```
# 设置值
# 参数：
    time，过期时间（数字秒 或 timedelta 对象）
```

psetex(name, time_ms, value)

```
# 设置值
# 参数：
    # time_ms，过期时间（数字毫秒 或 timedelta 对象
```

mset(*args, **kwargs)

批量设置值
如：
mset(k1='v1', k2='v2')
或
mget({'k1': 'v1', 'k2': 'v2'})

get(name)

获取值

mget(keys, *args)

批量获取
如：
```
    mget('k1', 'k2')
    或
    r.mget(['k3', 'k4'])
```

getset(name, value)

设置新值并获取原来的值

getrange(key, start, end)

```
# 获取子序列（根据字节获取，非字符）
# 参数：
    # name，Redis 的 name
    # start，起始位置（字节）
    # end，结束位置（字节）
# 如： "军军" ，0-2 表示 "军"所对应的字节，如需查看字符串，则需要转码。

print(str(data,encode='utf-8')
```

setrange(name, offset, value)

```
# 修改字符串内容，从指定字符串索引开始向后替换（新值太长时，则向后添加）
# 参数：
    # offset，字符串的索引，字节（一个汉字三个字节）
    # value，要设置的值
```

setbit(name, offset, value)

```
# 对 name 对应值的二进制表示的位进行操作

# 参数：
    # name，redis 的 name
    # offset，位的索引（将值变换成二进制后再进行索引）
    # value，值只能是 1 或 0

# 注：如果在 Redis 中有一个对应： n1 = "foo",
```

那么字符串 foo 的二进制表示为：01100110 01101111 01101111
所以，如果执行 setbit('n1', 7, 1)，则就会将第 7 位设置为 1，
那么最终二进制则变成 01100111 01101111 01101111，即："goo"

getbit(name, offset)

```
# 获取 name 对应的值的二进制表示中的某位的值 （0 或 1）
```

bitcount(key, start=None, end=None)

```
# 获取 name 对应的值的二进制表示中 1 的个数
# 参数：
    # key， Redis 的 name
    # start，位起始位置
    # end，位结束位置
```

bitop(operation, dest, *keys)

```
# 获取多个值，并将值做位运算，将最后的结果保存至新的 name 对应的值

# 参数：
    # operation,AND（并） 、 OR（或） 、 NOT（非） 、 XOR（异或）
    # dest， 新的 Redis 的 name
    # *keys,要查找的 Redis 的 name

# 如：
    bitop("AND", 'new_name', 'n1', 'n2', 'n3')
    # 获取 Redis 中 n1,n2,n3 对应的值，然后讲所有的值做位运算（求并集），然后将结果保存 new_name 对应的值中
```

strlen(name)

```
# 返回 name 对应值的字节长度（一个汉字 3 个字节）
```

incr(self, name, amount=1)

```
# 自增 name 对应的值，当 name 不存在时，则创建 name＝amount，否则，则自增。

# 参数：
    # name,Redis 的 name
    # amount,自增数（必须是整数）

# 注：同 incrby
```

incrbyfloat(self, name, amount=1.0)

```
# 自增 name 对应的值，当 name 不存在时，则创建 name＝amount，否则，则自增。

# 参数：
    # name,Redis 的 name
    # amount,自增数（浮点型）
```

decr(self, name, amount=1)

