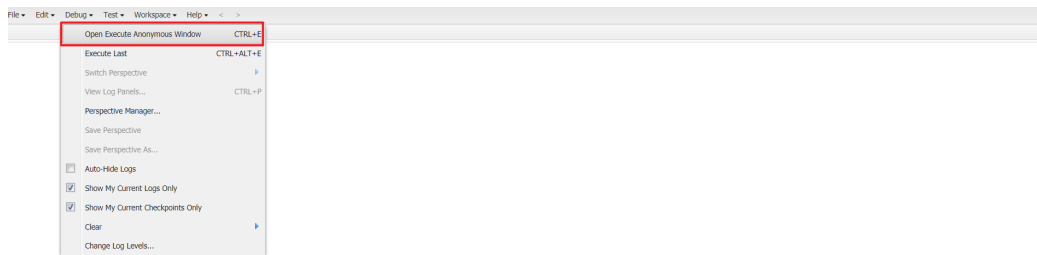


Salesforce Apex 零基础入门

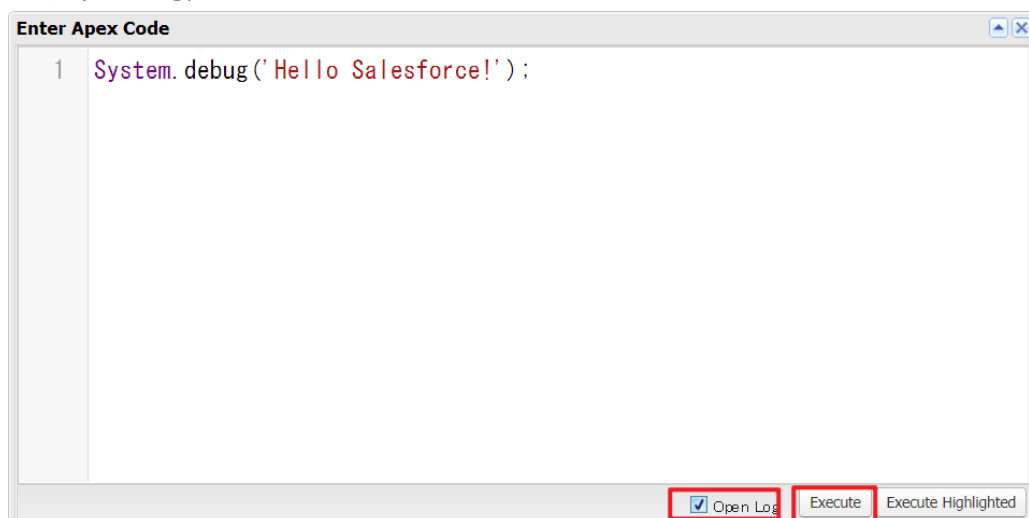
第1章 Hello Salesforce!

```
1 // 第1条语句
2 System.debug('Hello Salesforce');
```

- System开头字母大小写都可以，建议开头字母大写
- 要输出的字符用单引号括起来
- 在行尾要添加一个分号，表示程序的结束



选中Open Log，然后Execute



Execution Log		
Timestamp	Event	Details
07:24:41:001	USER_INFO	[EXTERNAL]0055800003b7B1software.01@gmail.com((GMT+09:00) 日本標準時 (Asia/Tokyo))(GMT+09:00)
07:24:41:001	EXECUTION_ST...	
07:24:41:001	CODE_UNIT_ST...	[EXTERNAL]execute_anonymous_apex
07:24:41:002	HEAP_ALLOCATE	[79]Bytes:3
07:24:41:002	HEAP_ALLOCATE	[84]Bytes:152
07:24:41:002	HEAP_ALLOCATE	[399]Bytes:408
07:24:41:002	HEAP_ALLOCATE	[412]Bytes:408
07:24:41:002	HEAP_ALLOCATE	[520]Bytes:48
07:24:41:002	HEAP_ALLOCATE	[139]Bytes:5
07:24:41:002	HEAP_ALLOCATE	[EXTERNAL]Bytes:1
07:24:41:002	STATEMENT_EX...	[1]
07:24:41:002	STATEMENT_EX...	[1]
07:24:41:002	HEAP_ALLOCATE	[1]Bytes:17
07:24:41:002	HEAP_ALLOCATE	[52]Bytes:5
07:24:41:002	HEAP_ALLOCATE	[58]Bytes:5
07:24:41:002	HEAP_ALLOCATE	[66]Bytes:7
07:24:41:002	USER_DEBUG	[1]DEBUGHello Salesforce!
07:24:41:002	CUMULATIVE_L...	
07:24:41:002	LIMIT_USAGE...	(default)
07:24:41:000	LIMIT_USAGE...	Number of SQL queries: 0 out of 100
07:24:41:000	LIMIT_USAGE...	Number of query rows: 0 out of 50000

第2章 变量及其命名规则

2.1 什么是变量

现实世界中，装水要有水桶，盛饭要用碗，在计算机的世界中，保存东西用的叫变量。

Integer studentNumber = 55;

变量名

2.2 变量的命名规则

变量的名字可以随便起，比如说学生的学号姓名，

- studentNo ~~abc1123~~ = ~~55~~;
- studentName studentName = 'JZY';

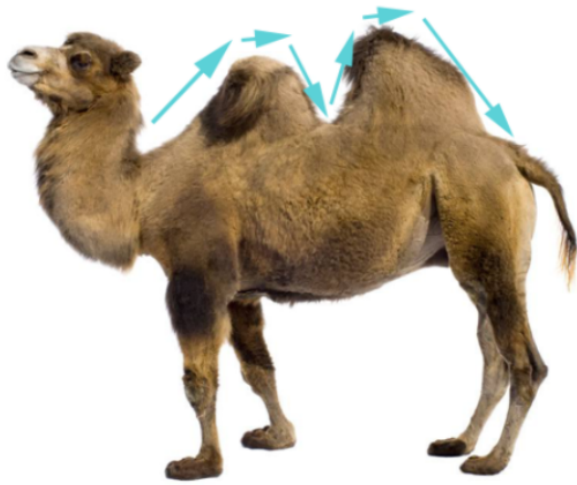
studentNo = 38;

① **小写字母开头**，开头不能用下划线（_），*，?，<

② **驼峰命名法**

什么是驼峰命名法？

变量名的第一个单词的首写字母小写，后面的单词的第一个字母大写。



第3章 变量类型 (Integer)

studentNumber就是一个变量名，不同类型的变量存储在计算机里的大小也是不一样的，一个数字的大小肯定和一大段文章，或者一段视频是不一样的，所以就有变量类型的概念

比如说我们要学的第一个变量类型Integer，就是一个整数类型。

Integer studentNumber = 55;

变量类型

当我写下Integer的时候，我就是在告诉计算机给我准备一个能存放整数类型的大小的空间，当然是放在内存里的。

比如说年龄，学生的数量，就是个数字，学生的姓名就是个字符串 (String)

```
1 Integer studentAge = 130;  
2 System.debug('studentAge'); // 推荐  
3 system.debug(studentAge);
```

- 加号起到拼接的作用
- 在程序中使用// 是注释的意思，为了加一些提示，里面的代码不执行
- +号表示连接

```
1 Integer studentAge = 130;
2 // 加号起到拼接的作用
3 // 学生的年龄是：39!
4 System.debug('学生的年龄是：' + studentAge);
```

第4章 关键字和保留字

```
1 // 有些单词被占用，不让我们使用
2 Integer system = 39;
3 Integer Integer = 39;
4
5 // 下面的是可以的
6 Integer integerAge = 39;
```

第5章 再说Integer

Integer文档

说起整数，一定要有大小，不能无限大，而且是有正负的

Integer最小值-2, 147, 483, 648、最大值2, 147, 483, 647

```
1 // Error 当是负的2147483648的时候，无法执行
2 // Integer i1 = -2147483648;
3 Integer i1 = -2147483647;
4 Integer i2 = 2147483647;
5 System.debug(i1);
6 System.debug(i2);
```

大家知道这两个数字的由来吗？

$-2, 147, 483, 648 = -2^{31}$;

$2, 147, 483, 647 = 2^{31} - 1$;

第6章 变量的声明和初始化

```
1 // 变量的声明
2 // 变量的初始化,就是给一个值
3 Integer i2 = 130;
4
5 // 变量的声明
6 Integer i1;
7
8 // 变量的初始化,就是给一个值
9 i1 = 130;
10 System.debug(i1);
```

```

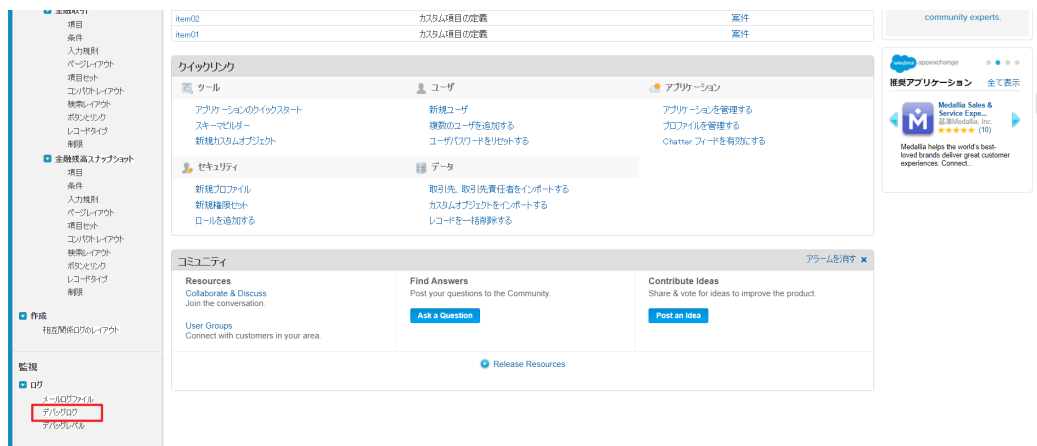
1 // 変量の声明
2 Integer i1;
3
4 // 在只声明，没有赋值的情况下，就是null
5 System.debug(i1);

