Technology Tarena 达内科技

自定义继承

实现自定义继承

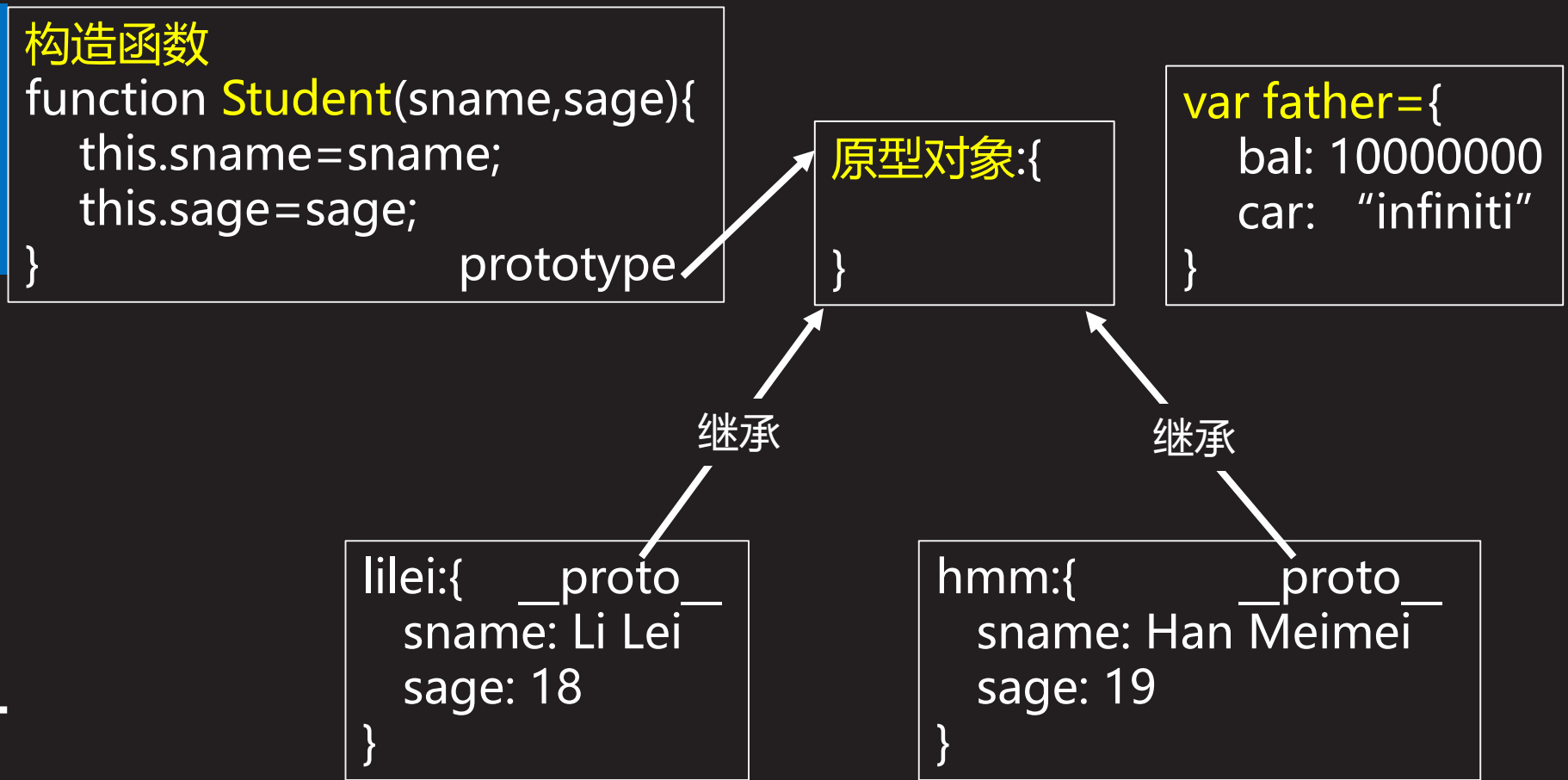
- JavaScript中主要通过修改原型对象来实现自定义继承。
- 主要有两种方式：
  - 1. 单独修改一个对象的原型，而不影响其他对象的原型。
  - 语法: 子对象.__proto__=新父对象
  - 或 Object.setPrototypeOf(子对象，新父对象);
  - 2. 统一修改一类对象的父对象。
  - 语法：构造函数.prototype=新父对象;
  - 时机: 必须在用构造函数创建子对象之前就修改