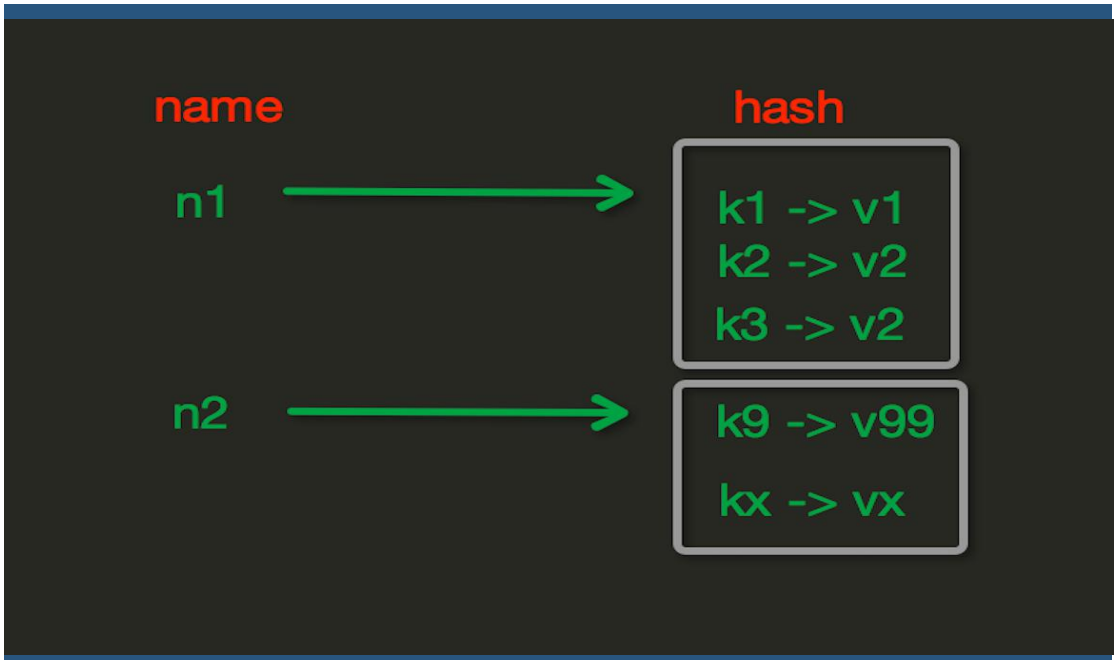
```
# 自减 name 对应的值，当 name 不存在时，则创建 name＝amount，否则，则自减。

# 参数：
    # name,Redis 的 name
    # amount,自减数（整数）
```

append(key, value)

```
# 在 redis name 对应的值后面追加内容

# 参数：
    key, redis 的 name
    value, 要追加的字符串
```

reids 的 hash 操作

Hash 操作，redis 中 Hash 在内存中的存储格式如下图：



hset(name, key, value)

```
# name 对应的 hash 中设置一个键值对（不存在，则创建；否则，修改）

# 参数：
    # name，redis 的 name
    # key，name 对应的 hash 中的 key
    # value，name 对应的 hash 中的 value

# 注：
    # hsetnx(name, key, value),当 name 对应的 hash 中不存在当前 key 时则创建（相当于添加）
```

hmset(name, mapping)

```
# 在 name 对应的 hash 中批量设置键值对

# 参数：
    # name，redis 的 name
    # mapping，字典，如：{'k1':'v1', 'k2': 'v2'}

# 如：
    # conn.hmset('xx', {'k1':'v1', 'k2': 'v2'})
```

hget(name,key)

```
# 在 name 对应的 hash 中获取根据 key 获取 value
```

hmget(name, keys, *args)

```
# 在 name 对应的 hash 中获取多个 key 的值

# 参数:
    # name，reids 对应的 name
    # keys，要获取 key 集合，如：['k1', 'k2', 'k3']
    # *args，要获取的 key，如：k1,k2,k3

# 如:
    # r.mget('xx', ['k1', 'k2'])
    # 或
    # print r.hmget('xx', 'k1', 'k2')
```

hgetall(name)

```
# 获取 name 对应 hash 的所有键值
print(re.hgetall('xxx').get(b'name'))
```

hlen(name)

```
# 获取 name 对应的 hash 中键值对的个数
```

hkeys(name)

```
# 获取 name 对应的 hash 中所有的 key 的值
```

hvals(name)

```
# 获取 name 对应的 hash 中所有的 value 的值
```

hexists(name, key)

```
# 检查 name 对应的 hash 是否存在当前传入的 key
```

hdel(name,*keys)

```
# 将 name 对应的 hash 中指定 key 的键值对删除
print(re.hdel('xxx','sex','name'))
```

hincrby(name, key, amount=1)

```
# 自增 name 对应的 hash 中的指定 key 的值，不存在则创建 key=amount
# 参数:
    # name，redis 中的 name
    # key， hash 对应的 key
    # amount，自增数（整数）
```

hincrbyfloat(name, key, amount=1.0)

```
# 自增 name 对应的 hash 中的指定 key 的值，不存在则创建 key=amount

# 参数:
    # name，redis 中的 name
    # key， hash 对应的 key
    # amount，自增数（浮点数）

# 自增 name 对应的 hash 中的指定 key 的值，不存在则创建 key=amount
```

hscan(name, cursor=0, match=None, count=None)

```
# 增量式迭代获取，对于数据大的数据非常有用，hscan 可以实现分片的获取数据，并非一次性将数据全部获取完，从而放置内存被撑爆

# 参数:
    # name，redis 的 name
    # cursor，游标（基于游标分批取获取数据）
    # match，匹配指定 key，默认 None 表示所有的 key
    # count，每次分片最少获取个数，默认 None 表示采用 Redis 的默认分片个数

# 如:
```

```
    # 第一次：cursor1, data1 = r.hscan('xx', cursor=0, match=None,
count=None)
    # 第二次：cursor2, data1 = r.hscan('xx', cursor=cursor1, match=None,
count=None)
    # ...
    # 直到返回值 cursor 的值为 0 时，表示数据已经通过分片获取完毕
```