```

第7章 System.debug() 实际应用

大多是在调试程序的时候使用，因为apex代码不像Java一样有很好的调试功能，因为Salesforce的代码在云上，所有有的使用为了定位问题，需要使用System.debug()

调试完程序可以删掉，也可以保留，只要不出log，对效率是不会有影响，但是推荐删除点，但是在测试的类里面无所谓。



デバッグログ

デバッグログには、データベース操作、システムプロセス、トランザクション実行時または単体テストの実行中に発生するエラーを記録します。ユーザーが追跡フラグを設定している場合、その開始日から終了日までの間に開始するトランザクションをユーザーが実行するたびに、システムがデバッグログを作成します。下に指定したユーザーのデバッグログを監視して取得できます。1つのSFDC DevConsole デバッグレベルが組織内のすべての DEVELOPER_LOG 追跡フラグで共有されます。

ビュー: すべて 新規ビューの作成

ユーザー追跡フラグ						
アクション	名前 ↑	LogType	要求者	開始日	有効期限	デバッグレベル名
削除 編集 フィルタ	金 宗宇①	DEVELOPER_LOG	宗宇① 金	2021/07/13 12:07	2021/07/13 13:30	SFDC_DevConsole
Edit - レコード 1 - 金 宗宇①						

前のページ | 次のページ

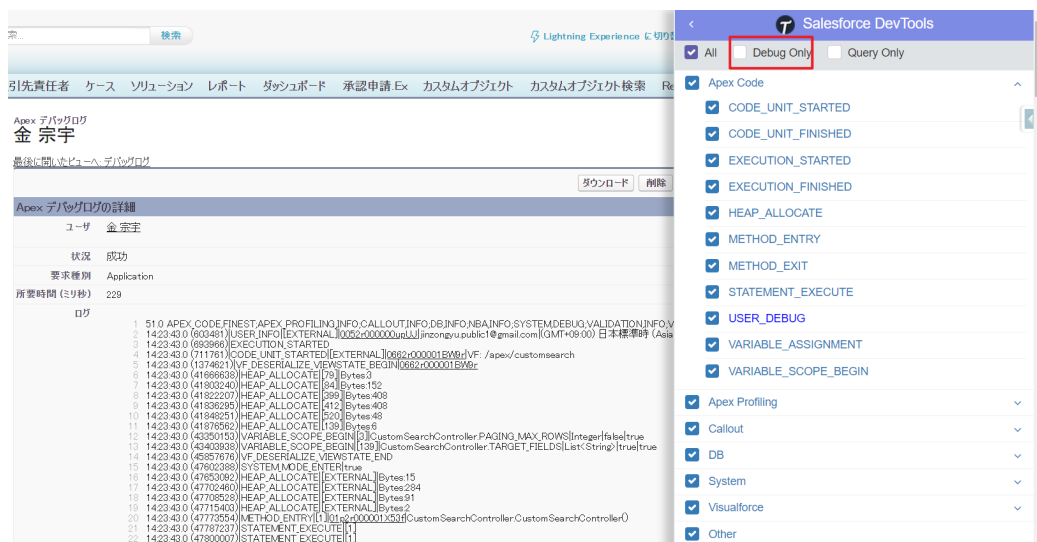
デバッグログ							
すべてを削除							
ユーザー	要求種別	アプリケーション	演算子	状況	所要時間 (ミリ秒)	ログサイズ (バイト)	開始時刻
すべてを削除							

デバッグレベルを設定します。

[キャンセル](#) [保存](#)

追跡対象エンティティ種別	ユーザー
追跡対象エンティティ名	金 宗宇①
開始日	2021/07/13 14:22 [2021/07/13 14:22]
有効期限	2021/07/14 14:22 [2021/07/13 14:22]
デバッグレベル	SFDC_DevConsole 新しいデバッグレベル

[キャンセル](#) [保存](#)



最新公開のビューヘ、デバグログ

ダウンロード 削除

Apex デバグログの詳細	
ユーザ	金 宗宇
状況	成功
要求種別	Application
所要時間 (ミリ秒)	229
ログ	100 1423.43.0 (67718518) (USER_DEBUG[20]) (DEBUG) (total count: 22)

ダウンロード 削除

第8章 基本数据类型

8.1 基本数据类型（11种）

① Integer（整数）

32 位整数。Integer 最小値-2, 147, 483, 648、最大値2, 147, 483, 647。

Integer 类

```
1 Integer age = 130;  
2
```

② Long（长整数）

64位整数。Longs 最小值是负的2的63次幂、最大值为2的63次幂-1。

比 Integer 更广泛的时候使用。

```
1 Long l = 2147483648L;
```

③ Double 没有 float

64位小数。Doubles 最小值是负2的63次幂、最大值为正2的63次幂-1。

```
1 Double d=3.14159;
```

④ Decimal

小数点を含む数値。Decimal は、任意の精度数です。**通貨項目には自動的に Decimal 型が割り当てられます。**

<input type="radio"/> URL	Web サイトのアドレスを入力できます。ユーザがこの項目をクリックすると、その URL が、必要アップロードのファイルに示されます。
<input type="radio"/> チェックボックス	True (チェック) または False (チェックなし) の値を入力できます。
<input type="radio"/> テキスト	文字列に数値のどちらも入力できます。
<input type="radio"/> テキスト (暗号化) ¹	数字や文字を任意の組み合わせで入力し、暗号化して保存できます。
<input type="radio"/> テキストエリア	複数行にわたって、255 文字まで入力できます。
<input type="radio"/> パーセント	「10」などのパーセントを表す数値を入力できます。また、パーセント記号が自動的に数値に追加されます。
<input type="radio"/> メール	メールアドレスを入力できます。入力されたアドレスは、入力形式が正しいかどうかを確認されます。クリックすると、自動的にメールアドレスが起動され、メールを作成して送信できます。
<input type="radio"/> テキストエリア (リッチ)	ユーザに、書式設定済みテキストの入力、画像とリンクの追加を許可します。複数行に分けて最大 131,072 文字です。
<input type="radio"/> リッチテキストエリア	複数行にわたって、131,072 文字まで入力できます。
<input type="radio"/> 時間	ユーザがローカル時刻を入力できます。たとえば、「2:40 PM」、「14:40」、「14:40:00」、および「14:40:50.600」はすべてこの項目で有効な時刻です。
<input type="radio"/> 数値	数値を入力できます。先頭の 0 は削除されます。
<input type="radio"/> 選択リスト	あらかじめ設定されたリストから値を選択する項目です。
<input type="radio"/> 選択リスト (複数選択)	ユーザは定義されたリストから複数の値を選択可能です。
<input type="radio"/> 地理位置情報	場所を定義できます。緯度および経度コンポーネントを含め、距離の計算に使用できます。
<input checked="" type="radio"/> 通貨	ドルまたはその他の通貨で金額を入力でき、自動的に通貨形式の金額になります。この形式は、エクスポート後の Excel や他のスプレッド形式のデータでも有効です。
<input type="radio"/> 電話	電話番号を入力できます。自動的に電話番号形式にします。
<input type="radio"/> 日付	日付を直接入力することも、ポップアップのカレンダーから選択することもできます。
<input type="radio"/> 日付/時間	日付/時刻を直接入力することも、ポップアップのカレンダーから選択することもできます。ポップアップから選択した場合は、選択した日付とそのときの時間が日付/時刻項目に入力されます。

⑤String

使用单引号, 比如说一段文字

String类型

```

1 String s = 'This is salesforce variable';
2
3 String s = 'Hello Maximillian';
4 String s2 = s.abbreviate(8);
5 System.debug('Hello...' + s2);
6 System.debug(s2.length());
7
8 String s = 'This is salesforce variable';
9 System.debug('s');
10 System.debug(s);

```

自己练习的时候:

- **assertEquals 换成debug**
- **第一个参数删掉**
- **参照下面的例子**

```

1 String s = 'Hello Maximillian';
2 String s2 = s.abbreviate(10);
3 System.assertEquals('Hello...', s2);
4 System.assertEquals(8, s2.length());
5
6 String s = 'Hello Maximillian';
7 String s2 = s.abbreviate(8);
8 System.debug(s2);
9 System.debug(s2.length());

```

⑥Boolean

只有True和False两种值，比如说性别

在チェックボックス的时候用

```

1 Boolean b = Boolean.valueOf('true');

```

```

2 System.Debug(b);
3
4 Boolean b = true;
5 System.Debug(b);