知识讲解

实现自定义继承

- 单独修改一个对象的原型
  - 比如: hmm.__proto__=father;
  - 或 Object.setPrototypeOf(hmm, father)

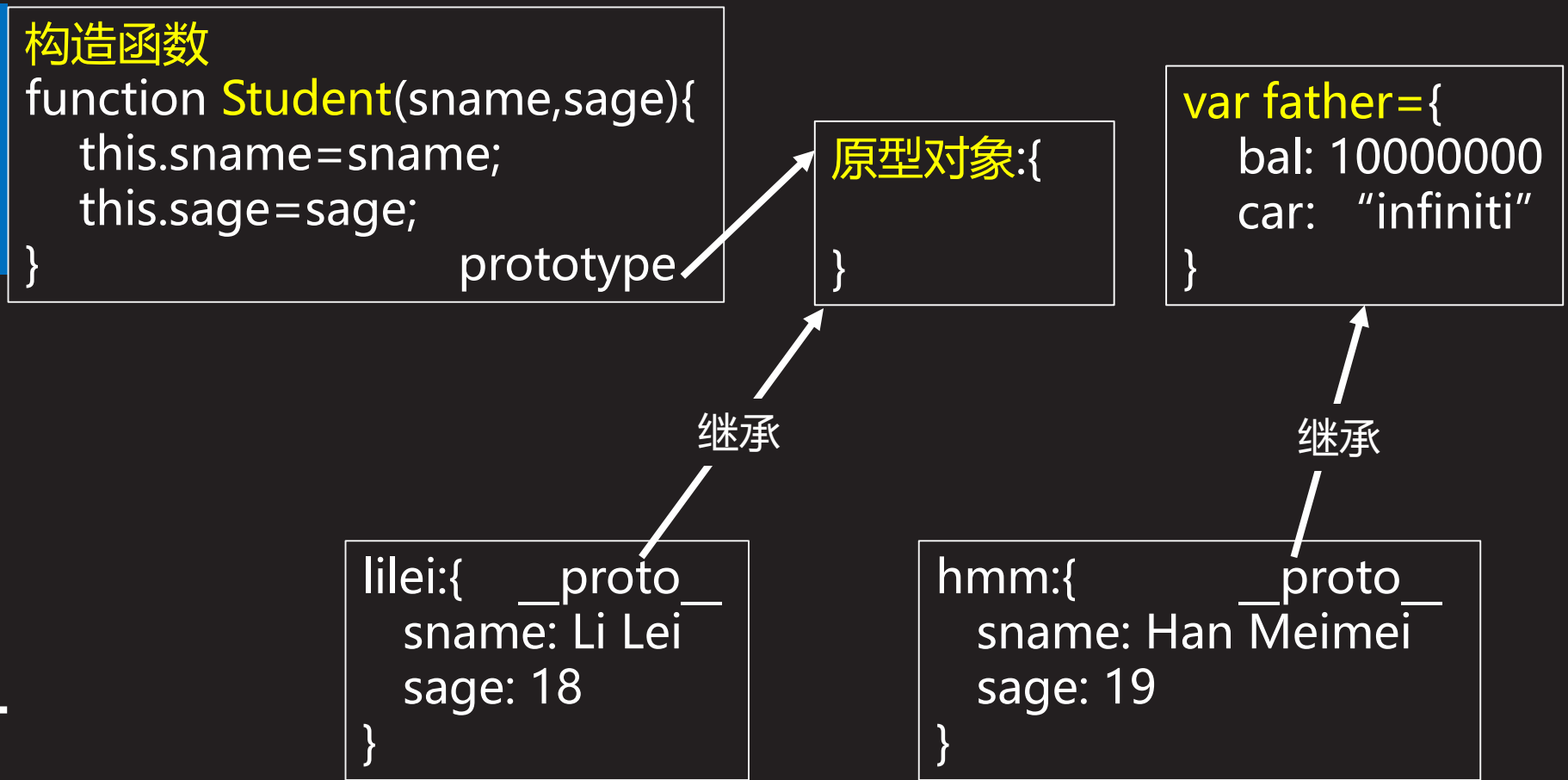
知识讲解



实现自定义继承

- 单独修改一个对象的原型
  - 比如: hmm.__proto__=father;
  - 或 Object.setPrototypeOf(hmm, father)

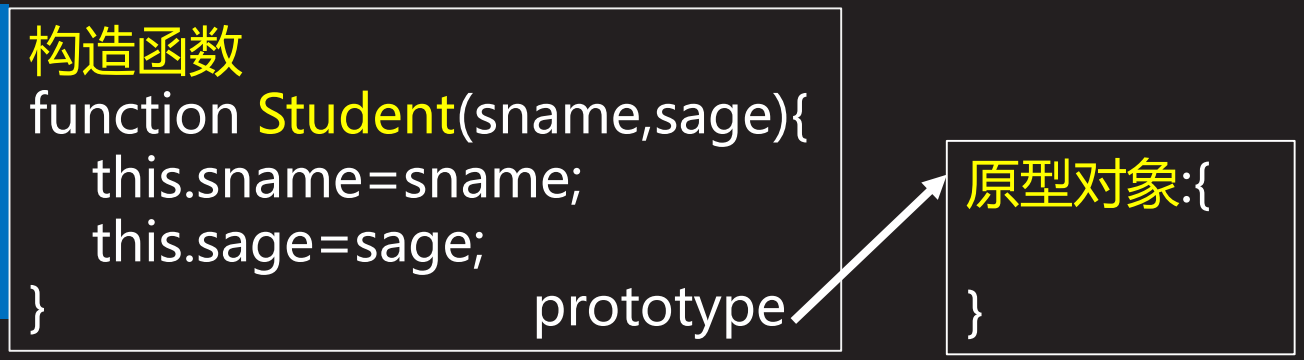
知识讲解



实现自定义继承

- 统一修改一类对象的父对象
  - 比如: 构造函数.prototype=新父对象;