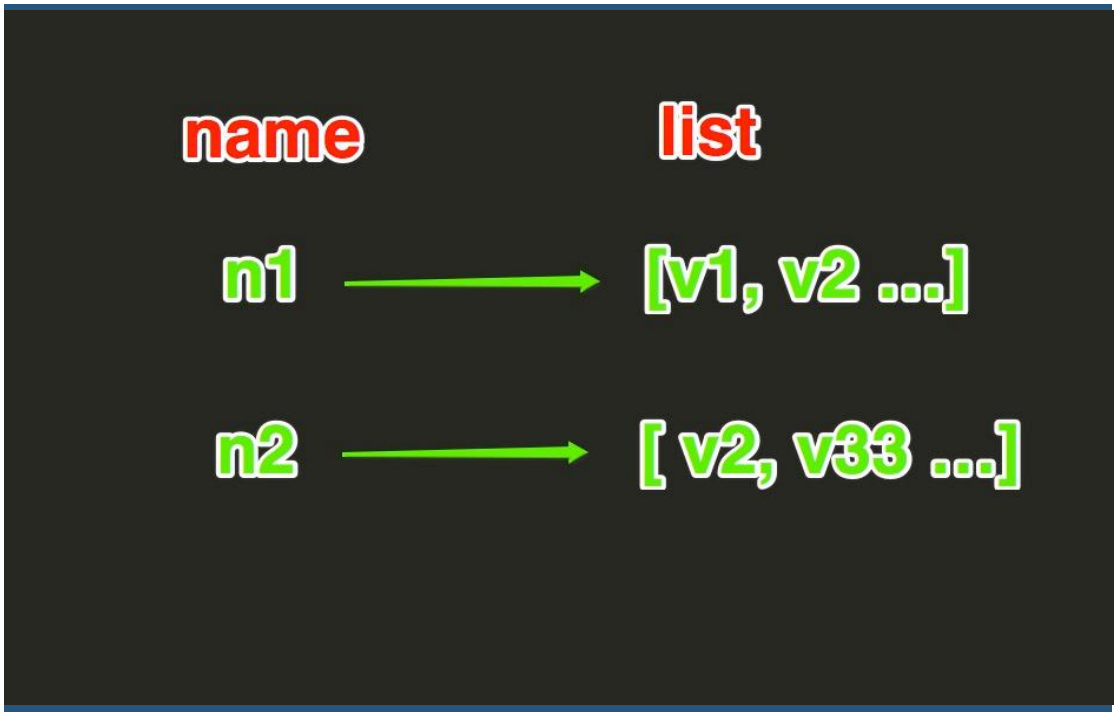
hscan_iter(name, match=None, count=None)

复制代码

```
# 利用 yield 封装 hscan 创建生成器，实现分批去 redis 中获取数据

# 参数：
    # match，匹配指定 key，默认 None 表示所有的 key
    # count，每次分片最少获取个数，默认 None 表示采用 Redis 的默认分片个
数

# 如：
    # for item in r.hscan_iter('xx'):
    #     print item
```

reids 的 list 操作

List 操作，redis 中的 List 在在内存中按照一个 name 对应一个 List 来存储。如图：



lpush(name,values)

```
# 在 name 对应的 list 中添加元素，每个新的元素都添加到列表的最左边

# 如：
    # r.lpush('oo', 11,22,33)
    # 保存顺序为: 33,22,11

# 扩展：
    # rpush(name, values) 表示从右向左操作
```

lpushx(name,value)

```
# 在 name 对应的 list 中添加元素，只有 name 已经存在时，值添加到列表的最左边

# 更多：
    # rpushx(name, value) 表示从右向左操作
```

llen(name)

```
name 对应的 list 元素的个数
```

linsert(name, where, refvalue, value))

```
# 在 name 对应的列表的某一个值前或后插入一个新值

# 参数：
    # name，redis 的 name
    # where，BEFORE 或 AFTER(小写也可以)
    # refvalue，标杆值，即：在它前后插入数据（如果存在多个标杆值，以找到的第一个为准）
    # value，要插入的数据
```