```

<input type="radio"/> URL	Web サイトのアドレスを入力できます。ユーザがこの項目をクリックすると、その URL が、別のブラウザのウィンドウに表示されます。
<input checked="" type="radio"/> チェックボックス	True (チェック) また False (チェックなし) の値を入力できます。
<input type="radio"/> テキスト	文字列と数値のどちらも入力できます。
<input type="radio"/> テキスト(暗号化) ⁱ	数字や文字を任意の組み合わせで入力し、暗号化して保存できます。
<input type="radio"/> テキストエリア	複数行にわたって、255 文字まで入力できます。
<input type="radio"/> パーセント	「10」などのパーセントを表す数値を入力できます。また、パーセント記号が自動的に数値に追加されます。
<input type="radio"/> メール	メールアドレスを入力できます。入力されたアドレスは、入力形式が正しいかどうかを確認されます。クリックすると、自動的にメールソフトが起動され、メールを作成して送信できます。
<input type="radio"/> テキストエリア (リッチ)	ユーザに、書式設定済みテキストの入力、画像とリンクの追加を許可します。複数行に分けて最大 131,072 文字です。
<input type="radio"/> HTMLテキストエリア	複数行にわたって、131,072 文字まで入力できます。
<input type="radio"/> 時間	ユーザがローカル時刻を入力できます。たとえば、「2:40 PMJ」、「14:40」、「14:40:00」、および「14:40:50.600」はすべてこの項目で有効な時刻です。
<input type="radio"/> 数値	数値を入力できます。先頭の 0 は削除されます。
<input type="radio"/> 選択リスト	あらかじめ設定されたリストから値を選択する項目です。
<input type="radio"/> 選択リスト (複数選択)	ユーザは定義されたリストから複数の値を選択可能です。
<input type="radio"/> 地理位置情報	場所を定義できます。緯度および経度エンポイントを含め、距離の計算に使用できます。
<input type="radio"/> 通貨	ドルまたはその他の通貨で金額を入力でき、自動的に通貨形式の金額になります。この形式は、エクスポート後の Excel や他のスプレッド形式のデータでも有効です。
<input type="radio"/> 電話	電話番号を入力できます。自動的に電話番号形式になります。

⑦Blob

二进制数据，在处理文档的时候使用

Blob类型

⑧Date

Date Class

```

1 取得当前日期：
2 Date dt = Date.today();
3 System.debug(dt);
4
5 测试的时候经常用：
6 Date myDate = date.NewInstance(1960, 2, 17);
7 System.debug(myDate);
8
9 判断闰年：
10 System.debug(Date.isLeapYear(2004) );
11
12 boolean bln1 = Date.isLeapYear(2004);
13 boolean bln2 = Date.isLeapYear(2021);
14
15 System.debug(bln1);
16 System.debug(bln2);

```

⑨Time

Time Class

```

1 Time myTime = Time.newInstance(18, 30, 2, 20);
2 System.debug(myTime);

```


⑩ Datetime

CMT时间：格林威治时间

```
1 DateTime dt = DateTime.now();
2 System.debug(dt);
3
4 ※因为标准的对象使用的是格林威治时间，自定义的对象也要使用格林威治时间
5 XXXDatetime__c = Datetime.now().addHours(9);
```

⑪ ID

ID Class

Salesforce中每一条资源都是有ID的

资源包括：实际的数据，Metadata (profile, tab等等)

Classic 15位

Lightning 18位

```
1 Account a = new Account(name = 'account');
2 insert a;
3 Id myId = a.id;
4 system.debug(myId.getSObjectType());
```

8.2 集合类型

8.2.1 List

List Class

1. 从第0条记录开始
2. 存储某种类型的记录的集合

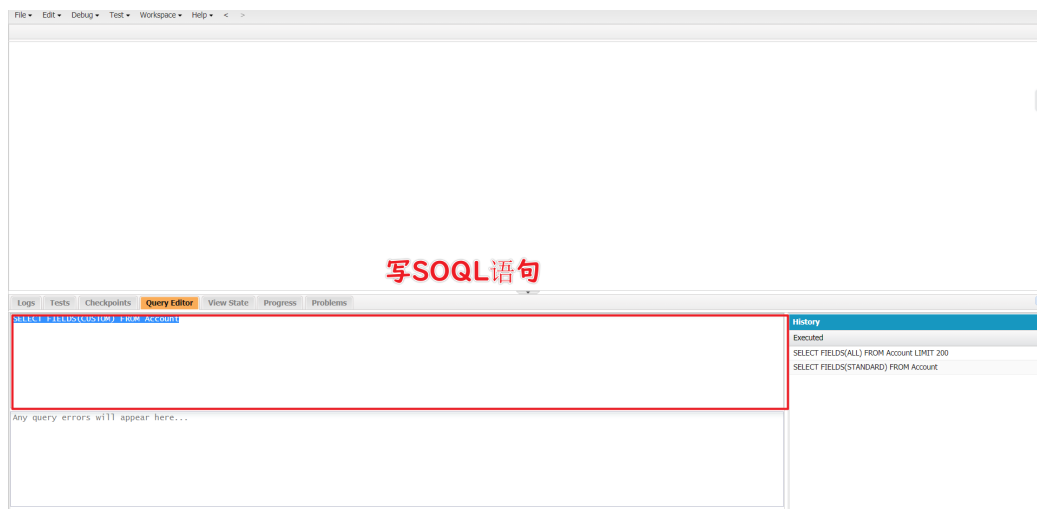
List<String>

String类型的集合

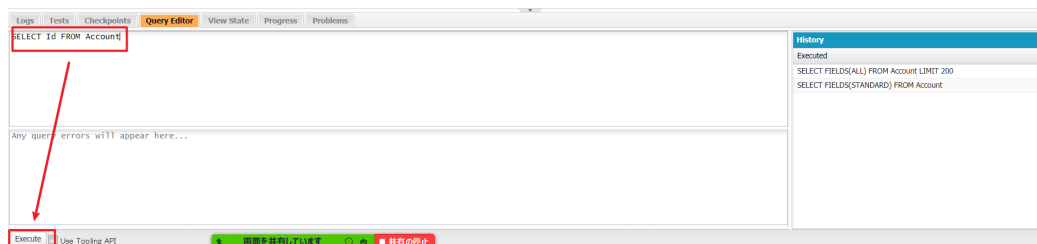
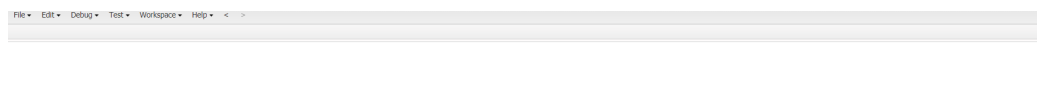
而且只能是同一种类型

Integer

String



SOQL::: Salesforce Object Query Language (SOQL)



大小写无所谓，一般第一个字母，或者关键字整个大写

SELECT: 选择 固定的

ID: 你要选需求的值 可以添加对象里面存在的项目

From 固定的，从后面的对象里面来选择

Account: 你的环境里面标准的对象，或者自定义的对象

- 1 SELECT Id FROM Account
- 2 SELECT Id, Name FROM Account order by id
- 3
- 4 如果没写order by，应该默认通过id排序

SOSL, DML

- 1 // list里放的是果篮
- 2 List<Account> accList = [SELECT Id, Name, MasterRecordId FROM Account];
- 3
- 4 // accList[0] : account
- 5 // MasterRecordId
- 6 System.debug(accList[0].Id);
- 7 System.debug(accList[0].Name);

· 创建 · 添加 · 取得 · 修改 · 删除

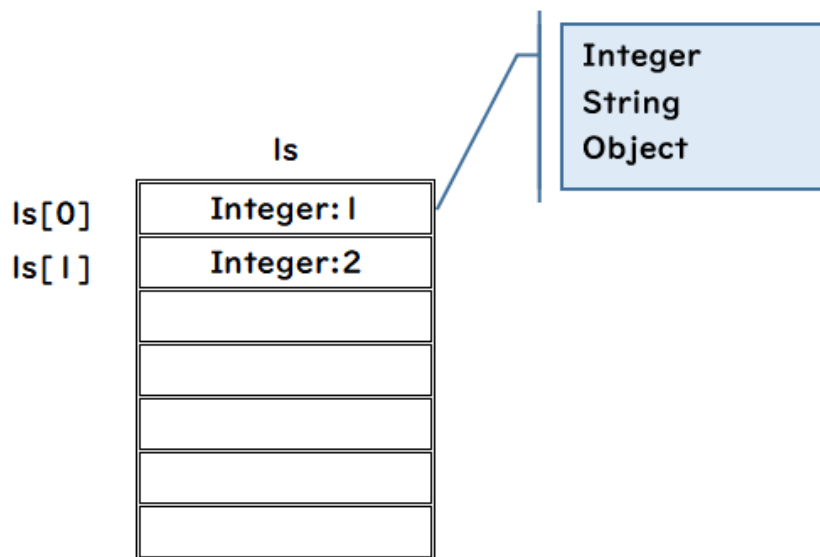
 直接创建

[//] 两个反斜杠相当于注释

```

1 List<Integer> ls = new List<Integer>();
2 //
3 //List<String> ls = new List<String>(); // 前后必须保持一致
4 //List<Integer> ls = new List<String>(); // 错误
5
6 ls.add(1);
7 ls.add(2);
8 ls.add(3);
9 // 计算机里面第一个集合元素是0开头
10 System.debug(ls[0]); //取得第一个元素  ls.add(1);
11 System.debug(ls[1]); //取得第二个元素  ls.add(2);
12 System.debug(ls[2]); //取得第三个元素  ls.add(3);
13
14 // ls[1] = ls.get(1) 相同
15 System.debug('ls.get(1):' + ls.get(1)); //取得
16

```



2021/08/08 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

Salesforce删除权限

ユーザがレコードを削除するには、3つのシナリオがあります：

- レコードを削除しようとしているユーザがレコードの所有者です。
- ユーザが所有者のロール階層において所有者よりも上位に位置します。
- レコードを削除しようとするユーザが「すべて変更」権限を持っています。

