1. 诊断与通信管理功能（Diagnostic and Communications Management）

这类服务主要用于控制诊断会话的状态、安全访问以及通信模式。

0x10 (Diagnostic Session Control)：诊断会话控制。用于切换 ECU 的会话模式（如默认会话、扩展会话、编程会话等）。

0x11 (ECU Reset)：ECU 重启。包括硬重启、软重启等。

0x27 (Security Access)：安全访问。用于解锁 ECU 的某些高级权限（如写数据或刷写软件），通常通过种子和密钥（Seed & Key）机制实现。

0x28 (Communication Control)：通信控制。用于打开或关闭 ECU 的某些报文收发。

0x3E (Tester Present)：测试仪在线。用于告诉 ECU 测试设备还在连接中，防止 ECU 因超时自动退回默认会话模式。

0x85 (Control DTC Setting)：控制 DTC（故障码）设置。用于开启或关闭 ECU 记录故障码的功能。

2. 数据传输功能（Data Transmission）

用于读取或写入 ECU 内部的数据。

0x22 (Read Data By Identifier)：通过标识符读取数据（配合 DID 使用，常用于读 VIN 码、软硬件版本号、传感器电压等）。

0x24 (Read Scaling Data By Identifier)：读取数据比例信息。

0x2A (Read Data By Periodic Identifier)：周期性读取数据。

0x2C (Dynamically Define Data Identifier)：动态定义数据标识符。

0x2E (Write Data By Identifier)：通过标识符写入数据（写入配置参数、VIN 码等）。

3. 存储数据传输/故障排查功能（Stored Data Transmission）

专门用于读取和清除车辆的故障码（DTC）。

0x14 (Clear Diagnostic Information)：清除诊断信息（即清空故障码）。

0x19 (Read DTC Information)：读取故障码信息（包括读取当前故障、历史故障、快照数据等）。

4. 输入输出控制功能（Input/Output Control）

0x2F (Input Output Control By Identifier)：输入输出控制。用于强制覆盖 ECU 的正常逻辑，直接控制执行器（如强制点亮车灯、启动风扇等）。

5. 例程控制功能（Routine Control）

0x31 (Routine Control)：例程控制。用于启动、停止或请求 ECU 内部某个特定程序/例程的运行结果（配合 RID 使用，常用于复杂的自检动作）。

6. 上传与下载功能（Upload/Download）

主要用于 ECU 的软件刷写（OTA 或线下烧录）。

0x34 (Request Download)：请求下载（测试设备向 ECU 发送数据前申请）。

0x35 (Request Upload)：请求上传（测试设备从 ECU 读取大段内存数据前申请）。

0x36 (Transfer Data)：传输数据（实际传输刷写的数据流）。

0x37 (Request Transfer Exit)：请求退出传输（数据传输完毕后结束流程）。

每个请求诊断的协议第一位就是 SID 后续就是数据

一、 发送方式（Addressing Modes)

测试仪向 ECU 发送请求时，主要有两种“喊话”方式：

1. **物理寻址（Physical Addressing)：点对点（1 对 1)**
    - 测试仪明确指定某一个具体的 ECU 进行通信。
    - **应用场景**：绝大多数诊断服务（如读写数据、安全访问、例程控制）。
    - **举例**：比如在 CAN 总线上，规定用 0x7E0 的 ID 发给发动机 ECU，发动机 ECU 固定用 0x7E8 的 ID 回复。
2. **功能寻址（Functional Addressing)：广播（1 对多)**
    - 测试仪向网络上的所有 ECU 同时发送同一个请求，支持该请求的 ECU 都会给出响应。
    - **应用场景**：通常用于需要多个节点同时执行的动作（如 Tester Present 0x3E 保持全车唤醒，或清除全车故障码 0x14）。
    - **举例**：在 CAN 总线上，测试仪通常用 0x7DF 的广播 ID 发送请求。

二、 请求与响应的报文结构

在应用层（不考虑底层 CAN/LIN 帧头的封装），UDS 数据的基本结构是由 **SID（服务标识符）**、**Sub-function（子功能）** 和 **Data（数据参数）** 组成的。

1. 发送请求（Request）

测试仪发送的数据格式通常为：

[SID] + [Sub-function] + [Data]