r.lset(name, index, value)

```
# 对 name 对应的 list 中的某一个索引位置重新赋值

# 参数：
    # name，redis 的 name
    # index，list 的索引位置
    # value，要设置的值
```

r.lrem(name, value, num)

```
# 在 name 对应的 list 中删除指定的值

# 参数：
    # name，redis 的 name
    # value，要删除的值  当存在这个删除的值时，与 num 搭配使用
    # num，   num=0，删除列表中所有的指定值；
            # num=2,从前到后，删除 2 个；  存在 value 这个值，从前外后，依次删除与之相同的两个 value 即可
            # num=-2,从后向前，删除 2 个；  从后往前，依次删除两个 value 即可
```

lpop(name)

```
# 在 name 对应的列表的左侧获取第一个元素并在列表中移除，返回值则是第一个元素

# 更多：
```

```
    # rpop(name) 表示从右向左操作
```

lindex(name, index)

在 name 对应的列表中根据索引获取列表元素

lrange(name, start, end)

```
# 在 name 对应的列表分片获取数据
# 参数：
    # name，redis 的 name
    # start，索引的起始位置
    # end，索引结束位置  print(re.lrange('aa',0,re.llen('aa')-1))    索引取值骨头有股味
```

ltrim(name, start, end)

```
# 在 name 对应的列表中移除没有在 start-end 索引之间的值
# 参数：
    # name，redis 的 name
    # start，索引的起始位置
    # end，索引结束位置（大于列表长度，则代表不移除任何）
```

rpoplpush(src, dst)

```
# 从一个列表取出最右边的元素，同时将其添加至另一个列表的最左边
# 参数：
    # src，要取数据的列表的 name
    # dst，要添加数据的列表的 name
```

blpop(keys, timeout)

```
# 将多个列表排列，按照从左到右去 pop 对应列表的元素

# 参数：
    # keys，redis 的 name 的集合
    # timeout，超时时间，当元素所有列表的元素获取完之后，阻塞等待列表内有数据的时间（秒），0 表示永远阻塞

# 更多：
```

```
    # r.brpop(keys, timeout)，从右向左获取数据
```

爬虫实现简单分布式：多个 url 放到列表里，往里不停放 URL，程序循环取值，但是只能一台机器运行取值，可以把 url 放到 redis 中，多台机器从 redis 中取值，爬取数据，实现简单分布式

eg:

```
  import redis
    pool = redis.ConnectionPool(host=xx,port=xx,password=xx)
    r = redis,Redis(connection_pool=pool)

r.lpush('test',11,22,33)
r.lpush('test1',44,55,66)
r.lpush('test2',77,88,99)

while True:
    a = r.blpop(['tets','test1','test2'],3)
    print(a)   # test 33    test 22   test 11   test1  66 ..... test2 77   none .....   因此需要加上一个 if 判断!
```

结果：会将 redis 里 test,test1,test2 里缓存的数据一个个的取出来，最终取完，redis 里对应的 name 字段就没有 values 了！

blpop 补充：

BLPOP 是阻塞式列表的弹出原语。 它是命令 LPOP 的阻塞版本，这是因为当给定列表内没有任何元素可供弹出的时候， 连接将被 BLPOP 命令阻塞。 当给定多个 key 参数时，按参数 key 的先后顺序依次检查各个列表，弹出第一个非空列表的头元素。

非阻塞行为:

当 BLPOP 被调用时，如果给定 key 内至少有一个非空列表，那么弹出遇到的第一个非空列表的头元素，并和被弹出元素所属的列表的名字 key 一起，组成结果返回给调用者。

当存在多个给定 key 时， BLPOP 按给定 key 参数排列的先后顺序，依次检查各个列表。 我们假设 key list1 不存在，而 list2 和 list3 都是非空列表。考虑以下的命令：

```
  BLPOP list1 list2 list3 0
```

BLPOP 保证返回一个存在于 list2 里的元素（因为它是从 list1 -> list2 -> list3 这个顺序查起的第一个非空列表）。

阻塞行为:

如果所有给定 key 都不存在或包含空列表，那么 BLPOP 命令将阻塞连接，直到有另一个客户端对给定的这些 key 的任意一个执行 LPUSH 或 RPUSH 命令为止。