- 所有者可以删除，也就是Owner

- Role阶层上位的人可以删除下位
- Object的「すべて変更」权限有的情况

※主从关系(A是主,AA是子)，共有设定选主从联动，通过sharing rule共享了权限，共享先删不掉共享元的主对象的数据，但是可以删掉从对象的数据

关于上节课的SOQL语句的补充

- Limit 关键字后面接数字，表示显示的最大件数
- Order By后面接字段名字 默认是升序

```
1 ASC: ascending    升序 默认写不写都可以
2 DESC: descending  降序 需要写上
3
4 SELECT id,name,Date01__c FROM FormulaConfirm__c Order By Date01__c,Name
5 可以添加多个排序项目
```

- Group By 分组 （还有个Having过滤，以后再说）

1. 单个字段分组
2. 多个字段分组
3. 常用聚合函 `count()` , `sum()` , `avg()` , `max()` , `min()`

```
Select YoteiDate__c,sum(Money__c),sum(Quantity__c) from SoqlTestObject__c Group
By YoteiDate__c
```

```
1 Select YoteiDate__c,sum(Money__c),sum(Quantity__c)
2 from SoqlTestObject__c Group By YoteiDate__c
3
4 Select calendar_month(YoteiDate__c),sum(Money__c),sum(Quantity__c)
5 from SoqlTestObject__c Group By calendar_month(YoteiDate__c)
6
7 calendar_month 08
8 CALENDAR_YEAR  2021
9
10 SELECT CreatedDate, Amount
11 FROM Opportunity
12 WHERE CALENDAR_YEAR(CreatedDate) = 2021
13
14 CALENDAR_YEAR之类的函数尽量用在where条件里面
15 要想在选择项目中表示的情况下，后面也要使用
```

https://developer.salesforce.com/docs/atlas.ja-jp.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select_date_functions.htm

Group By 后面的可以在显示的字段中表示，其他的一律不行
如果要显示其他字段，必须是加入集合函数 `sum`

为了排除干扰，我们来创建一个对象 SoqlTestObject

予定日	YoteiDate	日付
予定金額	Money	通貨 (18、0)
数量	Quantity	数値 (18、0)
Dummy	Dummy	テキスト (80)

<input type="checkbox"/> SoqlTestObject名 ↑	▼	予定金額	▼	予定日	▼	数量	▼
1	<input type="checkbox"/> A	¥ 100		2021/08/08		10	
2	<input type="checkbox"/> A	¥ 2,000		2021/08/08		20	
3	<input type="checkbox"/> B	¥ 2,001		2021/08/09		500	
4	<input type="checkbox"/> B	¥ 6,009		2021/08/10		123,213	

```
1 SELECT Id FROM Account
2 SELECT Id FROM Account Limit 1
3
4 // 标准Object
5 SELECT FIELDS(ALL) FROM Account LIMIT 200
6 SELECT FIELDS(CUSTOM) FROM Account LIMIT 200
7 SELECT FIELDS(STANDARD) FROM Account
8
9 // 自定义Object
10 SELECT FIELDS(ALL) FROM FormulaConfirm__c LIMIT 200
11 SELECT FIELDS(CUSTOM) FROM FormulaConfirm__c LIMIT 200
12 SELECT FIELDS(STANDARD) FROM FormulaConfirm__c
13
14 当使用FIELDS关键字的时候， FIELDS(ALL)和FIELDS(CUSTOM) 的时候 最大是200
15 不适用FIELDS关键字的时候，不能超过50000，一次查询实际数据超过50000条，
16 就会触发ガバナー制限（后面的课程会有实例代码）
```

```
1 SELECT FIELDS(STANDARD) FROM FormulaConfirm__c
```

NO	項目の表示ラベル	API 参照名	データ型	桁数	項目タイプ	必須	選択リスト値
1	カスタムオブジェクト ID	Id	Id		標準	必須	
2	所有者 ID	OwnerId	参照関係 (小组/用户)		標準	必須	
3	削除	IsDeleted	チェックボックス		標準		
4	数式確認用オブジェクト名称	Name	テキスト	80	標準		
5	レコードタイプ ID	RecordTypeId	参照関係 (记录类型)		標準		
6	作成日	CreatedDate	日付/時間		標準	必須	
7	作成者 ID	CreatedBy	参照関係 (用户)		標準	必須	
8	最終更新日	LastModifiedDate	日付/時間		標準	必須	
9	最終更新者 ID	LastModifiedBy	参照関係 (用户)		標準	必須	
10	System Modstamp	SystemModstamp	日付/時間		標準	必須	
11	最終閲覧日	LastViewedDate	日付/時間		標準		
12	最終参照日	LastReferencedDate	日付/時間		標準		
13	Formula01	Formula01__c	数式 (テキスト)	1300	カスタム		
14	Product.type	Product.type__c	ロングテキストエリア	255	カスタム		
15	Formula02	Formula02__c	数式 (テキスト)	1300	カスタム		
16	Days.Open	Days.Open__c	数値 (18、0)	18	カスタム		
17	Date01	Date01__c	日付		カスタム		
18	選択リスト	Field1__c	選択リスト		カスタム		AAA,BBB,CCC,DDD
19	連動1	Field2__c	選択リスト		カスタム		AAA,BBB,CCC,ABC
20	連動2	Field3__c	選択リスト		カスタム		AAA1,AAA2,AAA3,BBB1,BBB2,Q
21	list1	list1__c	選択リスト		カスタム		AAA,BBB
22	list2	list2__c	選択リスト		カスタム		AAA1,AAA2,BBB1,BBB2

```
1 SELECT FIELDS(CUSTOM) FROM FormulaConfirm__c LIMIT 200
```

NO	項目の表示ラベル	API 参照名	データ型	桁数	項目タイプ	必須	選択リスト値
1	カスタムオブジェクト ID	Id	id		標準	必須	
2	所有者 ID	OwnerId	参照関係 (小组/用户)		標準	必須	
3	削除	IsDeleted	チェックボックス		標準		
4	数式確認用オブジェクト名称	Name	テキスト	80	標準		
5	レコードタイプ ID	RecordTypeId	参照関係 (记录类型)		標準		
6	作成日	CreatedDate	日付/時間		標準	必須	
7	作成者 ID	CreatedBy	参照関係 (用户)		標準	必須	
8	最終更新日	LastModifiedDate	日付/時間		標準	必須	
9	最終更新者 ID	LastModifiedBy	参照関係 (用户)		標準	必須	
10	System Modstamp	SystemModstamp	日付/時間		標準	必須	
11	最終閲覧日	LastViewedDate	日付/時間		標準		
12	最終参照日	LastReferencedDate	日付/時間		標準		
13	Formula01	Formula01_c	数式 (テキスト)	1300	カスタム		
14	Product type	Product_type_c	ロングテキストエリア	255	カスタム		
15	Formula02	Formula02_c	数式 (テキスト)	1300	カスタム		
16	Days Open	Days_Open_c	数値 (18, 0)	18	カスタム		
17	Date01	Date01_c	日付		カスタム		
18	選択リスト	Field1_c	選択リスト		カスタム		AAA,BBB,CCC,DDD
19	連動1	Field2_c	選択リスト		カスタム		AAA,BBB,CCC,ABC
20	連動2	Field3_c	選択リスト		カスタム		AAA1,AAA2,AAA3,BBB1,BBB2,CCC
21	list1	list1_c	選択リスト		カスタム		AAA,BBB
22	list2	list2_c	選択リスト		カスタム		AAA1,AAA2,BBB1,BBB2

Where 语句

1. 字符串条件

```
Select id,name from SQOL__c where name = 'A'
```

```
Select id,name,Money__C from SQOL__c where Name = 'AA' and Money__C = 1
```

2. 各种符号

```
select id,name,Money__c from SQOL__c where Money__c > 1000 and Money__c < 3000
```

※没有between and语句

3. in、not in、or、and语句 常用

```
select id,name,Money__c from SQOL__c where Money__c in (1000,3000)
```

```
select id,name,Money__c from SQOL__c where Money__c = 1000 or Money__c = 3000
```

4. like语句

- “%” 为通配符，代表0 - n个任意字符
- “-” **下划线**代表一个任意字符（有的编程语言是问号）

```
select id,name,Money__c from SQOL__c where name like 'A__'
```

select id, name, Money__c from SQOL__c where Name like '%A%'

Id	Name	Money__c
a0GSH0000003YmGEAU	HHHAA999H	2000
a0GSH00000033uAEAA	A	1000
a0GSH0000003YhVEAU	AAA	1111
a0GSH0000003YhYEAU	BAA	1
a0GSH00000033uAEAA	A123	2005

Query Editor: Save Rows | Insert Row | Delete Row | Refresh Grid

Access in Salesforce: Create New | Open Detail Page | Edit Page

Logs | Tests | Checkpoints | **Query Editor** | View State | Progress | Problems

```
select id,name,Money__c from SQOL__c where Name like 'A%'
```

Any query errors will appear here...

History

Executed

Select id,name,Money__c from SQOL__c

Select id,name,Money__c from SQOL__c where Money__c > 1000 and ...

Select id,name,Money__c from SQOL__c where Money__c >= 1000 and ...

Select id,name,Money__c from SQOL__c

Select id,name,Money__c from SQOL__c where Money__c in (1000,1111,...