SID：1 字节。告诉 ECU 要做什么服务（如 0x10 会话控制，0x22 读数据）。

Sub-function：1 字节（可选）。告诉 ECU 具体怎么做。比如 0x10 服务下，0x01 代表默认会话，0x03 代表扩展会话。*（注：子功能字节的最高位 bit 7 通常是抑制肯定响应位（SPRM），如果置 1，ECU 执行成功后将保持沉默，不发肯定响应以节省总线带宽。）*

Data：若干字节（可选）。如具体的 DID、密码密钥等。

2. 肯定响应（Positive Response）

当 ECU 成功接收请求并顺利执行后，会返回肯定响应。其最核心的特征是：**响应 SID = 请求 SID + 0x40**。

[SID + 0x40] + [Sub-function] + [Data]

举例：

- 请求：10 03 （进入 0x03 扩展会话）
- 响应：50 03 ... （0x10 + 0x40 = 0x50，表示成功进入）

3. 否定响应（Negative Response）

当 ECU 收到请求，但因为某些原因（不支持、条件不满足、格式错误等）无法执行时，会返回否定响应。其格式是固定且统一的 3 个字节：

[0x7F] + [请求 SID] + [NRC]

0x7F：UDS 规定的固定否定响应标志。

请求 SID：告诉你是因为哪个请求被拒绝了。

NRC (Negative Response Code)：否定响应码，1 个字节。它是一本“错误字典”，明确告诉测试仪具体出错的原因。

三、 常见的 NRC（错误码）解析

看懂 NRC 是诊断排错的关键。以下是实际工程中最常见的几个 NRC：

0x11 (Service Not Supported)：服务不支持。你发了一个 ECU 根本不认识的 SID。

0x12 (Sub-function Not Supported)：子功能不支持。ECU 认识这个 SID，但不认识你带的 Sub-function。

0x13 (Incorrect Message Length or Invalid Format)：报文长度错误或格式无效。比如协议规定读 DID 要发 3 个字节，你只发了 2 个字节。

0x22 (Conditions Not Correct)：条件不满足。比如在车辆行驶中请求刷写软件，ECU 出于安全考虑会用这个码拒绝你。

0x31 (Request Out Of Range)：请求超出范围。比如你请求读取 DID 0xFFFF，但

ECU 代码里根本没定义这个 DID。

0x33 (Security Access Denied)：安全访问被拒绝。ECU 处于锁定状态，你必须先跑完 0x27 解锁流程，才能执行此操作。

否定响应码（NRC）	
0x10	该否定响应不是协议里已定义的
0x11	请求的服务压根就没做进去
0x12	ECU不支持当前请求的子功能
0x13	请求报文的长度或格式不正确
0x22	先决条件不满足
0x24	请求报文的顺序不正确
0x31	参数超出范围/数据ID不支持
0x33	不满足安全策略，请先解锁
0x35	密钥不匹配（27服务）
0x36	尝试解锁次数已达上限（27服务）
0x37	超时时间未到（27服务）
0x70	不允许上传/下载（34服务）
0x71	数据传输中止
0x72	擦除或烧写内存时错误
0x73	块序列计数错误
0x78	已收到请求，但会晚些响应
0x7E	当前会话下，该子功能不支持
0x7F	当前会话下，该服务不支持



请求和响应都要有源地址与目标地址

UDS 与 CAN 报文

诊断请求在 CAN 上的真实形态

- **UDS 意图**：发送 `22 02 07`（总共3个有效字节）。
- **真实的 CAN 报文**：

```
ID: 0x709 | DLC: 8 | Data: 03 22 02 07 00 00 00 00
```

解析：

- `0x709`：诊断仪的发送ID。
- `DLC: 8`：代表这条CAN报文装满了8个字节。
- `03`：这是网络层（CAN-TP）加上的**单帧（Single Frame）控制字节**。`0` 代表这是一帧就能发完的短数据，`3` 代表后面跟着 3个字节 的真实UDS数据。
- `22 02 07`：这是真正的 UDS 请求内容。
- `00 00 00 00`：填充字节（Padding）。因为不够8个字节，为了填满CAN帧，通常用 `0x00` 或 `0xAA` 占位，没有实际意义。