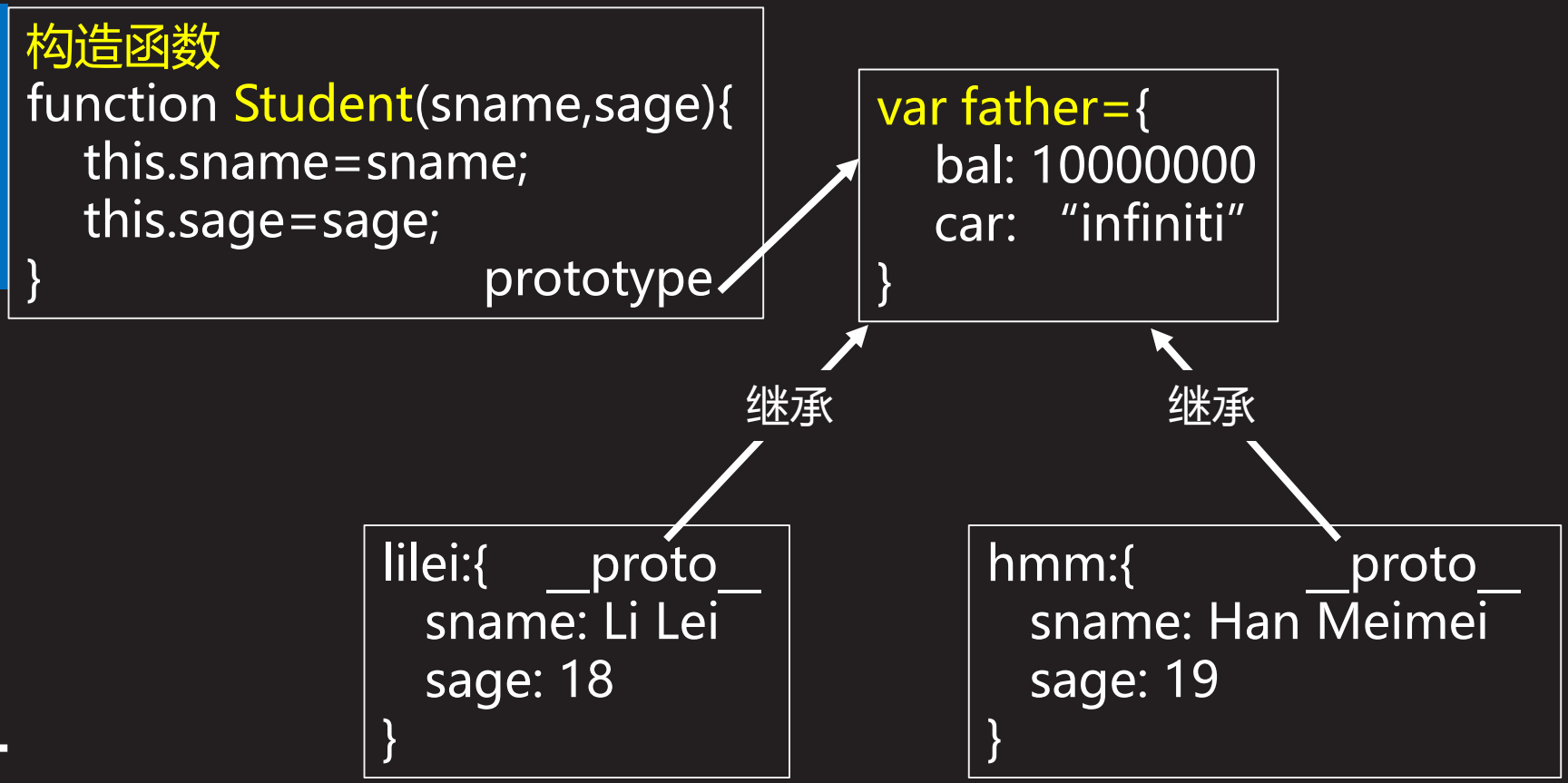
知识讲解



实现自定义继承

- 统一修改一类对象的父对象
  - 比如: 构造函数.prototype=新父对象;