Select id,name,Money__c from SQOL__c where Money__c = 1000 or Mo...

select id,name,Money__c from SQOL__c

select id,name,Money__c from SQOL__c where Name like 'A%'

select id,name,Money__c from SQOL__c where Name like 'A%'

select id,name,Money__c from SQOL__c where Name like '%A%'

5. = null · != null

```
select id,name,Money__c,AObject__c from SQL__c where AObject__c = null
```

- 不是is null
- 一个等号 (= null)

6. 转义字符\

```
select id,name from sql__c where name = 'A\'123'
```

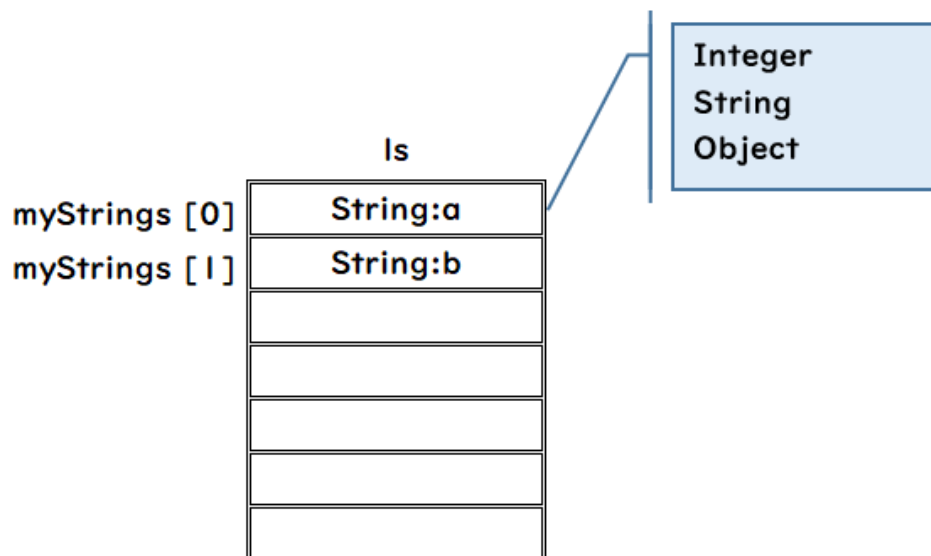
添加上转义字符 (\) 之后，在SOQL语句中'A\'123' 就相当于A'123

在线Salesforce编辑器

<https://my-aside.herokuapp.com/>

☒ ~~直接创建并完成初始化~~

```
1 List<String> myStrings = new List<String>{'a', 'b'};
2 System.debug(myStrings);
```



```
55     System.debug(acc);
56 }
57
58 Integer count = 11;
59 do {
60     System.debug(count);
61     count++;
62 } while (count < 11);
63
64 Integer count = 11;
65 while (count < 11) {
66     System.debug(count);
67     count++;
68 }
69
70 List<String> strList = new List<String>{'aaa','bbb'};
71 System.debug(strList);
```

☒ Open Log Execute Execute Highlighted

☒ 由其他List创建

```
1 List<Integer> ls1 = new List<Integer>();
2 ls1.add(1);
3 ls1.add(2);
4 // Create a list based on an existing one
5 List<Integer> ls2 = new List<Integer>(ls1);
6 // ls2 elements are copied from ls1
7 System.debug(ls2);
8
9 List<String> strList = new List<String>{'aaa','bbb'};
10 List<String> ls2 = new List<String>(strList);
11 System.debug(ls2);
12
```

☐ 通过Set创建

```
1 Set<Integer> s1 = new Set<Integer>();
2 s1.add(1);
3 s1.add(2);
4 // Create a list based on a set
5 List<Integer> ls = new List<Integer>(s1);
6 // ls elements are copied from s1
7 System.debug(ls);
```

☐ 通过数组创建

```
1 List<String> colors = new String[3];
2 colors[0] = 'Red';
3 colors[1] = 'Blue';
4 colors[2] = 'Green';
5 system.debug(colors);
```


☐ 添加List中的元素 (add)

☐ 修改List中的元素

```
1 List<Integer> ls = new List<Integer>();
2 ls.add(1);
3 ls.add(2);
4 System.debug('ls:' + ls);
5 ls.set(0, 3);
6 System.debug('ls:' + ls);
```

☐ 删除List中的元素

```
1 List<String> colors = new String[3];
2 colors[0] = 'Red';
3 colors[1] = 'Blue';
4 colors[2] = 'Green';
5 system.debug(colors);
6 String s1 = colors.remove(1);
7 system.debug(colors);
```

• 其他方法

```
1 clear();
2 clone();
3 contains(listElement);
4 deepClone()
5 isEmpty()
6 size()
7 sort()
```

8.2.2 Set

Set Class

和List类似，唯一区别是里面的数据不能重复，自动去重复元素（List中元素可重复）

Set无序

Set中没有发现修改方法

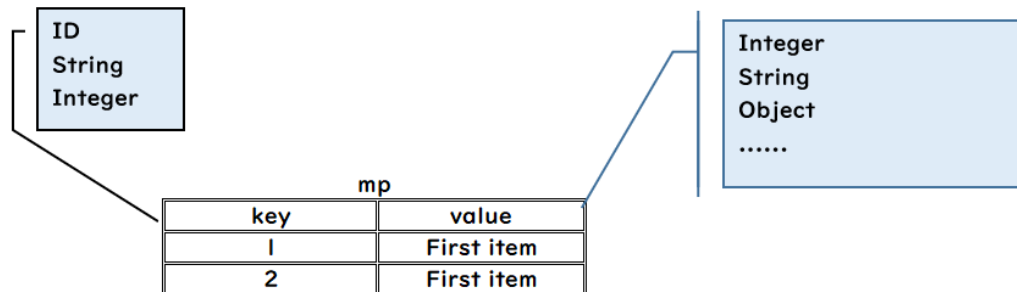
<https://keneloper.com/explain-how-to-use-apex-list-map-set/>

```
1 List<Integer> ls = new List<Integer>();
2 ls.add(1);
3 ls.add(1);
4 System.debug('ls:' + ls);
5
6
7 Set<Integer> st = new Set<Integer>();
8 st.add(1);
9 st.add(1);
10 System.debug('st:' + st);
```

11
12 实际项目用的多，因为可以去重复
13

8.2.3 Map

Map Class



• 三种创建方法

☐ 直接创建

```
1 Map<Integer, String> mp = new Map<Integer, String>();
2 mp.put(1, 'First item');
3 mp.put(2, 'Second item');
4 System.debug(mp);
5
6 {1=First item, 2=Second item}
7
8 Map<Integer, String> mp = new Map<Integer, String>();
9 mp.put(1, 'First item');
10 mp.put(1, 'Second item'); //覆盖掉key相同的元素
11 System.debug(mp);
12 {1=Second item}
```

☐ 通过其他map创建

```
1 Map<Integer, String> m1 = new Map<Integer, String>();
2 m1.put(1, 'First item');
3 m1.put(2, 'Second item');
4
5 Map<Integer, String> m2 = new Map<Integer, String>(m1);
6 System.debug(m2);
7
8 {1=First item, 2=Second item}
```

☐ 通过List创建

```
1 List<Account> ls = [select Id,Name from Account limit 1];
2 Map<Id, Account> mp = new Map<Id, Account>(ls);
3 System.debug(mp);
4
5 {0012r0000078jkJAAQ=Account:{Id=0012r0000078jkJAAQ, Name=Edge Communications}
```

❑ 取得Key方法 (keySet)

```
1 Map<String, String> colorCodes = new Map<String, String>();
2
3 colorCodes.put('Red', 'FF0000');
4 colorCodes.put('Blue', '0000A0');
5
6 Set <String> colorSet = new Set<String>();
7 colorSet = colorCodes.keySet();
8 System.debug(colorSet);
9
10 : {Blue, Red} //Map中是无序的
```

❑ 取得值方法 (get)

```
1 Map<String, String> colorCodes = new Map<String, String>();
2
3 colorCodes.put('Red', 'FF0000');
4 colorCodes.put('Blue', '0000A0');
5
6 System.debug(colorCodes.get('Blue'));
7
8 0000A0
```

❑ 添加方法 (put, putAll)

- **put**
- **putAll(fromMap)**

```
1 Map<String, String> map1 = new Map<String, String>();
2 map1.put('Red', 'FF0000');
3 Map<String, String> map2 = new Map<String, String>();
4 map2.put('Blue', '0000FF');
5 // Add map1 entries to map2
6 map2.putAll(map1);
7 System.assertEquals(2, map2.size());
```

- **putAll(subjectArray)**

```
1 List<Account> accts = new List<Account>();
2 accts.add(new Account(Name='Account1'));
3 accts.add(new Account(Name='Account2'));
4 // Insert accounts so their IDs are populated.
5 // 如果把下面这句话屏蔽掉，可以吗？
6 insert accts;
7 Map<Id, Account> m = new Map<Id, Account>();
8 // Add all the records to the map.
9 m.putAll(accts);
10 System.assertEquals(2, m.size());
```

```

1 Map<Integer, String> m1 = new Map<Integer, String>();
2 m1.put(1, 'First item');
3 m1.put(2, 'Second item');
4
5 Map<Integer, String> m2 = new Map<Integer, String>(m1);
6
7 m1.put(1, 'third');
8
9 System.debug(m1);
10 System.debug(m2);
11 //////////////////////////////////////////////////
12 Map<Integer, String> m1 = new Map<Integer, String>();
13 m1.put(1, 'First item');
14 m1.put(2, 'Second item');
15
16 Map<Integer, String> m2 = m1;
17
18 m1.put(1, 'third');
19
20 System.debug(m1);
21 System.debug(m2);
22 //////////////////////////////////////////////////
23
24 List<Integer> ls1 = new List<Integer>();
25 ls1.add(1);
26 ls1.add(2);
27 // Create a list based on an existing one
28 List<Integer> ls2 = new List<Integer>(ls1);
29
30 ls1.set(0, 3);
31 // ls2 elements are copied from ls1
32 System.debug(ls1);
33 System.debug(ls2);
34
35
36 List<Integer> ls1 = new List<Integer>();
37 ls1.add(1);
38 ls1.add(2);
39 // Create a list based on an existing one
40 List<Integer> ls2 = ls1;
41
42 ls1.set(0, 3);
43 // ls2 elements are copied from ls1
44 System.debug(ls1);
45 System.debug(ls2);