诊断响应在 CAN 上的真实形态

- **UDS 意图**：发送否定响应 `7F 22 11`（也是3个有效字节）。
- **真实的 CAN 报文**：

```
ID: 0x701 | DLC: 8 | Data: 03 7F 22 11 00 00 00 00
```

解析：
原理完全一样，`0x701` 是ECU发送的ID，第一个字节 `03` 代表有3个字节的有效回复，后面紧跟 `7F 22 11`，最后补零。

在 UDS 诊断协议中，“会话”（Session）的全称是**诊断会话（Diagnostic Session）**，它对应的服务 SID 是 **0x10 (Diagnostic Session Control)**。

通俗地讲，**会话就是 ECU 的“权限模式”或“工作状态”**。它决定了在当前状态下，诊断仪（Tester）可以对 ECU 执行哪些操作，不能执行哪些操作。

打个大家都熟悉的比方：这就好比你使用电脑时的**账户权限机制**：

不输密码时的“锁屏界面”（只能看个时间、接个电话）。

普通登录后的“访客模式”（能看网页，但不能装软件）。

输入管理员密码后的“超级管理员模式”（可以修改系统底层文件、重装系统）。

在汽车 ECU 中，为了保证行驶安全和数据安全，也设计了这样一套严格的层级。UDS 协议定义了多种会话模式，其中最常用、最核心的有以下三种：

1. 默认会话（Default Session - 0x01）

状态：相当于“锁屏/访客模式”。汽车一通电启动，所有 ECU 默认都会处于这个状态。

权限：**最低**。在这个模式下，ECU 一边正常控制汽车（比如喷油、换挡），一边只能接受最基础的诊断请求。

能做的事：读取一些非关键的通用数据（比如车速、水温）、读取排放相关的基本故障码。

不能做的事：不能写数据、不能强制控制执行器、绝对不能刷写软件。

2. 扩展诊断会话（Extended Diagnostic Session - 0x03）

状态：相当于“高级维修模式”。如果要修车或者进行复杂的标定，诊断仪必须发指令（10 03）让 ECU 切换到这个模式。

权限：**较高**。进入这个模式后，ECU 会开放更多的高级功能。

能做的事：读取和清除所有故障码、写入配置参数（如改写 VIN 码）、触发例程测试（比如强制启动电子风扇转动）。

注意：很多高级操作（比如写数据）在进入扩展会话后，还需要配合 **0x27 安全访问**（输入动态密码）才能最终执行。

3. 编程会话 (Programming Session - 0x02)

状态：相当于电脑的“BIOS 刷机模式”或手机的“DFU 恢复模式”。

权限：**最高且最危险**。

能做的事：专门用来给 ECU 刷写底层的软件代码（OTA 升级或线下烧录）。

代价：一旦进入编程会话，ECU 通常会**停止正常的应用层工作**（比如发动机 ECU 不再控制喷油，仪表盘可能会黑屏或全亮报错），全心全意接收诊断仪传来的几兆大小的软件升级包。

核心机制：会话的“防呆”退回（Timeout）

你可能会想：如果我让 ECU 进入了“扩展会话”或“编程会话”，然后我突然把诊断仪拔了，ECU 岂不是一直卡在这个危险的高权限状态？

为了防止这种情况，UDS 引入了一个极其重要的安全机制：**S3 定时器**。

当 ECU 处于非默认会话（如扩展或编程）时，它会在后台启动一个倒计时（通常是 5 秒左右）。

如果诊断仪在倒计时结束前，没有跟 ECU 说任何话，**ECU 就会认为诊断仪已经掉线，随后强制、自动退回到最安全的“默认会话”（0x01）**。

这就解释了为什么前面提到了 **0x3E (Tester Present，测试仪在线)** 这个服务。

如果诊断仪暂时不需要发指令，但又想保持高级会话权限，它就必须每隔两三秒给 ECU 发送一个 3E 报文，告诉 ECU："我还连着呢，别退回默认状态！"（相当于你每隔几秒晃一下鼠标，防止电脑自动息屏）。