一旦有新的数据出现在其中一个列表里，那么这个命令会解除阻塞状态，并且返回 key 和弹出的元素值。

当 BLPOP 命令引起客户端阻塞并且设置了一个非零的超时参数 timeout 的时候， 若经过了指定的 timeout 仍没有出现一个针对某一特定 key 的 push 操作，则客户端会解除阻塞状态并且返回一个 nil 的多组合值(multi-bulk value)。

timeout 参数表示的是一个指定阻塞的最大秒数的整型值。当 timeout 为 0 是表示阻塞时间无限制。

什么 key 会先被处理？是什么客户端？什么元素？优先顺序细节

当客户端为多个 key 尝试阻塞的时候，若至少存在一个 key 拥有元素，那么返回的键值对(key/element pair)就是从左到右数第一个拥有一个或多个元素的 key。 在这种情况下客户端不会被阻塞。比如对于这个例子 BLPOP key1 key2 key3 key4 0，假设 key2 和 key4 都非空， 那么就会返回 key2 里的一个元素。

当多个客户端为同一个 key 阻塞的时候，第一个被处理的客户端是等待最长时间的那个（即第一个因为该 key 而阻塞的客户端）。 一旦一个客户端解除阻塞那么它就不会保持任何优先级，当它因为下一个 BLPOP 命令而再次被阻塞的时候，会在处理完那些 被同个 key 阻塞的客户端后才处理它（即从第一个被阻塞的处理到最后一个被阻塞的）。

当一个客户端同时被多个 key 阻塞时，若多个 key 的元素同时可用（可能是因为事务或者某个 Lua 脚本向多个 list 添加元素）， 那么客户端会解除阻塞，并使用第一个接收到 push 操作的 key（假设它拥有足够的元素为我们的客户端服务，因为有可能存在其他客户端同样是被这个 key 阻塞着）。 从根本上来说，在执行完每个命令之后，Redis 会把一个所有 key 都获得数据并且至少使一个客户端阻塞了的 list 运行一次。这个 list 按照新数据的接收时间进行整理，即是从第一个接收数据的 key 到最后一个。在处理每个 key 的时候，只要这个 key 里有元素， Redis 就会对所有等待这个 key 的客户端按照“先进先出”(FIFO)的顺序进行服务。若这个 key 是空的，或者没有客户端在等待这个 key， 那么将会去处理下一个从之前的命令或事务或脚本中获得新数据的 key，如此等等。

brpoplpush(src, dst, timeout=0)

```
# 从一个列表的右侧移除一个元素并将其添加到另一个列表的左侧

# 参数：
    # src，取出并要移除元素的列表对应的 name
    # dst，要插入元素的列表对应的 name
```

```
    # timeout，当 src 对应的列表中没有数据时，阻塞等待其有数据的超时时间（秒），0 表示永远阻塞
```

自定义增量迭代

```
# 由于 redis 类库中没有提供对列表元素的增量迭代，如果想要循环 name 对应的列表的所有元素，那么就需要：
    # 1、获取 name 对应的所有列表
    # 2、循环列表
# 但是，如果列表非常大，那么就有可能在第一步时就将程序的内容撑爆，所有有必要自定义一个增量迭代的功能：
import redis
conn=redis.Redis(host='127.0.0.1',port=6379)
#
conn.lpush('test',*[1,2,3,4,45,5,6,7,7,8,43,5,6,768,89,9,65,4,23,54,6757,8,68])
# conn.flushall()      删除当前库下的所有缓存，所以要慎用！！！
def scan_list(name,count=2):
    index=0
    while True:
        data_list=conn.lrange(name,index,count+index-1)
        if not data_list:
            return
        index+=count
        for item in data_list:
            yield item
print(conn.lrange('test',0,100))
for item in scan_list('test',5):
    print('---')
    print(item)
```

redis 的 set 操作

Set 操作，Set 集合就是不允许重复的列表

sadd(name,values)

```
# name 对应的集合中添加元素
```

scard(name)

获取 name 对应的集合中元素个数

sdiff(keys, *args)

在第一个 name 对应的集合中且不在其他 name 对应的集合的元素集合

略。。。

其他操作

delete(*names)

根据删除 redis 中的任意数据类型

exists(name)

检测 redis 的 name 是否存在

keys(pattern='*')

```
# 根据模型获取 redis 的 name

# 更多：
    # KEYS * 匹配数据库中所有 key 。
    # KEYS h?llo 匹配 hello ， hallo 和 hxllo 等。
    # KEYS h*llo 匹配 hllo 和 heeeeello 等。
    # KEYS h[ae]llo 匹配 hello 和 hallo ，但不匹配 hillo
```

expire(name ,time)