知识讲解



自定义继承

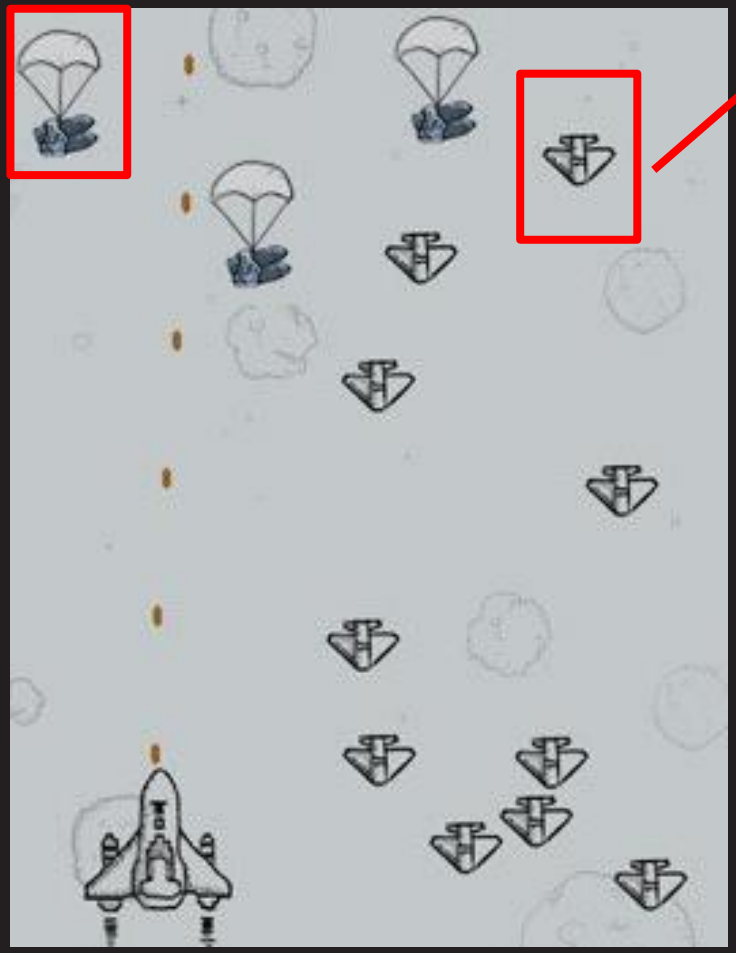
- 使用两种方式实现自定义继承

课堂练习

继承与扩展

- 问题： 用程序结构描述飞机大战中的两类敌人

名称: 小蜜蜂
速度: 50 km/h
奖励: 1次生命



名称: 敌机
速度: 1000 km/h
分数: 5分

知识讲解

继承与扩展

- 解决: 先定义敌机类型，描述所有敌机

构造函数

```
function Plane(fname, speed, score){
    this.fname=fname;
    this.speed=speed;
    this.score=score;
}
```

prototype

知识讲解

继承与扩展

- 解决: 先定义敌机类型，描述所有敌机

知识讲解

原型对象

```
Plane.prototype={
    fly:function(){
        console.log(`${this.fname} 以时速 ${this.speed} 飞行`);
    },
    getScore:function(){
        console.log(`击落${this.fname}得${this.score}分`)
    }
}
```

继承与扩展

- 解决: 再定义蜜蜂类型，描述所有蜜蜂

```
构造函数
function Bee(fname, speed, award){
    this.fname=fname;
    this.speed=speed;
    this.award=award;
}
```

prototype

知识讲解

继承与扩展

- 解决: 先定义敌机类型，描述所有敌机

知识讲解

原型对象

```
Bee.prototype={
    fly:function(){
        console.log(`${this.fname} 以时速 ${this.speed} 飞行`);
    },
    getAward:function(){
        console.log(`击落${this.fname}奖励${this.award} `)
    }
}
```

继承与扩展

- 测试: 实例化一个敌机对象和一个小蜜蜂对象，并调用方法

知识讲解

```
function Plane(...){
    this.fname=fname;
    this.speed=speed;
    this.score=score;
}
```

prototype

```
原型对象:{
    fly:function(){  ... ... },
    getScore:function(){ ... ... }
}
```

继承

```
var f16=new Plane(“F16”, 1000, 5);
f16.fly();  //F16 以时速 1000 飞行
f16.getScore();  //击落F16 得 5 分
```

```
f16: {          __proto__
  fname:”F16”,
  speed: 1000,
  score: 5
}
```

继承与扩展

- 测试: 实例化一个敌机对象和一个小蜜蜂对象，并调用方法

知识讲解

```
function Bee(...){
    this.fname=fname;
    this.speed=speed;
    this.award=award;
}
```

prototype

```
原型对象:{
    fly:function(){  ... ... },
    getAward:function(){ ... ... }
}
```

继承

```
var bee=new Bee(“小蜜蜂”, 50 , “1次生命”);
bee.fly(); //小蜜蜂 以时速 50 飞行
bee.getAward(); //击落小蜜蜂 奖励 1次生命
```

```
bee: {          __proto__
  fname:”小蜜蜂”,
  speed: 50,
  award: “一次生命”
}
```

继承与扩展

- 问题: Plane和Bee两种类型之间有部分相同的属性和方法

```
function Plane(...){
    this.fname=fname;
    this.speed=speed;
    this.score=score;
}
```

prototype

```
原型对象:{
    fly:function(){  ... ... },
    getScore:function(){ ... ... }
}
```

```
function Bee(...){
    this.fname=fname;
    this.speed=speed;
    this.award=award;
}
```

prototype

```
原型对象:{
    fly:function(){  ... ... },
    getAward:function(){ ... ... }
}
```