一帧 CAN 报文装不下的情况下





1 个字节（Byte，简称大写的 B） = 8 个比特（bit，简称小写的 b）

结合咱们刚才一直聊的 UDS 和 CAN 报文，我们可以从以下几个维度来直观地理解"字节"到底有多大，以及它在诊断中长什么样：

1. 最底层的本质：8 个二进制"开关"

比特（bit）是最小的信息单位，它就像一个开关，只有两种状态：0（关）或 1（开）。

字节（Byte）就是把 8 个这样的开关捆绑在一起作为一个基本单元。

比如，1 个字节在计算机底层线路里跑的时候，实际长这样：01001101。

2. 它能装下多少东西（容量）？

因为 1 个字节有 8 个 bit 位，每一位都有 0 或 1 两种可能，所以它可以组合出 $2^8 = 256$ 种不同的状态。

表示数字：它可以表示从 **0 到 255** 的十进制正整数。比如前面提到的 NRC（否定响应码）0x11，它就是一个字节，对应十进制的 17。

表示字符：在标准的 ASCII 编码中，1 个字节正好可以存下 **1 个英文字母或标点符号**（比如字母 A 在底层就是 65）。

3. 在 UDS 和 CAN 报文中的样子（十六进制）

人类去看 01001101 这种二进制太痛苦了，所以在分析 UDS 诊断报文时，工程师统一使用十六进制（Hex）来显示。

这里有一个极其重要的对应关系：**每 2 个十六进制字符，正好等于 1 个字节**。

咱们之前图片里看到的请求报文 **22 02 07**，总共是 **3 个字节**。

- 第 1 个字节是 22（读数据服务）。
- 第 2 个字节是 02。
- 第 3 个字节是 07。

我们说经典 CAN 报文一帧最多只能装 **8 个字节**，意思就是在抓包软件里，它的 Data 域最多只能写满 16 个十六进制字符（例如：03 22 02 07 00 00 00 00）。

一、 单帧 (SF - Single Frame) —— 首字节以 0 开头

当你要发送的 UDS 诊断数据长度 **≤7 个字节** 时，一帧 CAN 报文就能搞定。

格式：[0 + 长度] [UDS 数据 1] [UDS 数据 2] ...

拆解：首字节的高 4 位固定是 0（代表单帧），低 4 位代表后面跟着几个真实数据字节。

举例：发送 22 02 07（3 个字节的数据）。

- CAN 报文 Data 域：03 22 02 07 00 00 00 00。
- （首字节 03，0 代表单帧，3 代表后面有 3 个字节有效数据）。



二、 多帧传输的“三步曲”

当 UDS 数据长度 **>7 个字节** 时，就必须启动多帧传输流程。这个流程像极了现实中的“发大宗物流”，需要用到剩下三种类型的帧互相配合：

第一步：首帧（FF - First Frame）—— 发送方打招呼，首字节以 1 开头

发送方先发一个“预告”，告诉接收方：“我要给你发个大包裹了，总共有多大，并且先附上最开头的几个字节。”

格式：首字节的高 4 位是 1。接下来的 12 个 bit 用来表示**数据的总长度**。

特点：因为占用了 2 个字节做控制信息，所以首帧只能带 **6 个字节** 的真实 UDS 数据。

这是一个非常敏锐的问题！你准确地抓住了网络层（CAN-TP）最核心的“拼图游戏”——**如何在有限的空间里把控制信息和真实数据塞进去。**

为了让你彻底明白这“12 个 bit”是怎么算出来的，以及为什么“只剩下 6 个字节”，我们拿一个放大镜，把首帧（FF）的底层比特（bit）级别拆开来看。

一、 8 字节的“空间保卫战”

我们已知一个前提：**一帧经典的 CAN 报文，它的 Data 域总共只有 8 个字节（8 Bytes）。**

当诊断仪要发送一条很长的数据（比如 20 个字节）时，它发出的**首帧（First Frame）**需要承担两个任务：

1. **自我介绍**：告诉接收方“我是一个首帧，你要准备好接大包裹了，包裹总大小是 X”。
2. **带点干货**：既然车都跑一趟了，不能只喊话，得顺便把大包裹的**最前面一部分数据**先运过去。