为某个 redis 的某个 name 设置超时时间

rename(src, dst)

对 redis 的 name 重命名为

move(name, db))

将 redis 的某个值移动到指定的 db 下

randomkey()

随机获取一个 redis 的 name（不删除）

type(name)

获取 name 对应值的类型

scan(cursor=0, match=None, count=None)
scan_iter(match=None, count=None)

```
# 同字符串操作，用于增量迭代获取 key
```

管道

redis-py 默认在执行每次请求都会创建（连接池申请连接）和断开（归还连接池）一次连接操作，如果想要在一次请求中指定多个命令，则可以使用 pipline 实现一次请求指定多个命令，并且默认情况下一次 pipline 是原子性操作。

```
import redis

pool = redis.ConnectionPool(host='127.0.0.1', port=6379)

r = redis.Redis(connection_pool=pool)

# pipe = r.pipeline(transaction=False)
pipe = r.pipeline(transaction=True)
pipe.multi()
pipe.set('name', 'alex')
pipe.set('role', 'sb')

pipe.execute()
```

Django 中使用 redis

方式一：

utils 文件夹下，建立 redis_pool.py

```
import redis
POOL = redis.ConnectionPool(host='127.0.0.1',
port=6379,password='12345',max_connections=1000)
```

视图函数中使用：

```
import redis
from django.shortcuts import render,HttpResponse
from utils.redis_pool import POOL
```

```
def index(request):
    conn = redis.Redis(connection_pool=POOL)
    conn.hset('kkk','age',18)

    return HttpResponse('设置成功')
def order(request):
    conn = redis.Redis(connection_pool=POOL)
    conn.hget('kkk','age')

    return HttpResponse('获取成功')
```

方式二：

安装 django-redis 模块

pip3 install django-redis

setting 里配置：

```
# redis 配置
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "CONNECTION_POOL_KWARGS": {"max_connections": 100}
            # "PASSWORD": 12345, #    个人所设置的密码
        }
    }
}
```

视图函数：

```
from django_redis import get_redis_connection
from django.shortcuts import HttpResponse
def test(request):
    conn = get_redis_connection()
    data = conn.get('name')  # 从 redis 数据库里获取字段名为 name 的值
    return HttpResponse(data)
```

上述使用是在设置了 settings 配置后完成的！

补充：
关于使用：from django.core.cache import cache
时报错如下：
 django.core.exceptions.ImproperlyConfigured: Requested setting CACHES, but settings are not configured. You must either define the environment variable DJANGO_SETTINGS_MODULE or call settings.configure() before accessing settings.

虽然在 python 的命令行下 输入：
python manage.py shell 环境下能够使用，但是在真实环境下无法使用。

解决方式：
因是 pycharm 要你配置一个 环境变量 DJANGO_SETTINGS_MODULE 这个变量告诉 django 项目去找哪一个 settings 文件。 具体的步骤：
在 django 项目里存在一个 run 按钮，点击进去后，找到 python 下的 test 选项，（注意不是 django 那一个），，然后修改里面的 Environment variables 添加一项。名称是 DJANGO_SETTINGS_MODULE 值是 你的 settings,比如 mysite.settings 。

保存好配置后就可以了。

不够在 test.py 里使用时，发现下一次 set 得到数据会掩盖上一次的数据，且字符串数据会乱码，暂时无解！

整理的缓存数据的前后端传输小例子：

前后端分离获取缓存
```
from rest_framework.views import APIView
import time
from django.core.cache import cache
from django.http import JsonResponse

ctime = time.time()
cache.set('test_data', {'name': 'michael1', 'time': ctime}, 10)
cache.set('test_data2',{'name':'michael2','time':ctime},3000)
```

```
# 取数据
class Test(APIView):
    def get(self,request):
        return JsonResponse([cache.get('test_data'),cache.
                             get('test_data2'),1234,5678,'妹子
'],safe=False,json_dumps_params={'ensure_ascii':False})
```

jsonresponse 可以传输汉子，列表，字典，不过要加上相关的配置。

一、 实训方法

机房操作。

二、 考核办法

此部分实训内容采用全体考察的方法，以百分制为满分，具体评分标准如下：

1. 站点创建的规范性和熟练程度（25 分）
2. 草稿图的绘制（20 分）
3. 草稿图的载入（5 分）

布局视图的相关操作（50 分）

三、 思考和练习

布局视图和表格视图在版面布局中各有优势和劣势？对怎样的版面布局应该用布局视图，又对怎样的版面布局应该用表格视图？