```

☐ 删除方法 (remove)

```

1 Map<String, String> colorCodes = new Map<String, String>();
2
3 colorCodes.put('Red', 'FF0000');

```

```

4 colorCodes.put('Blue', '0000A0');
5
6 String myColor = colorCodes.remove('Blue');
7 String code2 = colorCodes.get('Blue');
8
9 System.assertEquals(null, code2);

```

```

Map<Integer, String> m1 = new Map<Integer, String>();
m1.put(1, 'First item');
m1.put(2, 'Second item');

Map<Integer, String> m2 = m1;

m1.put(1, 'third');

System.debug(m1);
System.debug(m2);

List<Integer> ls1 = new List<Integer>();
ls1.add(1);
ls1.add(2);
// Create a list based on an existing one
List<Integer> ls2 = new List<Integer>(ls1);

ls1.set(0, 3);
// ls2 elements are copied from ls1
System.debug(ls1);
System.debug(ls2);

```

8.2.4 clone() 与 deepClone()

- 对于基本类型(<>中是基本类型), 无法使用deepClone(), 只能使用clone
- 对于SObject 类型, deepClone可以复制出一个完全没有联系的副本

※如果要使用深度克隆, 有需要的话, 要创建一个自定义类

基本类型:

```

1 List<String> ls1 = new List<String>();
2 ls1.add('a');
3 ls1.add('b');
4 System.debug(ls1);
5
6 List<String> ls2 = ls1.clone();
7 System.debug(ls2);
8
9 ls1.set(0, 'c'); // 修改ls1中的第一个元素
10
11 System.debug('modify after:' + ls1);

```

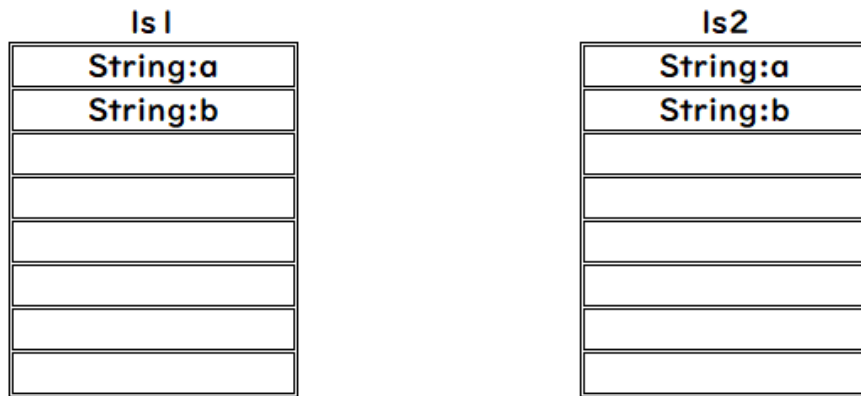
```
12 System.debug('modify after:' + ls2);
```

```
1 List<String> ls2 = ls1.deepclone();
```

```
2
```

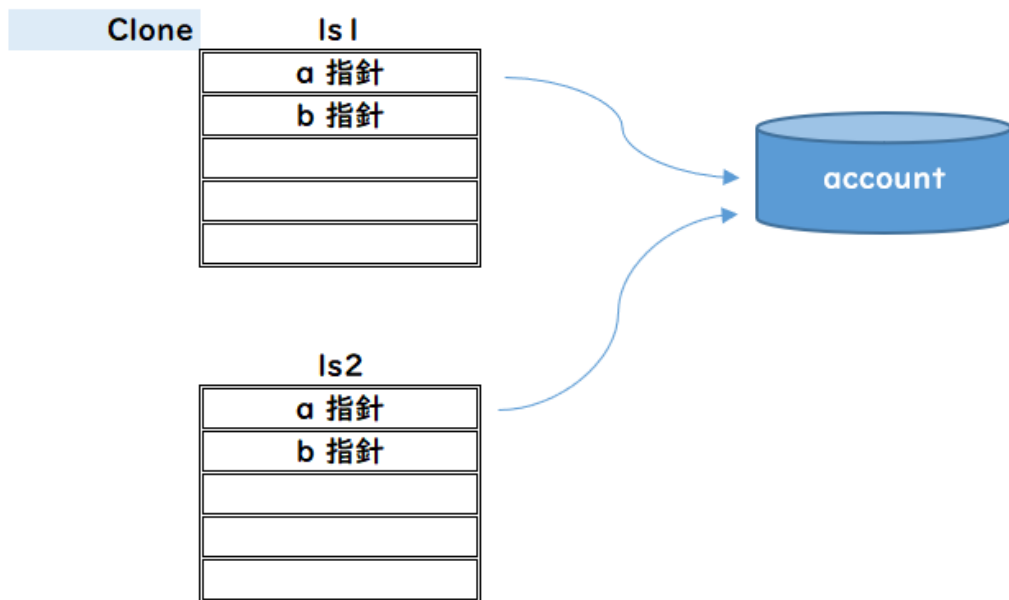
```
3 当使用deepClone的时候出现如下错误：
```

```
4 Operation only applies to SObject list types: List<String>
```



引用类型：

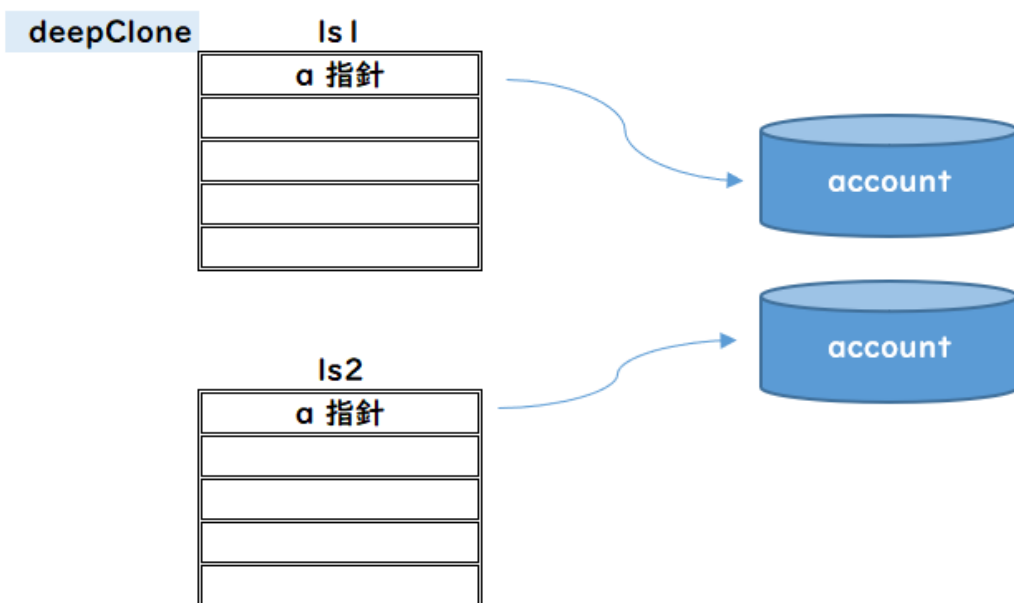
```
1 List<Account> ls1 = new List<Account>();
2
3 Account a = new Account(Name='Acme', BillingCity='New York');
4 ls1.add(a);
5
6 List<Account> ls2 = ls1.clone();
7 System.debug(ls1);
8 System.debug(ls2);
9
10 Account a1 = ls1.get(0);
11 a1.BillingCity = 'San Francisco';
12
13 System.debug('modify after ls1 :' + ls1);
14 System.debug('modify after ls2 :' + ls2);
15
```



```

1 List<Account> ls1 = new List<Account>();
2
3 Account a = new Account(Name='Acme', BillingCity='New York');
4 ls1.add(a);
5
6 List<Account> ls2 = ls1.deepClone();
7 System.debug(ls1);
8 System.debug(ls2);
9
10 Account a1 = ls1.get(0);
11 a1.BillingCity = 'San Francisco';
12
13 System.debug('modify after ls1 :' + ls1);
14 System.debug('modify after ls2 :' + ls2);