知识讲解

继承与扩展

- 解决: 第一步：多定义一个抽象父类型Enemy，集中保存共有的属性和方法

知识讲解

```
function Plane(...){
    this.fname=fname;
    this.speed=speed;
    this.score=score;
}              prototype
```

```
原型对象
Plane.prototype= {
    fly:function(){  ... ... },
    getScore:function(){ ... ... }
}
```

继承与扩展

- 解决: 第一步：多定义一个抽象父类型Enemy，集中保存共有的属性和方法

```
function Enemy(...){


}              prototype
```

```
原型对象
Enemy.prototype={

}
```

知识讲解

```
function Plane(...){
    this.fname=fname;
    this.speed=speed;
    this.score=score;
}              prototype
```

```
原型对象
Plane.prototype= {
    fly:function(){  ... ... },
    getScore:function(){ ... ... }
}
```

继承与扩展

- 解决: 第一步：多定义一个抽象父类型Enemy，集中保存共有的属性和方法

```
function Enemy(...){
    this.fname=fname;
    this.speed=speed;
}
```

prototype

```
原型对象
Enemy.prototype={
    fly:function(){  ... ... },
}
```

```
function Plane(...){
    this.fname=fname;
    this.speed=speed;
    this.score=score;
}
```

prototype

```
原型对象
Plane.prototype= {
    fly:function(){  ... ... },
    getScore:function(){ ... ... }
}
```

知识讲解

继承与扩展

- 解决: 第一步：多定义一个抽象父类型Enemy，集中保存共有的属性和方法

```
function Enemy(...){
    this.fname=fname;
    this.speed=speed;
}
```

prototype

```
原型对象
Enemy.prototype={
    fly:function(){  ... ... },
}
```

```
function Plane(...){
    this.score=score;
}
```

prototype

```
原型对象
Plane.prototype={
    getScore:function(){ ... ... }
}
```

知识讲解

继承与扩展

- 问题: 子对象会丢失原有部分属性和方法:

```
function Plane(...){
    this.score=score;
}
```

prototype →

```
原型对象:{
    getScore:function(){ ... ... }
}
```

```
var f16=new Plane( “F16”, 1000, 5);
```

继承

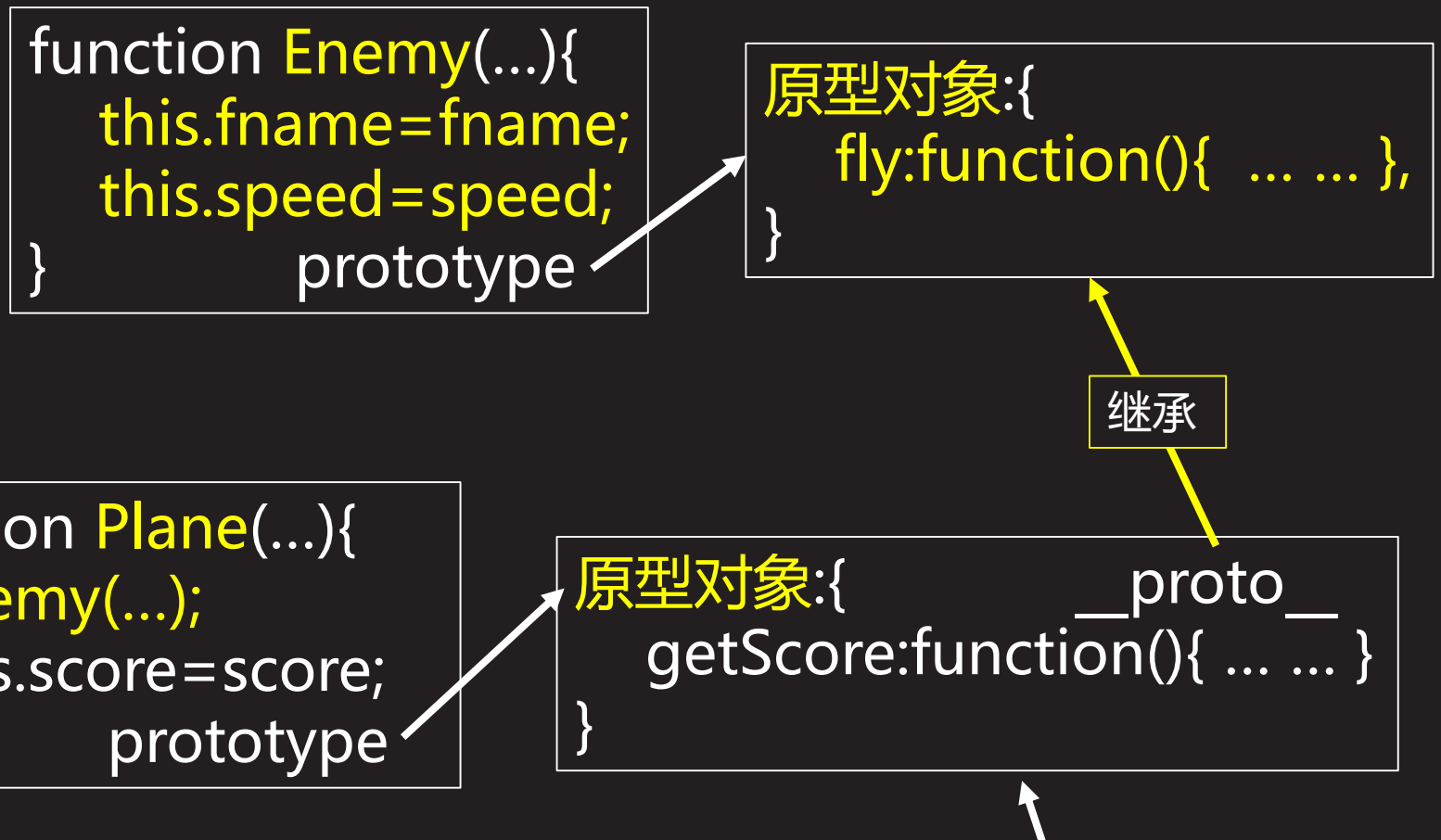
```
f16: {    __proto__
  score: 5
}
```