为了完成任务 1，协议规定必须从 8 个字节中**切出 2 个字节**来存放这个“自我介绍”的控制信息（这就是 PCI 控制字节）。

既然 8 个字节被占用了 2 个，那用来装真实 UDS 数据的空间自然就只剩下：**8 - 2 = 6 个字节**了。

二、 放大镜看前 2 个字节（16 个 bit）

我们重点来看看这被占用的前 2 个字节（Byte 0 和 Byte 1）。

1 个字节是 8 个 bit，2 个字节就是 **16 个 bit**。它们是怎么分配的呢？

字节位置	包含的 bit	功能分配	说明
Byte 0 (高半部分)	第 1 ~ 4 个 bit	**帧类型 (Frame Type)**	这 4 个 bit 用来定义帧的类型。首帧固定为 1（二进制就是 0001）。
Byte 0 (低半部分)	第 5 ~ 8 个 bit	**数据总长度 (最高位)**	
Byte 1 (完整)	第 9 ~ 16 个 bit	**数据总长度 (低位)**	

仔细看上面的表格，奇妙的化学反应发生了：

控制信息的总长度是 16 个 bit。

减去用来表示类型的 4 个 bit，用来表示**“大包裹总长度”**的位数正好剩下：**16 - 4 = 12 个 bit**！

这 12 个 bit 跨越了两个字节：它占用了 Byte 0 的后半段，以及整个 Byte 1。

三、 实战举例：发送 20 字节的数据

我们拿刚才“读取 17 字节 VIN 码”的例子来实战推演一下。

ECU 需要向诊断仪回复 20 个字节（即十六进制的 0x14）的数据。

1. 计算 12 bit 的长度信息：

20 转换成十六进制是 14。

因为我们要把它填进 12 个 bit 的格子里（相当于 3 个十六进制字符），所以要在前面补零，写成：**0 1 4**。

2. 拼装 Byte 0 和 Byte 1：

Byte 0：前半部分填帧类型 1，后半部分填长度的最高位 0。拼在一起就是十六进制的 **10**。

Byte 1：填入长度剩下的位 **14**。

3. 首帧的最终 CAN 报文形态：

10 14 [数据 1] [数据 2] [数据 3] [数据 4] [数据 5] [数据 6]

在这个报文里：

1 告诉诊断仪：我是首帧。

014 告诉诊断仪：这个大包裹一共 20 个字节，你准备好内存空间。

后面的 6 个位置：塞满了真实数据的头 6 个字节。

第二步：流控制帧（FC - Flow Control）—— 接收方定规矩，首字节以 3 开头

接收方收到首帧后，不能让发送方盲目往下发（万一发太快，接收方芯片处理不过来死机了怎么办？）。接收方必须回一个 FC 帧，告诉发送方自己目前的接收能力。 FC 帧

包含三个关键参数：

1. **流状态（FS）**：0 代表“继续发送（Clear to send）”，1 代表“请稍等（Wait）”，2 代表“缓存满了，溢出（Overflow）”。

2. **块大小（BS - Block Size）**：告诉发送方“你连续发几个包之后，必须停下来等我再发一次 FC 确认”。如果设为 0，代表“一口气全发过来，不用等我确认”。

3. **最小时间间隔（STmin）**：告诉发送方“你发每个连续帧之间，必须至少间隔多少



毫秒（比如 10ms）”。

第三步：连续帧（CF - Consecutive Frame）—— 发送方持续发货，首字节以 2 开头

发送方收到 FC（同意发送）后，开始按照 FC 规定的节奏，源源不断地发送剩下的数据。

格式：首字节高 4 位是 2，低 4 位是**流水号（Sequence Number，简称 SN）**。

流水号规则：SN 从 1 开始，1, 2, 3... 一直数到 15（即 16 进制的 F），然后再从

0 重新开始（0, 1, 2...）。流水号的目的是防止包裹在总线上丢了或者乱序。

特点：占用 1 个控制字节，每帧能带 **7 个字节** 的真实 UDS 数据。



三、 完整实战推演：读取 17 字节的 VIN 码

我们来模拟一次真实的读取 VIN 码（读取长数据）的对话。**场景**：诊断仪请求读取 VIN（DID 是 F1 90）。ECU 的 VIN 码是 17 个字符（加上 3 个字节的肯定响应前缀，总共要回传 20 个字节）。