```

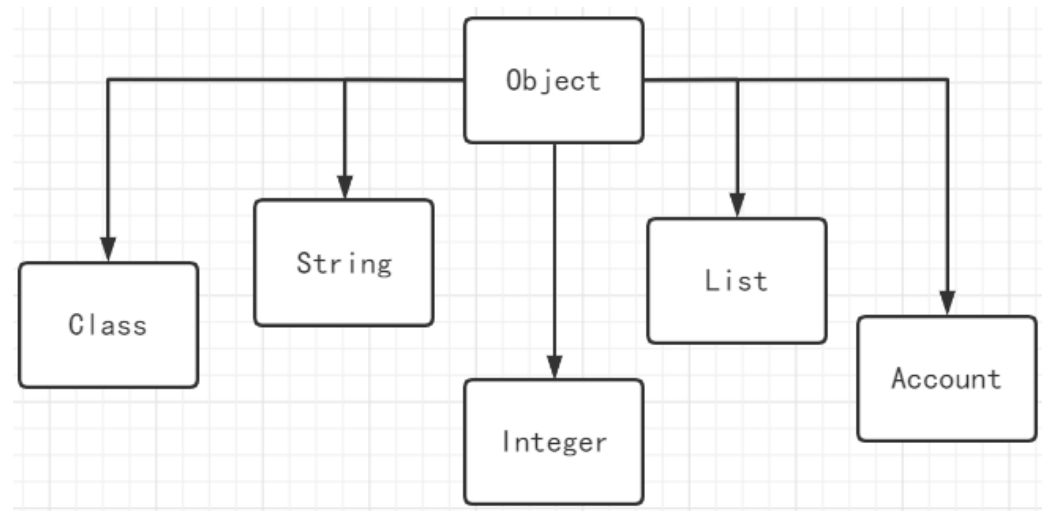


List<**String/Integer 11中基本类型**>的话只能用clone，而且只是分开保存的
List<**Account/自定义的都现象**>的话，可以用clone和deepclone，
第一种是同一个对象，ls1和ls2指向相同，
第二种完全是相当于两个不同的对象

8.3 Object类型

Object Class

※Object类型是一切类型的父类



```
1 Ex.1
2 Object obj = 10;
3 Integer i = Integer.valueOf(obj);
4 System.debug(i);
5
6 Ex.2
7 Object obj = 'Ex.2';
8 String s = (String)obj;
9 System.debug(s);
10
11 Ex.3
12 // 不要和主从关系搞混了，不是一回事
13 List<Object> accList = [SELECT Id, Name, MasterRecordId FROM Account];
14
15 if (accList[0] instanceof Account) {
16     System.debug('this is a Account instance!');
17 }
18
```

8.4 枚举类型

• Enum

枚举类型包括三个方法

1. values

2. name

3. ordinal

```
1 public enum Season {WINTER, SPRING, SUMMER, FALL}
2
3 List<Season> values = Season.values();
4 Integer i = Season.SUMMER.ordinal();
5 String s = Season.SUMMER.name();
6
7 // i:2
8 System.debug('i:' + i);
9
10 // s:SUMMER
11 System.debug('s:' + s);
12
13 // values:(WINTER, SPRING, SUMMER, FALL)
14 System.debug('values:' + values);
15
16 Season ss = Season.SPRING;
17 Season m(Integer x, Season o){
18     if(ss ==Season.SUMMER) return o;
19     //...
20 }
```

说明文档

- System.StatusCode
- System.XmlTag
- System.LoggingLevel
- System.RoundingMode
- System.SoapType
- System.DisplayType
- System.JSONToken
- ApexPages.Severity
- Dom.XmlNodeType

```
1 List values = StatusCode.values();
2 System.debug(values);
3
4 String s = StatusCode.MISSING_ARGUMENT.name();
5 System.debug(s);
6
7 Integer i = StatusCode.MISSING_ARGUMENT.ordinal();
8 System.debug(i);
9
10 //
11 UNKNOWN_EXCEPTION, TOO_MANY_APEX_REQUESTS, MISSING_ARGUMENT, INVALID_ARGUMENT
```

8.4.1 再说System.debug()

System.debug()是可以加第一个参数的

[Logging Level](#)

```
1 System.LoggingLevel level = LoggingLevel.ERROR;
2
3 System.debug(loggingLevel.INFO, 'MsgTxt');
```

第9章 定数（常量）

定数就理解成固定的数，就是不变的数，赋值之后，无法改变
使用static final关键字,static可以不用，
但是，但是一般都是两个关键字一起用
通常全部大写

- static final定义
- 赋值之后，无法改变

```
1 static final Integer PRIVATE_INT_CONST = 200;
2 static final Integer PRIVATE_INT_CONST2;
3
4 static final String PRIVATE_STR_CONST = 'AAAA';
5
6 PRIVATE_INT_CONST2 = 300;
7 // PRIVATE_INT_CONST2 = 400;
8
9 System.debug('PRIVATE_INT_CONST2:' + PRIVATE_INT_CONST2);
10 System.debug('PRIVATE_INT_CONST :' + PRIVATE_INT_CONST2);
11
12 //PRIVATE_INT_CONST2 = 400;
13 //System.debug('PRIVATE_INT_CONST2:' + PRIVATE_INT_CONST2);
```

第10章 运算符(演算子：えんざんし)

[说明文档](#)

10.0 基本演算子

```
1 1.Ex
2 Integer x = 10;
3 Integer y = 20;
4
```

```

5  x = y;
6  System.debug(x);
7
8  2.Ex
9  x += y;
10 x = x + y; //30
11
12 3.Ex 自增自减
13 i++;i--; 先赋值,后自增
14 ++i;--i; 先自增,后赋值
15
16 Integer base = 3;
17 Integer m = base++;
18 System.debug('m=' + m);
19 System.debug('base=' + base);
20 Integer n = ++base;
21 System.debug('n=' + n);
22 System.debug('base=' + base);
23
24

```

25 4.Ex (移位运算)

```

26 x >>= y
27
28 Integer x = 10;
29 System.debug(x >>= 2);
30
31 10的2进制是1010 向右移动1位 0101
32 0000      0
33 0001      1
34 0010      2
35 0011      3
36 0100      4
37 0101      5
38
39 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 10
40 0 , 1 , 2 , 3 , 10
41 0 , 1 , 2 , 3 , 4 , 5 , 6 , 10
42
43

```

44 冯·诺依曼

45 设计思想之一是二进制, 他根据电子元件双稳工作的特点, 建议在电子计算机中采用二进制。
 46 报告提到了二进制的优点, 并预言, 二进制的采用将大大简化机器的逻辑线路。

48 什么是2进制, 就是用0和1对应电子设备的开和关的两种状态 (只有两种)
 49 比如说上电 (通电) 就是1, 下电 (不通电) 就是0

51 计算机的世界都是【0, 1, 0, 1】这样的数字构成, 因为只有0和1两个字, 所以叫2进制
 52 也对应了电子设备的开和关的两种状态

54 现实世界是10进制, 所以有【0, 1, 2, 3, 4, 5, 6, 7, 8, 9】这10个数
 55 实际上是产生了进位, 变成了一零

56
57
58
59

不过，几千年前，易经中就用阴阳来划分世界万物，不知道冯·诺依曼是不是借鉴了易经的思想

10.1 按位与

「&」、「|」

- 可以用作位运算符

按位与规则：有零为零，相当与乘法

例子：12 & 5 = 4

```
1 System.debug(12 & 5); // 结论是：4
```

1	0	1	1	0		
0	1	0	1	0		
0	1	0	0			0100=4

- 也可以用逻辑运算符号

```
1 例子：true & false = falseSystem.debug(12 & 5); // 4
2 System.debug(true & false); OK
3
4 System.debug(true & 10); NG
```

结论：当你不涉到位运算的时候，尽量不要使用一个的。

(&&, ||)

10.2 按位或

```
1 System.debug(12 | 5); // 结论是13
```

1	0	1	1	0		
0	1	0	1	0		
1	1	0	1			
第3位	第2位	第1位	第0位			
$1 \times (2^3)$	$1 \times (2^2)$	$0 \times (2^1)$	$1 \times (2^0)$			$8+4+0+1=13$
	底都是2					

```
1 System.debug(true | false);
```

```
2 System.debug(false| false);
3 System.debug(true| true);
4 System.debug(false| true);
```

结论：当你不涉及到位运算的时候，尽量不要使用。

10.3 && 短路与

第一个如果是false，后面的语句不执行，结果为false

- 只要有一个为false，结果为false

```
1 Integer x = 10;
2 Integer y = 20;
3 Integer z = 30;
4 Integer m = 40;
5
6 // System.debug(只能是布尔类型);
7 System.debug(x > y);
8 System.debug(x > y && true);
9 System.debug(x > y && false);
```

10.4 || 短路或

第一个如果是true，后面的语句不执行，结果为true

- 只要有一个为True，结果为True

```
1 Integer x = 10;
2 Integer y = 20;
3 Integer z = 30;
4 Integer m = 40;
5
6 // System.debug(只能是布尔类型);
7 System.debug(x > y);
8 System.debug(x > y || true);
9 System.debug(x > y || false);
```

10.5 x == y

```
1 String x = '10';
2 String y = '20';
3
4 System.debug(x == y);
5
6 List<Integer> ls = new List<Integer>();
7 ls.add(1);
8
9 List<Integer> ls1 = new List<Integer>();
10 ls1.add(1);
11
```