知识讲解

继承与扩展

- 解决: 子类型Plane继承父类型:
  - 1. 子类型原型对象继承父类型原型对象——获得fly()方法
  - 2. 子类型构造函数中调用父类型构造函数——获得失去的属性

知识讲解



继承与扩展

```
var f16=new Plane(“F16”, 1000, 5);
f16.getScore();  //击落F16 得 5 分
f16.fly();  //F16 以时速 1000 飞行
```

知识讲解

继承与扩展

- 使用两种类型间的继承，定义敌机和小蜜蜂类型，避免重复。
- 实例化两种类型的对象，调用方法，观察结果是否正确
- 结果：

```
var f16=new Plane(“F16”, 1000, 5);
f16.getScore();  //击落 undefined 得 5 分
f16.fly();  // undefined 以时速 undefined 飞行
```

```
f16: {          __proto__
    score: 5
}
```

课堂练习

继承与扩展

- 问题: 抽象父类型的方法，子对象可用，但抽象父类型中的属性，子对象中没有!
- 原因: Enemy(...)前没有任何前缀，所以Enemy()中的this默认指向window，导致fname和speed泄露到全局中。

知识讲解

```
function Enemy(...){
  window.fname=fname;
  window.speed=speed;
}
```

prototype

```
原型对象:{
    fly:function(){ ... ... },
}
```

继承

```
function Plane(...){
  Enemy(...); //this->window
  this.score=score;
}
```

prototype

```
原型对象:{                    __proto__
    getScore:function(){ ... ... }
}
```