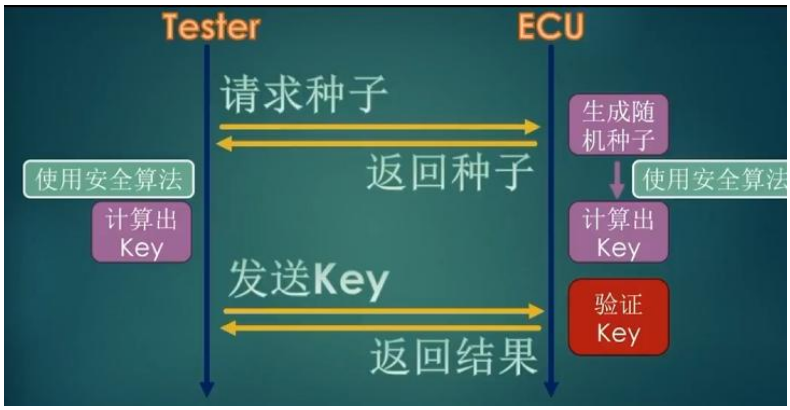
1. **诊断仪发送请求（单帧 SF）**：
    - 报文：03 22 F1 90 00 00 00 00 （我要读数据，只要 1 帧就够了）
2. **ECU 回复首帧（FF）**：
    - 报文：10 14 62 F1 90 57 41 55 （10 14 代表首帧，总数据长 0x14 即 20 字节。后面跟了前 6 个数据字节）
3. **诊断仪回复流控制帧（FC）**：
    - 报文：30 00 0A 00 00 00 00 00 （30 代表同意发送，00 代表一口气发完

不用分块，0A 代表每帧间隔 10 毫秒）

4. **ECU 发送连续帧 1 (CF1)**：
    - 报文：21 xx xx xx xx xx xx xx （21 代表第一帧，后面跟 7 个字节数据）
5. **ECU 发送连续帧 2 (CF2)**：
    - 报文：22 xx xx xx xx xx xx xx （22 代表第二帧，剩下最后的 7 个字节数据）