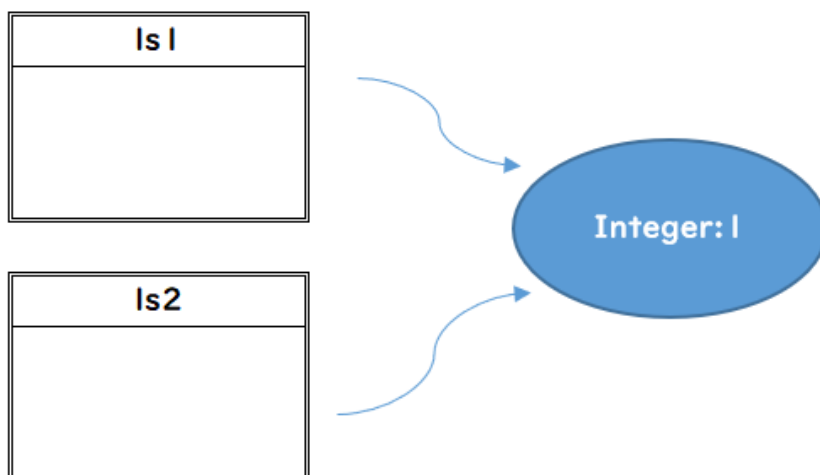
```
12 System.debug(ls == ls1);
```

10.6 $x == y$

这个操作符只能用于**引用类型**

比较严格，不仅要值相等，而且要内存中的位置也相等

```
1 List<Integer> ls = new List<Integer>();
2 ls.add(1);
3
4 List<Integer> ls1 = new List<Integer>();
5 ls1.add(1);
6
7 System.debug(ls == ls1);
8 System.debug(ls === ls1);
9
10 List<Integer> ls2 = ls1;
11
12 System.debug(ls1 == ls2);
13 System.debug(ls1 === ls2);
14
```



第11章 类型转换

11.1 自动类型转换

从小到大可以自动类型转换，反之则不行

```
1 Integer i = 10;
2 Long j = i;
3 Double k = j;
4 Decimal l = k;
5
6 System.debug('j:' + j);
7 System.debug('k:' + k);
8 System.debug('l:' + l);
```

```

9
10 从小到大可以隐式转换，或者说自动转换，颠倒就报错
11
12 类型转换的例子：
13 String a = '1001';
14 Integer aa = Integer.valueOf(a) + 1;
15 System.debug(aa);
16 String b = 'AA-' + String.valueOf(aa);
17 System.debug(b);

```

11.2 强制类型转换

最常用的一个函数 类型转换

1. 类型.valueOf(值) 个人建议用这种
2. (类型)值

```

1 String accId = String.valueOf(accList[0].Id);
2 String accId1 = (String)accList[0].Id;

```

```

1 Integer i = 10;
2 String str1 = String.valueOf(i);
3 System.debug(str1);
4
5 String str2 = '100';
6 Integer j = Integer.valueOf(str2);
7 System.debug(j);
8
9 Decimal a = 10.3;
10 Integer b = Integer.valueOf(a);
11 System.debug('b:' + b);

```

11.3 ID与String

- ID转换成String是可以的

```

1 List<Account> accList = [SELECT Id, Name, MasterRecordId FROM Account];
2 String accId = String.valueOf(accList[0].Id);
3 String accId1 = (String)accList[0].Id;
4 System.debug(accId1);
5

```

- String转换成ID是有危险的

```

1 String str = '0015h00000Bds32AAB';
2 Id id1 = Id.valueOf(stjava);
3 String str = 'aaa132';
4 Id id1 = Id.valueOf(str);
5 System.debug(id1);

```

- 需要转换之前用instanceof 来判断

```
1 // String str = '0015h00000Bds32AAB';
2 String str = 'aaa123';
3
4 if (str instanceof ID) {
5     Id id1 = Id.valueOf(str);
6     System.debug(id1);
7 } else {
8     System.debug('str is not a id!');
9 }
```

第12章 控制语句 (if, else, switch)

12.1 if else

// 中文就是如果, 否则

// 中文编程易语言和VB6.0很类似 做外挂

```
1 如果 ([Boolean_condition])    // 如果
2    // Statement 1
3 否则                            // 否则
4    // Statement 2
5
6 Integer score = 95;
7
8 if(score > 90) {
9     System.debug('有奖励');
10 } else {
11     System.debug('挨揍');
12 }
```

```
1 Integer x, sign;
2 // Your code
3 if (x <= 0) {
4     if (x == 0) {
5         sign = 0;
6     } else {
7         sign = -1;
8     }
9 }
```

```
1 if (place == 1) {
2     medal_color = 'gold';
3 } else if (place == 2) {
4     medal_color = 'silver';
5 } else if (place == 3) {
6     medal_color = 'bronze';
7 } else {
```



```
8 medal_color = null;
9 }
```

```
1 Integer score = 100;
2
3 if(score > 90) {
4     if(score > 95){
5         System.debug('游乐场一天');
6     }else {
7         System.debug('买个小玩具!');
8     }
9 } else {
10     System.debug('挨揍');
11 }
```

12.2 三目运算符

$x ? y : z$

相当于: if-then-else, 而且x不能是空

```
1 Integer score = 100;
2
3 String result = score > 90 ? 'OK':'NG';
4 System.debug(result);
5
```

12.3 switch

12.3.1 Ex1

```
1 switch on expression {
2     when value1 { // when block 1
3         // code block 1
4     }
5     when value2 { // when block 2
6         // code block 2
7     }
8     when value3 { // when block 3
9         // code block 3
10    }
11    when else { // default block, optional
12        // code block 4
13    }
14 }
```

12. 3. 2 Ex2

```
1 // Integer i; 不要给i赋值, 让i是null
2 Integer i = 2;
3 switch on i {
4   when 2 {
5     System.debug('when block 2');
6   }
7   when -3 {
8     System.debug('when block -3');
9   }
10  when null {
11    System.debug('bad integer');
12  }
13  when else {
14    System.debug('default');
15  }
16 }
17
18 // if 改写
19 Integer i = 2;
20 if (i == 2) {
21   System.debug('when block 2');
22 } else if (i == -3) {
23   System.debug('when block -3');
24 } else if (i == null) {
25   System.debug('bad integer');
26 } else {
27   System.debug('default');
28 }
```

12. 3. 3 Ex3

```
1 switch on i {
2   // if (i == 2 or i ==3 or i == 4)
3   when 2, 3, 4 {
4     System.debug('when block 2 and 3 and 4');
5   }
6   when 5, 6 {
7     System.debug('when block 5 and 6');
8   }
9   when 7 {
10    System.debug('when block 7');
11  }
12  when else {
13    System.debug('default');
14  }
15 }
```

12. 3. 4 Ex4

```

1 String str = 'a';
2 switch on str {
3   when 'a', 'b', 'ccc' {
4     System.debug('when block 2 and 3 and 4');
5   }
6   when else {
7     System.debug('default');
8   }
9 }

```

```

1 List<SObject> objList = [Select id,name from Account limit 1];
2
3 switch on objList[0] {
4   when Account a {
5     System.debug('account ' + a);
6   }
7   when Contact c {
8     System.debug('contact ' + c);
9   }
10  when null {
11    System.debug('null');
12  }
13  when else {
14    System.debug('default');
15  }
16 }

```

第13章 循环语句（for, while...）

- ☐ do {statement} while (Boolean_condition);
- ☐ while (Boolean_condition) statement;
- ☐ for (initialization; Boolean_exit_condition; increment) statement;
- ☐ for (variable : array_or_set) statement;
- ☐ for (variable : [inline_soql_query]) statement;

13.1 do while

do {操作内容} while(退出条件);

```

1 Integer count = 1;
2 do {
3   System.debug(count);
4   count++;
5 } while (count < 11);

```

分析：

Step1. 先执行一次do中的语句

Step1. 判断count是否小于11

Step2. 小于11，继续执行do中的语句，一定要有count++语句

Step3. 判断count是否小于11

Step4. 小于11，继续执行do中的语句，

13.2 while

while (退出条件) {操作内容}

```
1 Integer count = 1;
2 while (count < 11) {
3     System.debug(count);
4     count++;
5 }
```

分析：

Step1. 判断count是否小于11

Step2. 小于11，执行do中的语句，一定要有count++语句

Step3. 判断count是否小于11

Step4. 小于11，执行do中的语句，count为2

do {操作内容} while(退出条件);

while (退出条件) {操作内容};

※不同之处在于，do while先执行一次，把上述代码改成11，在分别执行一次

```
1 Integer count = 11;
2 do {
3     System.debug(count);
4     count++;
5 } while (count < 11);
6
7 Integer count = 11;
8 while (count < 11) {
9     System.debug(count);
10    count++;
11 }
```

13.3 for

13.3.1 Ex1

```
1 for (init_stmt; exit_condition; increment_stmt) {
```

```

2   code_block
3 }
4
5 for (Integer i = 0; i < 10; i++) {
6     System.debug(i);
7 }

```

分析:

Step1. Integer i = 0; 程序开始执行（初始化操作，所以只执行一次）

Step2. 判断i是否小于10，这个时候的i为0，小于10

Step3. 小于10，执行System.debug() 语句

Step4. 然后执行i++语句，i变为1

Step5. 判断i是否小于10，这个时候的i为1，小于10

Step6. 小于10，执行System.debug() 语句

Step7. 然后执行i++语句，i变为2

continue和break关键字

```

1 // 实际项目中，当循环查找某个值的之后，找到之后，就直接Break
2 for (Integer i = 0; i < 10; i++) {
3     if (i == 8) {
4         break;
5     }
6     System.debug(i);
7 }
8
9 for (Integer i = 0; i < 10; i++) {
10    if (i == 8) {
11        continue;
12    }
13    System.debug(i);
14 }

```

13.3.2 Ex2

```

1 for (variable : list_or_set) {
2     code_block
3 }
4
5 List<String> strList = new List<String>{'aa','bb'};
6
7 for (String str : strList) {
8     System.debug(str);
9 }

```

13.3.3 Ex3

```
1 for (variable : [sql_query]) {  
2     code_block  
3 }  
4  
5 for (Account acc : [Select Id,Name from Account]) {  
6     System.debug(acc);  
7 }
```