继承与扩展

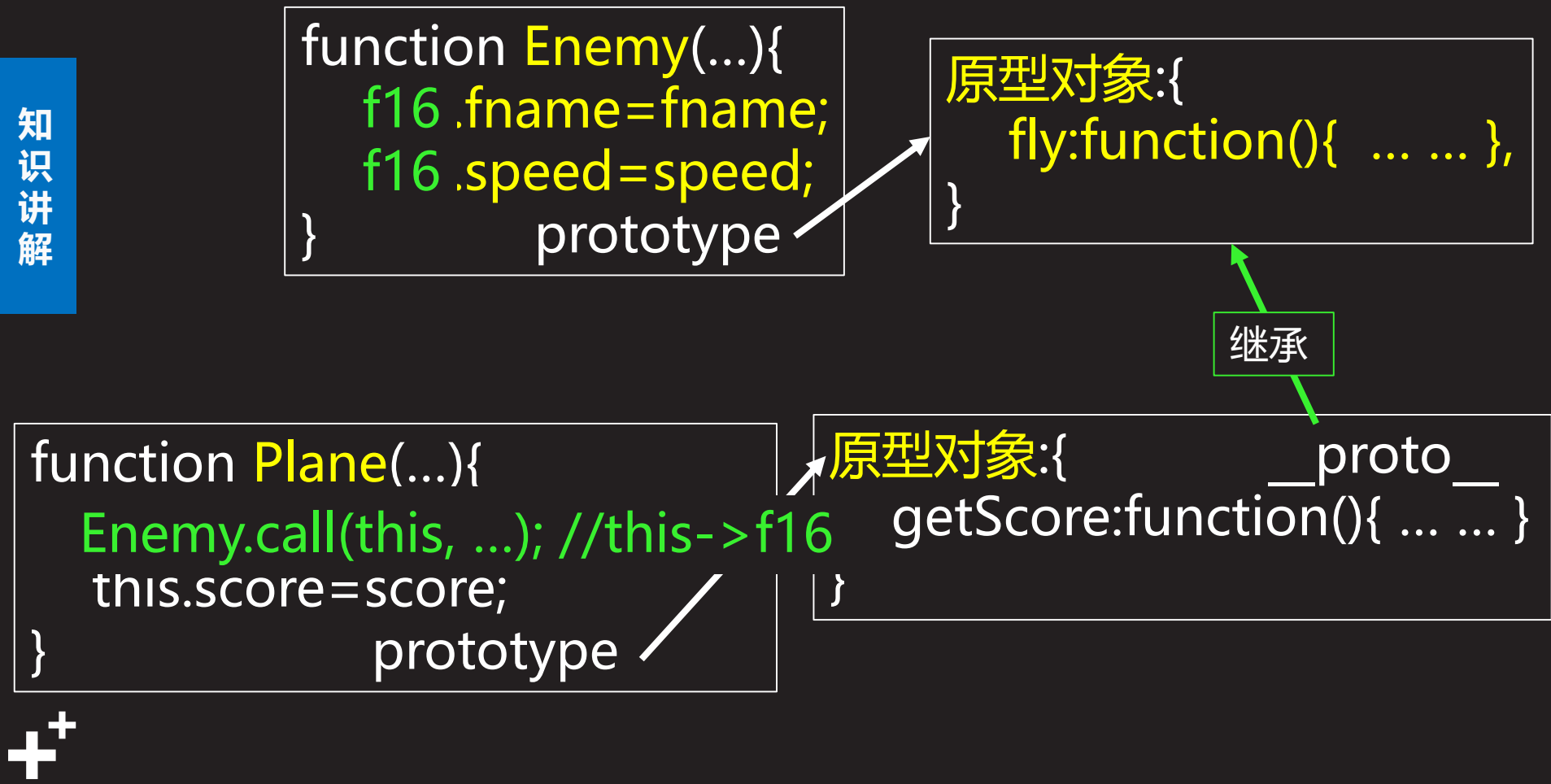
知识讲解

- 解决: 只要函数调用时，函数中的this不是想要的，就可用.call()调用，换成想要的对象
- 语法: 要调用的函数.call(替换this的对象, 其余参数值...)
- 比如:
  - Enemy(fname,speed); 中this->window
  - 实际执行: window.fname=fname; window.speed=speed;
  - 换成Enemy.call(f16, fname, speed); 中this->f16
  - 实际执行: f16.fname=fname; f16.speed=speed;

继承与扩展

- 解决: 只要函数中的this不是想要的，就可用.call()调用，换成想要的对象

知识讲解



继承与扩展

- 使用call修正this不正确的问题

课堂练习

Technology Tarena 达内科技

总结和答疑