整个传输结束！诊断仪收到所有报文后，把包拆掉，按顺序拼接，就得到了完整的 20 字节响应数据。

0x27 解锁 ECU

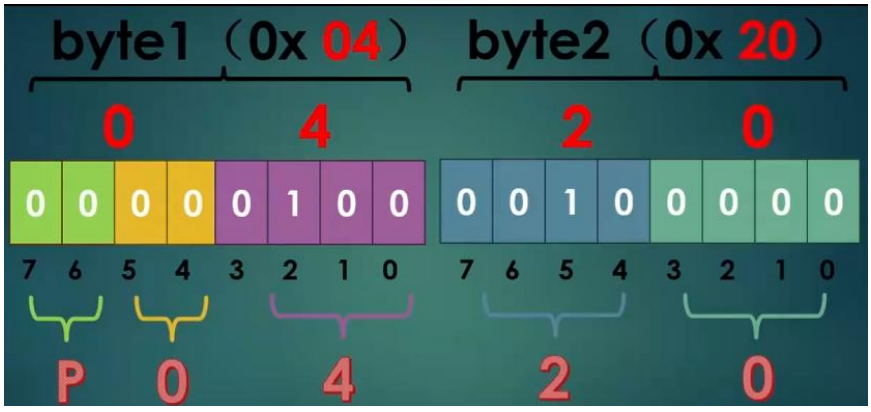


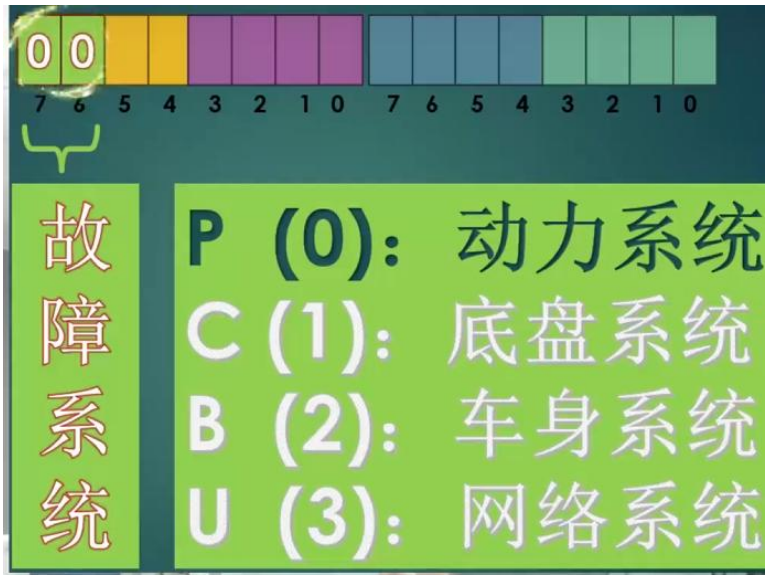
错误返回

NRC	描述
0x12	27服务的子功能不被当前ECU所支持
0x22	27服务的使用的条件不正确，没有处于扩展会话状态下
0x24	没有经过27 01“请求种子”服务，直接就27 02“发送Key”时
0x35	27 02 发送的Key不正确（ECU验证失败）
0x36	到达了ECU验证Key失败的最大次数
0x37	到达最大败次数后，ECU会处于不准解锁的计时等待阶段，此时Tester如果再尝试解锁ECU，ECU就会返回该NRC

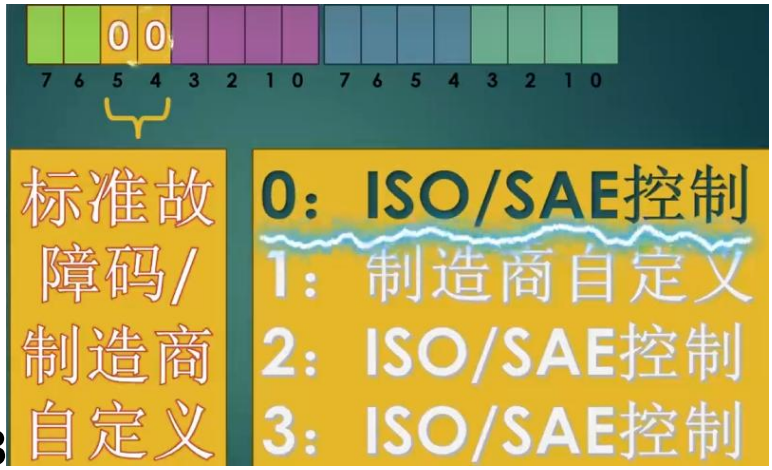
诊断故障码(5 位)



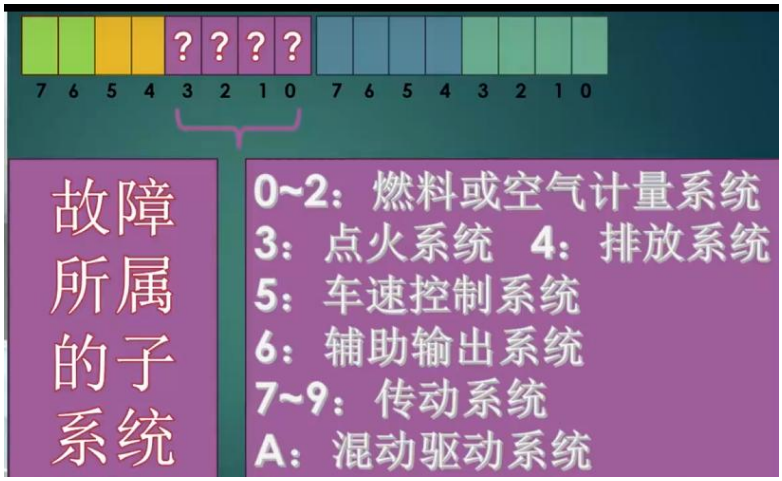




00=0　　01=1　10=2



11=3




? ? ? ? ? ? ? ?
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
发生故障的具体部件及类型
P0117：发动机水温电压信号太高
P0118：发动机水温电压信号太低
P0122：节气门位置传感器电压高
P0123：节气门位置传感器电压低

诊断故障码（3 位）